A Simple and Fast Algorithm for L1-norm Kernel PCA

Cheolmin Kim, Diego Klabjan

Abstract—We present an algorithm for L_1-norm kernel PCA and provide a convergence analysis for it. While an optimal solution of L_2-norm kernel PCA can be obtained through matrix decomposition, finding that of L_1-norm kernel PCA is not trivial due to its non-convexity and non-smoothness. We provide a novel reformulation of it through which an equivalent, geometrically interpretable problem is obtained. Based on geometric understandings, we present a "fixed-point" type algorithm that iteratively computes a binary weight for each observation. As the algorithm requires only inner products of data vectors, it is computationally efficient and the kernel trick is applicable. In the convergence analysis, we show that the algorithm converges to a local optimal solution in a finite number of steps. Moreover, we provide a rate of convergence analysis, which has never been done for any L_1-norm PCA algorithm, proving that the sequence of objective values converges to a local optimal value at a linear rate. In numerical experiments, we show the robustness of the algorithm in the presence of entry-wise perturbations and introduce an application to outlier detection where the model based on the proposed algorithm outperforms. Also, we provide a runtime comparison, attesting the scalability of the proposed algorithm.

Index Terms—Principal Component Analysis, Robustness, Kernel, Outlier Detection.

1 Introduction

Principal Component Analysis (PCA) is one of the most popular dimensionality reduction techniques [1]. Given a large set of possibly correlated features, it attempts to find a small set of features (*principal components*) that retain as much information as possible. To generate such new dimensions, it linearly transforms original features by multiplying *loading vectors* in a way that newly generated features are orthogonal and have the largest variances.

In traditional PCA, variances are measured using the L_2-norm. This has a nice property in that although the problem itself is non-convex, the optimal solution can be easily found through matrix factorization. With this property, together with its easy interpretability, PCA has been extensively used in a variety of applications. However, despite of its success, it still has some limitations. First, since it generates new dimensions through a linear combination of features, it is not able to capture non-linear relationships between features. Second, as it uses the L_2-norm for measuring variance, its solutions tend to be substantially affected by influential outliers. To overcome these limitations, the following two approaches have been proposed.

Kernel PCA The idea of kernel PCA is to map original features into a high-dimensional feature space, and perform PCA in that high-dimensional feature space [2]. With non-linear mappings, we can capture non-linear relationships among features, and this computation can be done efficiently using the *kernel trick*. With the kernel trick, computations of principal components can be done without an explicitly mapping.

L_1-norm PCA To alleviate the effects of influential observations, L_1-norm PCA uses the L_1-norm instead of the L_2-norm to measure variance. The L_1-norm is more advantageous than the L_2-norm when there are outliers having large feature values since it is less influenced by them. By utilizing this property, more robust results can be obtained through the L_1-norm based formulation in the presence of influential outliers.

In this paper, we combine the two approaches for the variance maximization version of L_1-norm PCA (which is not the same as minimizing reconstruction error with respect to the L_1-norm). In other words, we tackle a kernel version of L_1-norm PCA. Unlike L_2-norm kernel PCA, the kernel version of L_1-norm PCA is a hard problem in that it is not only non-convex but also non-smooth. However, through a reformulation, we make it a geometrically interpretable problem where the goal is to minimize the L_2-norm of a vector subject to a linear constraint involving the L_1-norm terms. For this reformulated problem, we present a "fixed point" type algorithm that iteratively computes a weight of -1 or 1 for each observation based on the kernel matrix and previous weights. We show that the kernel trick is applicable to this algorithm. Moreover, we prove the efficiency of the algorithm through a convergence analysis. We show that the proposed algorithm converges to a local optimal solution in a finite number of steps and the sequence of objective values converges to a local optimal value at a linear rate. Also, we computationally investigate the robustness of the algorithm and introduce an application to outlier detection. Lastly, we provide a runtime comparison of the proposed algorithm to other L_1-norm kernel PCA algorithms and L_2-norm kernel PCA.

Our work has the following contributions.

1. We provide a novel reformulation of L_1-norm kernel PCA to a geometrically interpretable problem and present an iterative algorithm based on geometric understandings. This framework is not specific to L_1-

• *Cheolmin Kim and Diego Klabjan are with the Department of Industrial Engineering and Management Science, Northwestern University, Evanston, IL, 60208.*
E-mail: cheolmkim@u.northwestern.edu, d-klabjan@northwestern.edu

Manuscript received August 30, 2017.

norm PCA since it can be applied to a more general problem. Particularly, the application of this framework to L_2-norm PCA results in Power iteration [3].
2. We not only prove convergence but also provide a rate of convergence analysis of the algorithm. Although many algorithms have been proposed for L_1-norm PCA, none of them provided a rate of convergence analysis. We stress that our analysis is for the kernel version which clearly provides an analysis for L_1-norm PCA. Through a novel analysis, we show that the algorithm attains a linear rate of convergence.
3. We introduce a methodology based on L_1-norm kernel PCA for outlier detection.

In what follows, we always refer to the variance maximization version of L_1-norm kernel PCA and assume that every variable in the input data is standardized with a mean of 0 and standard deviation of 1.

This paper is organized as follows. Section 2 reviews related works and points out how our work is different. Section 3 covers various formulations of L_1-norm kernel PCA. Through the reformulations, we offer geometric understandings of the problem and based on them we present an algorithm in Section 4. Section 5 gives a convergence analysis of the algorithm and experimental results are followed in Section 6.

2 Related Work

Extracting a low-rank representation from a large matrix is an important problem in machine learning and statistics. In a variety of contexts, many previous works [4], [5], [6], [7] have been proposed to address this problem. Recovering a low-rank matrix from a sampling of its entries is studied in [4]. Given that the number of sampled entries is sufficiently large, exact recovery is guaranteed with high probability by solving a simple convex optimization problem [4]. Assuming that a data matrix can be decomposed into the sum of a low-rank matrix L_0 and a sparse matrix S_0, a convex program (known as *robust PCA*) that minimizes a weighted combination of the nuclear norm of L_0 and the L_1 norm of S_0 is presented in [5]. Also, a variant of robust PCA that identifies outliers by additionally imposing a column-sparse structure on S_0 is considered in [6]. Under some mild conditions, exact recovery can be shown for both models [5], [6]. Moreover, exact recovery of mixture data is studied in [7], [8], [9], [10]. By utilizing a dictionary matrix, low-rank representation (LRR) [7] can better handle mixture data than robust PCA. While matrix recovery is the main focus of theses works, our work considers dimensionality reduction with emphasis on robustness, especially focusing on PCA with the L_1-norm.

To reduce the number of features in a robust manner, the L_1-norm has been involved in many PCA studies [11], [12], [13], [14], [15], [16], [17] or subspace estimation formulations [18], [19]. Finding a subspace onto which the L_1 projections of data vectors have the smallest sum of distances to the original data vectors is studied in [11]. Based on the observation that the L_1 projection occurs along an axis, it presents an algorithm for projecting d-dimensional data into the $(d-1)$-dimensional subspace, which minimizes the sum of distances of data vectors to their L_1 projections. In order to find the optimal axis and subspace, it solves d least absolute deviation regression problems, each having one dimension as a dependent variable while having the other dimensions as independent variables. With the use of linear programming, this algorithm finds a global optimal solution in polynomial time [11].

Minimizing reconstruction error with respect to the L_1-norm is considered in [13], [14], [18]. While the PCA problem of minimizing $\|M - XX^TM\|_1$ subject to $X^TX = I$ is considered in [13], the subspace estimation problem of minimizing $E(U,V) = \|M - UV\|_1$ is studied in [18] where M is a data matrix. In order to solve the former problem, an iterative algorithm that computes a weight for each data vector and applies L_2-norm PCA on the weighted data matrix is presented in [13]. On the other hand, the latter problem is solved using alternative convex minimization based on the observation that $E(U,V)$ becomes a convex function once U or V is known. It alternatively optimizes one matrix at a time while keeping the other one fixed, repeating this process until convergence. Also, a subspace estimation formulation that minimizes the reconstruction error with respect to the R_1-norm, $\|M - UV\|_{R_1} = \sum_{i=1}^{n} \|\mathrm{x}_i - U\mathrm{v}_i\|_2$ where the i^{th} column vector of M is x_i and that of V is v_i, is presented in [19]. This formulation minimizes the sum of reconstruction error with respect to the L_2-norm, and therefore it is different from L_2-norm PCA which minimizes the sum of squared reconstruction error with respect to the L_2-norm. Nonetheless, they share the same property in that they have a unique global solution which is rotational invariant [19].

Maximizing variance of projected data vectors with respect to the L_1-norm, which we refer to as L_1-norm PCA, is studied in [12], [15], [16], [17]. Our work also considers this formulation rather than the previous two since it has a favorable structure in that the optimal solution can be represented as a linear combination of data vectors with a weight of -1 or 1, making it possible to develop a kernel version. L_1-norm PCA is shown to be NP-hard in [17] and [16]. Nevertheless, an algorithm finding a global optimal solution is proposed in [17]. Utilizing the auxiliary-unit-vector technique [20], it computes a global optimal solution with complexity $\mathcal{O}(n^{pr+p-1})$ where n is the number of observations, r is the rank of the data matrix, and p is the desired number of principal components. Assuming r and p are fixed, the runtime of this algorithm is polynomial in n. However, if n, p, r are large, its computation time can be prohibitive. Instead of finding a global optimal solution which is intractable in general, our work focuses on developing an efficient algorithm finding a local optimal solution for L_1-norm kernel PCA.

Recognizing the hardness of L_1-norm PCA, an approximation algorithm is presented in [16] based on the known Nesterov's theorem [21]. In this work, L_1-norm PCA is relaxed to a semi-definite programming (SDP) problem and alternatively, the SDP relaxation is considered. After solving the relaxed problem, it generates a random vector and uses randomized rounding to produce a feasible solution. This randomized algorithm is a $\sqrt{2/\pi}$-approximate algorithm in expectation. To achieve this approximation guarantee with high probability, it performs randomized rounding multiple times and takes the one having the best objective

value. Rather than providing an approximation guarantee by solving a relaxed problem, our work directly considers the L_1-norm kernel PCA problem, and develops an efficient algorithm finding a local optimal solution.

Another approach utilizing a known mathematical programming model is introduced in [12]. Specifically, it proposes an iterative algorithm that solves a mixed integer programming problem in each iteration. Given an orthonormal matrix of loading vectors, it perturbs the matrix slightly in a way that the resulting matrix yields the largest objective value. After the perturbation, it uses singular value decomposition to recover orthogonality. The algorithm is completely different from the one proposed herein, since the objective values of the iterates do not necessarily improve over iterations. Our algorithm guarantees monotone convergence of the sequence of objective values as well as its linear convergence to a local optimal value.

A simple numerical algorithm finding a local optimal solution is proposed in [22]. In this work, the optimal solution is assumed to have a certain form, and weights involved in that form are updated at each iteration improving the objective values. A similar algorithm and its extended version that finds multiple loading vectors at once are derived in [15] utilizing an optimization algorithm for general L_1-norm maximization problems. In the case of linear kernel, our algorithm utilizes the same framework as the ones in [22] and [15]. However, while the algorithm in [22] is derived without any justification, we provide an understanding behind the algorithm, which is different from the derivation in [15]. Moreover, as opposed to [22] and [15], we provide a rate of convergence analysis and introduce a kernel version.

On other hand, the kernel version of L_1-norm PCA has been rarely studied. Due to the difficulty of applying the kernel trick to L_1-norm kernel PCA, an alternative method named *nonlinear projection trick* is applied to solve L_1-norm kernel PCA in [23]. Based on the finding that an optimal loading vector lies in the span of an orthonormal basis of $\Phi(A)^T U \Lambda^{-1/2}$ where $\Phi(A)$ is a high-dimensionally mapped data matrix and $U\Lambda U^T$ is the eigenvalue decomposition of the kernel matrix K, it substitutes $U\Lambda^{1/2}$ in place of $\Phi(A)$. Noting that the reformulated problem has the same form as L_1-norm PCA, it is solved by the algorithm in [22]. Another kernel extension of L_1-norm PCA is studied in [24]. In this algorithm, a linear system involving a kernel matrix is solved at each iteration and the resulting solution is used to update the iterate. The algorithms in [23] and [24] entail either eigenvalue decomposition or solving a linear system, which can be computationally costly in a large-scale setting. As opposed to them, our algorithm only requires a matrix-vector multiplication at each iteration, making it suitable in a large-scale setting.

3 KERNEL-BASED L_1-NORM PCA FORMULATIONS

We consider L_1-norm PCA in a high-dimensional feature space F. Suppose we map data vectors $a_i \in \mathbb{R}^d, i = 1, \ldots, n$ into a feature space F by a possibly non-linear mapping $\Phi : \mathbb{R}^d \to F$. Assuming $|\Phi(a_i)^T \Phi(a_j)| < \infty$ for every $i, j = 1, \ldots, n$, the kernel version of L_1-norm PCA can be formulated as follows.

$$\begin{aligned} \underset{\mathrm{x} \in F}{\text{maximize}} \quad & f(\mathrm{x}) = \sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{x}| \\ \text{subject to} \quad & \|\mathrm{x}\|_2 = 1 \end{aligned} \tag{1}$$

This formulation extends the variance maximization version of L_1-norm PCA and is also studied in other L_1-norm kernel PCA works [23], [24]. As shown in (1), we only consider extracting the first loading vector. This assumption is justifiable since subsequent loading vectors can be found by iteratively running the same algorithm. Specifically, each time a new loading vector is obtained, we update the kernel matrix K defined by $K_{ij} = \Phi(a_i)^T \Phi(a_j)$ by projecting $\Phi(a_i), i = 1, \ldots, n$ onto the space orthogonal to the most recently obtained loading vector and apply the same algorithm on the updated kernel matrix $\widetilde{K}$.

The problem (1) has a convex non-smooth objective function to maximize and the Euclidean unit ball constraint. To better understand this problem and derive an efficient algorithm, we reformulate (1) in the following way.

$$\begin{aligned} \underset{\mathrm{x} \in F}{\text{minimize}} \quad & g(\mathrm{x}) = \|\mathrm{x}\|_2 \\ \text{subject to} \quad & \sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{x}| = 1 \end{aligned} \tag{2}$$

Two optimization problems are said to be equivalent if there exists some mapping h such that if x^* is an optimal solution to one problem, then $h(\mathrm{w}^*)$ is an optimal solution to the other problem, and vice versa [25] for a possible different mapping function. Therefore, in order to prove the equivalence of (1) and (2), we show that an optimal solution of one formulation can be derived from an optimal solution of the other formulation by means of some mapping.

Proposition 1. *The followings hold.*

a) If x_1^* *is optimal to* (1)*, then*

$$\mathrm{x}_2^* = \frac{\mathrm{x}_1^*}{\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{x}_1^*|}$$

is an optimal solution to (2)*.*

b) If y_2^* *is optimal to* (2)*, then*

$$\mathrm{y}_1^* = \frac{\mathrm{y}_2^*}{\|\mathrm{y}_2^*\|_2}$$

is an optimal solution to (1)*.*

Proof. *a)* It is easy to check that x_2^* is a feasible solution to (2). Suppose that x_2^* is not optimal to (2). Then, there exists some z such that

$$\|\mathrm{z}\|_2 < \|\mathrm{x}_2^*\|_2.$$

As z is feasible to (2), we have

$$\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{z}| = 1.$$

Now, we consider $\mathrm{w} = \frac{\mathrm{z}}{\|\mathrm{z}\|_2}$. Then,

$$f(\mathrm{w}) = \sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{w}| = \frac{\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{z}|}{\|\mathrm{z}\|_2} = \frac{1}{\|\mathrm{z}\|_2}.$$

In the same way, we have

$$f(\mathrm{x}_1^*) = \frac{1}{\|\mathrm{x}_2^*\|_2}$$

from $\mathrm{x}_1^* = \frac{\mathrm{x}_2^*}{\|\mathrm{x}_2^*\|_2}$. This leads to

$$f(\mathrm{x}_1^*) < f(\mathrm{w}),$$

which contradicts the assumption that x_1^* is an optimal solution of (1).

b) Again, it is easy to check y_1^* is feasible to (1). To derive a contradiction, suppose that y_1^* is not optimal to (1). Then, there exists some w such that

$$\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{y}_1^*| < \sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{w}|.$$

Since $\|\mathrm{w}\|_2 = 1$, for

$$\mathrm{z} = \frac{\mathrm{w}}{\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{w}|},$$

we have

$$g(\mathrm{z}) = \frac{\|\mathrm{w}\|_2}{\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{w}|} = \frac{1}{\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{w}|}.$$

On the other hand,

$$\mathrm{y}_2^* = \frac{\mathrm{y}_1^*}{\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{y}_1^*|}$$

follows from $\mathrm{y}_1^* = \frac{\mathrm{y}_2^*}{\|\mathrm{y}_2^*\|_2}$ resulting in

$$g(\mathrm{y}_2^*) = \frac{1}{\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{y}_1^*|}.$$

Therefore, we obtain

$$g(\mathrm{y}_2^*) > g(\mathrm{z})$$

contradicting the assumption that y_2^* is optimal to (2). □

To understand formulation (2), we first examine the constraint set,

$$\partial P = \Big\{ \mathrm{x} | \sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{x}| = 1 \Big\}.$$

Geometrically, this constraint set is symmetric with respect to the origin and represents the boundary of the polytope

$$P = \Big\{ \mathrm{x} | \sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{x}| \leq 1 \Big\}.$$

It is easy to check that P is a polytope as it can be represented by the intersection of a finite set of linear inequalities each having the form of $\sum_{i=1}^{n} c_i \Phi(a_i)^T \mathrm{x} \leq 1$ where $c_i \in \{-1, 1\}$. Therefore, formulation (2) can be understood as a problem of finding the closest point to the origin from the boundary of the polytope ∂P. The following proposition proves that an optimal solution x^* must be perpendicular to one of the faces of P.

Proposition 2. *An optimal solution* x^* *is perpendicular to the face which it lies on.*

Proof. Suppose that an optimal solution of (2) is x^*. Letting

$$c_i^* = \mathrm{sgn}(\Phi(a_i)^T \mathrm{x}^*) = \begin{cases} 1, & \text{if } \Phi(a_i)^T \mathrm{x}^* \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

for $i = 1, \ldots, n$, we consider the face

$$E = \Big\{ \mathrm{x} | \sum_{i=1}^{n} c_i^* \Phi(a_i)^T \mathrm{x} = 1 \Big\} \cap P.$$

If x^* is not perpendicular to face E, then

$$\mathrm{w} = \frac{\sum_{i=1}^{n} \Phi(a_i) c_i^*}{\| \sum_{i=1}^{n} \Phi(a_i) c_i^* \|_2^2}$$

is the closest point to the origin from

$$\Big\{ \mathrm{x} | \sum_{i=1}^{n} c_i^* \Phi(a_i)^T \mathrm{x} = 1 \Big\}$$

having $\|\mathrm{w}\|_2 < \|\mathrm{x}^*\|_2$. Now, let us define its scalar multiple

$$\mathrm{z} = \frac{\mathrm{w}}{\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{w}|}.$$

By construction, z is a feasible solution to (2) and has the objective value of

$$g(\mathrm{z}) = \frac{\|\mathrm{w}\|_2}{\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{w}|}.$$

From

$$\begin{aligned}
&\sum_{i=1}^{n} |\Phi(a_i)^T (\sum_{j=1}^{n} \Phi(a_j) c_j^*)| - \| \sum_{i=1}^{n} \Phi(a_i) c_i^* \|_2^2 \\
&= \sum_{i=1}^{n} |\Phi(a_i)^T (\sum_{j=1}^{n} \Phi(a_j) c_j^*)| - \sum_{i=1}^{n} \Phi(a_i)^T c_i^* (\sum_{j=1}^{n} \Phi(a_j) c_j^*) \\
&= \sum_{i=1}^{n} \Big[|\Phi(a_i)^T (\sum_{j=1}^{n} \Phi(a_j) c_j^*)| - \Phi(a_i)^T c_i^* (\sum_{j=1}^{n} \Phi(a_j) c_j^*) \Big] \\
&\geq 0,
\end{aligned}$$

we have

$$\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{w}| = \frac{\sum_{i=1}^{n} |\Phi(a_i)^T (\sum_{j=1}^{n} \Phi(a_j) c_j^*)|}{\| \sum_{i=1}^{n} \Phi(a_i) c_i^* \|_2^2} \geq 1.$$

As a result,

$$g(\mathrm{z}) \leq \|\mathrm{w}\|_2 < \|\mathrm{x}^*\|_2$$

follows. This contradicts the assumption that x^* is an optimal solution to (2). Therefore, an optimal solution must be perpendicular to face E. □

Proposition 2 is important since it provides a way to characterize the form of an optimal solution. Specifically, we obtain the following corollary from Proposition 2.

Corollary 1. *An optimal solution* x^* *of* (2) *must have the form of*

$$\mathrm{x}^* = \frac{\mathrm{y}^*}{\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{y}^*|}$$

for some y^* *and* c^* *such that*

$$\mathrm{y}^* = \sum_{i=1}^{n} \Phi(a_i) c_i^*$$

and

$$c_i^* = sgn(\Phi(a_i)^T \mathrm{y}^*),$$

for $i = 1, ..., n$.

The characterization of an optimal solution x^* using a sign vector $\mathrm{c}^* = [c_1, \ldots, c_n]^T$ is first proposed in [22] without any justification. We provide a derivation based on the geometry of the set ∂P, which is different from the one in [15] utilizing the KKT conditions. Moreover, from

$$\begin{aligned}\|\mathrm{x}^*\|_2 &= \frac{\|\mathrm{y}^*\|_2}{\sum_{i=1}^n |\Phi(a_i)^T \mathrm{y}^*|} \\ &= \frac{\|\mathrm{y}^*\|_2}{\sum_{i=1}^n c_i^* \Phi(a_i)^T \mathrm{y}^*} \\ &= \frac{\|\sum_{i=1}^n \Phi(a_i) c_i^*\|_2}{\sum_{i=1}^n \sum_{j=1}^n c_i^* c_j^* \Phi(a_i)^T \Phi(a_j)} \\ &= \frac{\|\sum_{i=1}^n \Phi(a_i) c_i^*\|_2}{\|\sum_{i=1}^n \Phi(a_i) c_i^*\|_2^2} \\ &= \frac{1}{\|\sum_{i=1}^n \Phi(a_i) c_i^*\|_2}, \end{aligned} \tag{3}$$

we can further show that an optimal solution of formulation (2) can be derived from an optimal solution of the following binary problem,

$$\underset{\mathrm{c}\in\{-1,1\}^n}{\text{maximize}} \quad \|\sum_{i=1}^n \Phi(a_i) c_i\|_2^2. \tag{4}$$

Proposition 3. *Let an optimal solution of binary formulation* (4) *be* c^*. *Then,*

$$y^* = \sum_{i=1}^n \Phi(a_i) c_i^*$$

satisfies

$$c_i^* = sgn(\Phi(a_i)^T y^*),$$

for $i = 1, \ldots, n$. *Moreover, it follows that*

$$x^* = \frac{y^*}{\sum_{i=1}^n |\Phi(a_i)^T y^*|)}$$

is an optimal solution of formulation (2).

Proof. To deduce a contradiction, let us assume that there exists some nonempty set $J \subset \{1, \ldots, n\}$ such that

$$c_j^* = -\mathrm{sgn}(\Phi(a_j)^T \mathrm{y}^*)$$

for $j \in J$. Since c^* is an optimal solution of (4), flipping the sign of c_j^* for any $j \in J$ must not improve the objective value, $\|\sum_{i=1}^n \Phi(a_i) c_i^*\|_2^2$. However, for any $j \in J$, flipping the sign of c_j^* gives

$$\|\sum_{i\neq j}^n \Phi(a_i) c_i^* - \Phi(a_j) c_j^*\|_2^2 > \|\sum_{i=1}^n \Phi(a_i) c_i^*\|_2^2.$$

The above strict inequality follows from

$$\sum_{i\neq j}^n \Phi(a_i) c_i^* - \Phi(a_j) c_j^* = \mathrm{y}^* - 2\Phi(a_j) c_j^*$$

and

$$\begin{aligned}\|\mathrm{y}^* - 2\Phi(a_j) c_j^*\|_2^2 &= \|\mathrm{y}\|_2^2 - 4\mathrm{y}^T\big(\Phi(a_j) c_j^*\big) + 4\|\Phi(a_j)\|_2^2 \\ &= \|\mathrm{y}\|_2^2 + 4|\mathrm{y}^T(\Phi(a_j))| + 4\|\Phi(a_j)\|_2^2 \\ &> \|\mathrm{y}\|_2^2. \end{aligned}$$

This contradicts the assumption that c^* is an optimal solution to (4). Therefore, y^* must satisfy

$$c_i^* = \mathrm{sgn}(\Phi(a_i)^T \mathrm{y}^*)$$

for $i = 1, \ldots, n$. Since c^* maximizes $\|\sum_{i=1}^n \Phi(a_i) c_i^*\|_2^2$,

$$\mathrm{x}^* = \frac{\mathrm{y}^*}{\sum_{i=1}^n |\Phi(a_i)^T \mathrm{y}^*|}$$

is a minimizer of (2) due to Corollary 1 and (3). □

The following result has been shown in [17] for the linear kernel case but here we generalize it.

Corollary 2. *Formulation* (2) *is equivalent to formulation* (4).

Proof. Based on Corollary 1 and (3), we can formulate (2) as

$$\begin{aligned} \underset{\mathrm{c}\in\{-1,1\}^n}{\text{maximize}} \quad & \|\sum_{i=1}^n \Phi(a_i) c_i\|_2^2 \\ \text{subject to} \quad & \mathrm{y} = \sum_{i=1}^n \Phi(a_i) c_i \\ & c_i = \mathrm{sgn}(\Phi(a_i)^T \mathrm{y}), i = 1, ..., n. \end{aligned}$$

Since an optimal solution c^* to (4) satisfies the constraints by Proposition 3, the two formulations are essentially the same. □

It is interesting to note that we can reduce formulation (4) to the weighted max-cut problem. From

$$\|\sum_{i=1}^n \Phi(a_i) c_i\|_2^2 = \sum_{i,j=1}^n K_{ij} + \sum_{i,j=1}^n (-2K_{ij})\Big(\frac{1 - c_i c_j}{2}\Big), \tag{5}$$

we can alternatively consider the weighted max-cut problem on a complete graph with weight $w_{ij} = -K_{ij}$. From the above reduction, we can apply a popular approximation algorithm [26] for the weighted max-cut problem to solve (4). However, due to the additional constant terms in (5), this does not imply a constant worst case approximation ratio algorithm for (4).

4 ALGORITHM

In this section, we develop an efficient algorithm that finds a local optimal solution to problem (1) based on the findings in Section 3. Before giving the details of the algorithm, we first provide an idea behind the algorithm.

The main idea of the algorithm is to move along the boundary of P so that the L_2-norm of an iterate x_k successively decreases. Figure 1 illustrates a step of the algorithm. Starting with an iterate x^k, we first identify a hyperplane h^k which the current iterate x^k lies on. After identifying the equation of h^k, we find the closest point to the origin from h^k, which we denote by z^k. After that, we obtain x^{k+1} by projecting z^k to the constraint set ∂P, which is done by multiplying an appropriate scalar. We repeat this process until the sequence of iterates $\{\mathrm{x}^k\}$ converges.

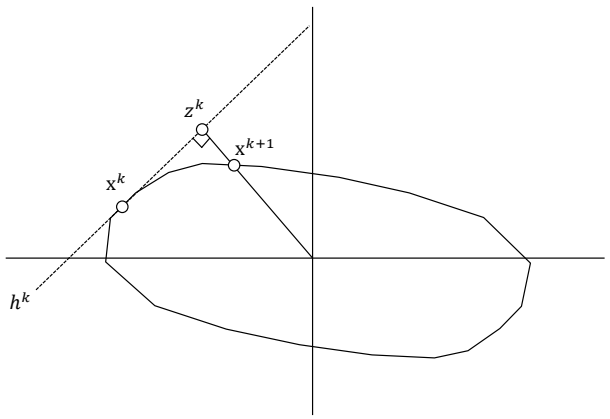


Fig. 1. Geometric interpretation of the algorithm

Now, we develop an algorithm based on the above idea. Let $K_{ij} = \Phi(a_i)^T\Phi(a_j)$. From Corollary 1, we know that an optimal solution x^* has the form of

$$\begin{aligned} x^* &= \frac{y^*}{\sum_{i=1}^n |\Phi(a_i)^T y^*|} \\ &= \frac{y^*}{\sum_{i=1}^n c_i \Phi(a_i)^T y^*} \\ &= \frac{\sum_{i=1}^n \Phi(a_i) c_i^*}{\sum_{i=1}^n \sum_{j=1}^n c_i c_j \Phi(a_i)^T \Phi(a_j)} \\ &= \frac{\sum_{i=1}^n \Phi(a_i) c_i^*}{(c^*)^T K c^*}. \end{aligned}$$

Noting that the optimal solution x^* can be characterized by the sign vector c^*, we characterize the initial iterate x^0 with the sign vector c^0 as

$$x^0 = \frac{\sum_{i=1}^n \Phi(a_i) c_i^0}{(c^0)^T K c^0}.$$

Since

$$y^k = \sum_{i=1}^n \Phi(a_i) c_i^k, \tag{6}$$

we can represent the equation of the hyperplane h^k by

$$(y^k)^T(x - x^k) = 0. \tag{7}$$

The closest point z^k to the origin among points in h^k has the form of $z^k = sy^k$. By plugging $z^k = sy^k$ into (7), we have

$$s = \frac{(y^k)^T(x^k)}{(y^k)^T(y^k)}, \quad z^k = \frac{(y^k)^T(x^k)}{(y^k)^T(y^k)} y^k.$$

Dividing z^k by $\sum_{i=1}^n |\Phi(a_i)^T z^k|$, we obtain

$$x^{k+1} = \frac{z^k}{\sum_{i=1}^n |\Phi(a_i)^T z^k|}. \tag{8}$$

From

$$(y^k)^T(x^k) = \sum_{i=1}^n \Phi(a_i)^T x^k c_i^k = \sum_{i=1}^n |\Phi(a_i)^T x^k| = 1, \tag{9}$$

we get

$$z^k = \frac{y^k}{\|y^k\|_2^2}, \tag{10}$$

$$x^{k+1} = \frac{y^k}{\sum_{i=1}^n |\Phi(a_i)^T y^k|}. \tag{11}$$

By plugging (6) into (11), we finally represent x^{k+1} as

$$x^{k+1} = \frac{y^k}{\sum_{i=1}^n |\Phi(a_i)^T y^k|} = \frac{\sum_{i=1}^n \Phi(a_i) c_i^k}{(c^k)^T K c^k}. \tag{12}$$

As x^{k+1} is a function of c^k, it is only necessary to update c^k at each iteration. From

$$\begin{aligned} c_i^{k+1} &= \mathrm{sgn}((\Phi(a_i))^T x^{k+1}) \\ &= \mathrm{sgn}((\Phi(a_i))^T y^k) \\ &= \mathrm{sgn}\Big(\sum_{j=1}^n K_{ij} c_j^k\Big), \end{aligned}$$

we can update c^{k+1} by

$$c^{k+1} = \mathrm{sgn}(Kc^k).$$

From

$$\|x^{k+1} - x^k\|_2^2 = 0 \iff (c^k - c^{k+1})^T K (c^k - c^{k+1}) = 0,$$

we get the termination criteria

$$(c^k - c^{k+1})^T K (c^k - c^{k+1}) = 0,$$

as the update needs to be repeated until $x^{k+1} = x^k$.

As the problem is non-convex, the algorithm can be stuck at a local optimum unless it is initialized close to the global optimum. In order to obtain a good initial iterate c^0, we consider each data vector a_j and select the one where $\Phi(a_j)/\|\Phi(a_j)\|_2$ yields the largest objective value, which is computed by

$$\frac{\Sigma_{i=1}^n |\Phi(a_i)^T \Phi(a_j)|}{\|\Phi(a_j)\|_2} = \frac{\Sigma_{i=1}^n |K_{ij}|}{\sqrt{K_{jj}}}.$$

Once we obtain the optimal data vector a_{j^*}, we set the initial sign vector

$$c^0 = \mathrm{sgn}(K_{\cdot j^*})$$

where $K_{\cdot j}$ represents j^{th} column-vector of matrix K. Since the optimal loading vector must be located somewhere between data vectors, the above initialization scheme often yields an initial iterate near the optimal solution.

Summarizing all the above, we get Algorithm 1.

Algorithm 1 L_1-norm Kernel PCA

Input: data vectors a_i, kernel matrix $K_{ij} = \Phi(a_i)^T\Phi(a_j)$
Find $j^* = \arg\max_{1 \le j \le n} \Sigma_{i=1}^n |K_{ij}|/\sqrt{K_{jj}}$
Initialize the sign vector c^0 as $c_i^0 = \mathrm{sgn}(K_{ij^*})$
$k \leftarrow -1$
repeat
  $k \leftarrow k + 1$
  Compute $c^{k+1} = \mathrm{sgn}(Kc^k)$
until $(c^k - c^{k+1})^T K (c^k - c^{k+1}) = 0$

After getting the output c^* from Algorithm 1, we can compute principal scores without explicit mapping $\Phi(a_i)$.

For example, the principal component of i^{th} observation can be computed by

$$\begin{aligned}\frac{\Phi(a_i)^T \mathrm{x}^*}{\|\mathrm{x}^*\|_2} &= \frac{\Phi(a_i)^T \mathrm{y}^*}{\|\mathrm{y}^*\|_2} \\ &= \frac{\sum_{j=1}^n \Phi(a_i)^T \Phi(a_j) c_j^*}{\sqrt{\sum_{i=1}^n \sum_{j=1}^n \Phi(a_i)^T \Phi(a_j) c_i^* c_j^*}} \\ &= \frac{K_{i\cdot}\mathrm{c}^*}{\sqrt{(\mathrm{c}^*)^T K \mathrm{c}^*}}.\end{aligned}$$

where $K_{i\cdot}$ represents the i^{th} row vector of the kernel matrix K. We can further proceed to find more principal components without explicit mapping $\Phi(a_i)$. As computing a loading vector and principal components only require the kernel matrix, it suffices to update the kernel matrix each time a new loading vector is found. We can update the kernel matrix without explicit mapping $\Phi(a_i)$ by

$$\begin{aligned}\widetilde{K}_{ij} &= \left(\Phi(a_i) - \frac{\Phi(a_i)^T \mathrm{x}^*}{\|\mathrm{x}^*\|_2^2}\mathrm{x}^*\right)^T \left(\Phi(a_j) - \frac{\Phi(a_j)^T \mathrm{x}^*}{\|\mathrm{x}^*\|_2^2}\mathrm{x}^*\right) \\ &= \Phi(a_i)^T\Phi(a_j) - \frac{\Phi(a_i)^T \mathrm{x}^* \Phi(a_j)^T \mathrm{x}^*}{\|\mathrm{x}^*\|_2^2} \\ &= K_{ij} - \frac{\left(\sum_{k=1}^n K_{i,k} c_k^*\right)\left(\sum_{k=1}^n K_{j,k} c_k^*\right)}{\left(\sum_{i=1}^n \sum_{j=1}^n K_{ij} c_i^* c_j^*\right)} \\ &= K_{ij} - \frac{K_{i\cdot}\mathrm{c}^* K_{j\cdot}\mathrm{c}^*}{(\mathrm{c}^*)^T K \mathrm{c}^*},\end{aligned}$$

which is equivalent to

$$\widetilde{K} = K - \frac{(K\mathrm{c}^*)(K\mathrm{c}^*)^T}{(\mathrm{c}^*)^T K \mathrm{c}^*}$$

in matrix form.

Since $\mathrm{y}_k = \nabla f(\mathrm{x}_k)$, update rule (11) can be understood as projecting a gradient $\nabla f(\mathrm{x}_k)$ to the constraint set ∂P in each iteration. In this sense, Algorithm 1 resembles Power iteration [3] for solving the eigenvalue problem, and interestingly, the application of our framework to the eigenvalue problem yields the same algorithm. Moreover, as the framework developed in this work such as the reformulation, geometric interpretation and algorithm derivation is not specific to our problem, we can actually extend it to solve more general problems. For example, our approach can be used to solve

$$\text{maximize} \quad f(\mathrm{x}) \quad \text{subject to} \quad \|\mathrm{x}\|_2 = 1$$

for any function f that is scale-invariant (*homogeneous* or *homothetic*). The application of our framework to this problem yields the following update rule

$$\mathrm{x}^{k+1} \leftarrow \nabla f(\mathrm{x}^k)/\|\nabla f(\mathrm{x}^k)\|_2.$$

Compared to the other L_1-norm kernel PCA algorithms [23], [24], which consider the same formulation (1), Algorithm 1 is simpler and computationally efficient as it involves just one matrix-vector multiplication in each iteration. In the case of L1-KPCA [24], it requires to solve a system of linear equations having the form of

$$K\eta = \Sigma_{j=1}^n \mathrm{c}_j^k K_{\cdot j}$$

in addition to one matrix-vector multiplication. Solving the above linear system is not only computationally costly but also numerically unstable since it is singular due to the presence of non-trivial solution c^k. On the other hand, KPCA-L1 [23] requires only one matrix-vector multiplication but it does not directly utilize the kernel matrix K. Instead, the eigenvalue decomposition of the kernel matrix $K = U\Lambda U^T$ is computed before finding each principal component and $U\Lambda^{1/2}$ is considered in the computation rather than the kernel matrix K. As Algorithm 1 entails neither solving a linear system nor computing the eigenvalue decomposition of K, it is computationally more efficient than the other algorithms.

When it comes to the initial starting point, the optimal solution of L_2-norm kernel PCA is chosen by L1-KPCA [24]. While KPCA-L1 [23] finds the data vector having the largest norm and let its normalized vector be the initial starting point, Algorithm 1 sets the data vector whose normalization has the largest objective value to be the initial point. As our initialization scheme is based on the objective function while the others are not, it is more likely to be initialized near a global optimal solution.

5 Convergence Analysis

In this section, we provide a convergence analysis of Algorithm 1. We first prove that the algorithm converges in a finite number of iterations, and then provide a rate of convergence analysis. Before proving the finite convergence of the algorithm, we first show that the sequence $\{\|\mathrm{x}^k\|_2\}$ generated by Algorithm 1 is non-increasing.

Lemma 1. *Let $\{\mathrm{x}_k\}$ be a sequence of iterates generated by Algorithm 1 and $\{\mathrm{z}_k\}$ be a sequence of vectors defined by* (10)*. Then, we have $\|\mathrm{x}^{k+1}\|_2 \le \|\mathrm{z}^k\|_2 \le \|\mathrm{x}^k\|_2$.*

Proof. The inequality $\|\mathrm{z}^k\|_2 \le \|\mathrm{x}^k\|_2$ follows from

$$\begin{aligned}\|\mathrm{x}^k\|_2^2 - \|\mathrm{z}^k\|_2^2 &= \|\mathrm{x}^k\|_2^2 - \frac{1}{\|\mathrm{y}^k\|_2^2} \\ &= \|\mathrm{x}^k\|_2^2 - \frac{((\mathrm{y}^k)^T(\mathrm{x}^k))^2}{\|\mathrm{y}^k\|_2^2} \\ &= \frac{\|\mathrm{x}^k\|_2^2\|\mathrm{y}^k\|_2^2 - ((\mathrm{y}^k)^T(\mathrm{x}^k))^2}{\|\mathrm{y}^k\|_2^2} \\ &\ge 0. \qquad (13)\end{aligned}$$

Here, the second equality is from (9) and the last inequality follows from Cauchy-Schwarz inequality where the equality

holds if x^k is a scalar multiple of y^k. Next, from

$$\begin{aligned}\sum_{i=1}^{n}|\Phi(a_i)^T\mathrm{z}^k| &= \sum_{i=1}^{n}\frac{|\Phi(a_i)^T(\mathrm{y}^k)|}{(\mathrm{y}^k)^T(\mathrm{y}^k)}\\ &= \frac{1}{(\mathrm{y}^k)^T(\mathrm{y}^k)}\sum_{i=1}^{n}|\Phi(a_i)^T\mathrm{y}^k|\\ &= \frac{\sum_{i=1}^{n}|\Phi(a_i)^T(\sum_{j=1}^{n}\Phi(a_j)c_j^k)|}{\sum_{i=1}^{n}\sum_{j=1}^{n}\Phi(a_i)^T\Phi(a_j)c_i^kc_j^k}\\ &= \frac{\sum_{i=1}^{n}|(\Phi(a_i)c_i^k)^T(\sum_{j=1}^{n}\Phi(a_j)c_j^k)|}{\sum_{i=1}^{n}\sum_{j=1}^{n}\Phi(a_i)^T\Phi(a_j)c_i^kc_j^k}\\ &= \frac{\sum_{i=1}^{n}|\sum_{j=1}^{n}\Phi(a_i)^T\Phi(a_j)c_i^kc_j^k|}{\sum_{i=1}^{n}\sum_{j=1}^{n}\Phi(a_i)^T\Phi(a_j)c_i^kc_j^k} \quad (14)\\ &\geq 1, \quad (15)\end{aligned}$$

we have

$$\|\mathrm{x}^{k+1}\|_2^2 = \frac{\|\mathrm{z}^k\|_2^2}{(\sum_{i=1}^{n}|\Phi(a_i)^T\mathrm{z}^k|)^2} \leq \|\mathrm{z}^k\|_2^2.$$

□

Lemma 2. *If*

$$\|\mathrm{x}^k\|_2 = \|\mathrm{x}^{k+1}\|_2,$$

then, we have

$$\mathrm{x}^k = \frac{\mathrm{y}^k}{\|\mathrm{y}^k\|_2^2},$$
$$\mathrm{y}^k = \frac{\mathrm{x}^k}{\|\mathrm{x}^k\|_2^2},$$

resulting in

$$\mathrm{x}^k = \mathrm{x}^{k+1}.$$

Proof. From Lemma 1, we have $\|\mathrm{z}^k\|_2 = \|\mathrm{x}^k\|_2$. Then, by (13), x^k is a scalar multiple of y^k. Letting $\mathrm{x}^k = r\mathrm{y}^k$, we have

$$r = \frac{1}{\|\mathrm{y}^k\|_2^2}$$

from (9) resulting in

$$\mathrm{x}^k = \frac{\mathrm{y}^k}{\|\mathrm{y}^k\|_2^2}.$$

In the same way, we are able to show

$$\mathrm{y}^k = \frac{\mathrm{x}^k}{\|\mathrm{x}^k\|_2^2}.$$

Since $\mathrm{z}^k = \mathrm{x}^k$ holds by (10), we have

$$\begin{aligned}\mathrm{x}^{k+1} &= \frac{\mathrm{z}^k}{\sum_{i=1}^{n}|\Phi(a_i)^T\mathrm{z}^k|}\\ &= \frac{\mathrm{x}^k}{\sum_{i=1}^{n}|\Phi(a_i)^T\mathrm{x}^k|}\\ &= \mathrm{x}^k\end{aligned}$$

where the last equality follows from the feasibility of x^k. □

Theorem 1. *The sequence $\{\mathrm{x}^k\}$ converges in a finite number of steps.*

Proof. Suppose the sequence $\{\mathrm{x}^k\}$ does not converge. As an iterate x^k is solely determined by a sign vector $\mathrm{c}^k \in \{-1,+1\}^n$, the number of possible vectors that x^k can take is finite. Therefore, if the sequence $\{\mathrm{x}^k\}$ does not converge, some vectors must appear more than once. Without loss of generality, let $\mathrm{x}^l = \mathrm{x}^{l+m}$. By Lemma 1, we have

$$\|\mathrm{x}^{l+m}\|_2 = \|\mathrm{x}^l\|_2 \geq \|\mathrm{x}^{l+1}\|_2 \geq ... \geq \|\mathrm{x}^{l+m}\|_2$$

forcing us to have

$$\|\mathrm{x}^l\|_2 = \|\mathrm{x}^{l+1}\|_2 = ... = \|\mathrm{x}^{l+m}\|_2.$$

As this implies

$$\mathrm{x}^l = \mathrm{x}^{l+1} = ... = \mathrm{x}^{l+m}$$

by Lemma 2, we get a contradiction to the assumption that the sequence $\{\mathrm{x}^k\}$ does not converge. Therefore, the sequence $\{\mathrm{x}^k\}$ generated by Algorithm 1 must converge in a finite number of steps. □

Next, we show that the sequence of objective values $\{\|\mathrm{x}^k\|_2\}$ generated by Algorithm 1 converges at a linear rate. While finite convergence ensures that the algorithm converges in a finite number of steps, due to the combinatorial structure of the problem, it may take an exponential number of steps to converge, making it not appropriate in a large-scale setting. To address this issue, we additionally prove linear convergence of the algorithm. Linear convergence ensures that the optimality gap decreases no worse than a certain rate $\rho < 1$ and this implies that an ϵ-optimal local solution can be attained after $\mathcal{O}(\frac{1}{1-\rho}\log(1/\epsilon))$ iterations. With linear convergence, we are ensured to obtain a near-optimal solution after a sufficient number of iterations without waiting for an exponential number of steps.

Theorem 2. *Let Algorithm 1 start from x^0 and terminate with x^* at iteration k^*. Then, for some $\rho < 1$, we have*

$$\|\mathrm{x}^k\|_2 - \|\mathrm{x}^*\|_2 \leq \rho^k(\|\mathrm{x}^0\|_2 - \|\mathrm{x}^*\|_2)$$

for $k < k^*$.

Proof. From (8), we have

$$\|\mathrm{x}^k\|_2 = \frac{\|\mathrm{z}^{k-1}\|_2}{\sum_{i=1}^{n}|\Phi(a_i)^T\mathrm{z}^{k-1}|}.$$

As $\|\mathrm{z}^{k-1}\|_2 \leq \|\mathrm{x}^{k-1}\|_2$ holds by Lemma 1, we obtain

$$\|\mathrm{x}^k\|_2 \leq \frac{\|\mathrm{x}^{k-1}\|_2}{\sum_{i=1}^{n}|\Phi(a_i)^T\mathrm{z}^{k-1}|}. \quad (16)$$

Subtracting $\|\mathrm{x}^*\|_2$ to (16), we obtain

$$\begin{aligned}\|\mathrm{x}^k\|_2 - \|\mathrm{x}^*\|_2 &\leq \frac{\|\mathrm{x}^{k-1}\|_2}{\sum_{i=1}^{n}|\Phi(a_i)^T\mathrm{z}^{k-1}|} - \|\mathrm{x}^*\|_2\\ &\leq \frac{1}{\sum_{i=1}^{n}|\Phi(a_i)^T\mathrm{z}^{k-1}|}(\|\mathrm{x}^{k-1}\|_2 - \|\mathrm{x}^*\|_2) \quad (17)\end{aligned}$$

where the last inequality follows from (15). By induction on (17), we have

$$\|\mathrm{x}^k\|_2 - \|\mathrm{x}^*\|_2 \leq (\|\mathrm{x}^0\|_2 - \|\mathrm{x}^*\|_2)\prod_{l=1}^{k}\frac{1}{\sum_{i=1}^{n}|\Phi(a_i)^T\mathrm{z}^{l-1}|}. \quad (18)$$

From (15), we know that

$$\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{z}^{l-1}| \geq 1.$$

If

$$\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{z}^{l-1}| = 1,$$

by (14), we have

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \Phi(a_i)^T \Phi(a_j) c_i^{l-1} c_j^{l-1} = \sum_{i=1}^{n} |\sum_{j=1}^{n} \Phi(a_i)^T \Phi(a_j) c_i^{l-1} c_j^{l-1}|$$

resulting in

$$c_i^{l-1} = \mathrm{sgn}\Big(\sum_{j=1}^{n} \Phi(a_i)^T \Phi(a_j) c_j^{l-1}\Big).$$

As this implies

$$\mathrm{c}^l = \mathrm{sgn}(K\mathrm{c}^{l-1}) = \mathrm{c}^{l-1},$$

we have

$$\mathrm{x}^l = \mathrm{x}^{l+1}$$

from (12). Therefore, as long as $l < k^*$, we must have

$$\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{z}^{j-1}| > 1.$$

For $\mathrm{c} \in \{-1, 1\}^n$, let

$$\rho(\mathrm{c}) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} \Phi(a_i)^T \Phi(a_j) c_i c_j}{\sum_{i=1}^{n} |\sum_{j=1}^{n} \Phi(a_i)^T \Phi(a_j) c_i c_j|}$$

and

$$\rho = \max_{\mathrm{c}\in\{-1,1\}^n} \rho(\mathrm{c}) \text{ subject to } \rho(\mathrm{c}) < 1.$$

Then, for $l < k^*$, we have

$$\frac{1}{\sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{z}^{j-1}|} = \rho(\mathrm{c}^{j-1}) < \rho < 1$$

where the first equality follows form (14). By combining it with (18), we get the desired result. □

As shown in Theorem 2, no matter where the algorithm starts, the sequence of objective values converges at a linear rate. Now, we show that we can obtain a local optimal solution of (1) by scaling the output of Algorithm 1.

Theorem 3. *Let the output of Algorithm 1 be* x^*. *Then,*

$$\bar{\mathrm{x}}^* = \frac{\mathrm{x}^*}{\|\mathrm{x}^*\|_2}$$

is a local optimal solution of (1).

Proof. It is easy to see that $\bar{\mathrm{x}}^*$ is feasible. Since x^* is the output of Algorithm 1,

$$\mathrm{y}^* = \frac{\mathrm{x}^*}{\|\mathrm{x}^*\|_2^2}$$

holds by Lemma 2. Next, consider

$$L(\lambda, \mathrm{x}) = \sum_{i=1}^{n} |\Phi(a_i)^T \mathrm{x}| - \lambda(\|\mathrm{x}\|_2^2 - 1).$$

From

$$\nabla_{\mathrm{x}} L(\lambda, \mathrm{x}) = \sum_{i=1}^{n} \mathrm{sgn}(\Phi(a_i)^T \mathrm{x})\Phi(a_i) - 2\lambda \mathrm{x},$$

we have

$$\begin{aligned}
\nabla_{\mathrm{x}} L(\lambda, \bar{\mathrm{x}}^*) &= \sum_{i=1}^{n} \mathrm{sgn}(\Phi(a_i)^T \bar{\mathrm{x}}^*)\Phi(a_i) - 2\lambda \bar{\mathrm{x}}^* \\
&= \sum_{i=1}^{n} \mathrm{sgn}(\Phi(a_i)^T \mathrm{x}^*)\Phi(a_i) - 2\lambda \bar{\mathrm{x}}^* \\
&= \mathrm{y}^* - 2\lambda \bar{\mathrm{x}}^* \\
&= \frac{\mathrm{x}^*}{\|\mathrm{x}^*\|_2^2} - 2\lambda \bar{\mathrm{x}}^* \\
&= \Big(\frac{1}{\|\mathrm{x}^*\|_2} - 2\lambda\Big)\bar{\mathrm{x}}^*.
\end{aligned}$$

Therefore, with

$$\lambda^* = \frac{1}{2\|\mathrm{x}^*\|_2},$$

we have

$$\nabla_{\mathrm{x}} L(\lambda^*, \bar{\mathrm{x}}^*) = 0$$

meaning that $(\lambda^*, \bar{\mathrm{x}}^*)$ satisfies the first-order necessary conditions. Moreover, from

$$\nabla_{\mathrm{xx}} L(\lambda^*, \bar{\mathrm{x}}^*) = -2\lambda^* I \prec 0,$$

the second-order sufficient condition is also satisfied. As $(\lambda^*, \bar{\mathrm{x}}^*)$ satisfies the first and second order conditions, by the theory of constrained optimization, $\bar{\mathrm{x}}^*$ is a local optimal solution of (1). □

6 Experimental Results

In this section, we assess the robustness and scalability of Algorithm 1 by running it on several tasks together with other kernel PCA algorithms. First, we run the kernel PCA algorithms on datasets having entry-wise perturbations and investigate how well each algorithm extracts principal components in a noisy setting. Next, we introduce an outlier detection model based on kernel PCA algorithms and compare their performance with other popular outlier detection models. Lastly, we provide a runtime comparison to assess scalability.

In addition to Algorithm 1, the two other L_1-norm kernel PCA algorithms (KPCA-L1 [23], L1-KPCA [24]), the kernel version of R_1-norm PCA (R1-KPCA [19]) and L_2-norm kernel PCA (L2-KPCA [2]) are considered in the experiments. While R_1-norm PCA [19] was not originally designed to incorporate kernels, we include it as it is straightforward to develop a kernel variant. We considered other L_1-norm PCA algorithms as well but they are excluded as it is not simple to develop a kernel version for them.

6.1 Robust Extraction of PCs

To measure robustness, we first run kernel PCA algorithms on datasets having entry-wise perturbations (noisy datasets) to obtain loading vectors. After that, we compute how much variation in the perturbation-excluded datasets (normal datasets) is explained by the loading vectors obtained

from the noisy datasets. For this experiment, we generate synthetic datasets that contain entry-wise perturbations in a way that the loading vectors obtained by running L_2-norm kernel PCA on noisy and normal datasets are different from each other.

To generate synthetic datasets, we first construct a 1000×50 data matrix with the rank of 10 following the data generation procedure in [13]. While the largest size in [13] is 300×50, we choose the size of 1000×50 to consider larger datasets. In order to generate entry-wise perturbations, we corrupt $r\%$ of observations by adding some random noises. We refer to the resulting dataset as a noisy dataset and the noisy dataset without the entry-wise perturbations as a normal dataset. For each value of $r \in \{5, 10, 15, 20, 25, 30\}$, we generate 10 instances.

Let K denote a kernel matrix of a normal dataset and $\mathrm{x}_1, \ldots, \mathrm{x}_p$ be the p loading vectors obtained by running L_2-norm kernel PCA on K. Also, let $\tilde{K}$ be a kernel matrix of a noisy dataset and $\tilde{\mathrm{x}}_1, \ldots, \tilde{\mathrm{x}}_p$ be the loading vectors obtained by running one of the kernel PCA algorithms (Algorithm 1, KPCA-L1, L1-KPCA, R1-KPCA, L2-KPCA) on $\tilde{K}$. Assuming that the normal dataset is standardized,

$$\sum_{j=1}^{p}\sum_{i=1}^{n}(\Phi(a_i)^T\tilde{\mathrm{x}}_j)^2 = \sum_{j=1}^{p}\tilde{\mathrm{x}}_j^T K\tilde{\mathrm{x}}_j \tag{19}$$

represents the amount of variation of the normal dataset explained by the p loading vectors $\tilde{\mathrm{x}}_1, \ldots, \tilde{\mathrm{x}}_p$ where n is the number of observations in the normal dataset. After dividing (19) by $\sum_{j=1}^{p}\mathrm{x}_j^T K\mathrm{x}_j$, which is the maximum amount of variation p orthogonal vectors can explain, and multiplying by 100, we get the following measure:

$$\text{(Total Explained Variation)} \quad 100 \times \frac{\sum_{j=1}^{p}\tilde{\mathrm{x}}_j^T K\tilde{\mathrm{x}}_j}{\sum_{j=1}^{p}\mathrm{x}_j^T K\mathrm{x}_j}. \tag{20}$$

Metric (20) captures how well the loading vectors obtained from the noisy dataset explain variation of the normal dataset with respect to the L_2-norm, and therefore it can be used to measure robustness of a kernel PCA algorithm in the presence of entry-wise perturbations. For example, if an algorithm has a value close to one, then it is robust with respect to entry-wise perturbations. Using this metric, we compare the robustness of Algorithm 1 with that of KPCA-L1, L1-KPCA, R1-KPCA, and L2-KPCA. For each value of r, we compute (20) for the ten datasets with $p = 4$ and average them. We arbitrarily choose $p = 4$ since the result is consistent regardless of the choice of p. Figure 2 shows the results for the linear kernel and Figure 3 shows the results for the Gaussian kernel with the width parameter σ varying from 10 to 25.

In the case of the linear kernel, R1-KPCA achieves the best performance for all values of r followed by the L_1-norm kernel PCA algorithms and L2-KPCA. While the loading vectors from L2-KPCA explain about 90% of the variation, those from R1-KPCA, Algorithm 1, KPCA-L1, and L1-KPCA explain around 96%,95%,94%, and 93% of the variation, respectively. This demonstrates the robustness of the R_1-norm and L_1-norm based kernel PCA models with respect to the presence of entry-wise perturbations. Among the three L_1-norm kernel PCA algorithms, Algorithm 1 consistently outperforms KPCA-L1 and L1-KPCA by 1% and 2%, respectively. As the percentage of corrupted observations ($r\%$) increases, the total explained variation tends to decrease for every model but the gaps between them remain the same.

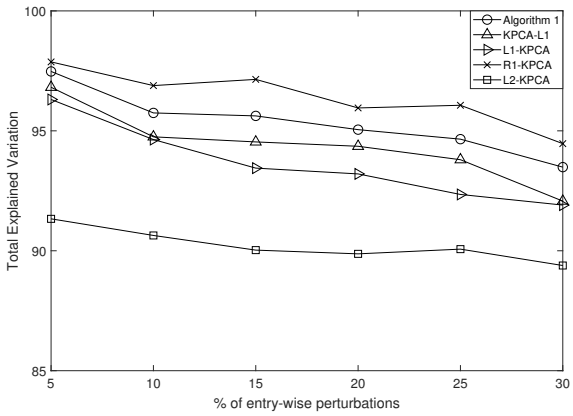


Fig. 2. Robust Extraction of PCs (Linear Kernel)

When the Gaussian kernel is used, the results are slightly different depending on the value of r and σ. If r and σ are small, the effects of entry-wise perturbations are relatively small so that all the algorithms give pretty similar results. However, if r or σ is large, the influences of entry-wise perturbations are clearly present in the kernel matrix, and therefore, the results are different depending on the robustness of each algorithm. As shown in Figure 3, the three L_1-norm kernel PCA algorithms and R1-KPCA outperform L2-KPCA as in the case of the linear kernel. However, while R1-KPCA achieves the best performance for the linear kernel, the L_1-norm kernel PCA algorithms work better than R1-KPCA when the Gaussian kernel is used. Especially, Algorithm 1 outperforms the other algorithms if r exceeds 20. The superior performance of Algorithm 1 ranges from 1% to 5% in these cases.

6.2 Outlier Detection

L_2-norm PCA has been shown to be effective for anomaly detection [27]. The idea is to extract loading vectors using datasets consisting of only normal samples and use these loading vectors for developing a detection model. Specifically, a boundary of the normal samples is derived from the loading vectors and the boundary is used to discriminate normal and abnormal samples.

We extend this principle to outlier detection, i.e. its unsupervised counterpart. In the outlier detection setting, sample labels are not given when the model is built. Therefore, it is not possible to build a detection model solely based on normal samples. Given this context, we run a robust kernel PCA algorithm on the entire dataset (with outliers) and use the resulting loading vectors to characterize a boundary of normal samples. Since these loading vectors are less influenced by outliers as illustrated in Section 6.1, we expect that they would better construct a normal boundary. We compare the performance of our algorithm based model to that of KPCA-L1, L1-KPCA, R1-KPCA, and L2-KPCA based models as well as other popular outlier detection models [28] [29].

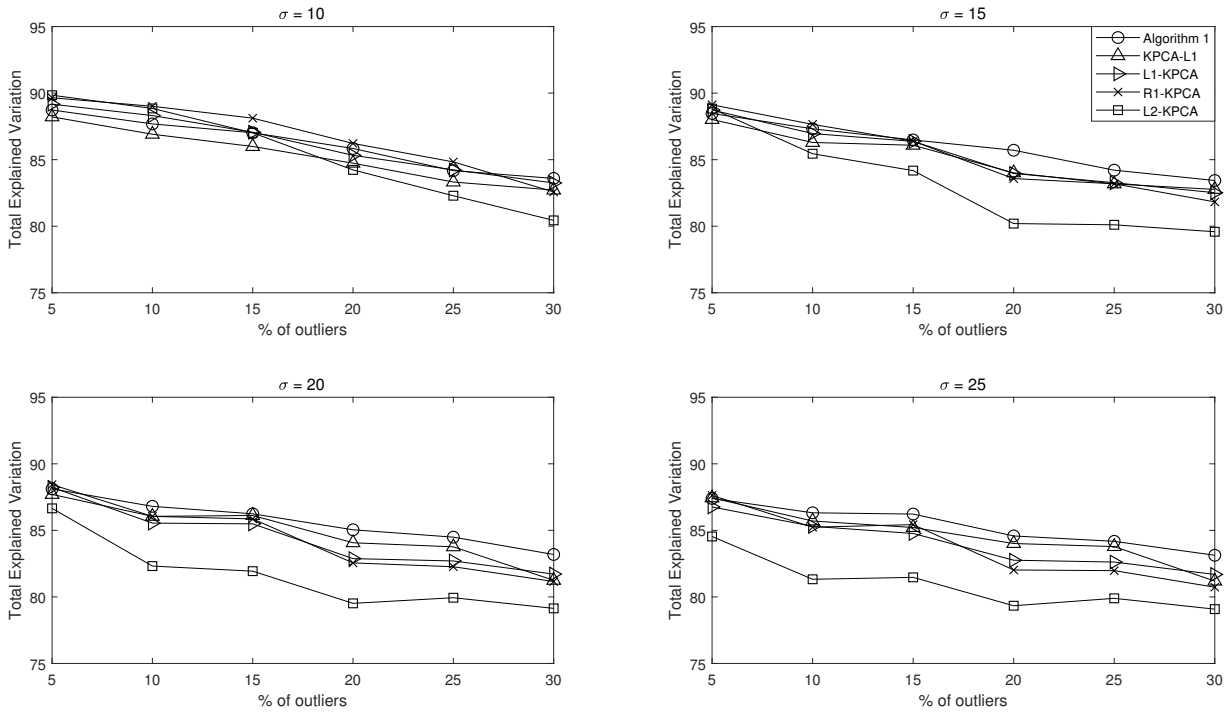


Fig. 3. Robust Extraction of PCs (Gaussian Kernel with σ ranging from 10 to 25)

6.2.1 Toy Examples

We first illustrate the advantage of using robust PCA models for outlier detection using the following two-dimensional toy examples.

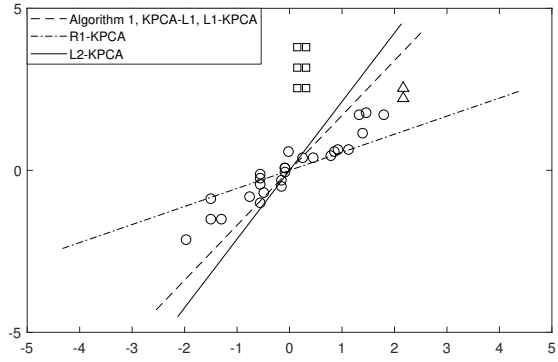


Fig. 4. The first toy example - original space

Figure 4 shows the distribution of normal samples and outliers with the first loading vectors of each algorithm. To extract them, the linear kernel is used as the underlying pattern is linear. As the three L_1-norm kernel PCA algorithms yield the same loading vectors for this example, we express their loading vectors using a single dashed line. In the figure, the normal samples are distributed forming a linear pattern and the outliers are scattered exhibiting two different patterns; the two triangle points are outliers due to their scale and the six square points are outliers as they do not follow the linear pattern. If a loading vector exactly matches the linear pattern, outliers can be easily detected in the principal space; the triangle points can be detected due to the large first principal component and the square points can be detected from the large second principal component. However, due to the presence of the outliers, it is hard to find such loading vector. Given this context, we utilize robust kernel PCA algorithms to obtain a loading vector which deviates less from the linear pattern.

Figure 5 displays the PCA results of the five kernel PCA algorithms. In the figure, the x-axis, y-axis represents the first, second principal component, respectively. As shown in the figure, the triangle outliers can be easily separated by the first principal component of each algorithm. However, while the square outliers can be discriminated by the second principal components of L_1-norm kernel PCA and R1-KPCA, there exists some overlap between the normal samples and the square outliers in the range of the second principal component of L2-KPCA. Two outliers appear closer to the origin than some normal samples making the circular boundary of the normal samples include them as shown in Figure 5. On the other hand, all the normal samples are clearly separated from the outliers in the result of Algorithm 1 and R1-KPCA, demonstrating the advantage of using Algorithm 1 or R1-KPCA in outlier detection. This is consistent with findings from Figure 2.

In order to see if the same result holds for the Gaussian kernel, we consider another example. As shown in Figure 6, the second example has a spiral pattern consisting of normal samples as well as two types of outliers. As in the previous example, it has both the trivial outliers (the triangle points) and more challenging outliers (the square points). In

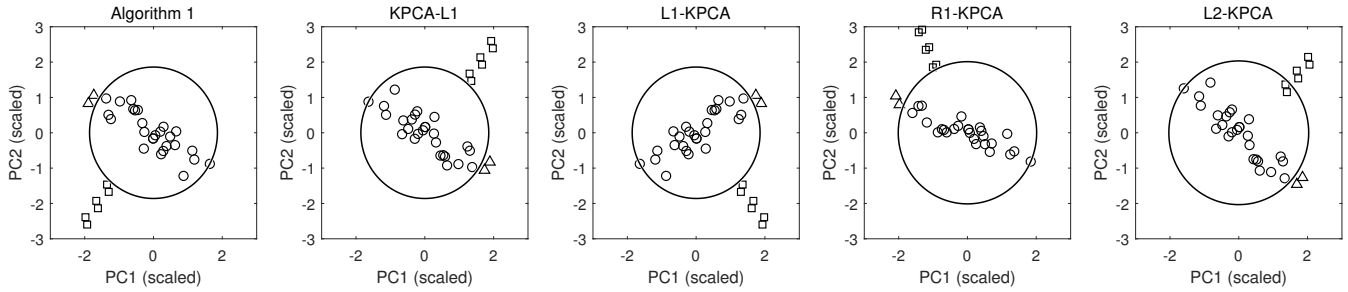


Fig. 5. The first toy example - principal space

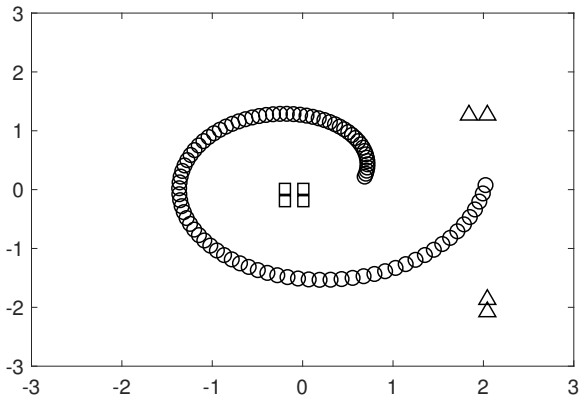


Fig. 6. The second toy example - original space

order to obtain nonlinear principal components, we apply the five kernel PCA algorithms with the Gaussian kernel. As Figure 7 displays, only Algorithm 1 succeeds to exclude the square outliers from the boundary while the other kernel PCA algorithms include them within the boundary. This superior performance of Algorithm 1 with the Gaussian kernel is consistent with the results in Section 6.1 and attests the effectiveness of using it for outlier detection, especially with the Gaussian kernel.

6.2.2 Real-world Datasets

For outlier detection, we use datasets from the UCI Machine Learning Repository [30] and the ODDS Library [31], see Table 1.

TABLE 1
Real-world Datasets for Outlier Detection

Data set	# samples	# features	# outliers
WBC	378	30	21 (7.6%)
Ionosphere	351	33	126 (36%)
BreastW	683	9	239 (35%)
Cardio	1831	21	176 (9.6%)
Musk	3062	166	97 (3.2%)
Mnist	7603	100	700 (9.2%)

In this experiment, we use a similar detection rule as in [27] where it is applied for anomaly detection. Let $Y \in \mathbb{R}^{n\times p}$ denote p principal components, and let m_j and λ_j be the mean and variance of j^{th} principal components, respectively. In order to detect outliers, we consider the following model:

$$\text{Classify } i^{th}\text{sample as an outlier, if } \sum_{\{j:\lambda_j \geq \alpha\}} \frac{(Y_{ij}-m_j)^2}{\lambda_j} > c. \tag{21}$$

The metric appearing on the left-hand side of (21) represents the squared Euclidean distance to the origin in the reduced standardized principal space consisting of principal components whose variance is greater than or equal to α. Our model can be understood as drawing a circular boundary (as illustrated in Figures 5 and 7) on the reduced standardized principal space. As we are assuming the outlier detection setting, sample labels are unknown at the stage of building a model so that selecting an appropriate c is not trivial. Therefore, we compute precision and recall with varying c and evaluate the performance of a model using AUC under the precision-recall curve. We compare AUC of Algorithm 1 based model to that of KPCA-L1, L1-KPCA, R1-KPCA, and L2-KPCA based models as well as that of the two popular outlier detection models, Local Outlier Factor (LOF) [28] and Isolation Forest (iForest) [29].

Since principal components having a small variance contain minor information, we only consider the principal components whose sample variance is greater than or equal to α. We select α be to the largest α' such that

$$0.8 \times \sum_{j=1}^{d} \lambda_j \leq \sum_{\{j:\lambda_j \geq \alpha'\}} \lambda_j,$$

where d is the number of features. In other words, we select the top principal components which explain more than 80% of variation in the dataset. For the choice of the kernel function, we consider both linear and Gaussian kernels with the width parameter σ of the Gaussian kernel to be d. On the other hand, we set the number of nearest neighbors to 10 in LOF, and set the number of trees, the size of subsample, and the number of rounds to 100, 256, 10, respectively in iForest since these parameter values are widely used.

Table 2 displays the AUCs of the 12 different models. The numbers in bold present the highest AUC cases (there can be several similar top performances). If the outliers are obvious, every kernel PCA based model works well as seen in the case of Breastw and Musk. However, when the outliers are not obvious, Algorithm 1 based model tends to outperform the other detection models. Especially, when Algorithm 1 is used with the Gaussian kernel, it consistently achieves top AUC values. Compared to the kernel PCA

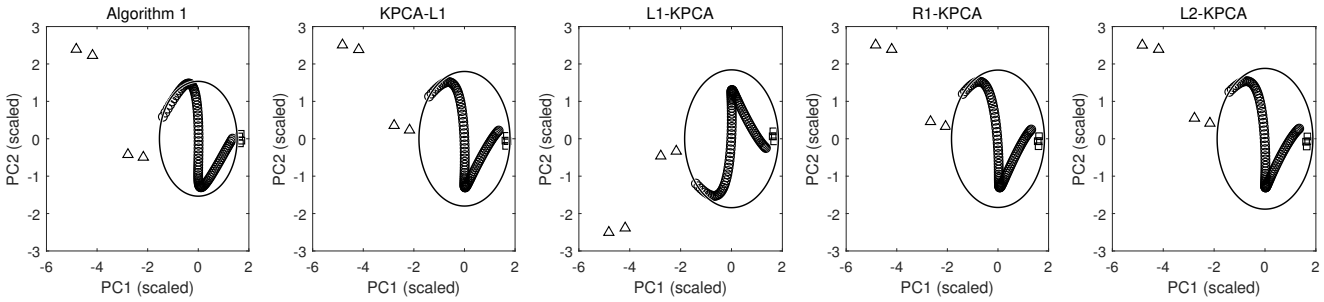


Fig. 7. The second toy example - principal space

TABLE 2
AUC of the Outlier Detection Models

Datasets	AUC											
	Linear					Gaussian					LOF	iForest
	Algo 1	KPCA -L1	L1-KPCA	R1-KPCA	L2-KPCA	Algo 1	KPCA -L1	L1-KPCA	R1-KPCA	L2-KPCA		
WBC	0.5208	0.5288	0.5320	0.4658	0.4798	0.5292	**0.5340**	**0.5337**	0.5072	0.5224	0.3451	**0.5525**
Ionosphere	0.6625	**0.7319**	0.6834	**0.7642**	0.7057	**0.7238**	0.6806	0.6887	0.7041	0.6992	0.7032	0.7067
Breastw	0.9250	0.9125	0.9218	0.9269	0.9152	**0.9428**	0.9287	0.9354	**0.9521**	0.9309	0.3750	**0.9513**
Cardio	0.5790	0.5551	**0.5799**	0.4265	0.5066	**0.6096**	0.5752	**0.5963**	0.5116	0.4664	0.1921	0.5114
Musk	**0.9947**	**0.9947**	**0.9947**	0.8055	0.9358	**0.9947**	**0.9947**	**0.9947**	0.9916	**0.9947**	0.0925	0.7596
Mnist	**0.3985**	**0.4002**	N/A	N/A	0.3914	**0.3966**	0.3913	N/A	N/A	0.3639	0.1924	0.3380

* N/A: The experiments can not be completed within the period of 12 hours.

based detection models, LOF and iForest do not work well. LOF never achieves the top performance and iForest is not competitive for high-dimensional datasets such Must and Mnist although it yields the top AUC values for WBC and Breastw. Unlike iForest, Algorithm 1 based model with the Gaussian kernel consistently works well regardless of the size of the problems, demonstrating its effectiveness in outlier detection.

6.3 Runtime Comparison

Finally, we compare the runtime of Algorithm 1 to that of KPCA-L1, L1-KPCA, R1-KPCA, and L2-KPCA. We run them on the six real-world datasets presented in Table 1 and measure the time taken to get all the principal components.

As shown in Table 3, the runtime largely varies across the algorithms. Among the L_1-norm kernel PCA algorithms, Algorithm 1 has the smallest runtime for all datasets. Actually, it is much faster than the other two algorithms since it requires only matrix-vector multiplication while the other algorithms entail either eigen-decomposition or solving a system of equations. R1-KPCA is also not as fast as Algorithm 1 since it involves QR-decomposition at each iteration to make the loading vectors orthogonal. Among the robust kernel PCA algorithms, only Algorithm 1 is computationally comparable to L2-KPCA, making it the best choice for robust kernel PCA in a large-scale setting.

7 Conclusion

In this work, we propose a simple algorithm for L_1-norm kernel PCA and provide its convergence analysis. To derive the algorithm, we first reformulate the problem so that it can be geometrically understood. Based on the geometric understanding, we derive an algorithm under which the kernel trick is applicable. As this framework is not problem specific, it can be applied to other problems as well and its application to L_2-norm PCA results in Power iteration [3]. In the convergence analysis, we prove that our algorithm converges in a finite number of steps together with the rate of convergence being linear. Moreover, we prove that the output of our algorithm satisfies the local optimality conditions.

Computational experiments demonstrate the robustness of our algorithm and the runtime comparison shows that the proposed algorithm takes much less time for L_1-norm kernel PCA than other robust kernel PCA algorithms. Also, its application to outlier detection outperforms all outlier detection algorithms. The model based on our algorithm is not only better than that of the other kernel PCA algorithms but also outperforms LOF and iForest, especially when the high-dimensional datasets are considered.

References

[1] I. Jolliffe, *Principal Component Analysis*. Wiley Online Library, 2002.
[2] B. Schölkopf, A. Smola, and K.-R. Müller, "Kernel Principal Component Analysis," in *International Conference on Artificial Neural Networks*, 1997, pp. 583–588.
[3] G. H. Golub and C. F. Van Loan, *Matrix Computations*. JHU Press, 2012, vol. 3.
[4] E. J. Candès and B. Recht, "Exact Matrix Completion via Convex Optimization," *Foundations of Computational Mathematics*, vol. 9, no. 6, p. 717, 2009.
[5] E. J. Candès, X. Li, Y. Ma, and J. Wright, "Robust Principal Component Analysis?" *Journal of the ACM*, vol. 58, no. 3, p. 11, 2011.
[6] H. Xu, C. Caramanis, and S. Sanghavi, "Robust PCA via outlier pursuit," in *Advances in Neural Information Processing Systems*, 2010, pp. 2496–2504.

TABLE 3
Runtime Comparison

Datasets	Runtime (minutes)									
	Linear					Gaussian				
	Algo 1	KPCA -L1	L1-KPCA	R1-KPCA	L2-KPCA	Algo 1	KPCA -L1	L1-KPCA	R1-KPCA	L2-KPCA
WBC	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.1	0.1	0.0
Ionosphere	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.0	0.1	0.0
Breastw	0.0	0.0	0.1	0.1	0.0	0.0	0.0	0.1	0.1	0.0
Cardio	0.0	0.1	3.6	1.8	0.0	0.0	0.7	5.0	1.7	0.0
Musk	0.2	28.7	362.4	349.9	0.1	0.2	4.9	401.6	337.8	0.0
Mnist	0.9	263.3	>720	>720	1.8	1.0	263.2	>720	>720	1.8

[7] G. Liu, Z. Lin, S. Yan, J. Sun, Y. Yu, and Y. Ma, "Robust Recovery of Subspace Structures by Low-Rank Representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 171–184, 2013.

[8] G. Liu and P. Li, "Recovery of Coherent Data via Low-Rank Dictionary Pursuit," in *Advances in Neural Information Processing Systems*, 2014, pp. 1206–1214.

[9] G. Liu, H. Xu, J. Tang, Q. Liu, and S. Yan, "A Deterministic Analysis for LRR," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 3, pp. 417–430, 2016.

[10] G. Liu, Q. Liu, and P. Li, "Blessing of Dimensionality: Recovering Mixture Data via Dictionary Pursuit," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 1, pp. 47–60, 2017.

[11] J. P. Brooks, J. H. Dulá, and E. L. Boone, "A Pure L1-norm Principal Component Analysis," *Computational Statistics & Data Analysis*, vol. 61, pp. 83–98, 2013.

[12] Y. W. Park, "Optimization for Regression, PCA, and SVM: Optimality and Scalability," Ph.D. dissertation, Northwestern University, 2015.

[13] Y. W. Park and D. Klabjan, "Iteratively Reweighted Least Squares Algorithms for L1-Norm Principal Component Analysis," in *IEEE International Conference on Data Mining*, 2016, pp. 430–438.

[14] ——, "Three iteratively reweighted least squares algorithms for L_1-norm principal component analysis," *Knowledge and Information Systems*, vol. 54, no. 3, pp. 541–565, 2018.

[15] F. Nie, H. Huang, C. Ding, D. Luo, and H. Wang, "Robust Principal Component Analysis with Non-Greedy L_1-Norm Maximization," in *Proceedings of the International Joint Conference on Artificial Intelligence*, vol. 22, no. 1, 2011, pp. 1433–1438.

[16] M. McCoy and J. A. Tropp, "Two Proposals for Robust PCA using Semidefinite Programming," *Electronic Journal of Statistics*, vol. 5, pp. 1123–1160, 2011.

[17] P. P. Markopoulos, G. N. Karystinos, and D. A. Pados, "Optimal Algorithms for L_1-subspace Signal Processing," *IEEE Transactions on Signal Processing*, vol. 62, no. 19, pp. 5046–5058, 2014.

[18] Q. Ke and T. Kanade, "Robust L_1 Norm Factorization in the Presence of Outliers and Missing Data by Alternative Convex Programming," in *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005, pp. 739–746.

[19] C. Ding, D. Zhou, X. He, and H. Zha, "R_1-PCA: Rotational Invariant L_1-norm Principal Component Analysis for Robust Subspace Factorization," in *Proceedings of the 23rd International Conference on Machine Learning*. ACM, 2006, pp. 281–288.

[20] G. N. Karystinos and A. P. Liavas, "Efficient Computation of the Binary Vector That Maximizes a Rank-Deficient Quadratic Form," *IEEE Transactions on Information Theory*, vol. 56, no. 7, pp. 3581–3593, 2010.

[21] Y. Nesterov, "Semidefinite Relaxation and Nonconvex Quadratic Optimization," *Optimization Methods and Software*, vol. 9, no. 1-3, pp. 141–160, 1998.

[22] N. Kwak, "Principal Component Analysis based on L_1-norm Maximization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 9, pp. 1672–1680, 2008.

[23] ——, "Nonlinear projection trick in kernel methods: An alternative to the kernel trick," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 12, pp. 2113–2119, 2013.

[24] Y. Xiao, H. Wang, W. Xu, and J. Zhou, "L1 norm based KPCA for novelty detection," *Pattern Recognition*, vol. 46, no. 1, pp. 389–396, 2013.

[25] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[26] M. X. Goemans and D. P. Williamson, "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming," *Journal of the ACM (JACM)*, vol. 42, no. 6, pp. 1115–1145, 1995.

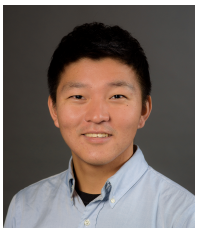
[27] M.-L. Shyu, S.-C. Chen, K. Sarinnapakorn, and L. Chang, "A Novel Anomaly Detection Scheme Based on Principal Component Classifier," in *IEEE International Conference on Data Mining*, 2003, pp. 172–179.

[28] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "LOF: Identifying Density-Based Local Outliers," in *ACM SIGMOD Record*, vol. 29, no. 2, 2000, pp. 93–104.

[29] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," in *IEEE International Conference on Data Mining*, 2008, pp. 413–422.

[30] M. Lichman, "UCI Machine Learning Repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[31] S. Rayana, "ODDS Library," 2016. [Online]. Available: http://odds.cs.stonybrook.edu



Cheolmin Kim is a Ph.D student in Industrial Engineering and Management Science at Northwestern University. He received his B.S in Industrial Engineering and B.A in Economics from Seoul National University. His research interests are at the interaction of optimization and machine learning. He is interested in designing a new machine learning model, developing an training algorithm and analyzing the efficiency of the algorithm from an optimization perspective.

Diego Klabjan is a professor at Northwestern University, Department of Industrial Engineering and Management Sciences. He is also Founding Director, Master of Science in Analytics. After obtaining his doctorate from the School of Industrial and Systems Engineering of the Georgia Institute of Technology in 1999 in Algorithms, Combinatorics, and Optimization, in the same year he joined the University of Illinois at Urbana-Champaign. In 2007 he became an associate professor at Northwestern and in 2012 he was promoted to a full professor. His research is focused on machine learning, deep learning and analytics with concentration in finance, transportation, sport, and bioinformatics. Professor Klabjan has led projects with large companies such as Intel, Baxter, Allstate, AbbVie, FedEx Express, General Motors, United Continental, and many others, and he is also assisting numerous start-ups with their analytics needs. He is also a founder of Opex Analytics LLC.