AI INSPIRED ALGORITHMS FOR SEVERAL COMBINATORIAL OPTIMIZATION PROBLEMS IN DATA SCIENCE

by

Guihong Wan

APPROVED BY SUPERVISORY COMMITTEE:

Dr. Haim Schweitzer, Chair

Dr. Ovidiu Daescu

Dr. Xiaohu Guo

Dr. Sriraam Natarajan

To my family.

AI INSPIRED ALGORITHMS FOR SEVERAL COMBINATORIAL OPTIMIZATION PROBLEMS IN DATA SCIENCE

by

GUIHONG WAN, BS, MS

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN

COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS

August 2021

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Dr. Haim Schweitzer, for his guidance and continuous support. I deeply appreciate that he took me as his student. It has been a great time working with him.

I would like to thank my committee members Dr. Ovidiu Daescu, Dr. Xiaohu Guo, and Dr. Sriraam Natarajan for their insightful comments and feedback.

I would like to thank Baokun He, Swair Shah, Crystal Maung, and Gordon Arnold for their contributions to the joint work. Special thanks to Crystal Maung for her encouragement.

I would like to thank my teachers Dr. Xiaojun Liu, Prof. Greg Ozbirn, and Ni Yao for helping me in my early career.

Thanks to my roommates and friends for their companionship and care, especially during the COVID-19 pandemic.

Last but not least, thanks to my family for their support and love.

June 2021

AI INSPIRED ALGORITHMS FOR SEVERAL COMBINATORIAL OPTIMIZATION PROBLEMS IN DATA SCIENCE

Guihong Wan, PhD
The University of Texas at Dallas, 2021

Supervising Professor: Dr. Haim Schweitzer, Chair

Combinatorial optimization is a class of problems that consists of finding an optimal solution from a finite set of feasible solutions. Many important problems in Data Science can be viewed as combinatorial optimization problems, typically described in terms of selecting a small number of items from a much larger set. We describe Artificial Intelligence (AI) inspired combinatorial optimization algorithms to three selection problems that have important practical applications. The first is the "unsupervised column subset selection problem", which has important applications to dimensionality reduction. The second is the "supervised column subset selection problem", which can be viewed as a direct generalization of the unsupervised case. The third is the "outlier detection for Principal Component Analysis (PCA)". We use ideas from AI to derive new algorithms for these classical problems that are known to be NP-hard. Our algorithms compare favorably with the current state of the art and come with guarantees on the quality of the solutions.

In the unsupervised column subset selection problem, one attempts to represent an entire matrix as a linear combination of a small fraction of its columns. We study a generalization that approximates the matrix with both selected and extracted features. We show that an optimal solution to this hybrid problem involves a combinatorial search, and cannot be trivially obtained even if one can optimally solve the separate problems of selection and

extraction. Our approach that gives optimal and approximate solutions uses a combinatorial search in a setting similar to the weighted A* algorithm.

In the supervised column subset selection problem, we study the approximation of a "target" matrix in terms of several selected columns of another matrix, sometimes called a "dictionary" matrix. We propose the first nontrivial optimal algorithm for this problem, using a combinatorial search setting similar to the classical A^* algorithm. We also propose practical sub-optimal algorithms in a setting similar to the classical weighted A^* algorithm. Experimental results show that our sub-optimal algorithms compare favorably with the current state of the art. Previously proposed fastest nontrivial algorithms have a running time proportional to the product of the number of columns of the two matrices. We describe a significantly faster algorithm with complexity proportional to the sum of the number of columns of the two matrices.

Outliers negatively affect the accuracy of data analysis. Algorithms that attempt to detect outliers and remove them from the data prior to applying PCA are sometimes called "Robust PCA" algorithms. We propose a new algorithm to detect outliers for PCA that combines two ideas. The first is "chunk recursive elimination" that was used effectively to accelerate feature selection, and the second is combinatorial search, in a setting similar to the weighted A* algorithm. Our main result is showing how to combine these two ideas to balance speed and accuracy. The resulting algorithm is called Chunk-A^*, with variants that compute optimal and sub-optimal solutions. We also propose a fast algorithm to address this problem. The main idea is to rank each data point by looking ahead and evaluating the change in the global PCA error when an inlier is converted into an outlier. We show that this lookahead procedure can be implemented efficiently, and it is much more accurate than the current state-of-the-art algorithms.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Linear dimensionality reduction techniques are the cornerstone of analyzing high-dimensional data and have been applied in many domains such as machine learning, data science, and pattern recognition. See, e.g., (Donoho et al., 2000; Cunningham and Ghahramani, 2015; Espadoto et al., 2019; Ayesha et al., 2020; Wan et al., 2020). Given a set of high-dimensional data as input, the goal of linear dimensionality reduction is to produce a linear mapping for the data from the high-dimensional space into a low-dimensional space such that some objective is optimized. See, e.g., (Burges, 2010; Kokiopoulou et al., 2011; Cunningham and Ghahramani, 2015). Linear dimensionality reduction can be unsupervised or supervised. Unsupervised dimensionality reduction techniques (e.g., Cunningham and Ghahramani, 2015; Gray, 2017) take advantage of the patterns and structures of the data, such as data distribution and variance. Supervised dimensionality reduction techniques (e.g., Cui and Fan, 2012; Vepakomma et al., 2018) compute a feature subset to predict label information.

Studies of dimensionality reduction techniques distinguish between feature selection and feature extraction. See, e.g., (Guyon et al., 2008; Khalid et al., 2014; Ayesha et al., 2020). Feature selection is the process of selecting a subset of the most useful features to be used in model construction in tasks such as regression, classification, and clustering. The problem can be formalized as the column subset selection in unsupervised and supervised settings. Unlike feature selection, feature extraction creates a new, small set of features, which are the results of applying arbitrary functions to the data. The extracted features are not constrained to be a subset of the original features. Even though the approximation obtained by using feature selection is worse than the approximation obtained by using feature extraction, there are advantages of feature selection that make it the preferred choice in many situations. For example, selected features generalize better than extracted features in many machine learning tasks (e.g., Guyon and Elisseeff, 2003); the results obtained by feature extraction

are "notoriously difficult to interpret in terms of the underlying data" (e.g., Drineas et al., 2008).

PCA (Pearson, 1901) is arguably one of the most popular unsupervised feature extraction techniques. It preserves the maximum amount of variance of the original data. Recent algorithmic approaches that use randomization give algorithms that can handle large datasets. See, e.g., (Halko et al., 2011; Musco and Musco, 2015; Li et al., 2017). Unfortunately, PCA is known to be sensitive to outliers (e.g., Hadi et al., 2009; Shah et al., 2018; Simonov et al., 2019). It can be made significantly more accurate if it is allowed to ignore a fraction of the data, considered to be outliers. There is a large number of studies on outlier detection and removal specifically for PCA. They are sometimes called "Robust Principal Component Analysis" (RPCA) algorithms, or, alternatively, "Robust Subspace Recovery" (RSR) algorithms. For recent surveys that discuss many variants of these robust algorithms, see (Vaswani and Narayanamurthy, 2018; Lerman and Maunu, 2018c). The first reference reviews techniques that consider outliers as partially corrupt observations, while the second reviews techniques that consider each data point as either an outlier or an inlier. We follow the same interpretation of outliers as in the second reference.

In this dissertation, we focus on three problems: unsupervised column subset selection, supervised column subset selection, and outlier detection for PCA. They can be viewed as combinatorial optimization problems, specifically described in terms of selecting a solution from a large set of feasible solutions such that the objective is optimized:

$$\begin{aligned} &\min_{S} E(S) \\ &\text{subject to } S \in \Phi, \end{aligned} \tag{1.1}$$

where S is a selection, and Φ is the admissible space of all feasible solutions. In the following sections we discuss the three problems in detail.

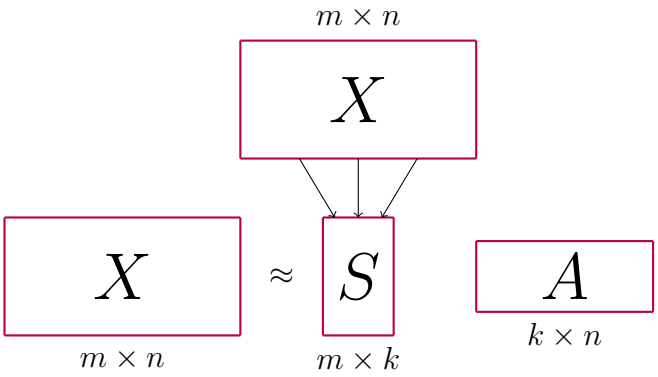

Figure 1.1: The unsupervised column subset selection problem. Given X and k, compute S and A to linearly approximate X, where S contains k selected columns of X.

1.1 Unsupervised Column Subset Selection

Let $X = (x_1 \dots x_n)$ be the data matrix of m rows and n columns. Consider selecting $k \leq n$ columns of X that are used to linearly approximate the original matrix X:

$$X \approx SA.$$

Here $S = (x_{s_1} \dots x_{s_k})$ is the selection matrix of size $m \times k$, and A is the coefficient matrix of size $k \times n$, as illustrated in Figure 1.1. The quality of the selected columns in S is measured by the following reconstruction error in the Frobenius norm:

$$E(S) = \min_{A} \|X - SA\|_F^2.$$

Then the unsupervised column subset selection problem can be stated as selecting k columns of X such that the reconstruction error is minimized:

$$\begin{aligned} &\min_{S,A} \|X - SA\|_F^2 \\ &\text{subject to } S \in \Phi, \end{aligned} \tag{1.2}$$

where Φ is the set of all possible column subsets of X with size k. The problem is known to be NP-hard (Shitov, 2017a).

When each column in X corresponds to a data point, the problem is called unsupervised representative selection. See, e.g., (Zaeemzadeh et al., 2019; Joneidi et al., 2020). When each column in X corresponds to a data feature, the problem is called unsupervised feature selection. See, e.g., (Arai et al., 2015; Solorio-Fernández et al., 2020). Unsupervised feature selection formulated as the column subset selection has attracted a lot of attention (e.g., Businger and Golub, 1965; Maung and Schweitzer, 2013; Paul et al., 2015). Recent studies using classical AI tools of combinatorial search derived optimal and near-optimal algorithms in the Frobenius norm. See Arai et al. (2015, 2016).

We study a generalization that approximates the data matrix X with both selected and extracted features, and the reconstruction error is measured in all unitarily invariant norms, including the Frobenius norm. The model we propose has r_1 selected features and r_2 extracted features. The approximation of X is given by:

$$X \approx SA_1 + VA_2,$$

where S consists of r_1 columns from X, and V consists of r_2 columns that are unconstrained. A_1 and A_2 are the coefficient matrices. Let Θ be a matrix norm. The reconstruction error can be computed as follows:

$$E(S, V) = \min_{A_1, A_2} \Theta(X - SA_1 - VA_2).$$

The problem can be described as:

$$\begin{aligned} &\min_{S,V,A_1,A_2} \Theta(X - SA_1 - VA_2) \\ &\text{subject to } S \in \Phi, \end{aligned} \tag{1.3}$$

where Φ is the set of all possible column subsets of X with size r_1, and V can be any matrix with size $m \times r_2$. We refer to the representation as the "Hybrid Low Rank", or HLR. Observe that the HLR has simple feature selection and simple feature extraction as special

cases. Thus, one would expect the HLR to have some desired properties of feature selection combined with some desired properties of feature extraction. For example, r_1 of the HLR features are easy to interpret (as in feature selection), and only r_2 of them are hard to interpret (as in feature extraction).

The problem is NP-hard since the subproblem of feature selection is known to be NP-hard (Shitov, 2017a). Our main result is an algorithm that computes HLR for any unitarily invariant error criteria. It uses a combinatorial search and gives optimal and approximate solutions with guarantees. This work was done in collaboration with Baokun He, Swair Shah, Crystal Maung, Gordon Arnold, and Haim Schweitzer. The results were published in (He et al., 2019).

1.2 Supervised Column Subset Selection

Supervised column subset selection problem can be viewed as a direct generalization of the unsupervised case described in the problem (1.2). Instead of approximating the original data matrix X, it selects several columns of X to approximate another target matrix Y. Let $Y = (y_1 \dots y_N)$ be the target matrix of m rows and N columns. Consider selecting $k \leq n$ columns of X that are used to linearly approximate Y:

$$Y \approx SA,$$

where S is the selection matrix which contains k columns of X, and A is the coefficient matrix of size $k \times N$, as illustrated in Figure 1.2. The reconstruction error of Y in terms of S is defined by:

$$E(S) = \min_A \|Y - SA\|_F^2.$$

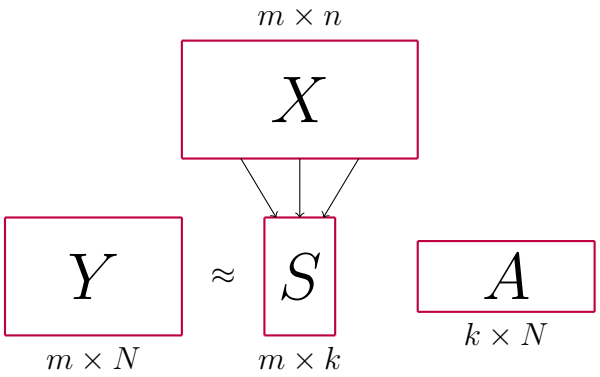

Figure 1.2: The supervised column subset selection problem. Given X, Y and k, compute S and A to linearly approximate Y, where S contains k columns of X.

The goal of supervised column subset selection is to find k columns of X to approximate Y such that the reconstruction error of Y is minimized:

$$\begin{aligned} &\min_{S,A} \|Y - SA\|_F^2 \\ &\text{subject to } S \in \Phi, \end{aligned} \tag{1.4}$$

where Φ is the set of all possible column subsets of X with size k. Optimal solutions, as well as approximations within a constant, are known to be NP-hard even for the case where $N = 1$ (Natarajan, 1995; Davis et al., 1997; Amaldi and Kann, 1998). This implies that algorithms that find optimal solutions cannot be efficient. Previous studies have developed optimal algorithms that are exponential in the worst case but improve on exhaustive search, and fast algorithms that are not optimal.

Extensive studies were directed at the $N = 1$ case. See, e.g., (Mallat, 1999; Tropp, 2004; Furnival and Wilson, 1974; Zhang, 2009; Qian et al., 2015). Applications include signal processing (e.g., Mallat, 1999; Pati et al., 1993; Neff and Zakhor, 1997; Tropp, 2004) and supervised feature selection in linear regression (e.g., Furnival and Wilson, 1974; Hastie et al., 2009; Neff and Zakhor, 1997; Zhang, 2009; Qian et al., 2015). An optimal algorithm proposed in (Furnival and Wilson, 1974) finds the best solution by a leaps and bounds procedure, but it is not feasible for large k and large datasets.

In the general case where $N > 1$, we observe that one cannot simply apply an $N = 1$ algorithm separately to each column of Y. The challenge is to find columns in X that can simultaneously approximate all columns of Y. Previously proposed algorithms for the general case are typically greedy. They include (Maung and Schweitzer, 2015; Belmerhnia et al., 2014; Tropp et al., 2006; Çivril and Magdon-Ismail, 2012; Chen and Huo, 2006).

We solve the problem (1.4) optimally by using a combinatorial search in a setting similar to the classical A* algorithm (Hart et al., 1968). To the best of our knowledge it is the first nontrivial optimal algorithm for the general case where $N > 1$. We also propose practical sub-optimal algorithms in a setting similar to the classical weighted A^* algorithm (Pohl, 1970). These results are published in (Wan and Schweitzer, 2021c). Previously proposed fastest nontrivial algorithms have a running time $O(mnN)$ (Maung and Schweitzer, 2015). In addition, we describe a significantly faster algorithm with complexity $O(km(n + N))$, which is published in (Wan and Schweitzer, 2021b).

1.3 Outlier Detection for PCA

PCA is an unsupervised linear dimensionality reduction technique. Given the data matrix X of size $m \times n$ with each column as a data point, and the desired rank r, the rank-r PCA computes a linear mapping to reduce the dimension of data points from m to r: $A = V^T X$, where $r \leq \min(m, n)$. The inverse of this mapping gives approximate reconstructions of the data points in X:

$$X \approx VA,$$

where V of size $m \times r$ is not constrained, and A is the coefficient matrix of size $r \times n$. As in the classical development of PCA (e.g., Jolliffe, 2002), the quality of the mapping can be measured by the reconstruction errors of all data points:

$$E(V) = \min_A \|X - VA\|_F^2.$$

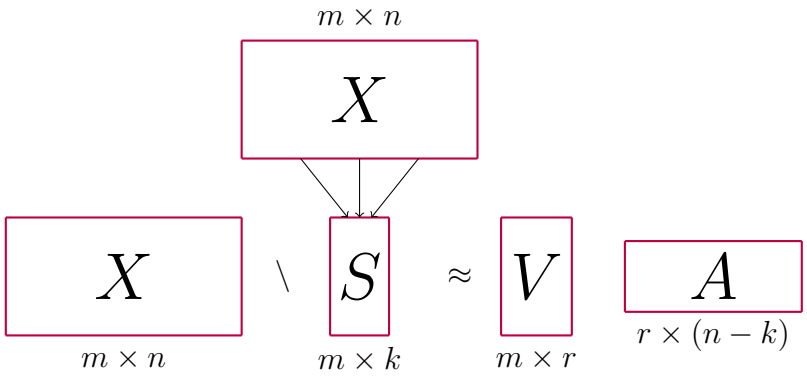

Figure 1.3: The outlier detection for PCA problem. Given X, k and r, detect the outlier set S, and then compute the rank-r PCA from the inlier set, where columns of V are unconstrained.

It is known (e.g., Jolliffe, 2002) that the columns of V can be computed as the r eigenvectors of the matrix $B = XX^T$ corresponding to its r largest eigenvalues.

Suppose that X is partitioned into a subset of k outliers S and the reminder inlier subset $X\backslash S$. The desired rank-r PCA is computed from the inlier set, as illustrated in Figure 1.3. Accordingly, the PCA error is computed from the inlier subset:

$$E(S, V) = \min_A \|X\backslash S - VA\|_F^2,$$

where V is of size $m \times r$, and A is the coefficient matrix of size $r \times (n-k)$. Then the problem of the robust PCA is to detect the outlier set S such that the rank-r PCA error for the inlier set is minimized:

$$\begin{aligned} &\min_{S,V,A} \|X\backslash S - VA\|_F^2 \\ &\text{subject to } S \in \Phi, \end{aligned} \tag{1.5}$$

where Φ is the set of all possible outlier subsets of X with size k. The problem is known to be NP-hard (e.g., Hardt and Moitra, 2013).

There is a large number of studies on outlier detection and removal specifically for computing robust PCA. A recent review paper is (Lerman and Maunu, 2018c). An important

way to solve the robust PCA problem is to first filter outliers and then fit a subspace to the remaining data by using classical PCA. Many studies view the data as coming from a fixed distribution, and attempt to detect outliers as data points at the margins of the distribution. See, e.g., (Roberts, 1999; Scheirer et al., 2011; Hubert and Engelen, 2004; Xu et al., 2010; Zhang et al., 2015a). Another common approach is to rank the data points based on some criterion, and detect outliers as data points with low scores. See, e.g., (Chen and Lerman, 2009; Soltanolkotabi et al., 2012; Rahmani and Atia, 2017; You et al., 2017; Shah et al., 2018; Rahmani and Li, 2019a).

We describe an algorithm that uses an accelerated combinatorial search framework. It produces optimal and approximate solutions with bounds on sub-optimality. This work is published in (Wan and Schweitzer, 2021a). We also derive a fast algorithm by using a lookahead procedure. The preliminary results are published in (Wan and Schweitzer, 2021d).

1.4 Main Tools

Combinatorial search algorithms on graphs such as A^* (Hart et al., 1968) and the weighted A^* (Pohl, 1970) are widely used for solving problems in Artificial Intelligence. Our approaches to solve the three selection problems we focus on are mainly based on these algorithms.

1.4.1 A^* Search Algorithm

The A^* search algorithm is a well known heuristic search algorithm. See, e.g., (Hart et al., 1968; Pearl, 1984; Russell and Norvig, 2010). The goal of the algorithm is to compute a minimum-cost path from a root node to a goal node in a graph, guided by a function $f(n_i)$ that can be computed at each node n_i. The standard formulation of the A^* search on graphs uses two lists: a Fringe list (F) and a Closed list (C). Initially, F contains the root node, and C is empty. In each iteration, the most promising open node from F is expanded and moved into C. If the node is not a goal node, its children nodes are added into F. Thus, F

contains the nodes that are still need to be expanded, and C contains the expanded nodes, which is used to avoid revisiting nodes.

The algorithm expands nodes according to a criterion $f(n_i)$, which is computed for each node n_i by the following formula:

$$f(n_i) = g(n_i) + h(n_i),$$

where $g(n_i)$ is the cost between the root node and the node n_i, and the heuristic function $h(n_i)$ is problem specific. With certain conditions on $h(n_i)$ (e.g., consistency), the A^* search algorithm is guaranteed to find an optimal solution.

1.4.2 Weighted A^* Search Algorithm

A standard approach to accelerate the A^* search algorithm is to use the weighted heuristic. Specifically, the weighed A* search algorithm expands nodes according to a criterion $f'(n_i)$ given by the following formula:

$$f'(n_i) = g(n_i) + (1+\epsilon)h(n_i) = f(n_i) + \epsilon h(n_i), \quad \epsilon \geq 0.$$

For $\epsilon > 0$ the weighted A* search algorithm is not guaranteed to find an optimal solution even if the heuristic function $h(n_i)$ is consistent. Theoretical analysis and experimental results (e.g., Pearl, 1984; Russell and Norvig, 2010) show that the weighted A^* algorithm can be much faster than the A^* algorithm.

1.5 Contributions

In this dissertation, the main contributions are summarized below:

- We formulate the problem of combining feature selection and feature extraction in an unsupervised setting. The unsupervised column subset selection problem is a special case of this hybrid problem. Using a combinatorial graph search framework we address

this hybrid problem optimally and approximately. The optimal variant is the first nontrivial optimal algorithm to this problem. The approximate variant gives solutions with bounds on their accuracy.

- We describe an algorithm for the supervised column subset selection problem by using a combinatorial graph search framework. The heuristic functions are different from the ones for the unsupervised case. Our algorithm is the first nontrivial optimal algorithm for the general case where the target matrix contains multiple columns. Practical sub-optimal variants with optimality guarantees are also proposed. Experimental results show that these sub-optimal variants outperform the current state of the art.

- We describe a fast spectral pursuit algorithm to the supervised column subset selection problem, which has a linear time complexity w.r.t the number of columns of the two matrices. This improves the current state-of-the-art algorithms of time complexity proportional to the product of the number of columns. We show experimentally that our algorithm compares favorably with the current state-of-the-art algorithms.

- We describe a new algorithm to detect outliers for PCA, called Chunk-A^*. The algorithm combines the "chunk recursive elimination" and the combinatorial search framework to balance accuracy and speed. It computes optimal and approximate solutions. When comparing with the current state of the art, our algorithm has advantages in terms of accuracy and speed. It also comes with guarantees on the sub-optimality of the solutions.

- We describe a fast outlier detection algorithm for PCA that uses a novel lookahead procedure. The efficient implementation of the algorithm requires a new rank-one modification formula for eigenvalues of centered matrices. The algorithm is very fast, and the accuracy can outperform the current state of the art.

1.6 Dissertation Outline

This dissertation is produced in accordance with guidelines which permit the inclusion of papers published or submitted for publication. The dissertation is organized as follows:

- Chapter 2 is the paper "Heuristic Search Algorithm for Dimensionality Reduction Optimally Combining Feature Selection and Feature Extraction". It solves the HLR problem with the unsupervised column subset selection problem as a special case.
- Chapter 3 is the paper "Heuristic Search for Approximating One Matrix in Terms of Another Matrix". It solves the supervised column subset selection problem.
- Chapter 4 is the supplementary material for "Heuristic Search for Approximating One Matrix in Terms of Another Matrix".
- Chapter 5 is the paper "A Fast Algorithm for Simultaneous Sparse Approximation". It introduces a fast algorithm for the supervised column subset selection problem.
- Chapter 6 is the paper "Accelerated Combinatorial Search for Outlier Detection with Provable Bounds on Sub-Optimality". It solves the outlier detection problem for PCA.
- Chapter 7 is the supplementary material for "Accelerated Combinatorial Search for Outlier Detection with Provable Bounds on Sub-Optimality".
- Chapter 8 is the paper "A Lookahead Algorithm for Robust Subspace Recovery". It introduces a fast algorithm for the outlier detection problem for PCA.
- Chapter 9 summarizes the results of this dissertation.

CHAPTER 2

HEURISTIC SEARCH ALGORITHM FOR DIMENSIONALITY REDUCTION OPTIMALLY COMBINING FEATURE SELECTION AND FEATURE EXTRACTION

Authors – Baokun He, Swair Shah, Crystal Maung, Gordon Arnold,

Guihong Wan, Haim Schweitzer

Department of Computer Science, EC 31

The University of Texas at Dallas

800 West Campbell Road

Richardson, Texas 75080-3021

Published in Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019.

Abstract

The following are two classical approaches to dimensionality reduction: 1. Approximating the data with a small number of features that exist in the data (feature selection). 2. Approximating the data with a small number of arbitrary features (feature extraction). We study a generalization that approximates the data with both selected and extracted features. We show that an optimal solution to this hybrid problem involves a combinatorial search, and cannot be trivially obtained even if one can solve optimally the separate problems of selection and extraction. Our approach that gives optimal and approximate solutions uses a "best first" heuristic search. The algorithm comes with both an *a priori* and an *a posteriori* optimality guarantee similar to those that can be obtained for the classical weighted A* algorithm. Experimental results show the effectiveness of the proposed approach.

2.1 Introduction

The representation of data in terms of a small number of features is a fundamental tool in data analysis. The compact representation allows for efficient manipulation, and may reveal relations in the data that are harder to identify. We study the unsupervised case, where a typical criterion of quality for the representation is the accuracy with which the data can be reconstructed from the compact representation.

Let m be the number of data items, each specified in terms of n features, so that the data can be viewed as the matrix X of m rows and n columns. A compact representation with r features is given by a matrix V of size $m \times r$, with $r \leq n$. The reconstruction of X from V is computed by $X \approx VA$, where A is the $r \times n$ coefficients matrix. We note that the matrix VA is of rank r, so that X is being approximated by a rank r matrix. Conversely, any rank r matrix can be expressed as the product VA, and thus gives a compact representation in terms of r features.

Input: the matrix X, the integer r.
Output: the r vectors $v_1, \ldots, v_r$.
1 Compute the matrix $B = XX^T$.
2 $v_1, \ldots, v_r$ are the top r eigenvectors of B.

Figure 2.1: The algorithm for optimal feature extraction

2.1.1 Previous Work

Studies of dimensionality reduction distinguish between the case where the columns of V must also be columns of X (feature selection), and the case in which this constraint is not enforced (feature extraction). We review these two approaches and then propose to combine them. We show that applying selection followed by extraction or vice versa does not give the optimal hybrid representation.

Let Θ be a matrix norm, then the error of approximating the matrix X by VA is given by:

$$X \approx VA, \quad \text{error} = \Theta(X - VA) \tag{2.1}$$

Feature extraction.

The well-known algorithm for optimal feature extraction is shown in Fig.2.1. See, e.g., (Jolliffe, 2002; Li et al., 2017). Applications of this algorithm include the technique of principal component analysis (**PCA**), which is arguably the most popular feature extraction technique. With recent advances in numerical techniques for computing eigenvectors (e.g., (Halko et al., 2011; Li et al., 2017)) the algorithm in Fig.2.1 can be implemented efficiently even for large amounts of data.

Among the topics of current research are attempts to minimize the approximation error (2.1) in norms that are not unitarily invariant. This turns out to be very challenging. In particular, minimizing the entry-wise l_1 or l_0 norms is expected to improve the robustness of the estimation, but unfortunately the problem formulated in these norms turns out

to be NP-hard. See, e.g., Gillis (Gillis and Vavasis, 2018), Song (Song et al., 2017), and Bringmann (Bringmann et al., 2017). For the more general case of entrywise l_p norms see Chierichetti (Chierichetti et al., 2017).

Feature selection.

In feature selection, the columns of V are constrained to be columns of X. This is sometimes known as the Column Subset Selection Problem (**CSSP**). See, e.g. (Golub and Van-Loan, 2013). Even though the approximation obtained by feature selection is worse than the approximation obtained by feature extraction, there are advantages of feature selection that make it the preferred choice in many situations. For example:

- Unlike feature selection, the results obtained by feature extraction are "notoriously difficult to interpret in terms of the underlying data" (Drineas et al., 2008).
- Selected features generalize better than extracted features in machine learning tasks. See, e.g. (Guyon and Elisseeff, 2003).
- Functions computed from extracted features depend on all the features and are typically more expensive to evaluate than functions computed from few selected features.
- Feature selection retains the data sparsity.

To describe current and previous results we need the following notation. Let $E_{\mathrm{FE}}, E_{\mathrm{FS}}$ be the smallest errors obtainable by feature extraction and by feature selection respectively. Consider an algorithm α that produces a selection S from the matrix X. Its error is given by $E_\alpha(S, X) = \min_A \Theta(X - SA)$. For such algorithm one can define:

$$p_\alpha(X) = \frac{E_\alpha(S, X)}{E_{\mathrm{FE}}}, \quad p_\alpha = \max_X p_\alpha(X)$$

Then the value of p_α indicates the estimation quality in the worst-case (e.g., Boutsidis (Boutsidis et al., 2009), Golub (Golub and Van-Loan, 2013)). The motivation behind this definition is that for any algorithm α and a matrix X we have: $1 \leq \frac{E_\alpha(S,X)}{E_{\mathrm{FE}}} \leq p_\alpha$. Therefore,

small values of p_α imply better worst-case performance. For example, Deshpande (Deshpande and Rademacher, 2010) showed that for the Frobenius norm error in selecting r features $p_\alpha = \sqrt{r+1}$. Thus, we say that an algorithm α is optimal if $E_\alpha(S, X)$ is the smallest possible, and it is worst-case optimal if its p_α is the smallest possible.

Unsupervised feature selection formulated as CSSP has attracted a lot of attention, with the first algorithm (pivoted QR) being developed more than 50 years ago (Businger and Golub, 1965). Recent results improve the accuracy, the running time, and the number of passes (e.g., (Paul et al., 2015; Maung and Schweitzer, 2013). The problem was recently proved NP-hard (Shitov, 2017a). There are, however, polynomial algorithms that are worst-case optimal, and nontrivial optimal algorithms that run much faster than exhaustive search.

Numerical linear algebra studies focus on algorithms for minimizing the Spectral norm. The deterministic algorithm with the best worst-case error can be found in (Gu and Eisenstat, 1996). A randomized algorithm with an improved worst-case accuracy for the Spectral norm is described in (Boutsidis et al., 2009). The theoretical computer science community produced worst-case optimal and near optimal randomized algorithms for the Frobenius norm. These include, among others, (Deshpande et al., 2006; Guruswami and Sinop, 2012). A worst-case optimal deterministic algorithm for the Frobenius norm is given in (Deshpande and Rademacher, 2010; Guruswami and Sinop, 2012).

The algebraic approach taken by most researchers was shown effective in deriving worst-case optimal algorithms, but so far has not produced optimal algorithms. Recent studies using classical AI tools of combinatorial search were used to derive optimal and near optimal algorithms in the Frobenius norm. See Arai (Arai et al., 2015, 2016).

Hybrid low rank representation.

As discussed above feature extraction and feature selection each have unique advantages and disadvantages. A hybrid representation that includes both extracted and selected features

was previously proposed in (Kneip and Sarda, 2011) and (Wang, 2012). The main idea is that feature extraction works well in situations where the features are highly correlated, while feature selection works well in situations where the data is uncorrelated. Therefore, these studies apply feature extraction to remove the correlated components and follow it by feature selection. As we show this approach is not optimal.

2.1.2 Our Results

In our model we fix both the number of selected features and the number of extracted features, and attempt to perform selection and extraction to minimize the approximation error in various norms. We show that the optimal combination of extraction and selection cannot be obtained by separate optimal algorithms for selection and extraction and requires a combinatorial search. To the best of our knowledge we are the first to make this observation.

The model we propose has r_1 selected features and r_2 extracted features. The approximation of X is given by:

$$X \approx SA_1 + VA_2, \text{ error} = \min_{A_1,A_2} \Theta(X - SA_1 - VA_2) \tag{2.2}$$

where S consists of r_1 columns from X and the r_2 columns of V are unconstrained. We refer to the representation in (2.2) as the "Hybrid Low Rank", or **HLR**. Our main result is an algorithm that computes HLR for any unitarily invariant error criteria. Observe that the HLR has simple feature extraction and simple feature selection as special cases.

The algorithm.

An obvious approach to obtain a hybrid low rank representation is to start with the selection of r_1 features and follow it with the extraction of r_2 features. Another alternative is to have the order of selection and extraction reversed. However, it turns out (see Section 2.2.1) that *neither of these approaches is optimal*. Instead, we propose to use variants of a "best first" heuristic search to find optimal and near optimal HLR solutions.

The algorithm that we develop is based on the combinatorial approach to feature selection described in Arai (Arai et al., 2016). The authors define a search graph for subsets, and use variants of A* to find a solution. The key to their algorithm is the introduction of heuristic functions that use eigenvalues. We show that the solution to the HLR can be found in a similar way, but with different heuristic functions.

Main contributions.

- A heuristic search algorithm for computing optimal and near optimal Hybrid Low Rank (HLR).
- *A priori* and *a posteriori* bounds for the algorithm.
- Since feature selection is a special case of the HLR ($r_2 = 0$), our HLR algorithm can also be used for optimal feature selection in all unitarily invariant norms. In particular this gives the first nontrivial optimal feature selection algorithm for the spectral norm and for the nuclear norm.

2.2 Hybrid Low Rank Representations

To simplify expressions related to matrices that are sometimes used as sets of columns we use the following notation: For two matrices A, B with the same number of rows we write $A \subset B$ to indicate that the columns of A are a subset of the columns of B. We write $|A|$ for the number of columns in A, and $[A|B]$ for the matrix consisting of the columns of A followed by the columns of B.

Let Θ be an error criterion. We consider the following approximation errors:

$$\boxed{\begin{aligned} &E_{\mathrm{FE}}(X,r) = \min_{V,A} \Theta(X - VA) \\ &\qquad \text{subject to } |V| = r \\ &E_{\mathrm{FS}}(X,r) = \min_{S,A} \Theta(X - SA) \\ &\qquad \text{subject to } S \subset X,\ |S| = r \\ &E_{\mathrm{HLR}}(X,r_1,r_2) = \min_{S,A_1,V,A_2} \Theta(X - SA_1 - VA_2) \\ &\qquad \text{subject to } S \subset X,\ |S| = r_1,\ |V| = r_2 \end{aligned}} \tag{2.3}$$

From (2.3) it easily follows that with $r = r_1 + r_2$ we have:

$$E_{\mathrm{FE}}(X,r) \leq E_{\mathrm{HLR}}(X,r_1,r_2) \leq E_{\mathrm{FS}}(X,r) \tag{2.4}$$

Thus, one would expect the HLR to have some desired properties of feature selection combined with some desired properties of feature extraction. For example, r_1 of the HLR features are easy to interpret (as in feature selection), and only r_2 of them are hard to interpret (as in feature extraction).

2.2.1 Greedy HLR Is Not Optimal

Suppose we are given a black box algorithm that computes optimal selection, and another black box algorithm that computes optimal extraction. We show by example that one cannot perform optimal selection followed by optimal extraction, or vice versa, to compute the optimal HLR. Consider the following two matrices:

$$X_1 = \begin{pmatrix} 100 & 0 & 1 \\ 0 & 1 & 100 \\ 0 & 100 & 50 \end{pmatrix}, \quad X_2 = \begin{pmatrix} 20 & 0 & 12 \\ -5 & 0 & 100 \\ 10 & 30 & 0 \end{pmatrix}$$

The goal for both matrices is to optimally select one column and extract one feature ($r_1 = 1, r_2 = 1$). If optimal selection of one feature is applied to X_1, the best selection (in Frobenius norm) is Column 3 (the error is $E_{\text{FS}}(X_1, 1)$=133.9). Combining the selection of Column 3 with an optimally extracted single feature reduces the error to 89.0. This, however, is not optimal. The selection of Column 1 with one extracted feature reduces the error to $E_{\text{HLR}}(X_1, 1, 1)$=77.4 which is optimal. This shows that optimal selection followed by optimal extraction does not guarantee the optimal HLR.

Similarly, if optimal extraction of one feature is applied to X_2 followed by optimal selection, the error is reduced to 20.44. The selection in this case is Column 2. This is not optimal since it is possible to extract a feature followed by the selection of Column 3 and reduce the error to $E_{\text{HLR}}(X_2, 1, 1)$=18.8. This shows that optimal extraction followed by optimal selection does not guarantee the optimal HLR.

2.3 HLR by Heuristic Search

A recent paper (Arai et al., 2016) has shown how to solve CSSP with the weighted A* algorithm for the Frobenius norm. They create a graph of subsets and perform the search on that graph. We use the same graph to convert the HLR into a graph search problem and study the performance of graph search algorithms for this problem. We propose two heuristics in a standard "best-first" setting. The first heuristic, that we call u, is an upper bound on the optimal HLR value. As we show, selecting graph nodes according to u gives a fast greedy algorithm.

The second heuristic, that we call f, is a lower bound on the optimal HLR value. We prove that using f by itself gives an algorithm that is guaranteed to find the optimal solution. Experimental results show that the algorithm runs much faster than exhaustive search (and produces the same results).

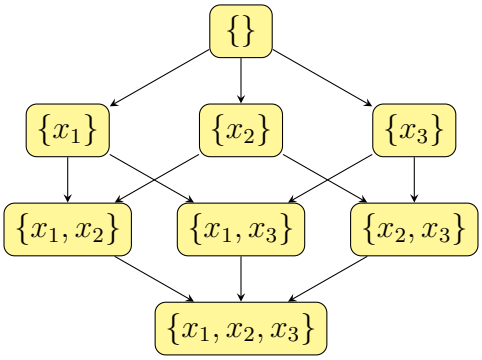

Figure 2.2: Example of the subsets graph

We linearly combine f and u to create the following heuristic: $f' = f + \epsilon u$. This gives a much faster algorithm than using f by itself. This is similar to the weighted A* approach, and we prove that the solution found by our algorithm comes with guaranteed bounds on its accuracy.

2.3.1 The Subset Graph

The subsets graph is created with nodes corresponding to column subsets. There is an edge from subset S_i to subset S_j if adding one column to S_i creates S_j. The graph generated for the matrix $X = (x_1, x_2, x_3)$ is shown in Fig.2.2.

Even though a subset graph is not a tree, it has two properties that are typically associated with trees. The first property is that it has a root, corresponding to the empty subset. The second is that all paths leading from the root to a node can be considered equivalent. For example, if the goal node $\{x_1, x_3\}$ is found, it is irrelevant if it is reached by the path $\{\} \rightarrow \{x_1\} \rightarrow \{x_1, x_3\}$ or by the path $\{\} \rightarrow \{x_3\} \rightarrow \{x_1, x_3\}$. This is similar to the case of a tree where the choice of path leading to a node is irrelevant since there is a unique path leading from the root to any node.

Input: X,r_1,r_2, and a heuristic function $f'(n)$.
Output: a subset S of selected columns.
Data Structures: Each node n_i keeps the subset S_i, and f'_i. Two global lists: the fringe list L, and the closed nodes list C.
Initialization: Put an empty subset into L.

```
1  while L is nonempty do
2  |  Pick n_i with the smallest f'_i from L. Ties are resolved in favor of the larger |S_i|
   |   (depth).
3  |  if S_i contains r_1 columns then
4  |  |  Stop and return S_i as the solution subset.
5  |  else
6  |  |  Add n_i to C.
7  |  |  for each child n_j of n_i do
8  |  |  |  if n_j is not in C or L then
9  |  |  |  |  Compute f'_j from X, S_j, r_1, and r_2.
10 |  |  |  |  Put n_j with its corresponding f'_j in L.
11 |  |  |  end
12 |  |  end
13 |  end
14 end
```

Figure 2.3: The best-first search algorithm.

2.3.2 The Heuristic Search Algorithm

The algorithm in Figure 2.3 performs the search for the optimal HLR. It is similar to the standard "best-first" algorithm except for the following notable difference. The standard graph search algorithm updates a node in the fringe if a better path to it is found (in Line 8 of the algorithm, when n_j is in the fringe). In our algorithm there is no such update. As explained in Section 2.3.1, all paths to the same subset are equivalent, and the value of the node depends only on the subset and not on the path leading to the subset.

2.3.3 Heuristic Functions

The HLR is defined in terms of X, r_1, r_2. At each node n_i the subset S_i and its size $k_i = |S_i|$ are known. Recall that the error E_{HLR} is the smallest error of approximating X by a selection

of r_1 columns and the best possible additional r_2 unconstrained vectors. The function d_i at node n_i is defined as the smallest error of approximating X by a selection of r_1 columns that include S_i and the best possible additional r_2 unconstrained vectors. This is the value at the best goal node below n_i. The function u_i at node n_i is defined as the smallest error of approximating X by the selection S_i and the best possible additional r_2 unconstrained vectors. The function f_i at node n_i is defined as the smallest error of approximating X by the selection S_i and the best possible additional $r_1 + r_2 - k_i$ unconstrained vectors, where $k_i = |S_i|$.

$$
\begin{aligned}
E_{\mathrm{HLR}}(X, r_1, r_2) &= \min_{S, A_1, V, A_2} \Theta(X - SA_1 - VA_2) \\
&\quad \text{subject to} \;\; S \subset X, |S| = r_1, |V| = r_2 \\
d_i = d(n_i, r_1, r_2) &= \min_{S, A_1, V, A_2} \Theta(X - SA_1 - VA_2) \\
&\quad \text{subject to} \;\; S_i \subset S \subset X, |S| = r_1, |V| = r_2 \\
u_i = u(n_i, r_2) &= \min_{A_1, V, A_2} \Theta(X - S_iA_1 - VA_2) \\
&\quad \text{subject to} \;\; |V| = r_2 \\
f_i = f(n_i, r_1, r_2) &= \min_{A_1, V, A_2} \Theta(X - S_iA_1 - VA_2) \\
&\quad \text{subject to} \;\; |V| = r_1 + r_2 - k_i
\end{aligned}
\tag{2.5}
$$

Observe that E_{HLR} and d_i cannot be calculated efficiently since the optimal selection S is unknown. By contrast, u_i and f_i use only the partial selection available at the given node, and as shown later can be computed efficiently. Clearly, the best heuristic choice for the algorithm is $f'_i = d_i$. But since it cannot be efficiently calculated we consider other choices using f_i and u_i. The motivation behind these choices is that both f_i and u_i can be viewed as approximations of d_i, as shown in Proposition 1.

Proposition 1: For each node n_i:

$$f(n_i, r_1, r_2) \leq d(n_i, r_1, r_2) \leq u(n_i, r_2)$$

and at a goal node (where k_i=r_1) the inequalities become equalities.

Proof:

- To see that $f_i \leq d_i$ observe that both f_i and d_i use the same number of vectors (r_1+r_2), and both use the k_i vectors in S_i. The rest of the vectors are unconstrained in f_i but partially constrained in d_i. This proves the left hand side inequality.
- To see that $d_i \leq u_i$, let S_i, V be the vector subsets that are used to calculate u_i. The minimum in the definition of d_i includes the subsets S_i and V (and additional vectors). This proves the right hand side inequality.
- If $k_i = r_1$ then the definitions of f_i and u_i are identical.

■

2.4 Three Variants of the Algorithm

Proposition 1 shows that the optimal heuristic d_i is "sandwiched" between f_i and u_i. We consider three different options for running the algorithm. The first is the choice $f'_i = u_i$, the second is the choice $f'_i = f_i$, and the third takes f'_i between f_i and u_i. Specifically, for the third choice we observe that taking $f'_i = (1-\beta)f_i + \beta u_i$ with $0 \leq \beta \leq 1$ is equivalent to taking $f'_i = f_i + \epsilon u_i$ with $\epsilon = \frac{\beta}{1-\beta}$, $\epsilon \geq 0$.

The Greedy HLR algorithm: $f'_i = u_i$.

We prove in Theorem 1 that using f'_i=u_i gives a greedy algorithm that examines exactly r_1 nodes before terminating with a solution.

The Optimal HLR algorithm: $f'_i = f_i$.

We prove in Theorem 2 using f'_i=f_i gives an algorithm that is guaranteed to find the optimal solution.

The Suboptimal HLR algorithm: $f'_i = f_i + \epsilon u_i$.

We prove in Theorem 3 that using f'_i=f_i+ϵu_i guarantees a solution "close" to the optimum.

Bounds on the distance between the optimum and the solution can be calculated *a priori* before the algorithm is executed, and *a posteriori*, after the algorithm terminates.

2.4.1 Proofs

Theorem 1. With the choice $f'_i = u_i$, the algorithm terminates after examining r_1 nodes.

Theorem 2. With the choice $f'_i = f_i$, the algorithm terminates with an optimal solution. The optimal solution error is $E_{\mathrm{HLR}}(X, r_1, r_2)$.

Theorem 3. Let n_* be an optimal solution node for the HLR. Let $e^* = E_{\mathrm{HLR}}(X, r_1, r_2)$ be the error at n_*. Suppose the algorithm is using $f'_i = f_i + \epsilon u_i$, with $\epsilon \geq 0$. Let n_{**} be the goal node found by the algorithm. Let e^{**} be the error at n_{**}, and f_{**} be the value of f at n_{**}. Let $u_{\max}$ be the largest value of u in the nodes remaining at the Fringe list after the goal node is reached. Then:

$$e^{**} \leq e^* + \epsilon(u_{\max} - f_{**}) \tag{2.6}$$

Lemma 1. f_i is monotonically increasing along any path.

Proof of Lemma 1: Suppose n_j is a child of n_i, so that $S_j = [S_i|x]$, where x is the added column. We need to show:

$$f(n_j, r_1, r_2) = \min_{|V_j|=r_1+r_2-k_i-1} \min_A \Theta(X - [S_i|x|V_j]A)$$

$$\geq \min_{|V_i|=r_1+r_2-k_i} \min_A([S_i|V_i]A) = f(n_i, r_1, r_2)$$

This follows because the minimum on the right hand side has one unconstrained vector that is constrained on the left hand side. ■

Lemma 2. The value of u_i is monotonically decreasing along any path.

Proof of Lemma 2: We need to show that if n_j is the child of n_i then $u_j \leq u_i$. From the definition in (2.5) the right hand side reduces the error with the subset S_i while the left hand side reduces the error with the subset S_j, which includes S_i and one additional column. Clearly, the additional column can only reduce the error. ■

Lemma 3. Consider the choice $f'_i = u_i$. Let n_i be the node picked at Line 2 of the algorithm. Let n_j be a child of n_i. The following two properties hold:

a. The depth $|S_j|$ of n_j is larger than the depth of all other nodes currently in the fringe.

b. The next node to be picked is a child of n_i.

Proof of Lemma 3: The proof is by induction. Property a follows trivially from Property b. To prove Property b observe that from Lemma 2, u_i is monotonically decreasing (non-increasing) along any path. Therefore, the f' values of the children of n_i will be no greater than the f' values of all the nodes currently in the fringe. Property a guarantees that the tie breaker will always be decided in favor of a child, so that the child of n_i will be selected next. ■

Lemma 4. Suppose Theorem 3 is false. Then for any node n_z on the path from the root to n_* the following condition holds: $f'_z < f'_{**}$.

Proof of Lemma 4: The falsehood of Theorem 3 can be written as follows: $e^{**} > e^* + \epsilon(u_{\max} - e^{**})$. Since both n_* and n_{**} are goal nodes Proposition 1 implies: $e^{**} = f_{**} = u_{**}$ and $e^* = f_* = u_*$. Using this and some algebra it can be shown that an equivalent falsehood condition is: $f_{**} > f_* + \frac{\epsilon}{1+\epsilon}(u_{\max} - f_*)$. The lemma can now be proved as follows:

$$
\begin{aligned}
f'_{**} &= f_{**} + \epsilon u_{**} = (1+\epsilon) f_{**} && (c_1)\\
&> (1+\epsilon) f_* + \epsilon(u_{\max} - f_*) = f_* + \epsilon u_{\max} && (c_2)\\
&\geq f_z + \epsilon u_{\max} \geq f_z + \epsilon u_z = f'_z && (c_3)
\end{aligned}
$$

c_1 : from the definition of f'. c_2 : from the equivalent falsehood assumption. c_3 : from Lemma 1 $f_* > f_z$. ■

Proof of Theorem 1:

The proof follows trivially from Lemma 3. ■

Proof of Theorem 2:

The proof follows as a corollary of Theorem 3 with $\epsilon = 0$. ■

Proof of Theorem 3:

If the theorem is false then from Lemma 4 it follows that all nodes on the path from the root to n_* have smaller f'_i values than f'_{**}. Since at any given time at least one of them is in the fringe list, they should all be selected before n_{**} is selected. But this means that n_* is selected as the solution and not n_{**}. ■

2.4.2 *A priori* and *A posteriori* Bounds

Both the Greedy HLR and the Suboptimal HLR are not guaranteed to produce the optimal solution. We proceed to show how to obtain bounds on how close their solution is to the optimal. We call a bound *a priori* if it can be calculated before the run of the algorithm and *a posteriori* if it can only be calculated after the run of the algorithm.

Consider a run of a nonoptimal algorithm producing the nonoptimal value of f_{**}, while the optimal value is f_*. The value of f_{**} can be bounded as follows:

$$f_{**} \leq f_* + B, \quad B \geq f_{**} - f_*$$

We refer to the value of B as a bound, where a smaller B indicates a better bound, and $B = 0$ implies an optimal solution.

The *a posteriori* bounds that we describe require the examination of the fringe list after the run of the algorithm. In particular we compute the following two values from the fringe list:

$$f_{\min} = \min_{n_i \in F} f_i, \quad u_{\max} = \max_{n_i \in F} u_i$$

In addition, the *a posteriori* bounds use the value f_{**} at the (nonoptimal) goal node.

From Lemma 1 it follows that $f_* \geq f_{min}$, so that $B_1 = f_{**} - f_{\min}$ is an *a posteriori* bound for all variants of the algorithm.

Greedy HLR.

Greedy HLR has the following *a priori* bound: $B_2 = u_{\text{root}} - f_{\text{root}}$. This bound follows from Proposition 1. The only *a posteriori* bound of Greedy HLR is B_1.

Suboptimal HLR.

Suboptimal HLR has the following *a priori* bound: $B_3 = \epsilon u_{\text{root}}$. This bound follows from Theorem 3 and Lemma 2 by observing that:

$$\epsilon(u_{\max} - f_{**}) \leq \epsilon u_{root}$$

Suboptimal HLR has two *a posteriori* bounds: B_1 and $B_4 = \epsilon(u_{\max} - f_{**})$. Clearly, its effective bound is the minimum of the two.

2.4.3 Using *A posteriori* Bounds to Improve the Result

A paper by Thayer and Ruml (Thayer and Ruml, 2008) shows how to use the *a posteriori* bounds to improve the output of the classic weighted A* algorithm. The idea is to run the weighted A* algorithm to convergence, and then identify the node in the fringe list that affects the bound the most. That node is then expanded, its children are added to the fringe, and the weighted A* algorithm continues with the new fringe. Typically, a single iteration

Input: X,r_1,r_2, ϵ, T.
Output: a subset S of selected columns.

Start with an empty fringe F and a Closed list C.
for $t = 1, \ldots, T$ **do**
    Run either the Greedy HLR or the Suboptimal HLR to convergence, using F and C.
    Go over the fringe F, identify the nodes n_{b1} and n_{b4}, and compute the values of B_1, B_4.

$$n_{b1} = \arg\min_{n_i \in F} f_i, \quad n_{b4} = \arg\max_{n_i \in F} u_i$$

    if $B_1 < B_4$ **then**
        Expand n_{b1}.
    else
        Expand n_{b4}.
    end
end

Figure 2.4: Optimistic Search Algorithm.

of this algorithm would either improve goal node or improve the *a posteriori* bound. Fig.2.4 describes the algorithm in detail.

2.5 Relationship to Previous Work

In this section we discuss the relationship between the algorithm presented here and classical work on the weighted A* algorithm. We also compare our work to the results of (Arai et al., 2016).

There are many similarities between our model and the classical weighted A* graph search algorithm (e.g. (Pearl, 1984)). The most important one is introduction of the heuristic function f with the following three key properties: 1. f is a lower bound on the true value at the goal. 2. f is monotonically increasing. 3. At a goal node the value of f is the value that one attempts to minimize. Although a heuristic function is also introduced in the classical theory of (weighted) A* search, its definition is entirely different. On the other hand, there is no function in our setting that corresponds naturally to the functions g (distance from the

root) or h (heuristic) in the classical theory. Similarly, there is no natural function in the classical theory that corresponds to the function u in our setting.

The similarity in the properties of f makes our suboptimality proofs similar to the classical proofs of weighted A* suboptimality (e.g. (Pearl, 1984)). However, since the heuristic functions used here are different from those used in graph search, one cannot use the classical proofs "as is" and apply them to our case. In particular, our Lemma 1 has a corresponding lemma in the classical theory, and our proof idea of Lemma 4 is similar (but not identical) to the classical theory. However, there is no correspondence to our Proposition 1 (right hand side), Lemma 2, and Theorem 1. The bound obtained in Theorem 3 is also different. The result for the classical weighted A* algorithms are in terms of a relative bound, while the guarantees in our case are in terms of an additive bound. Still, the similarity between the approaches enables us to map ideas that were developed in the classical theory to our setting. We demonstrated this with the Optimistic Search Algorithm that can be applied almost verbatim in our case. (The only difference is the exact formulas for the *a posteriori* bounds.)

Our work is motivated by the study described in Arai (Arai et al., 2016). The main difference is that our results are for the HLR, and do not use any norm specific assumptions. By contrast, the Arai proofs are for the CSSP which is a special case of the HLR, and they make use of the Frobenius norm assumption.

2.6 Experimental Results

Efficiently computing f_i and u_i.

The optimality proof does not use any properties of the error criterion Θ. However, an efficient computation requires the norms to be unitarily invariant. From the definition of f_i, u_i in (2.5) the challenge is the computation of the coefficient matrices A_1, A_2, and the

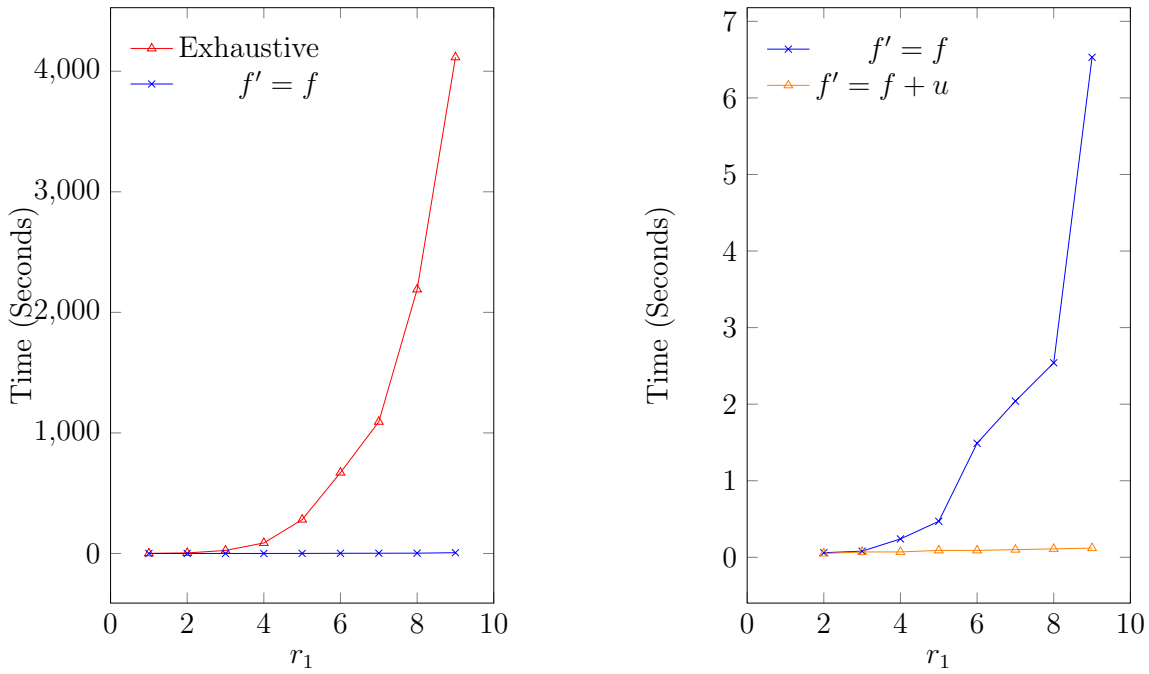

Figure 2.5: Run-time results HLR on the dataset *vehicle*. x-axis shows r_1 and r_2=10 − r_1. Error criterion is the Schatten p-Norm with p=0.25.

Table 2.1: Accuracy comparison under Nuclear norm and Spectral norm. The minimum error is highlighted.

Error Criterion	r_1	$f' = f$	$f' = f + 0.2u$	$f' = f + 0.4u$	$f' = f + 0.8u$	$f' = u$	$f' = u$ *a priori*	$f' = u$ *a posteriori*	ARSS	GE
vehicle dataset ($m = 846$, $n = 18$)										
Nuclear	5	**1399.20**	1402.64	1569.49	1569.49	1569.49	24490.7	270.83	3465.75	-
Spectral	5	**247.58**	326.12	326.12	326.12	326.12	19600.32	82.66	-	248.58
Nuclear	10	**466.85**	520.18	520.18	520.18	520.18	25371.7	105.55	1682.16	-
Spectral	10	**112.19**	138.80	144.99	144.99	148.60	19744.0	48.85	-	131.68
spectf dataset ($m = 267$, $n = 45$)										
Nuclear	5	**3814.14**	**3814.14**	3816.69	3817.42	3821.42	8334.75	435.57	4598.65	-
Spectral	5	**252.69**	257.91	290.60	290.60	280.45	6841.58	78.09	-	348.24
Nuclear	15	-	-	2297.04	**2292.79**	**2292.79**	9850.00	457.51	3091.28	-
Spectral	15	-	152.12	**151.83**	165.41	1883.42	6938.17	82.43	-	154.62
libras dataset ($m = 360$, $n = 90$)										
Nuclear	4	**68.44**	68.53	68.53	68.48	71.55	135.88	11.03	91.79	-
Spectral	4	**8.558**	9.954	9.954	9.954	13.182	84.62	4.80	-	11.863
Nuclear	30	-	**6.134**	6.185	6.322	6.322	189.90	1.89	8.235	-
Spectral	30	-	**0.343**	0.351	0.351	0.712	92.80	0.50	-	0.4211

unconstrained matrix V. For all unitarily invariant norms A_1, A_2 can be calculated with a pseudo inverse, and the matrix V by calculating eigenvectors. For other norms it is not immediately clear how to compute these values. Specifically, for the entry-wise l_0, l_1 these calculations are known to be NP-hard. See Gillis (Gillis and Vavasis, 2018).

Running time.

Fig.2.5 shows running-time on the dataset *vehicle*. The left panel shows that the algorithm with $f_i' = f_i$ is significantly faster than exhaustive search. The right panel shows that using $f_i + u_i$ runs much faster than f_i.

Optimal feature selection.

As discussed in Section 2.1.2 feature selection is a special case of the HLR. Our algorithm is the first nontrivial algorithm for optimal feature selection for unitarily invariant error criteria besides Frobenius. The results for various norms are shown in Table 2.1. We do not include the results when algorithms run more than five minutes. They are compared with two algorithms. The column ARSS shows results obtained by the algorithm of Zhu (Zhu et al., 2015). Their algorithm cannot be used to compute the Spectral norm, so we use the algorithms of Gu and Eisenstat (Gu and Eisenstat, 1996) instead.

Minimizing entry-wise l_0 and l_1 norms.

As discussed in Section 2.1.1 a current topic of interest is the computation of low rank representation minimizing entrywise l_0 and l_1 norms. We found experimentally that feature selection typically gives lower errors for entry-wise l_0 and l_1 norms than feature extraction, though feature extraction performs better in terms of the unitarily invariant norm used as the error criterion. The hybrid low rank approach allows us to balance this trade-off, reducing the unitarily invariant norm while at the same time reducing the error in the l_0 and/or l_1

Table 2.2: Reduction in l_0 and l_1 entrywise norms with increased r_1.

Norm	r_1	r	l_0 error	l_1 error
spectf dataset ($m = 267$, $n = 45$)				
Nuclear	1	30	0.693	1.16
	3	30	0.671	1.15
	5	30	0.647	1.13
$p = 0.25$	1	30	0.691	1.16
	3	30	0.675	1.16
	5	30	0.649	1.14
vehicle dataset ($m = 846$, $n = 18$)				
Frobenius	1	10	0.562	0.92
	5	10	0.472	0.831
	9	10	0.342	0.71
$p = 0.4$	1	10	0.562	0.92
	5	10	0.465	0.819
	9	10	0.342	0.71

Table 2.3: Greedy HLR on TechTC01 data with relative bounds.

r_1	*Bound*	$r_2 = 0$	$r_2 = 5$	$r_2 = 10$
100	*a priori*	530.37	122.62	100.46
	a posteriori	0.19	0.15	0.13
	solution error	21354.43	15511.81	11278.12
120	*a priori*	1606.61	430.05	391.93
	a posteriori	0.24	0.16	0.12
	solution error	7056.89	4445.01	2909.46
140	*a priori*	9410.57	4065.81	9779.79
	a posteriori	0.28	0.17	0.07
	solution error	1205.26	470.98	116.84

norms. Table 2.2 shows that for a fixed r, increasing r_1 indeed reduces the entry-wise l_0 and l_1 norms.

Experiments with big sparse data.

We describe experiments with the Greedy HLR algorithm applied to the TechTC dataset. The matrix size in this case is 163×29261. This means that the algorithm selection is from

Table 2.4: Relative *a posteriori* bounds of the Greedy HLR with Optimistic Search Algorithm on the TechTC01 dataset.

$r_1 : r_2$	Iterations			solution error
	1	10	100	
42:0	0.09173	0.09171	0.09156	273585.83
42:5	0.06124	0.06122	0.06105	214917.10
42:10	0.04434	0.04434	0.04416	170169.98
5:0	0.04592	0.04528	0.03664	2.02e6
5:5	0.01126	0.01033	0.00854	1.16e6
5:10	0.00388	0.00385	0.00299	8.25e5

29261 features. Exhaustive search algorithms are clearly not practical in this case. (For example, there are approximately 10^{288} subsets of selecting 100 features out of 29261, which is significantly more than the number of atoms in the universe.)

The results are shown in Table 2.3. The value of the bounds is given as the ratio between the bounds and the errors at the goal node.

Experiments with the Optimistic Search Algorithm.

We do experiments with the Optimistic Search Algorithm, as discussed in Section 2.4.3. The results are shown in Table 2.4. Observe that the solution error does not change, but the relative error bound is being reduced (slightly) with additional iterations.

2.7 Concluding Remarks

This paper introduces the "Hybrid Low Rank" (HLR) representation of a matrix as a low rank matrix representation that uses both selected features and extracted features. It was shown that an optimal HLR representation cannot be obtained by first selecting features and then extracting features, or vice versa. Instead, it requires a combinatorial search.

An algorithm that uses the "best-first" heuristic search approach was described. Three variants, optimal, suboptimal and greedy, were described, This heuristic search technique

allows us to compute *a priori* and *a posteriori* bounds, which show how close the results are to the optimal solution. *A priori* bounds can be computed before the run of the algorithm. *A posteriori* bounds can be computed after termination. The paper also shows how to use the *a posteriori* bound to improve the solution accuracy.

A short abstract describing some of the results in this paper appears in (Shah et al., 2018). Similar ideas were also used to derive new algorithms for robust PCA. See (Shah et al., 2017, 2018).

CHAPTER 3

HEURISTIC SEARCH FOR APPROXIMATING ONE MATRIX IN TERMS OF ANOTHER MATRIX

Authors – Guihong Wan, Haim Schweitzer

Department of Computer Science, EC 31

The University of Texas at Dallas

800 West Campbell Road

Richardson, Texas 75080-3021

Published in Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI 2021.

Supplementary material is provided in the next chapter.

Abstract

We study the approximation of a target matrix in terms of several selected columns of another matrix, sometimes called "a dictionary". This approximation problem arises in various domains, such as signal processing, computer vision, and machine learning. An optimal column selection algorithm for the special case where the target matrix has only one column is known since the 1970's, but most previously proposed column selection algorithms for the general case are greedy. We propose the first nontrivial optimal algorithm for the general case, using a heuristic search setting similar to the classical A^* algorithm. We also propose practical sub-optimal algorithms in a setting similar to the classical Weighted A^* algorithm. Experimental results show that our sub-optimal algorithms compare favorably with the current state-of-the-art greedy algorithms. They also provide bounds on how close their solutions are to the optimal solution.

3.1 Introduction

Let $Y = (y_1 \dots y_N)$ be a "target" matrix of m rows and N columns. Let $X = (x_1 \dots x_n)$ be a "dictionary" matrix of m rows and n columns. We consider the problem of selecting $k \leq n$ columns from X that can be used to linearly approximate Y as follows:

$$Y \approx SA, \tag{3.1}$$

where $S = (x_{s_1} \dots x_{s_k})$ is the selection matrix of size $m \times k$, and A is the coefficient matrix of size $k \times N$. We evaluate the quality of the selected columns in S with the Frobenius norm:

$$E(S) = \min_A \|Y - SA\|_F^2. \tag{3.2}$$

This formulation includes several well-known special cases. Supervised column subset selection corresponds to the case where $N = 1$. See, e.g., (Hastie et al., 2009; Boutsidis et al.,

2013). When $N > 1$, the problem is sometimes known as sparse multi-target prediction, or simultaneous sparse approximation (e.g., (Tropp, 2004; Maung and Schweitzer, 2015; Wan and Schweitzer, 2021b)). When $Y = X$, it is known as the unsupervised column subset selection problem, or unsupervised feature selection problem (e.g., (Golub and Van-Loan, 2013; Arai et al., 2015)).

Optimal solutions, as well as approximations within a constant, are known to be NP-hard even for the $N = 1$ case (Natarajan, 1995; Davis et al., 1997; Amaldi and Kann, 1998), and for the unsupervised case where $Y = X$ (Shitov, 2017b). This implies that optimal solutions cannot be efficient. Previous studies have developed optimal algorithms that are exponential in the worst case but improve on exhaustive search, and fast algorithms that are not optimal.

Optimal algorithms for the unsupervised case ($Y = X$) were studied in (Arai et al., 2015; He et al., 2019). Optimal algorithms for the $N = 1$ case were studied in (Furnival and Wilson, 1974; Bertsimas et al., 2016, 2020). We are not aware of previous studies of optimal algorithms for the $N > 1$ case that we focus on in this paper.

The case in which the matrix to be approximated contains only one column ($N = 1$) has received a lot of attention. See, e.g., (Furnival and Wilson, 1974; Qian et al., 2015; Bertsimas et al., 2016; Hou et al., 2017; Qin et al., 2018; Huang et al., 2020). Applications include signal processing (e.g., (Selesnick, 2017; Qin et al., 2018)) and supervised feature selection in linear regression (e.g., (Furnival and Wilson, 1974; Qian et al., 2015; Hou et al., 2017; Bertsimas et al., 2016)). Convex relaxation approaches to feature selection replace some natural constraints (sometimes defined in terms of l_0 norm) with other convex constraints. An example is that the l_1 norm is used in the Lasso technique (Tibshirani, 1996). Another recent variant is the Pareto technique (Qian et al., 2015), where the authors treated the subset selection as a bi-objective optimization problem. The algorithm was proved to be optimal for data drawn from Exponential Decay distributions.

Heuristic search algorithms were recently applied to the unsupervised case where $Y = X$. See (Arai et al., 2015, 2016; He et al., 2019). They model the selection as a graph search

problem, and apply the A^* or Weighted A^* to solve it. Our method is motivated by this approach. The algorithms developed in these studies can be viewed as special cases of our method.

In the general case, the matrix Y contains $N > 1$ columns. We observe that one cannot simply apply an algorithm for the N=1 case separately to each column of Y. The challenge of the general case is to find columns in X that can simultaneously approximate all columns of Y. Previously proposed algorithms for this general case are generally greedy. They include (Maung and Schweitzer, 2015; Belmerhnia et al., 2014; Çivril and Magdon-Ismail, 2012; Tropp et al., 2006). Some of these algorithms are generated from the greedy algorithms for the $N = 1$ case. For example, the Simultaneous Orthogonal Matching Pursuit (SOMP) Algorithm (Tropp et al., 2006) was generated from the Orthogonal Matching Pursuit (OMP) (Tropp, 2004). Similarly, the Simultaneous Orthogonal Least Squares (SOLS) Algorithm (Cotter et al., 2005) is a direct generalization of the Orthogonal Least Squares (OLS) Algorithm (Chen et al., 1989). An algorithm established in (Çivril and Magdon-Ismail, 2012) improves the SOLS in term of runtime at the cost of increased memory. A recursive formulation in (Maung and Schweitzer, 2015) is used to improve the speed of the SOLS. A spectral pursuit algorithm presented in (Wan and Schweitzer, 2021b) is efficient when n and N are large. The running time and the memory requirements of these algorithms are summarized in Table 3.1. The complexity of our greedy variant is discussed in Section 3.4.

3.1.1 Our Approach

We propose a heuristic search approach for solving the general problem of selecting a subset of k columns from the dictionary matrix to simultaneously approximate the entire target matrix. Our heuristic functions are based on eigenvalues of related matrices. With proper selection of the heuristics, our algorithm can be tuned to give an optimal solution as well as

Table 3.1: Complexity of various approximate algorithms. T is the number of iterations. $r_x \leq \min(m, n)$.

Algorithms	**Time complexity**	**Memory complexity**
SOMP (Tropp et al., 2006)	$O(kmnN)$	$O(m(N+k))$
SSBR (Belmerhnia et al., 2014)	$O(Tkmn N)$	$O(m(N+k))$
SOLS (Cotter et al., 2005)	$O(kmnN)$	$O(m(n+N))$
CM (Çivril and Magdon-Ismail, 2012)	$O(nN(m+k))$	$O(m(n+N))+nN$
ISOLS (Maung and Schweitzer, 2015)	$O(mnN)$	$O(km)+2n$
SPXY (Wan and Schweitzer, 2021b)	$O(km(n+N))$	$O(m(n+N))$
Greedy variant (this work)	$O((k^2+m)r_x n)$	$O(r_x n)$

approximate solutions. The optimal variant runs slow and works only for small k or small dictionary matrices. The suboptimal variants run much faster and produce solutions with guarantees on how close the solutions are to the optima.

Main Contributions

The main contributions are summarized below:

- We describe the first nontrivial algorithm guaranteed to produce an optimal solution for the general case $N \geq 1$.
- We describe suboptimal algorithms that are more accurate than the greedy variants. Their solutions come with bounds on how far the solutions are from the optimal solution.
- We describe a greedy algorithm that produces results of similar accuracy to the current state of the art with a superior running time. It also comes with a bound on how far the solution is from the optimum.

3.2 The Proposed Algorithms

Heuristic search algorithms on graphs, such as A^* (Hart et al., 1968) and Weighted A^* (Pohl, 1970), are widely used for solving search problems that can be modeled as graph search. The goal is to find a path from a root node to a goal node minimizing the cost associated with

Algorithm 1: The Search Algorithm.

Input:

Y: a target matrix of N columns.

X: a dictionary matrix of n columns.

k: the desired number of columns to be selected from X.

$f(\cdot)$: a heuristic function.

r_y (optional): the desired rank of Y. Default:$r_y = \min(m, N)$.

r_x (optional): the desired rank of X. Default:$r_x = \min(m, n)$.

Output: a subset S of k columns.

Data Structures: Two global node lists: the fringe list F, and the closed list C.

Preprocessing: Dimensionality reduction.

Initialization: Put the empty subset into F.

```
1:  while F is nonempty do
2:    Pick node n_i (associated with subset S_i of size k_i) with the smallest heuristic f_i
      from F. Ties are resolved in favor of the larger k_i.
3:    if S_i contains k columns then
4:      Stop and return S_i as the solution subset.
5:    else
6:      for each child n_j of n_i do
7:        if n_j is not in C then
8:          Compute f_j for n_j.
9:          Put n_j in C.
10:         Put n_j in F.
11:       end if
12:     end for
13:   end if
14: end while
```

the path. The search is guided by functions associated with each node that are known as "Heuristics". With some heuristics, A^* is guaranteed to be optimal, but it may be very slow. Experimental results and theoretical analysis (e.g., (Pearl, 1984)) show that Weighted A^* runs faster than A^*. The algorithm we propose in this paper is similar to the standard A^* and Weighted A^*.

We describe the search algorithm in terms of a general heuristic function that will be defined later. It is shown in Algorithm 1. The same search procedure (with different heuristics) was also used in (Arai et al., 2015, 2016; He et al., 2019; Wan and Schweitzer, 2021a).

In our case, the graph is constructed as follows. A node contains a subset of selected columns from X and the corresponding f value based on the approximation error of Y. The root node corresponds to the empty subset. A goal node corresponds to a subset of k columns. The children of a node are created by adding a new column into its parent subset. Without loss of generality, we do not distinguish between a node and a subset. The goal of the algorithm is to search a node of k columns on the graph, minimizing the heuristic function f. The fringe list F is a list of nodes that need to be further evaluated, and the closed list C is a list of visited nodes that need not to be added into F again. Employing the standard "best first" strategy the algorithm selects the node from F with the smallest f value to be expanded.

There are two differences between the search procedure in Algorithm 1 and the typical (Weighted) A^* procedure. The first is in Line 9, where the node is immediately inserted into the closed list, and the second is in Line 10, where the classical A^* algorithm first checks if the node is already in the fringe list, and if so decides whether its value needs to be updated. The justification for these differences is that in our case all paths leading from the root node to a node are equivalent. Once a subset node is added into the fringe list other nodes with the same subset (obtained through a different path) will have the exact same heuristic values and therefore can be ignored. This implies that the fringe list is a subset of the closed list. (These observations show that the closed list can be implemented by a hash table, and the fringe list by a heap.)

3.3 Heuristic Functions

Recall that the error of approximating Y by a linear combination of k columns in S is given by (5.3). Let n_i be an arbitrary subset node. Let $k_i<k$ be the number of columns selected at n_i, and let S_i be the matrix formed by these columns. Let $\overline{k}_i=k-k_i$ be the number of columns

that still need to be selected. We consider the error that can be obtained by completing S_i to size k. Let S_i of size $\overline{k}_i$ be such completion. Its error is given by:

$$e(S_i) = E(S_i \cup S_i) = \min_{A_i, \overline{A}_i} \|Y - S_i A_i - S_i \overline{A}_i\|_F^2. \tag{3.3}$$

In the equation above, $E()$ is the error defined in (5.3), and $A_i, \overline{A}_i$ are arbitrary coefficient matrices. Thus, the smallest error of a solution subset that contains S_i would be:

$$d_i = \min_{S_i} e(S_i). \tag{3.4}$$

This shows that the best choice for the heuristic value f_i for a node n_i is $f_i = d_i$. Unfortunately, computing d_i is too expensive since it requires going over all the subsets S_i of size $\overline{k}_i$. However, we show that there is an effective computational approach to approximate d_i. We define two approximations l_i and u_i, and show that they bound d_i from below and above.

$$\begin{aligned} d_i &= \min_{S_i, A_i, \overline{A}_i} \|Y - S_i A_i - S_i \overline{A}_i\|_F^2, \\ u_i &= \min_{A_i} \|Y - S_i A_i\|_F^2 = \|Y - S_i A_i^*\|_F^2 = \|Y_i\|_F^2, \\ l_i &= \|Y_i - \overline{Y}_i\|_F^2, \end{aligned} \tag{3.5}$$

where $A_i^* = \arg\min_{A_i} \|Y - S_i A_i\|_F^2$, $Y_i = Y - S_i A_i^*$, $\overline{Y}_i$ is the best rank $\overline{k}_i$ approximation to Y_i.

Lemma 5. For any selection S_i of size $k_i \leq k$:

$$l_i \leq d_i \leq u_i \tag{3.6}$$

with equality if $k_i = k$.

Proof:

- To prove the right hand side inequality, observe that for any S_i the expression for u_i is the same as d_i if the matrix $\overline{A}_i$ is taken to be identically 0. This ignores the contribution of additional columns that may reduce the error further.

- To prove the left hand side inequality:

$$l_i = \|Y_i - \overline{Y}_i\|_F^2 \leq \min_{S_i, \overline{A}_i} \|Y_i - S_i\overline{A}_i\|_F^2 \qquad \text{(note1)}$$

$$= \min_{S_i, A_i, \overline{A}_i} \|Y - S_iA_i - S_i\overline{A}_i\|_F^2 = d_i$$

  The justification for the inequality in (note1) is that the rank of $S_i\overline{A}_i$ is at most $\overline{k}_i$.

- It remains to show that the inequalities become equalities when $k_i = k$. Under this condition, we have: $\overline{k}_i = 0$, which implies that $\overline{Y}_i = 0$ and $S_i = 0$. Therefore, l_i, d_i and u_i have same expression: $\|Y_i\|_F^2$. ■

3.3.1 Three Variants of the Search Algorithm

Clearly, the best choice for the heuristic function is $f_i = d_i$. However since it cannot be efficiently calculated, we consider the linear combination between l_i and u_i. Three options of the heuristic function are considered. The resulting variants are stated as in the following theorems.

Theorem 4. (the optimal variant) If $f_i = l_i$ then the algorithm is guaranteed to terminate with an optimal solution.

Theorem 5. (the greedy variant) If $f_i = u_i$ then the algorithm is greedy and terminates after expanding k nodes.

Theorem 6. (the weighted variant) If $f_i = l_i + \gamma u_i$, where $\gamma \geq 0$, and the algorithm terminates with the subset S^{**} that has an error of e^{**} then

$$e^{**} \leq e^* + \gamma\|Y\|_F^2,$$

where e^* is the smallest possible error.

3.3.2 Lemmas

Our proofs of the theorems require the following results.

Lemma 6. If n_j is a child of n_i then $l_i \leq l_j$.

Lemma 7. If n_j is a child of n_i then $u_j \leq u_i$.

Lemma 8. Consider the choice $f_i = u_i$. Let n_i be the node picked at Line 2 of the algorithm. Then the following two properties hold:

a. For each child n_j of n_i, the selection size $|S_j|$ is larger than the selection size of all nodes currently in the fringe list.

b. The next node to be picked up will be a child of n_i.

Instead of proving Theorem 6 directly, we prove a stronger version with a tighter bound:

Theorem 7. (the tighter weighted variant) If $f_i = l_i + \gamma u_i$, where $\gamma \geq 0$, and the algorithm terminates at the node n_{**} with the subset S^{**} that has an error of e^{**}, then:

$$e^{**} \leq e^* + \gamma(u_{\max} - l_{**})$$

where e^* is the smallest possible error, l_{**} is the value of the lower bound for n_{**}, and $u_{\max}$ be the largest value of the upper bound for the nodes remaining in the fringe after the goal node is reached.

To see that Theorem 7 implies Theorem 6, observe that $u_{\max} \leq \|Y\|_F^2$ and $l_{**} \geq 0$.

Lemma 9. Suppose Theorem 7 is false. Then for any node n_z on the path from the root to an optimal goal node n_*, the following condition holds: $f_z < f_{**}$.

Similar theorems were proved in (He et al., 2019) for the unsupervised case. However, the proof technique used in their study does not appear strong enough to prove these theorems

for the supervised case. The tool we use to prove the lemmas is the interlacing property of eigenvalues (Golub and Van-Loan, 2013).

The proof of Theorem 4 follows as a corollary of Theorem 6 with $\gamma = 0$. Using Lemma 8, Theorem 5 can be proved. Theorem 6 follows from Theorem 7, and Theorem 7 can be proved by contradiction and using the results in Lemma 9. Detailed proofs are given in the supplementary material.

3.4 Efficient Heuristics Calculation

The following theorem summarizes the computational formulas for heuristic values.

Theorem 8. At the node n_i where the k_i columns in S_i have already been selected from X, additional $\overline{k}_i{=}k{-}k_i$ columns still need to be selected. Define: $B_i = Y_i Y_i^T$, where Y_i is defined in (3.5). Let $\lambda_1, \ldots, \lambda_m$ be the eigenvalues of B_i in non-increasing order. Then the values of l_i, u_i defined in (3.5) can be calculated by:

$$\begin{aligned} u_i &= \sum_{j=1}^{m} \lambda_j = \text{Trace}\{B_i\}, \\ l_i &= \sum_{j=\overline{k}_i+1}^{m} \lambda_j = \text{Trace}\{B_i\} - \sum_{j=1}^{\overline{k}_i} \lambda_j. \end{aligned} \tag{3.7}$$

The proof is given in the supplementary material. From the formulas in Theorem 8, it is clear that only the top $\overline{k}_i$ eigenvalues of B_i need to be calculated in addition to its trace. But they have to be calculated for each node, which is impractical. We proceed to show that there is a matrix related to B_i with a special structure that enables efficient computation of these eigenvalues.

Observe that we only need heuristics for the children of the picked node (parent) at Line 8 in the algorithm. At that point the parent node n_i is known. We show how to efficiently calculate the heuristics for the children by first performing an expensive eigendecomposition

of the parent. However the expensive eigendecomposition of the parent can be reduced by computing eigendecomposition in the initial preprocessing step. In summary, we discuss three different types of eigendecompositions. The first one is the one in the preprocessing step, which only needs to be performed once. The second is for each parent, and the third is for the children, where only the eigenvalues are needed.

3.4.1 Initial Eigendecomposition

In the preprocessing step we compute the eigenvalues and the eigenvectors of the matrices $B_y = YY^T$ and $B_x = XX^T$. The eigenvalue decompositions give:

$$B_y = U_y D_y U_y^T, \quad B_x = U_x D_x U_x^T,$$

where U_y, U_x are the eigenvectors and D_y, D_x are diagonal matrices with the eigenvalues as the diagonal elements.

Let r_y be the rank of Y then $r_y \leq \min(m, N)$. Let r_x be the rank of X then $r_x \leq \min(m, n)$. As we show later we can ignore zero eigenvalues and their corresponding eigenvectors, and can replace X, Y with the following information:

$$\begin{aligned} &W_x = U_x^T X, \quad \text{where } W_x \text{ is of size } r_x \times n. \\ &D_y \quad \text{reduced to size } r_y \times r_y. \\ &P = D_y^{\frac{1}{2}} U_y^T U_x, \quad \text{where } P \text{ is of size } r_y \times r_x. \end{aligned} \tag{3.8}$$

With the advancement of randomized algorithms for matrix decompositions, this initial step can be performed efficiently. For example, using the algorithm described in (Halko et al., 2011), the time complexity is $O(mr_yN + mr_xn)$; the memory complexity is $O(mr_y + mr_x + r_xn)$ for the preprocessing step.

3.4.2 Eigendecomposition for Parent Nodes

Instead of working with the matrix B_i as defined in Theorem 8, we use a related matrix H_i which has same eigenvalues as B_i. The special structure of H_i makes the calculations of these eigenvalues more efficient.

Lemma 10. Let Q_i be an orthonormal basis of the current selection S_i (selected from X) of size k_i. Let $\tilde{S}_i$ be the corresponding selection from $W_x = U_x^T X$, and $\tilde{Q}_i$ be the orthonormal basis of $\tilde{S}_i$, then: $Q_i = U_x \tilde{Q}_i$.

Proof: Straightforward.

Lemma 11. Let B_i be the matrix whose eigenvalues are used in (3.7) to calculate the heuristics. Let $\tilde{Q}_i$ be an orthonormal basis of $\tilde{S}_i$ of size k_i. Given D_y and P from (3.8), define the following $r_y \times r_y$ matrix:

$$H_i = D_y - Z_i Z_i^T = D_y - \sum_{j=1}^{k_i} z_j z_j^T, \tag{3.9}$$

where $Z_i = P\tilde{Q}_i$, $z_i = P\tilde{q}_i$, and $\tilde{q}_i$ is the ith column of $\tilde{Q}_i$. Then H_i and B_i have the same eigenvalues (and trace).

Lemma 11 shows that H_i can be computed from k_i times rank-one updates of the diagonal eigenvalue matrix, which enables specialized routines to compute its eigenpairs (e.g., (Bunch et al., 1978)). The proof is given in the supplementary material.

3.4.3 Eigenvalues for Children Nodes

To compute the heuristics at Line 8 in Algorithm 1, eigenvalues for children nodes along with their traces are needed. It is the most expensive part of the algorithm. We show how to compute those values efficiently.

Let n_p be the node picked at Line 2 of the algorithm and let $\tilde{S}_p$ of size k_p be the corresponding selection. Let $\tilde{Q}_p$ be the orthonormal basis of $\tilde{S}_p$. Let H_p be the matrix computed

Table 3.2: Accuracy comparison for the case $N = 1$. The minimum errors are highlighted. No results (-) is shown if the runtime is longer than 30 minutes. The "error" means that there is an error thrown out during the run of the algorithm. Our optimal variant ($f = l$) and Leaps are guaranteed to produce a best selection. Our weighted variant ($f = l + \gamma u$) are more accurate than other non-optimal algorithms.

k	f=l	f=l+u	f=l+5u	f=u	Forward	Backward	Leaps	OMP	FoBa	POSS	ISOLS
eunite2001 X:367×16 Y:367×1											
5	**1.154e6**	**1.154e6**	**1.154e6**	1.169e6	1.169e6	1.161e6	**1.154e6**	1.509e6	1.329e6	1.226e6	1.169e6
10	**0.920e6**	**0.920e6**	0.931e6	1.035e6	1.035e6	**0.920e6**	**0.920e6**	1.043e6	1.043e6	1.013e6	1.035e6
spectf X:267×44 Y:267×1											
5	**38.64**	**38.64**	**38.64**	39.62	39.62	39.20	**38.64**	40.52	40.52	38.87	39.62
7	**37.73**	**37.73**	38.10	38.36	38.36	38.41	**37.73**	38.74	37.83	**37.73**	38.36
libras X:360×90 Y:360×1											
3	**5192.12**	**5192.12**	**5192.12**	**5192.12**	**5192.12**	5333.00	**5192.12**	5334.94	5334.94	5194.66	**5192.12**
5	**4723.07**	**4723.07**	4778.88	4796.08	4796.08	5086.07	**4723.07**	4953.25	4953.25	4777.82	4796.08
duke breast cancer X:$44 \times 7,129$ Y:44×1											
2	**14.67**	**14.67**	14.94	14.94	error	error	error	14.94	14.94	16.07	14.94
5	-	**4.92**	5.14	5.14	error	error	error	5.36	5.36	5.14	5.14

according to (3.9). Suppose a child node n_c is created by adding a column w from W_x to $\tilde{S}_p$. From (3.9) the associated matrix H_c for the child can be computed by:

$$H_c = H_p - z_c z_c^T \tag{3.10}$$

where $z_c = P\tilde{q}_c$, $\tilde{q}_c = \bar{w}_c/\|\bar{w}_c\|$, and $\bar{w}_c = (I - \tilde{Q}_p\tilde{Q}_p^T)w$.

The key observation here is that H_c is a rank-one modification of H_p. In this case the eigenvalues of the updated matrix can be computed efficiently in $O(kr_y)$ from the eigendecomposition of H_p. See, e.g., (Bunch et al., 1978). Since computing z_c takes $O(kr_x)$ this adds up to $O(k(r_x + r_y)n)$ for computing the heuristic values for all children of a parent node.

3.4.4 Complexity

Suppose there are T parent nodes, then the overall time complexity is $O(mNr_y + mnr_x + T \cdot k(r_x + r_y)n)$; the memory complexity is $O(mr_y + mr_x + r_x n + T \cdot n)$. For the greedy variant, since $T = k$ and no fringe/closed lists are needed, the time complexity is $O(mr_y N + mr_x n + k^2(r_x + r_y)n)$; the memory complexity is $O(mr_y + mr_x + r_x n)$. Assuming $r_x \geq r_y$, $n \geq N$ and $n \geq m$, then the time complexity is $O((k^2 + m)r_x n)$; the memory complexity is $O(r_x n)$.

Table 3.3: Accuracy comparison for the $N > 1$ case. Our optimal variant ($f = l$) is guaranteed to produce a best selection. The suboptimal results come with bounds. The bounds are normalized by the solution errors.

k	f=l	f=l+u		f=l+2u		f=l+10u		f=u		SOMP	SSBR	ISOLS
	error	error	bound:b	error	bound:b	error	bound:b	error	bound:b			
libras X: 360×45 Y: 360×46												
3	**6,010**	**6,010**	**0**	**6,010**	0.947	**6,010**	0.949	6,169	0.95	6,047	6,169	6,169
5	**5,587**	**5,587**	**0.941**	5,594	0.987	5,623	0.987	5,686	0.987	5,632	5,686	5,686
spectf X: 267×22 Y: 267×23												
5	**423,909**	428,524	**0.635**	433,697	0.639	433,697	0.639	433,697	0.639	431,650	433,697	433,697
10	**374,453**	377,282	0.842	377,282	0.842	377,282	0.842	377,282	0.842	384,156	377,313	377,282
duke breast cancer X: $44 \times 3,565$ Y: $44 \times 3,565$												
5	-	**62,040**	**0.071**	**62,040**	**0.071**	62,191	0.074	62,191	0.074	62,976	62,191	62,191
25	-	**18,040**	**0.159**	18,700	0.189	18,700	0.189	18,700	0.189	213,17	18,700	18,700
scm20d X: $8,966 \times 61$ Y: $8,966 \times 16$												
5	**5.727e9**	**5.727e9**	**0.727**	5.786e9	0.748	6.007e9	0.758	6.007e9	0.758	6.159e9	6.007e9	6.007e9
7	-	**5.205e9**	**0.882**	5.239e9	0.883	5.367e9	0.886	5.367e9	0.886	5.24e9	5.367e9	5.367e9
mediamill X: $43,907 \times 120$ Y: $43,907 \times 101$												
5	**116,292**	**116,292**	**0**	**116,292**	0.395	116,899	0.398	117,894	0.403	118,744	117,894	117,894
10	-	-	-	-	-	**112,975**	0.579	113,931	0.583	114,403	113,931	113,931
oes97 X: 334×263 Y: 334×16												
5	-	**2.120e9**	**0.597**	2.126e9	0.598	2.126e9	0.598	2.126e9	0.598	2.498e9	2.126e9	2.126e9
7	-	**1.704e9**	**0.756**	1.708e9	0.757	1.708e9	0.757	1.708e9	0.757	2.002e9	1.708e9	1.708e9

3.5 Bound on Sub-optimality

Both the greedy variant and the weighted variant are not guaranteed to produce an optimal solution. We proceed to show how to obtain bounds on how close their solutions are to the optimal. The technique we use was originally proposed by (Hansen and Zhou, 2007).

Consider a run of a non-optimal variant producing the non-optimal selection S^{**}. Then size$(S^{**}) = k$, and from Lemma 5 it follows that $l_{**} = u_{**} = E(S^{**})$. The value of l_{**} is related to the optimal value $l_* = u_* = E(S^*)$ by: $l_* \leq l_{**}$. Let b be a value satisfying: $l_{**} \leq l_* + b$, or, equivalently $b \geq l_{**} - l_*$. We refer to b as a bound, where a smaller b indicates a better bound, and in particular $b = 0$ implies an optimal solution. An important observation is that in heuristic search one can always compute such values. Let F be the fringe list after the algorithm terminates. Going over all the remaining nodes in the fringe list we can compute: $l_{\min} = \min_{n_i \in F} l_i$. From Lemma 6 it follows that $l_* \geq l_{min}$, so that we

can take:

$$b = l_{**} - l_{\min}, \quad \text{where: } l_{\min} = \min_{n_i \in F} l_i. \tag{3.11}$$

The b is a nontrivial bound on $l_{**} - l_*$, and can be calculated. This gives a provable bound on sub-optimality.

3.6 Experimental Results

We describe experiments on various datasets that are publicly available. For the N=1 case we compare the proposed algorithm with the following methods: Leaps (Furnival and Wilson, 1974); Forward (Hastie et al., 2009); Backward (Hastie et al., 2009); OMP (Mallat, 1999); FoBa (Zhang, 2009); POSS (Qian et al., 2015). For the general case ($N \geq 1$) we compare our algorithm with the following algorithms: SOMP (Tropp et al., 2006); SSBR (Belmerhnia et al., 2014); SOLS (Chen and Huo, 2006); CM (Çivril and Magdon-Ismail, 2012); ISOLS (Maung and Schweitzer, 2015). The results for SOLS, CM and ISOLS are same. (They are different in terms of runtime.) The results for ISOLS are shown. The implementations used for Leaps, Forward, and Backward are the functions in the R library "leaps". SOMP and SSBR are implemented in Python. Other implementations are publicly available. Experiments are conducted on iMac with Processor Intel Quad-Core i7 and Memory 32GB.

3.6.1 Comparison

As discussed in Section 3.3.1, with different choices of the heuristic function, our algorithm produces optimal, suboptimal and greedy results. We compare those results with the current state-of-the-art algorithms for the N=1 case and the general case where N>1.

Comparison for the N=1 case.

The results for various datasets are shown in Table 3.2. As expected, our optimal variant (f=l) gives identical results to Leaps. Recall that our optimal variant also works when N>1. The errors for the greedy variant (f=u) are same as Forward and ISOLS. The errors for the weighted variant (f=l+γu) are between the optimal solution and the greedy solution. They are more accurate than other non-optimal algorithms.

Comparison for the general case.

The results for various datasets are shown in Table 3.3. The first three datasets are split evenly and experimented without intercept. The remaining datasets are real multi-target datasets tested with intercept. (The constant vector **1** is added into the dictionary matrix X.)

Our optimal variant (f=l) is guaranteed to produce optimal solutions. The errors for the greedy variant (f=u) are same as the results of ISOLS. The errors for f=l+γu are between the optimal solution and the greedy solution.

Observe that the sub-optimality bounds are meaningful. In fact, in some cases the sub-optimality bound is 0 which implies that the solution is optimal even though the algorithm is not guaranteed to produce optimal solutions.

Comparison of the running time.

To evaluate the running time we use very large datasets. Table 3.4 shows the results. Taking f=l+u with the Day1 dataset as an example, the total cost is 24 minutes: 17 minutes for initial processing plus 7 minutes for searching. Other algorithms are not practical for big datasets.

Table 3.4: Runtime comparison on big datasets with k=20. The "-" indicates that the algorithm did not terminate after 50 minutes.

	SOMP	SSBR	ISOLS	f=l+5u	f=l+u
duke breast cancer: $44 \times 7,129$; $X = Y$					
error / b	58,330	52,629	52,629	52,034 / 0.13	52,629 / 0.14
time (s)	4	30	42	1.9	1.8
Sift: dense $128 \times 1,000,000$; $X = Y$					
error / b	-	-	-	5.15e10 / 0.23	5.18e10 / 0.23
time (min)	-	-	-	17 = 2 + 15	17 = 2 + 15
Day1: sparse $20,000 \times 3,231,957$; $X = Y$; $r_x = r_y = 500$					
error / b	-	-	-	284,222 / 0.097	284,455 / 0.098
time (min)	-	-	-	24 = 17 + 7	24 = 17 + 7

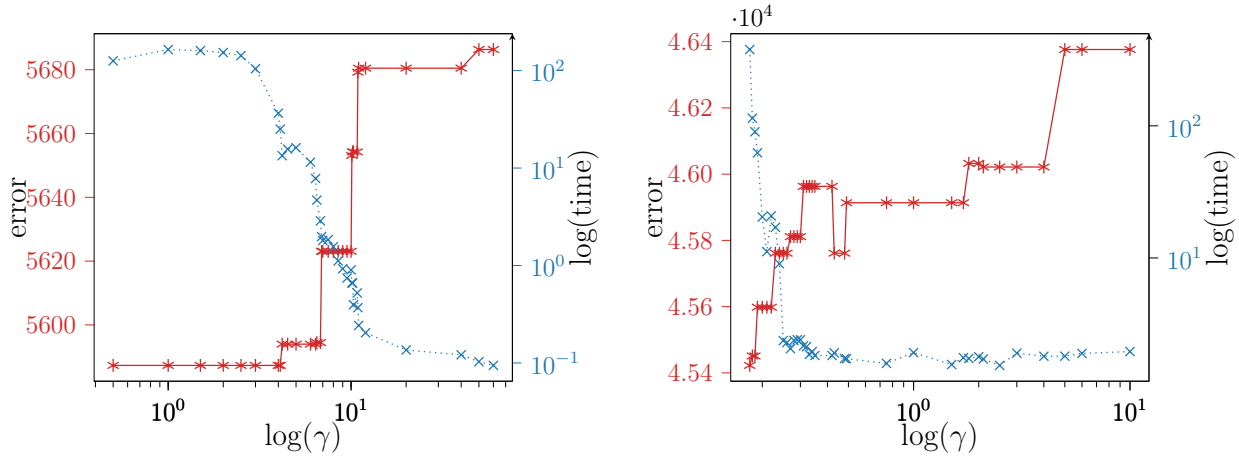

Figure 3.1: The effect of γ on accuracy and runtime for the general case. The left panel: libras dataset N=46, k=5. The right panel: duke breast cancer N=3,565, k=10.

3.6.2 The Effect of the Parameter γ

We investigated the influence of the parameter γ on the accuracy and runtime. The results are shown in Figure 3.1. The red curve corresponds to the change of errors as the increase of γ. The blue curve is for the change of the runtime.Observe that as the increase of γ, the error goes up from an optimal solution to a greedy solution, while the runtime decreases.

3.7 Concluding Remarks

We study a common setting where several columns of one matrix are selected to simultaneously approximate all the columns of another matrix. While many algorithms were developed for the case where the target matrix has a single column, we are only aware of approximate algorithms for the general case. We observe that the general case cannot be reduced to the one column case, as the challenge is to select columns that approximate the entire target matrix.

The algorithms that we develop are similar to the (Weighted) A^* algorithm. We show that with 0 assigned to a weight parameter, the algorithm is guaranteed to be optimal, but its running time may be very slow. Other nonzero choices give practical algorithms that beat the accuracy of the current state of the art algorithms.

In addition to producing a solution, our algorithms calculate bounds on how far the solutions are from the optimum. The quality of these bounds are data dependent. In some cases they provide no useful information, but in other cases it shows that the computed result is very close to the optimal solution. This is the case even for the greedy variant of our algorithm. It produces the same result as other known algorithms but gives the additional information of a bound.

While there is a significant similarity between our algorithm and the classical theory of the (Weighted) A^*, we are not aware of direct applications of A^* to the problem discussed here. In particular, our upper bound does not seem to have a parallel in the general theory.

CHAPTER 4

HEURISTIC SEARCH FOR APPROXIMATING ONE MATRIX IN TERMS OF ANOTHER MATRIX: SUPPLEMENTARY MATERIAL

4.1 Proofs of the Three Variants in Section 3.3.1

In this section we give the technical proofs of lemmas and theorems in Section 3.3.1. The tool we use here is the interlacing property of eigenvalues. We use the following special case:

Theorem 9. Suppose $\tilde{Z} = Z - cc^T$ where $Z \in \mathbb{R}^{d\times d}$ is symmetric, and $c \in \mathbb{R}^d$. Let $\lambda_i(\cdot)$ be the ith eigenvalue of a matrix, and all eigenvalues of a matrix are ordered such that $\lambda_{i+1}(\cdot) \leq \lambda_i(\cdot)$. Then:

$$\lambda_{i+1}(Z) \leq \lambda_i(\tilde{Z}) \leq \lambda_i(Z).$$

This follows from the more general representation as given, for example, in Chapter 8 of (Golub and Van-Loan, 2013) which has the condition:

$$\tilde{Z} = Z + \epsilon c_2 c_2^T, \quad \text{where } \|c_2\| = 1.$$

Observe that the special case is obtained with the choice $\epsilon = -\|c\|^2$, and $c_2 = c/\|c\|$. As an immediate corollary we get:

Corollary 1.

$$\sum_{i=\tau+1}^{d} \lambda_i(Z) \leq \sum_{i=\tau}^{d} \lambda_i(\tilde{Z}),$$

$$\text{where } 1 \leq \tau \leq d.$$

4.1.1 Proofs of the Lemmas

Our proofs have similar structure to the proofs in (He et al., 2019). When both matrices X and Y are same their result is a special case of our result. However, the tools used in (He

et al., 2019) may not be powerful enough to prove our result, which relies on the interlacing property.

Lemma 6: If n_j is a child of n_i then $l_i \leq l_j$.

Proof: Let S_i be the subset associated with node n_i of size k_i, and set $\overline{k}_i = k - k_i$. We need to show that if $S_j \supset S_i$ (so that S_j is a child of S_i) then l_j is not less than l_i. Set $Z = Y_i Y_i^T$, $\tau = \overline{k}_i$. Then the lemma follows from the inequality in Corollary 1. ■

Lemma 7: If n_j is a child of n_i then $u_j \leq u_i$.

Proof: The child node n_j is created by adding a new column into the selection S_i at node n_i. Clearly, the additional column can only reduce the error. ■

Lemma 8: Consider the choice $f_i = u_i$. Let n_i be the node picked at Line 2 of the algorithm. Then the following two properties hold:

a. For each child n_j of n_i the selection size $|S_j|$ is larger than the selection size of all nodes currently in the fringe list.

b. The next node to be picked up will be a child of n_i.

Proof: The proof is by induction. Property **a** follows trivially from Property **b**. To prove Property **b** observe that from Lemma 7, u_i is monotonically decreasing along any path. Therefore, the f values of the children of n_i will be no greater than the f values of all the nodes currently in the fringe list. Since ties are resolved in favor of a child, then the child of n_i with the smallest f value will be selected next. ■

Lemma 9: Suppose Theorem 7 is false. Then for any node n_z on the path from the root to an optimal goal node n_* the following condition holds: $f_z < f_{**}$.

Proof: The falsehood of Theorem 7 can be written as follows: $e^{**} > e^* + \gamma(u_{\max} - l_{**})$. Since both n_* and n_{**} are goal nodes their corresponding subsets S^* and S^{**} are both of size k. Lemma 5 implies: $e^{**} = l_{**} = u_{**}$ and $e^* = l_* = u_*$. Using this and some algebra it can be shown that an equivalent falsehood condition is: $l_{**} > l_* + \frac{\gamma}{1+\gamma}(u_{\max} - l_*)$. The lemma

can now be proved as follows:

$$\begin{aligned} f_{**} &= l_{**} + \gamma u_{**} = (1+\gamma) l_{**} && (c_1) \\ &> (1+\gamma) l_* + \gamma (u_{\max} - l_*) && (c_2) \\ &= l_* + \gamma u_{\max} \geq l_z + \gamma u_{\max} && (c_3) \\ &\geq l_z + \gamma u_z = f_z \end{aligned}$$

c_1 : from the definition of f. c_2 : from the equivalent falsehood assumption. c_3 : from Lemma 6, $l_* \geq l_z$. ■

4.1.2 Proofs of the Theorems

Theorem 4: (the optimal variant) If $f_i = l_i$ then the algorithm is guaranteed to terminate with an optimal solution.

Proof: The proof follows as a corollary of Theorem 6 with $\gamma = 0$. ■

Theorem 5: (the greedy variant) If $f_i = u_i$ then the algorithm is greedy and terminates after expanding k nodes.

Proof: From Lemma 8, when a node n_i is picked at Line 2 of the algorithm, one of its children will be examined next. This shows that the algorithm is greedy, and terminates after examining k nodes. ■

Theorem 7: (the tighter weighted variant) If $f_i = l_i + \gamma u_i$, where $\gamma \geq 0$, and the algorithm terminates at the node n_{**} with the subset S^{**} that has an error of e^{**} then

$$e^{**} \leq e^* + \gamma (u_{\max} - l_{**})$$

where e^* is the smallest possible error, l_{**} is the value of the lower bound for n_{**}, and $u_{\max}$ be the largest value of the upper bound for the nodes remaining in the fringe after the goal node is reached.

Proof: The proof is done by contradiction. If the theorem is false then from Lemma 9 it follows that all nodes on the path from the root to S^* have smaller f values than f_{**}. Since at any given time at least one of them is in the fringe list, they should all be selected before S^{**} is selected. But this means that S^* is selected as the solution and not S^{**}. ■

Theorem 6: (the weighted variant) If $f_i = l_i + \gamma u_i$, where $\gamma \geq 0$, and the algorithm terminates with the subset S^{**} that has an error of e^{**} then

$$e^{**} \leq e^* + \gamma \|Y\|_F^2,$$

where e^* is the smallest possible error.

Proof: The proof follows from Theorem 7 by observing that $u_{\max} - l_{**} \leq u_{\max} \leq u_{\text{root}} = \|Y\|_F^2$.

4.2 Proof of Theorem 8 in Section 3.4

Proof: Recall the relation between the trace and the Frobenius norm. For any matrix M, $||M||_F^2 = \text{trace}\{MM^T\}$. For the upper bound: $u_i = \|Y_i\|_F^2$, so that $u_i = \text{trace}\{Y_i Y_i^T\} = \text{trace}\{B_i\}$.

To prove the formula for the lower bound we observe that the best rank $\overline{k}_i$ approximation to Y_i can be written as: $\overline{Y}_i = UU^T Y_i$. See, e.g., (Golub and Van-Loan, 2013). Here U is formed by the $\overline{k}_i$ eigenvectors of $Y_i Y_i^T$ corresponding to the $\overline{k}_i$ largest eigenvalues. Therefore:

$$
\begin{aligned}
l_i &= ||Y_i - \overline{Y}_i||_F^2 = \|(I - UU^T)Y_i\|_F^2 \\
&= \text{trace}\{(I - UU^T)Y_iY_i^T(I - UU^T)\} \\
&= \text{trace}\{(I - UU^T)B_i(I - UU^T)\} \\
&= \text{trace}\{B_i - UU^TB_i - B_iUU^T + UU^TB_iUU^T\} \\
&= \text{trace}\{B_i\} - \text{trace}\{UU^TB_i\} - \text{trace}\{B_iUU^T\} \\
&\quad + \text{trace}\{UU^TB_iUU^T\} \\
&= \text{trace}\{B_i\} - \text{trace}\{U^TB_iU\} \qquad \text{(note1)} \\
&= \text{trace}\{B_i\} - \sum_{j=1}^{\overline{k}_i} \lambda_j(B_i)
\end{aligned}
$$

The simplification in the line marked with (note1) was obtained using the cyclic invariant property of the trace which implies that:

$$\text{trace}\{UU^TB_i\} = \text{trace}\{B_iUU^T\} = \text{trace}\{UU^TB_iUU^T\} = \text{trace}\{U^TB_iU\}.$$

■

4.3 Proof of Lemma 11

Proof: From (3.5) $A_i^* = \arg\min_{A_i} \|Y - S_iA_i\|_F^2$. Therefore $A_i^* = S^+Y$, where + indicates the pseudo inverse. Plugging into (3.5) gives $Y_i = Y - S_iS_i^+Y$. Since S_i is thin it follows that $S_iS_i^+ = Q_iQ_i^T$ where Q_i can be any orthonormal basis of S_i. Then $Y_i = Y - Q_iQ_i^TY$. Then B_i can be written as follows:

$$
\begin{aligned}
B_i &= Y_iY_i^T \\
&= (I - Q_iQ_i^T)YY^T(I - Q_iQ_i^T)^T \\
&= (I - Q_iQ_i^T)U_yD_yU_y^T(I - Q_iQ_i^T)^T \\
&= G_iD_y^{\frac{1}{2}}(G_iD_y^{\frac{1}{2}})^T
\end{aligned}
$$

where $G_i = (I - Q_i Q_i^T)U_y$. Since the order of matrix product between $G_i D_y^{\frac{1}{2}}$ and $(G_i D_y^{\frac{1}{2}})^T$ does not affect the eigenvalues, the following matrix that we call H_i has the same eigenvalues as B_i:

$$\begin{aligned}
H_i &= (G_i D_y^{\frac{1}{2}})^T G_i D_y^{\frac{1}{2}} \\
&= D_y^{\frac{1}{2}} G_i^T G_i D_y^{\frac{1}{2}} \\
&= D_y^{\frac{1}{2}} U_y^T (I - Q_i Q_i^T)^T (I - Q_i Q_i^T) U_y D_y^{\frac{1}{2}} \\
&= D_y^{\frac{1}{2}} U_y^T (I - Q_i Q_i^T) U_y D_y^{\frac{1}{2}} \qquad \text{(note1)} \\
&= D_y - D_y^{\frac{1}{2}} U_y^T Q_i Q_i^T U_y D_y^{\frac{1}{2}} \\
&= D_y - D_y^{\frac{1}{2}} U_y^T U_x \tilde{Q}_i (D_y^{\frac{1}{2}} U_y^T U_x \tilde{Q}_i)^T \\
&= D_y - P\tilde{Q}_i (P\tilde{Q}_i)^T = D_y - Z_i Z_i^T
\end{aligned}$$

where $P = D_y^{\frac{1}{2}} U_y^T U_x$. The simplification in the line marked with (note1) was obtained as $(I - QQ^T)^T(I - QQ^T) = I - QQ^T - QQ^T + QQ^T QQ^T = I - QQ^T$. This completes the proof. ■

CHAPTER 5

A FAST ALGORITHM FOR SIMULTANEOUS SPARSE APPROXIMATION

Authors – Guihong Wan, Haim Schweitzer

Department of Computer Science, EC 31

The University of Texas at Dallas

800 West Campbell Road

Richardson, Texas 75080-3021

Published by Advances in Knowledge Discovery and Data Mining, PAKDD 2021.

Abstract

Simultaneous sparse approximation problems arise in several domains, such as signal processing and machine learning. Given a dictionary matrix X of size $m \times n$ and a target matrix Y of size $m \times N$, we consider the classical problem of selecting k columns from X that can be used to linearly approximate the entire matrix Y. The previous fastest nontrivial algorithms for this problem have a running time of $O(mnN)$. We describe a significantly faster algorithm with a running time of $O(km(n+N))$ with accuracy that compares favorably with the slower algorithms. We also derive bounds on the accuracy of the selections computed by our algorithm. These bounds show that our results are typically within a few percentage points of the optimal solution.

5.1 Introduction

Sparse approximation, or alternatively, sparse representation, has attracted significant attention in fields of signal processing, image processing, and machine learning (e.g., (Mairal et al., 2014; Wright et al., 2010; Zhang et al., 2015)). It originally arose in the study of linear regression in which a target vector is approximated by a linear combination of several selected features to foster interpretability and avoid overfitting.

The problem that we discuss in this paper is an extension of the classical sparse approximation problem to multiple targets. The goal is to identify a small number of "atoms" (columns of the dictionary matrix) that can be used to linearly approximate all the columns of another "target" matrix. This formulation is referred to as simultaneous sparse approximation (e.g., (Malioutov et al., 2005; Tropp et al., 2006)). It has various applications, such as supervised feature selection for multi-target regression (Zhu et al., 2019), human action recognition in videos (Guha and Ward, 2011), multi-sensor image fusion (Zhang et al., 2018), and hyperspectral image unmixing (Xu and Shi, 2017).

We present an efficient selection algorithm with a running time that depends linearly on the number of columns of the relevant matrices. This significantly improves on previous algorithms which have a running time depending on the product of these parameters.

The main idea is to select columns from the dictionary matrix whose span is close to the dominant spectral components of the target matrix. The following two-stage procedure is proposed. In the first stage, a greedy algorithm is used to select k columns from the dictionary matrix. In each iteration, a column that is most related to the first left eigenvector of the residual target matrix is selected. In the second stage, an iterative "bidirectional selection" algorithm is used to improve the results produced in the first stage.

5.1.1 Problem Formulation

The problem of simultaneous sparse approximation can be stated as follows. Let $Y = (y_1 \dots y_N)$ be a "target" matrix of m rows and N columns. Let $X = (x_1 \dots x_n)$ be a "dictionary" matrix of m rows and n columns (n is typically very large). Consider an approximation of Y in terms of X:

$$Y \approx X\tilde{A}, \tag{5.1}$$

where $\tilde{A}$ is the coefficient matrix of size $n \times N$. The approximation in (5.1) is considered a sparse approximation if only a small number of rows of $\tilde{A}$ are nonzero. See, e.g., (Çivril and Magdon-Ismail, 2012; Maung and Schweitzer, 2015; Tropp, 2004; Tropp et al., 2006). Let $k \leq n$ be the number of nonzero rows in $\tilde{A}$, then the approximation in (5.1) can be written as:

$$Y \approx SA, \tag{5.2}$$

where $S = (x_{s_1} \dots x_{s_k})$ is the $m \times k$ selection matrix consisting of the k columns of X corresponding to the k nonzero rows of $\tilde{A}$, and A is the coefficient matrix of size $k \times N$, created from the k nonzero rows of $\tilde{A}$. Thus, the goal of the problem described in (5.2) is to select

k columns from X such that all the columns of Y can be simultaneously approximated by a linear combination of the selected columns in S. The quality of the selected columns in S is measured by the following (Frobenius norm) error:

$$E(S) = \min_{A} \|Y - SA\|_F^2, \quad \text{where } S \text{ is a subset of } X \text{ columns.} \tag{5.3}$$

The following cases are special sub-problems:

- When N=1, the problem is called sparse approximation, where Y is a single vector and X is a dictionary. It is also called supervised subset selection when Y corresponds to labels for the data matrix X. See, e.g., (Furnival and Wilson, 1974; Natarajan, 1995).
- When X=Y, the problem is known as "unsupervised feature selection" if each column corresponds to a data feature. It is known as "representative selection" if each column corresponds to a data point. See, e.g., (Arai et al., 2015; He et al., 2019; Zaeemzadeh et al., 2019).

Optimal solutions for the optimization in (5.3), as well as approximate solutions within a constant are known to be NP-hard even when N=1 (Davis et al., 1997; Natarajan, 1995).

5.1.2 Related Work

Extensive studies were directed at the N=1 case (e.g., (Tropp, 2004; Mallat, 1999; Furnival and Wilson, 1974; Zhang, 2009; Qian et al., 2015)). Applications include signal processing (e.g., (Tropp, 2004; Mallat, 1999)) and supervised feature selection in linear regression (e.g., (Furnival and Wilson, 1974; Zhang, 2009; Qian et al., 2015)). An optimal algorithm proposed in (Furnival and Wilson, 1974) finds the best solution, but it is not feasible for large k and large datasets. Previously proposed approximate solutions can be roughly divided into three groups: forward selection, backward elimination, and convex relaxation. Forward selection sequentially adds columns that improve the quality the most. Backward elimination starts

with the full selection and sequentially removes the columns that affect the quality the least. As shown in (Zhang, 2009), these two techniques have limitations due to their greedy behavior. The author of (Zhang, 2009) introduced an adaptive algorithm that combines these two ideas to alleviate the flaws. The convex relaxation approaches replace some natural constraints (sometimes defined in terms of the l_0 norm) with convex constraints. An example is the l_1 norm used in the Lasso technique (Tibshirani, 1996). In (Qian et al., 2015), the authors treated the subset selection as a bi-objective optimization problem. Their algorithm is optimal for data drawn from Exponential Decay distribution but not for the general case.

The case where $X{=}Y$ is known as the unsupervised feature selection or unsupervised column subset selection. See, e.g., (Golub and Van-Loan, 2013; Çivril and Magdon-Ismail, 2012; Arai et al., 2015, 2016; Zaeemzadeh et al., 2019; Joneidi et al., 2020). Recently, a fast algorithm (Zaeemzadeh et al., 2019) was proposed, which greedily selects columns that are closest to the first left eigenvector of a residual matrix. Another recent study in (Joneidi et al., 2020) proposes to iteratively improve a selection using a bi-directional stepwise refinement. Both these studies address the $X{=}Y$ case. Our method is motivated by these two recent studies, generalizing them to the supervised multiple-target case. We show that the speed advantage of this approach over other approaches becomes bigger for large N.

In the multi-target case, the target matrix Y has $N > 1$ columns. We observe that one cannot simply apply an $N{=}1$ algorithm separately to each column of Y. The challenge is to find columns in X that can simultaneously approximate all the columns of Y. Previously proposed algorithms for this general case are typically greedy. See, e.g., (Tropp et al., 2006; Maung and Schweitzer, 2015; Çivril and Magdon-Ismail, 2012; Chen and Huo, 2006; Belmerhnia et al., 2014). Some of these algorithms are generated from the $N{=}1$ case. For example, the Simultaneous Orthogonal Matching Pursuit (SOMP) (Tropp et al., 2006) is generated from the Orthogonal Matching Pursuit (OMP) (Tropp, 2004; Soussen et al., 2013) to handle a target matrix of N columns. Similarly, the Simultaneous Orthogonal Least

Table 5.1: Complexity of various algorithms. T is the number of iterations.

Algorithms	**time complexity**	**memory complexity**
SOMP (Tropp et al., 2006)	$O(kmnN)$	$O(m(N+k))$
S-SBR (Belmerhnia et al., 2014)	$O(Tkmn N)$	$O(m(N+k))$
SOLS (Cotter et al., 2005)	$O(kmnN)$	$O(m(n+N))$
CM (Çivril and Magdon-Ismail, 2012)	$O(nN(m+k))$	$O(m(n+N))+nN$
ISOLS (Maung and Schweitzer, 2015)	$O(mnN)$	$O(km)+2n$
SPXY (this work)	$O(km(n+N))$	$O(m(n+N))$

Squares (SOLS) is a direct extension of the Orthogonal Least Squares (OLS) (Soussen et al., 2013). See (Chen and Huo, 2006; Çivril and Magdon-Ismail, 2012) for the analysis of the SOLS algorithm. An algorithm established in (Çivril and Magdon-Ismail, 2012) improves the SOLS algorithm in terms of speed at the cost of increased memory. In (Maung and Schweitzer, 2015), the authors improved the speed of the SOLS algorithm by a recursive formulation. The running time and the memory requirements of some of these algorithms are summarized in Table 5.1.

5.1.3 Our Approach

Suppose the matrix S in (5.3) is not constrained to be a submatrix of X. Then it is known that the k columns of the optimal solution matrix S are the left eigenvectors of Y corresponding to the k largest singular values (see e.g., (Golub and Van-Loan, 2013)). Our algorithmic approach is based on this result.

Motivated by (Zaeemzadeh et al., 2019) and (Joneidi et al., 2020), we propose a two-stage algorithm that we call the Spectral Pursuit for the matrices X and Y (SPXY). It runs significantly faster than previously proposed algorithms, and its accuracy compares very favorably with the current state of the art. We also show how to derive a bound on how far the selection computed by SPXY is from the optimal solution. (Recall that the computation of the optimal solution is NP-hard.)

In the first stage we use a greedy technique to select k columns from the dictionary matrix. The algorithm runs k iterations. In each iteration the column selected is the one most similar to the left eigenvector corresponding to the largest singular value. The two matrices are then projected on the null space of the columns that were already selected. The null space of the selected columns indicates a subspace that the selected columns cannot span. This first stage algorithm is greedy, and gets "stuck" in a local minimum. In the second stage we use a bidirectional stepwise technique to further improve the results.

Similar to many other data analysis techniques, the performance of our SPXY algorithm is data-dependent. Since the general problem is NP-hard, there can be situations where one may be tempted to use other more accurate algorithms, such as exhaustive search, in the hope of significantly improving the accuracy of the result. To this end, we derive bounds on how far the results given by our algorithm are from the optima. As we show, in many practical problems our results are provably within a small percentage of the optima. Thus, in such cases, even the exhaustive search can provide almost no improvement.

Main Contributions

In summary our main contributions are as follows:

- We introduce the SPXY algorithm that has a linear running time in terms of the numbers of columns of the two matrices X and Y. This running time is significantly faster than the current state of the art that requires running time proportional to the product of the numbers of columns of X and Y.
- The accuracy of the SPXY algorithm compares favorably with the current state of the art.
- We derive bounds on how far the solutions produced by SPXY are from the optimal solutions.

Algorithm 2: Spectral Pursuit XY algorithm

procedure SPXY(X, Y, k)
  $S \leftarrow$ SELECT(X, Y, k). {select k columns from X to approximate Y.}
  $S \leftarrow$ IMPROVE(X, Y, k, S). {improve the selection of previous stage.}
  return S
end procedure

Algorithm 3: The selection algorithm

procedure SELECT(X, Y, k)
  $S = \{\}$.
  while $|S| < k$ **do**
    $u =$ the first left eigenvector of Y corresponding to its largest singular value.
    $i =$ index of the column from X, which most correlates with u.
    $S = S \cup \{i\}; \quad q_i = x_i / \|x_i\|$.
    $Y = Y - q_i q_i^T Y; \quad X = X - q_i q_i^T X$.
  return S
end procedure

5.2 The Proposed Algorithm

In this section, a computationally efficient algorithm is proposed for approximating the solution to the NP-hard selection problem of minimizing (5.3).

The top view of the proposed algorithm is shown as Algorithm 2, with two stages involved. In the first stage, a greedy algorithm is introduced to select k columns from the dictionary matrix. In the next stage, an efficient non-greedy algorithm is used to improve the results.

5.2.1 The Selection Algorithm

We first observe that the vectors: $q_1 \dots q_k$, generated by Algorithm 3, are mutually orthogonal. Define $Q = (q_1 \dots q_k)$, where $Q \in \mathbb{R}^{m \times k}$. The projection of Y on Q can be written as $Q^T Y$. The reconstruction of Y from this projection can be written as: $\hat{Y} = QQ^T Y$. Therefore, the optimization problem (5.3) can be restated as:

$$Q = \text{argmin}_Q \|Y - QQ^T Y\|_F^2, \tag{5.4}$$

where Q is restricted to be the basis of k columns from X. As mentioned before, this is an NP-hard problem. If the constraint on Q is relaxed (it can be any matrix with size $m \times k$ with orthonormalized columns), then it is known that the minimizer of problem (5.4) is the k left eigenvectors of Y corresponding to the k largest singular values (e.g., (Golub and Van-Loan, 2013)). We call the k eigenvectors corresponding to the k largest singular values as the first k eigenvectors.

In the selection algorithm, we modify (5.4) into two sub-problems. The first one relaxes the constraint that Q must be the basis of k columns from X. This relaxation makes finding the solution tractable at the expense of resulting in a solution that may not correspond to columns in X. To fix this problem, we introduce the second sub-problem that reimposes the underlying constraint that selects the column that has highest correlation with the vector computed in the first sub-problem. These two sub-problems are formulated as follows:

$$\begin{aligned} u &= \text{argmin}_u \|Y - uu^T Y\|_F^2, \ s.t. \ \|u\| = 1, \\ i &= \text{argmin}_i |u^T q_i|, \end{aligned} \tag{5.5}$$

where $q_i = x_i / \|x_i\|$. The resulting i is the index of the selected column. The first sub-problem is equivalent to computing the first left eigenvector of Y (e.g., (Golub and Van-Loan, 2013)). After solving for u (which is not necessarily one of the columns in X), we find the column that matches u the most (has the least angle with u).

After selecting the first column x_i, we project all columns onto the orthogonal space of this selected column. This forms the residual matrices: $Y = Y - q_i q_i^T Y$, $X = X - q_i q_i^T X$. In the future iterations, we solve problem (5.5) on the residual matrices. This process continues until k columns are selected.

Algorithm 3 shows the selection algorithm. To simplify notation, we do not distinguish between the selection matrix and the selection indexes. To evaluate the algorithm complexity we observe that there are several recently proposed efficient algorithms for computing eigenvectors. In particular, the randomized eigendecomposition algorithm (Halko et al., 2011)

Algorithm 4: The improvement algorithm

procedure IMPROVE(X, Y, k, S)
  $S_{\text{opt}} \leftarrow S$; $\text{error}_{\text{min}} \leftarrow$ the current error value.
  iter $= 0$.
  while a stopping criterion is not met **do**
    $i = \text{mod}(\text{iter}, k)$.
    $Q_{\bar{i}} =$ basis of columns in S except the ith column of S.
    $R_Y = Y - Q_{\bar{i}}Q_{\bar{i}}^T Y, \quad R_X = X - Q_{\bar{i}}Q_{\bar{i}}^T X$.
    $u =$ the first left eigenvector of R_Y.
    $j =$ index of the most correlated column in R_X with u.
    $q_j =$ the normalized jth column of R_X; error $= \|R_Y\|_F^2 - \|q_j^T R_Y\|_F^2$.
    if error $<$ $\text{error}_{\text{min}}$ **then:**
      $S[i] = j$. {Improvement}
      $S_{\text{opt}} \leftarrow S$; $\text{error}_{\text{min}} =$ error.
    iter += 1.
  end while
  return S_{opt}
end procedure

can be used to compute u in $O(mN)$ time. With this we conclude that the time complexity of Algorithm 3 is $O(km(n+N))$, and its memory complexity is $O(m(n+N))$.

5.2.2 The Improvement Algorithm

Since Algorithm 3 is greedy we propose to improve its result by iteratively revising the selection as long as it can be locally improved. Similar to the selection algorithm in Section 5.2.1, we modify the problem into two sub-problems. The first one is built upon the assumption that $k-1$ columns from X are already selected and the objective is to select next best column. The sub-problems are formulated as:

$$\begin{aligned} u_i &= \text{argmin}_u \|Y - Q_{\bar{i}}Q_{\bar{i}}^T Y - uu^T Y\|_F^2, \ s.t. \ \|u\| = 1, \\ j &= \text{argmin}_j |u_i^T q_j|. \end{aligned} \tag{5.6}$$

Here $Q_{\bar{i}}$ is the basis of $S_{\bar{i}}$ obtained by removing the ith selection in S. The first sub-problem is equivalent to finding the first left eigenvector of the residual matrix of Y: $R_Y = Y - Q_{\bar{i}}Q_{\bar{i}}^T Y$.

The residual matrix of X is given by: $R_X{=}X{-}Q_{\bar{i}}Q_{\bar{i}}^T X$, and $q_j{=}r_X^j/\|r_X^j\|$ is the normalized jth column of R_X. The index j found in this way is the new selection which will replace the ith selection in S.

The improvement algorithm is shown as Algorithm 4. The stopping criterion that we use is a pre-defined maximum number of iterations. The algorithm can be efficiently implemented by using the rank-one update for QR factorization (Daniel et al., 1976) with complexity per iteration of $O(km(n+N))$, and memory complexity of $O(m(n+N))$. In our implementation we limit the algorithm to run no more than 30 iterations. The algorithm terminates if there is no improvement in 5 iterations. The convergence behavior is studied in Section 5.5.

5.3 Fractional Bound

We proceed to show how to obtain nontrivial bound on how close the computed result is to the optimal solution. As described in (5.4) the algorithms minimize the following error:

$$E(S) = E(Q) = \|Y - QQ^T Y\|_F^2 = \|Y\|_F^2 - \|Q^T Y\|_F^2 = \|Y\|_F^2 - G(S),$$

where Q is an orthogonal basis of S, and $G(S) = \|Q^T Y\|_F^2$.

Since $\|Y\|_F^2$ is independent of S, the minimization of $E(S)$ is equivalent to the maximization of $G(S)$. Let E_{opt} be the smallest possible error, and let G_{opt} be the largest possible value of $G(S)$. They are related by: $E_{\text{opt}} = \|Y\|_F^2 - G_{\text{opt}}$. We define the fractional bound in terms of $G(S)$ as follows:

$$\tilde{b}_f(S) = (G_{\text{opt}} - G(S))/G_{\text{opt}}.$$

A smaller $\tilde{b}_f$ value indicates a better result, and in particular, $\tilde{b}_f = 0$ implies an optimal solution. Unfortunately both G_{opt} and E_{opt} are unknown. However, we observe that $\tilde{b}_f(S)$ is monotonically increasing as a function of G_{opt}. This implies that any nontrivial upper bound of G_{opt} can be used to obtain a nontrivial estimate of $\tilde{b}_f(S)$ as follows: If $G_{\text{upper}} \geq G_{\text{opt}}$ then:

$$\tilde{b}_f(S) = \frac{G_{\text{opt}} - G(S)}{G_{\text{opt}}} = 1 - \frac{G(S)}{G_{\text{opt}}} \leq 1 - \frac{G(S)}{G_{\text{upper}}}.$$

Let U_k be the matrix computed from the first k left eigenvectors of Y. Then for any S we have: $E(U_k) \leq E(S)$, so that for any S, $G(U_k) = \|Y\|_F^2 - E(U_k) \geq G(S)$. This shows that $G(U_k) \geq G_{\text{opt}}$, and gives the following formula for provable fractional bound:

$$b_f = 1 - G(S)/G(U_k). \tag{5.7}$$

5.4 Robustness

The sparse approximation algorithms are vulnerable to outliers. The span of outliers usually covers a bigger subspace, which may not be desirable to be represented. In our algorithm, at each iteration, we compute the first left eigenvector of the residual target matrix to guide the selection. If this eigenvector is robust, outliers in the target matrix X will be automatically rejected. We show that this eigenvector is the most robust spectral component against perturbation of matrix the Y. Additionally, we can use robust principal component analysis algorithms (e.g., (Wan and Schweitzer, 2021d; Lerman and Maunu, 2018a)) to compute this eigenvector.

Consider the second moment matrix of Y: $B = YY^T = \sum_{i=1}^{N} y_i y_i^T$. The first eigenvector of this matrix is same as the first left eigenvector of Y. If an outlier is added, it will perturb this matrix. The following lemma proven in (Zaeemzadeh et al., 2019) shows the robustness of ith eigenvector of B against the perturbation.

Lemma 12. Let B be a symmetric matrix with rank r, $(\lambda_1, \ldots, \lambda_r)$ be its eigenvalues (in decreasing order) and $(u_1, \ldots, u_r)$ be the corresponding eigenvectors. Then: $\frac{\|\partial u_i\|_2}{\|\partial B\|_F} \leq \sqrt{\sum_{j \neq i} \frac{1}{(\lambda_i - \lambda_j)^2}}$.

Define the sensitivity of the ith eigenvector as: $s_i = \sqrt{\sum_{j \neq i} \frac{1}{(\lambda_i - \lambda_j)^2}}$. It is known that s_1 is smaller than other $s_i, \forall i > 1$ if the gap between consecutive eigenvalues is monotonously decreasing (Zaeemzadeh et al., 2019).

Table 5.2: Comparison when data is split with proportion to 1:1. "-" indicates that the algorithm runs more than 30 minutes without results.

k	Random error	SOMP error / time (s)	S-SBR error / time (s)	ISOLS error / time (s)	SPXY (this work) error	bound	time (s)
Duke breast cancer X:$44 \times 3,565$ Y:$44 \times 3,565$							
10	40.3	30.7 / 0.73	29.6 / 3.83	29.6 / 9.13	**29.1**	**4.2%**	**0.13**
20	22.4	18.5 / 1.42	16.7 / 9.97	16.7 / 9.37	**16.3**	**3.1%**	**0.16**
30	11.3	9.4 / 2.15	**7.9** / 18.04	**7.9** / 9.41	8.0	**1.8%**	**0.24**
YearPredictionMSD X:$91 \times 257,672$ Y:$91 \times 257,673$							
10	9.2	-	-	-	**5.6**	**0.9%**	**28**
20	5.3	-	-	-	**2.7**	**0.6%**	**37**
30	2.8	-	-	-	**1.5**	**0.5%**	**45**
Sift X:$128 \times 500,000$ Y:$128 \times 500,000$							
10	36.3	-	-	-	**27.2**	**6.1%**	**76**
20	24.7	-	-	-	**19.8**	**5.5%**	**97**
30	21.7	-	-	-	**15.4**	**5.2%**	**117**
Sift:transpose X:$500,000 \times 128$ Y:$500,000 \times 128$							
10	**60.3**	-	-	-	60.5	**49%**	**61**
20	60.4	-	-	-	**59.7**	**53%**	**159**
30	59.4	-	-	-	**59.3**	**54%**	**150**

5.5 Experimental Results

We describe experiments on various datasets that are publicly available, and compare the proposed algorithm with the following algorithms: SOMP (Tropp et al., 2006); SOLS (Cotter et al., 2005); S-SBR (Belmerhnia et al., 2014); CM (Çivril and Magdon-Ismail, 2012); ISOLS (the exact version is used) (Maung and Schweitzer, 2015). Besides, the results of the random selection are shown. The error and bound are defined in (5.3) and (5.7), respectively. In experimental results, they are shown in percentage: error $= \frac{E(S)}{\|Y\|_F^2} * 100$; bound $= b_f * 100$. The computational efficiency and the selection accuracy of our algorithm are demonstrated.

5.5.1 Quantitative Comparison

In the first experiment, we evenly split the datasets into two matrices, serving as X and Y. The results are shown in Table 5.2. In the second experiment, we randomly split the datasets with the proportion of X:Y=3:1. The results are shown in Table 5.3. We choose

Table 5.3: Comparison when data is split with proportion to 3:1.

k	Random error	SOMP error / time(s)	S-SBR error / time(s)	ISOLS error / time(s)	SPXY (this work) error	bound	time (s)
Duke breast cancer X:$44 \times 5,347$ Y:$44 \times 1,783$							
10	44.9	31.4 / 0.84	30.3 / 4.85	30.3 / 6.89	**29.8**	**4.5%**	**0.15**
20	24.3	18.5 / 1.65	17.0 / 11.56	17.0 / 7.15	**16.9**	**3.4%**	**0.22**
30	12.4	9.5 / 2.46	8.0 / 20.01	8.0 / 7.29	**7.9**	**1.6%**	**0.26**
YearPredictionMSD X:$91 \times 386,508$ Y:$91 \times 128,873$							
10	11.0	-	-	-	**5.4**	**0.7%**	**23**
20	5.3	-	-	-	**2.7**	**0.6%**	**31**
30	2.9	-	-	-	**1.4**	**0.4%**	**37**
Sift X:$128 \times 750,000$ Y:$128 \times 250,000$							
10	38.9	-	-	-	**26.8**	**5.7%**	**63**
20	27.3	-	-	-	**19.7**	**5.5%**	**110**
30	19.9	-	-	-	**15.1**	**4.9%**	**134**
Extended Yale Face B X:$32,256 \times 1,488$ Y:$32,256 \times 496$							
10	23.6	18.6 / 214	18.0 / 176	18.0 / 433	**17.6**	**7.4%**	**18**
20	18.2	15.1 / 432	**13.9** / 552	**13.9** / 362	14.1	**8.2%**	**39**
30	15.5	12.8 / 649	**11.7** / 1105	**11.7** / 372	12.1	**8.1%**	**53**

to split the datasets, since the learning of the dictionary matrix is another big topic and task-dependent (e.g., (Xu et al., 2017)). As mentioned in (Maung and Schweitzer, 2015), the results of SOLS, CM, and ISOLS are exactly same (different in terms of speed). We show the results of ISOLS, as it is the fastest among these three algorithms. The parameter γ for S-SBR is 0, since k is known in our case.

Observe that our algorithm is much faster than other algorithms. The running time of ISOLS almost does not change as the increase of k. However, its initial step can be very expensive for big dense datasets. Taking the Duke breast cancer dataset in Table 5.2 as an example, its initial step takes 8.94 seconds. The overall running time for our algorithm is less than 1 second.

The errors given by our algorithm are typically smaller or similar to the errors of ISOLS. Additionally, our errors come with bounds indicating how close the solutions are to the optima. The bounds are usually within 10%. The reason for high bound values on Sift:transpose dataset is that X cannot explain Y well, since the errors are very big and do not change

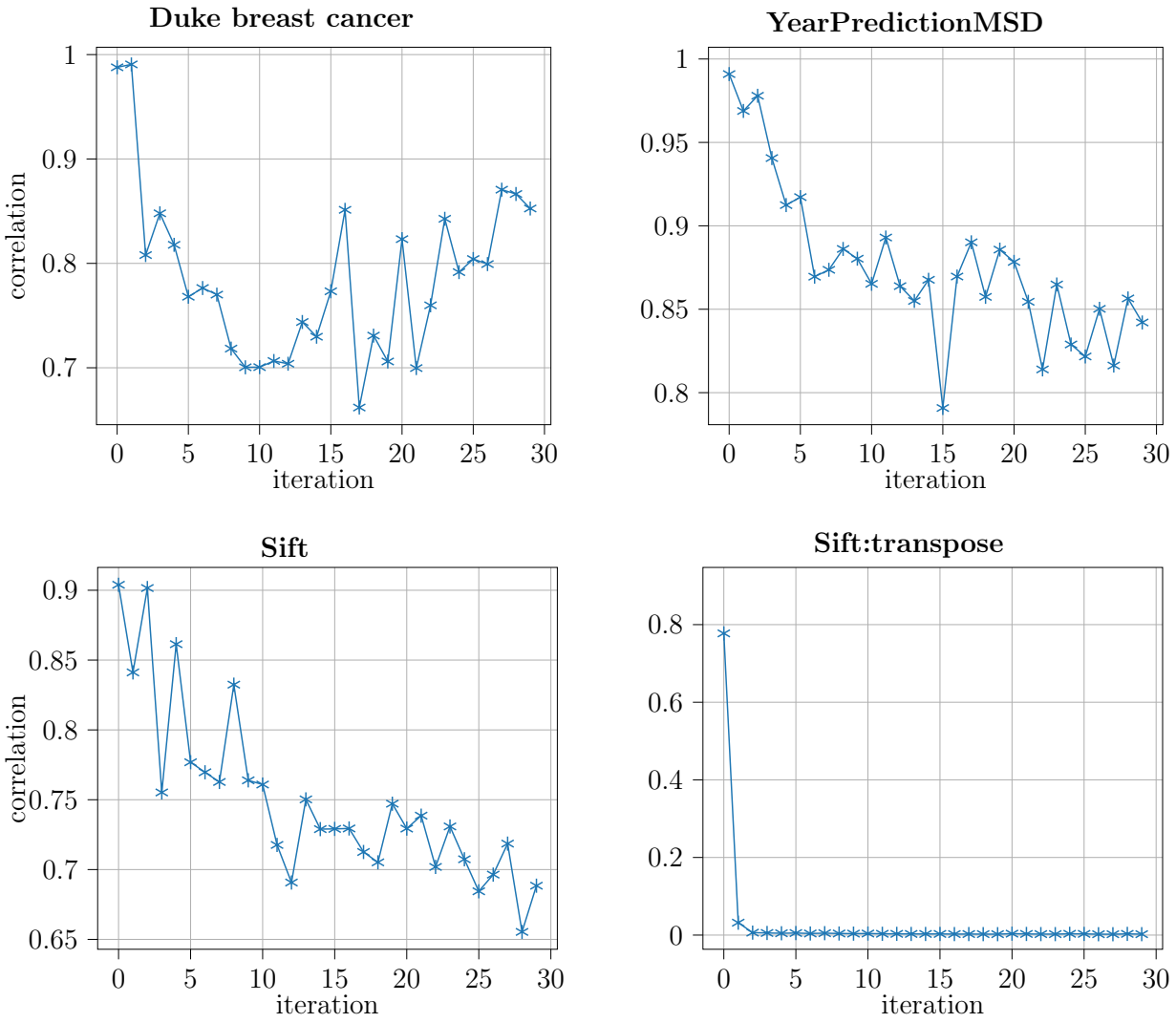

Figure 5.1: Correlation values of selected columns for various datasets.

as the increase of k. It is further discussed in Section 5.5.2. The errors given by SOMP are much bigger. One of the reasons is that SOMP does not update the matrix X on the null-space in each iteration. Clearly, other algorithms are not practical for big datasets.

5.5.2 Correlation Values of Algorithm 3

In the first stage, Algorithm 3 greedily selects the column with highest correlation value in each iteration. Fig. 5.1 shows the correlation values of the selected columns as the run of

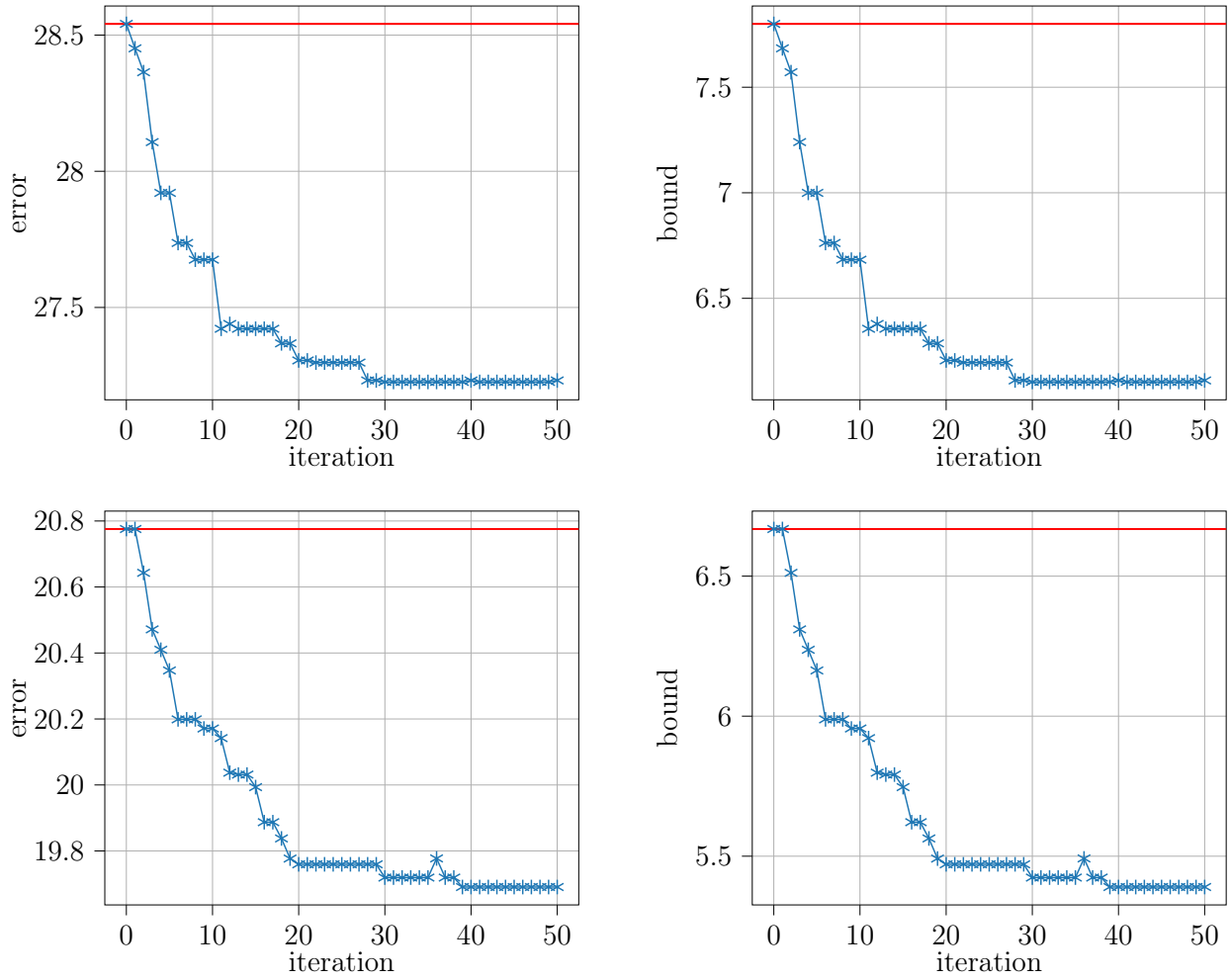

Figure 5.2: Convergence of Algorithm 4 on Sift dataset (1:1). First row: $k = 10$; second row: $k = 20$.

Algorithm 3 for various datasets (split with proportion to 1:1) with $k = 30$. Generally, as the increase of the iteration, the maximum correlation value between the first left eigenvector of the residual matrix of Y and the normalized residual columns of X decreases. For the transpose of Sift dataset, the correlation value for the first iteration is 0.78, but for later iterations, they are very small (almost 0). It shows that X cannot explain Y well.

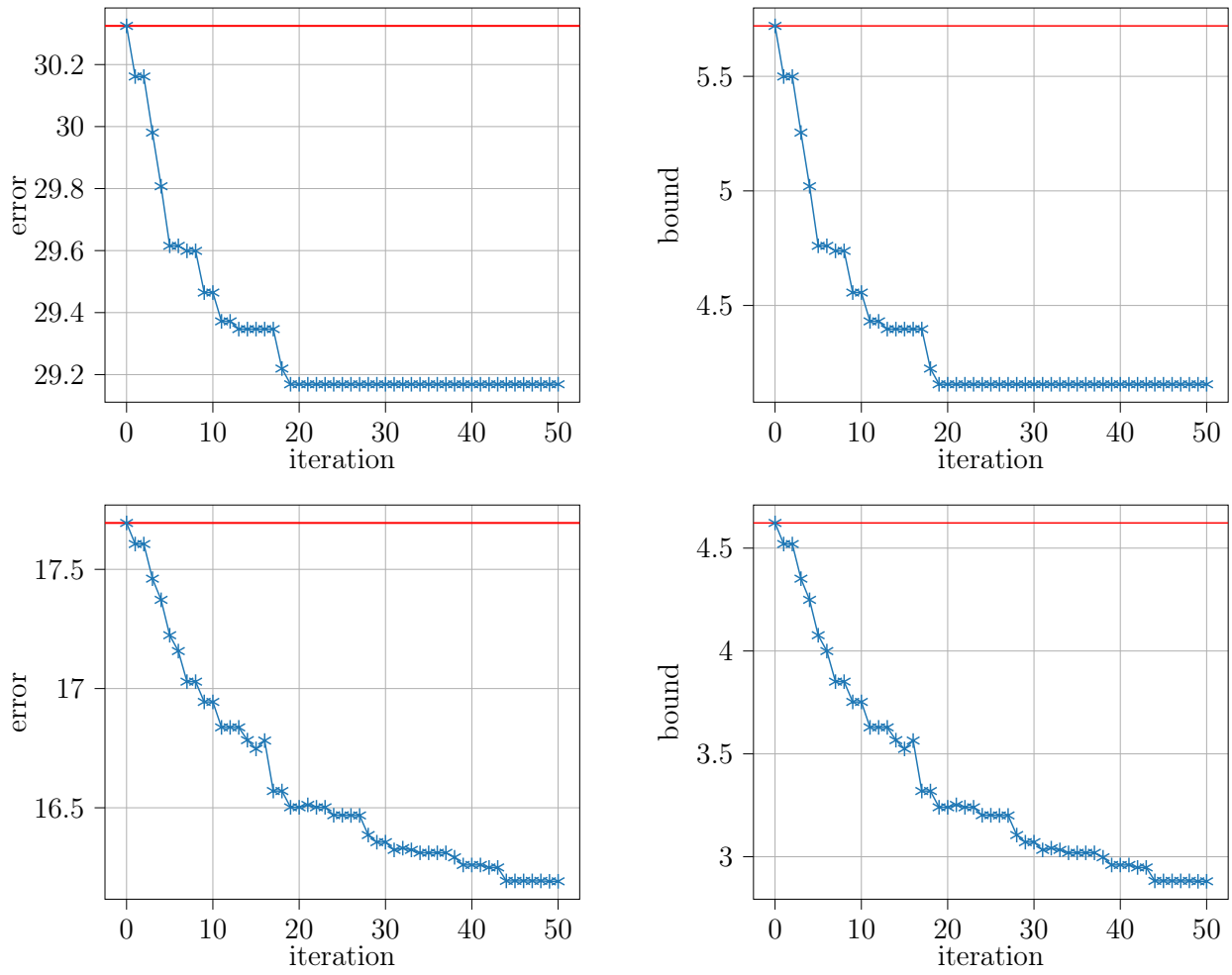

Figure 5.3: Convergence of Algorithm 4 on Duke breast cancer dataset (1:1). First row: $k = 10$; second row: $k = 20$.

5.5.3 Convergence of Algorithm 4

In this experiment, we investigate the convergence of Algorithm 4 in the second stage. Here we set the maximum number of iterations to be 50 without early termination. The results are shown in Fig. 5.2 and 5.3. The red lines correspond to errors or bounds given in the first stage. As expected, the algorithm improves errors and bounds sharply at the beginning.

5.6 Concluding Remarks

The problem discussed in this paper, simultaneously approximating one entire matrix in terms of a small number of selected columns from another matrix, is well-known, and appears to have many practical applications.

A novel two-stage selection algorithm, referred to as Spectral Pursuit for X and Y (SPXY), is presented, which selects columns capturing the spectral characteristics of the target matrix. What we found surprising is that it is possible to efficiently implement the algorithms with linear complexity w.r.t. the size of the two matrices. We show experimentally that our algorithm can outperform the state-of-the-art methods.

In addition to producing a solution, our algorithm produces a bound on how far the solution is from the optimum. The quality of the bound is data-dependent. In some cases (e.g., when X cannot explain Y well), it is loose, but in other cases, it shows that the computed result is very close to the optimal solution.

CHAPTER 6

ACCELERATED COMBINATORIAL SEARCH FOR OUTLIER DETECTION WITH PROVABLE BOUNDS ON SUB-OPTIMALITY

Authors – Guihong Wan, Haim Schweitzer

Department of Computer Science, EC 31

The University of Texas at Dallas

800 West Campbell Road

Richardson, Texas 75080-3021

Published in Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021.

Supplementary material is provided in the next chapter.

Abstract

Outliers negatively affect the accuracy of data analysis. In this paper we are concerned with their influence on the accuracy of Principal Component Analysis (PCA). Algorithms that attempt to detect outliers and remove them from the data prior to applying PCA are sometimes called Robust PCA, or Robust Subspace Recovery algorithms. We propose a new algorithm for outlier detection that combines two ideas. The first is "chunk recursive elimination" that was used effectively to accelerate feature selection, and the second is combinatorial search, in a setting similar to A*. Our main result is showing how to combine these two ideas. One variant of our algorithm is guaranteed to compute an optimal solution according to some natural criteria, but its running time makes it impractical for large datasets. Other variants are much faster and come with provable bounds on sub-optimality. Experimental results show the effectiveness of the proposed approach.

6.1 Introduction

An important challenge in the analysis of data is the design of a simple model that fits the data. In practical situations this requires handling data outliers. Typically, the data model can be made significantly more accurate if it is allowed to ignore a small fraction of the data items, considered to be outliers. See, e.g., (Aggarwal, 2016; Hodge and Austin, 2004).

The particular data model that we discuss in this paper is the one produced by Principal Component Analysis (PCA). Consider for example the data shown in Figure 6.1, consisting of 9 points. In this case, rank-1 PCA gives a bad model for all 9 points, but a good model for the 8 non-outliers. Observe the huge effect of a single outlier.

PCA is arguably the most widely used dimension reduction technique, with a variety of applications. See, e.g., (Jolliffe, 2002; Burges, 2010; Cabral et al., 2013; Vidal et al., 2016; Boutsidis et al., 2016; Gray, 2017). Recent algorithmic approaches that use randomization

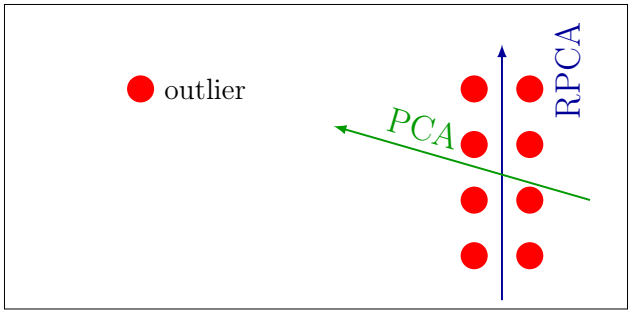

Figure 6.1: The direction of the dominant principal component computed from 9 points with one outlier. The direction labeled PCA (green arrow) was computed from the entire data (9 points). The direction labeled RPCA (blue arrow) was computed from the 8 non-outliers.

give algorithms that can easily handle large datasets (e.g., (Halko et al., 2011,?; Musco and Musco, 2015; Li et al., 2017)). However, computing robust variants that ignore outliers is still a big challenge.

There is a large number of studies on outlier detection and removal specifically for computing robust PCA. The algorithms are sometimes called "Robust Principal Component Analysis" (RPCA), or, alternatively, "Robust Subspace Recovery" (RSR). A recent review paper is (Lerman and Maunu, 2018c).

One way to solve the RPCA problem is to first filter outliers and then fit a subspace to the remaining data by using classical PCA. Many studies view the data as coming from a fixed distribution, and attempt to detect outliers as data points at the margins of the distribution (e.g., (Roberts, 1999; Scheirer et al., 2011; Hubert and Engelen, 2004; Xu et al., 2010; Zhang et al., 2015a)). Another common approach is to rank the data points based on some criterion, and detect outliers as points with low scores. See, e.g., (Chen and Lerman, 2009; Soltanolkotabi et al., 2012; Rahmani and Atia, 2017; You et al., 2017; Rahmani and Li, 2019a). For example, the "Cop" algorithm (Rahmani and Atia, 2017) computes the scores from the Gram matrix of the normalized data. The score for a data point is the norm of the corresponding row in the Gram matrix. The resulting algorithm runs very fast but requires a large amount of memory. The "R-graph" algorithm (You et al., 2017) measures

Table 6.1: Complexity of various algorithms. n is the number of data items. m is the dimension of each item. r is the number of principal components. k is the number of outliers. T is the number of iterations.

Algorithm	**time**	**memory**
Cop (Rahmani and Atia, 2017)	$O(mn^2)$	$O(n^2+mn)$
R-graph (You et al. 2017)	$O(mn^2+n^3)$	$O(n^2+mn)$
iSearch (Rahmani and Li, 2019a)	$O(mn^2)$	$O(mn)$
FMS (Lerman and Maunu, 2018b)	$O(T \cdot rmn)$	$O(mn)$
fastest Shah (Shah et al., 2018)	$O(krmn)$	$O(mn)$
fastest Chunk-A^* (this paper)	$O(rmn)$	$O(mn)$

the points based on the self-expressiveness property. The scores are computed by solving an elastic net minimization problem. The "Innovation" of each point is used to rank the points in (Rahmani and Li, 2019a). The authors refer to the algorithm as "iSearch".

A different approach was taken in (Shah et al., 2018). The authors view outlier detection as a graph search problem, and apply the weighted A^* algorithm to solve it. The goal of the search is to minimize the PCA error. As reported in (Shah et al., 2018) the algorithm comes with an accuracy parameter called ϵ. With $\epsilon = 0$ the algorithm is guaranteed to terminate with an optimal outlier selection. With larger ϵ values the algorithm terminates with outliers that are typically less accurate than the optimal, but the running time is much faster. Unfortunately, as we show in Section 6.7, even with $\epsilon = \infty$ the algorithm is slow, and cannot be applied to large datasets.

As we show experimentally in Section 6.7, current state-of-the-art algorithms that compute outliers encounter difficulties when applied to large datasets with hundreds of thousands/millions of data points. Most algorithms run too slow or require an unacceptably large amount of memory. Table 6.1 shows the complexity of some recent algorithms. We improve on previously proposed algorithms in terms of both runtime and accuracy. The time complexity of the fastest variant of our proposed Chunk-A^* is $O(k/c \cdot rmn)$, where c is the chunk size (introduced later). The memory complexity is $O(mn)$. The approach we take is outlined below.

6.1.1 Our Approach

Our goal is to design a scalable algorithm capable of detecting outliers in large datasets. A useful algorithm for the closely related feature selection problem (Guyon and Elisseeff, 2003) is "backward elimination", that requires a measure of the importance of each individual feature. Such a measure is typically called a "filter". Using filters, a typical implementation of backward elimination for selecting k features to be eliminated from among n features is as follows:

> *Run k iterations. In the jth iteration, rank all remaining features ($n-j+1$ features left) according to their filter values and eliminate the least important feature.*

Observe that this requires roughly nk filter evaluations which may be impractical. Instead, classical implementations of this approach (for example the well known SVM-RFE algorithm (Guyon et al., 2002)) eliminate a chunk of multiple features in each iteration. A chunk consists of a fraction of the features that are ranked as the least important. For sufficiently big chunks this reduces the number of filter evaluations to $O(n)$. We refer to this approach as the "Chunk-RFE", for "Chunk Recursive Feature Elimination".

Our main result is showing how to wrap a combinatorial search framework around the Chunk-RFE. This combines the speed of Chunk-RFE with the accuracy of the combinatorial search approach. The resulting algorithm that we call "Chunk-A^*" has two parameters that control the balance between speed and accuracy: the chunk size that we denote by c, and the ϵ parameter of the weighted A^*.

6.1.2 Our Results

We describe an outlier detection algorithm for detecting k outliers. (k is assumed to be known.) The algorithm error is defined as the PCA error of modeling the data without the outliers (see Section 6.2). The algorithm accuracy and running time are controlled by the two parameters ϵ, c, where $0 \leq \epsilon \leq \infty$ is the optimality parameter, and $1 \leq c \leq k$

is the chunk size. In general, larger values of ϵ and c result in a faster running time, and smaller values give more accurate results.

In addition to identifying outliers, the algorithm computes sub-optimality bounds on how close the results are to the optima. We are not aware of any other outlier detection algorithm that computes such bounds on sub-optimality.

The following are special cases that we prove:

- If $\epsilon = 0$ the algorithm terminates with an optimal outlier selection regardless of the value of the chunk size c.
- If $c = 1$ the algorithm is identical to (Shah et al., 2018). Still, even in this case, our algorithm has the advantage of producing sub-optimality bounds which are not computed by (Shah et al., 2018).

6.1.3 Paper Organization

The paper is organized as follows. The PCA error of modeling the data is defined in Section 6.2. The proposed algorithm is described in Section 6.3. The three variants of the algorithm along with a correctness proof are given in Section 6.4. An intuitive algorithm is introduced to further improve the results in Section 6.5. The sub-optimality bound is discussed in Section 6.6. Experimental results are shown in Section 6.7.

6.2 The Error of Modeling Data by PCA

Let $X = (x_1 \dots x_n)$ be the data matrix of size $m \times n$, where n is the number of data items and m is the dimension of each item. The PCA computes a linear mapping from the m dimensional x_i to the r dimensional y_i, where $r \leq \min(m, n)$. The inverse of this mapping gives an approximate reconstruction of x_i from y_i. We denote the reconstruction error of x_i by e_i. As in the classical development of PCA (e.g., (Jolliffe, 2002)) the quality of the

Algorithm 5: The Chunk-A^* algorithm.

Input:

X, a data matrix of n columns.

r, the desired number of principal components.

k, the desired number of outliers.

$f(S)$, a "filter" function. Here S is an outlier subset, and $f(S)$ is an estimate to the PCA error if the outliers in S are included in the solution subset.

$c(S)$, a criterion for determining the chunk size.

r_d (optional), the rank for dimensionality reduction. Default: $r_d = \min(m, n)$.

Output: a subset S of k outliers.

Preprocessing: dimensionality reduction (if desired).

Data Structures: Two global lists of subsets: the fringe list F, and the closed list C.

Initialization: Put the empty subset into F.

```
1  while F is nonempty do
2      Pick S_i with the smallest f(S_i) value from F. Ties are resolved in favor of the
         larger size(S_i).
3      if S_i contains k outliers then
4          Stop and return S_i as the solution subset.
5      else
6          Create U_i as the list of all subsets that can be obtained by adding a single
             column to S_i.
7          Remove from U_i all the subsets that are in C.
8          Compute f(S_j) for each subset S_j ∈ U_i.
9          Select chunk size c(S_i) satisfying: 1 ≤ c(S_i) ≤ k−size(S_i).
10         Compute the subset S̄_j as the union of the c(S_i) subsets in U_i with the
             smallest f(S_j).
11         Add S̄_j to F and C.
12         Add to F and C all the subsets S_j ∈ U_i that are not used in the creation of
             S̄_j.
13     end
14 end
```

mapping can be measured by the sum of reconstruction errors as follows:

$$e_i(r) = \|x_i - VV^T x_i\|^2, \quad E(r) = \sum_{i=1}^{n} e_i(r).$$

It is known (e.g., (Jolliffe, 2002)) that the columns of V can be computed as the r eigenvectors of the matrix $B = XX^T$ corresponding to its r largest eigenvalues. A known result about matrix eigenvalues gives an explicit expression to this error in terms of the eigenvalues of B.

Suppose $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m$ are the eigenvalues of B, and V is $m \times r$ then:

$$E(r) = \sum_{j=r+1}^{m} \lambda_j = \text{trace}(B) - \sum_{j=1}^{r} \lambda_j.$$

Suppose X is partitioned into a subset of outliers S and the reminder inliers subset $\tilde{X} = X \backslash S$. The desired rank-$r$ PCA is computed from the inliers. Accordingly, the PCA error is computed from the inliers subset:

$$\begin{aligned} e_i(S,r) &= \|x_i - \tilde{V}\tilde{V}^T x_i)\|^2, \\ E(S,r) &= \sum_{i \notin S} e_i(S,r) = \sum_{j=r+1}^{m} \tilde{\lambda}_j = \text{trace}(\tilde{B}) - \sum_{j=1}^{r} \tilde{\lambda}_j, \end{aligned} \tag{6.1}$$

where $\tilde{\lambda}_j$ and $\tilde{V}$ are computed from $\tilde{B} = \tilde{X}\tilde{X}^T$.

6.3 The Chunk-A^* Algorithm

In this section we describe our algorithm in terms of a general filter function that will be defined later. We refer to it as the Chunk-A^*. It is closely related to standard combinatorial search algorithms such as the A^* (Pearl, 1984; Russell and Norvig, 2010; He et al., 2019). The major difference is that previous studies do not have the Chunk-RFE part, as shown in lines 9-11 of Algorithm 5.

The goal of the algorithm is to compute a subset of k outliers, minimizing the "filter" function f. It maintains in F a list of subsets that need to be further evaluated and in C a list of visited subsets that need not to be added into F again. In the standard "best first" strategy, the algorithm selects the subset from F with the smallest f value to be expanded. However, unlike standard combinatorial search algorithms in our variant not all of the direct expansions of the selected subset are treated equally. Instead, the algorithm "guesses" that the best c among them are part of the solution, and packages them together in a "chunk". We refer to the "chunk" as the "super child", and the children created by adding a new column as "direct children".

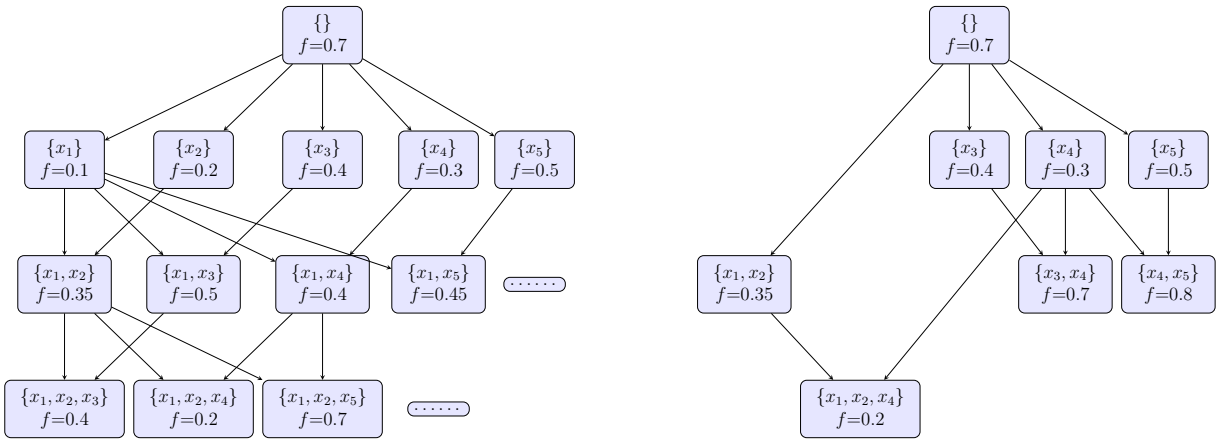

Figure 6.2: Examples of subset graphs with $n = 5$. The left graph is for c=1; the right graph is for c=2. For instance, when the root node is expanded, in the left graph all direct children are added into F; in the right graph, the super child $\{x_1, x_2\}$ is created by taking the union of $\{x_1\}$ and $\{x_2\}$. This allows the algorithm to quickly evaluate nodes in deeper levels.

6.3.1 The Subset Graph

The algorithm can be viewed as searching on a graph created with nodes corresponding to outlier subsets, similar to the subset graph proposed in (Arai et al., 2015, 2016). With the chunk size $c = 1$, there is an edge from the subset S_i to the subset S_j if adding one column to S_i creates S_j. When $c > 1$, super children are created for each node. Two subset graphs for the matrix $X = (x_1, \ldots, x_5)$ and $k = 3$ are shown in Figure 6.2.

Note that all paths leading from the root to a node can be considered equivalent, since the order of outliers in a subset does not matter. For example, if the goal node $\{x_1, x_2, x_4\}$ in the right graph is found, it is irrelevant if it is reached by the path $\{\}\rightarrow\{x_1, x_2\}\rightarrow\{x_1, x_2, x_4\}$ or by the path $\{\}\rightarrow\{x_4\}\rightarrow\{x_1, x_2, x_4\}$. This property makes the chunking meaningful.

6.3.2 The Chunk Size

The chunk size is used in Line 9 of the algorithm. The expectation is that using big chunks would result in fast convergence of the algorithm, while small chunks would give more accurate results. As we show, our theory requires that the chunk size should satisfy the following

condition:

$$1 \leq c(S_i) \leq k - \text{size}(S_i).$$

The right part $k-\text{size}(S_i)$ is the remaining number of outliers that still have to be selected. With chunk size $c = 1$, there is no acceleration at all.

6.3.3 The Filter Function

From Equation (6.1) it is clear that the subset with the smallest error can be found by exhaustively evaluating all the subsets of size k, but this is clearly impractical. Instead, we use filters within a search algorithm that evaluates only some of these subsets.

Suppose we are given a subset S_i of size $k_i < k$. We consider the error that can be obtained by completing it to size k. The best error that an outlier subset S_i can achieve is:

$$d(S_i, r, k) = \min_{\text{size}(S_i \cup T) = k} E(S_i \cup T, r). \tag{6.2}$$

Ideally, we may wish to use $d(S_i, r, k)$ as the filter function, but computing it is inefficient since it involves going over all subsets of size $k-k_i$. Instead, we approximate it using lower and upper bounds defined as follows. Let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_m$ be the eigenvalues of the matrix $\tilde{B}$ computed from the inliers at a graph node. Define:

$$\begin{aligned} l(S_i, r, k) &= E(S_i, r + k - k_i) = \sum_{j=r+k-k_i+1}^{m} \lambda_j, \\ u(S_i, r, k) &= E(S_i, r) = \sum_{j=r+1}^{m} \lambda_j. \end{aligned} \tag{6.3}$$

These are similar to the functions defined in (He et al., 2019).

Lemma 13. For any subset S_i satisfying $\text{size}(S_i) \leq k$:

$$l(S_i, r, k) \leq \; d(S_i, r, k) \; \leq u(S_i, r, k)$$

when $\text{size}(S_i) = k$ both inequalities become equalities.

The proof is based on the interlacing property of eigenvalues (e.g., (Hill and Parlett, 1992; Golub and Van-Loan, 2013)). Details will be given in the supplementary material of this paper.

6.4 Three Variants of the Algorithm

To simplify notations we write $l_i = l(S_i, r, k)$ and $u_i = u(S_i, r, k)$. Lemma 13 shows that the best choice for a filter is "sandwiched" between l_i and u_i. Following (He et al., 2019) we express the filter function f_i as a linear combination of l_i and u_i, with three distinct options: 1. $f_i = l_i$; 2. $f_i = u_i$; and 3. $f_i = l_i + \epsilon u_i$, where $\epsilon \geq 0$. In each case the Chunk-A^* has distinct characteristics as stated in the theorems below:

Theorem 10. (Optimal variant) If $f_i = l_i$, then the Chunk-A^* terminates with an optimal solution.

Theorem 11. (Greedy variant) If $f_i = u_i$, then the Chunk-A^* is greedy and terminates after at most k node expansions.

Theorem 12. (Suboptimal variant) If $f_i = l_i + \epsilon u_i$, then the Chunk-A^* guarantees a solution "close" to the optimum. Let S_* be an optimal solution subset of outliers. Let $e^* = E(S_*, r)$ be the corresponding error. Let S_{**} be the solution subset of outliers found by the algorithm. Let e^{**} be the error for S_{**}. Let l_{**} be the value of $l(S_{**}, r, k)$. Let $u_{\max}$ be the largest value of $u(S_i, r, k)$ for the subsets S_i remaining in the fringe list F after the goal node is reached. Then:

$$e^{**} \leq e^* + \epsilon(u_{\max} - l_{**}). \tag{6.4}$$

6.4.1 Proof of Theorems

In this section we give the technical proofs of the theorems stated in Section 6.4. Similar theorems were stated and proved in (He et al., 2019) for feature selection. However, their

proof strategy cannot be directly applied to the outlier detection case that we study here. Our proofs use several lemmas that are not fully proved. Complete proof details will be given in the supplementary material of this paper.

Lemma 14. The value of l_i is monotonically increasing along any path.

(The proof follows from the interlacing property of eigenvalues.)

Lemma 15. The value of u_i is monotonically decreasing along any path.

(The proof follows from the interlacing property of eigenvalues.)

Lemma 16. Consider the choice $f_i = u_i$. The f value of the super child is smaller than the f value of any of its siblings.

Lemma 17. Consider the choice $f_i = u_i$. Let n_i be the node picked at Line 2 of the algorithm. Let n_j be a child of n_i. The following three properties hold:

a. The size of S_j associated with node n_j is larger than the size of all other nodes currently in the fringe list.

b. The next node to be picked is a child of n_i.

c. If the chunk size $c > 1$, the next node to be picked is the super child of n_i.

(The proof is by induction.)

Lemma 18. Suppose Theorem 12 is false. Then for any node n_z on the path from the root to an optimal node n_* (corresponding to an optimal outlier subset S_*), the following condition holds: $f_z < f_{**}$.

(The proof is similar to the corresponding lemma in (He et al., 2019).)

Lemma 19. Let S_* be an optimal outlier subset of size k. If the algorithm always uses chunk size $c(S_i)$ satisfying: $1 \leq c(S_i) \leq k-\text{size}(S_i)$, then during the run of the algorithm the fringe list F always contains a subset of S_*.

(The proof is by induction.)

Proof of Theorem 10:

The proof follows as a corollary of Theorem 12 with $\epsilon = 0$. ■

Proof of Theorem 11:

From Lemma 17 when a node n_i is picked in Line 2 of the algorithm, one of its children will be examined next. If the chunk size $c > 1$, then the super child will be examined next. This shows that the algorithm is greedy. It detects c outliers in each iteration, and terminates after $\lceil \frac{k}{c} \rceil$ iterations. ■

Proof of Theorem 12:

The proof is by contradiction. Suppose the theorem is false. From Lemma 18 it follows that all subsets on the path from the root to an optimal subset S_* have smaller f values than f_{**}. Since from Lemma 19 at any given time at least one of them is in the fringe list, they should be selected before S_{**} is selected. But this means that S_* is selected as the solution and not S_{**}. ■

6.4.2 Bounds

Both the Greedy variant and the Suboptimal variant are not guaranteed to produce an optimal solution. We proceed to show how to obtain a bound on how close their solution is to the optimum. The technique we use was originally proposed by (Hansen and Zhou, 2007).

Consider a run of a non-optimal variant, producing the non-optimal subset S_{**}. Then size$(S_{**}) = k$, and from Lemma 13 it follows that $l_{**} = u_{**} = E(S_{**}, r)$. The value of l_{**} is related to the optimal value l_* by: $l_* \leq l_{**}$. Let B be a value satisfying: $l_{**} - l_* \leq B$. We refer to B as a bound, where a smaller B indicates a better bound, and in particular, B=0 implies an optimal solution. An important observation is that in many combinatorial search techniques it is possible to compute such values.

Let F be the fringe list after the algorithm terminates. Going over all the remaining nodes in F we can compute: $l_{\min} = \min_{S_i \in F} l_i$. Then from Lemma 14 it follows that: $l_* \geq l_{\min}$. Therefore we can take:

$$B = l_{**} - l_{\min}, \quad \text{where: } l_{\min} = \min_{S_i \in F} l_i. \tag{6.5}$$

6.5 Improving the Accuracy

We propose a simple algorithm that can improve the accuracy of the results obtained by the Chunk-A^* algorithm. The idea is similar to the well known k-means technique, and we refer to it as Algorithm 6. Running it on the results obtained by the Chunk-A^* will always decrease (never increase) the PCA error, but it typically gets "stuck" in a local minimum after a small number of iterations.

Algorithm 6: Improving Outliers
Input: X: a matrix of n columns. r: the PCA rank.
S: a subset of current outlier selection.
$T_{\max}$: the maximum number of iterations.
Output: a subset R of X satisfying: $|R| = |S|$ and $E(R, r) \leq E(S, r)$.

1 Set $k = |S|$.
2 **repeat**
3 Set old error = the current error.
4 Compute V as the rank-r PCA of $X \backslash S$.
5 Compute $e_i = \|x_i\|^2 - \|V^T x_i\|^2$ for all columns x_i of X.
6 Replace the k outliers of S by the k columns of X with the largest e_i.
7 Set new error = $\sum_{x_i \notin S} e_i$.
8 **until** *new error = old error* ***or*** *reach* T_{max};
9 Set $R = S$ and return.

The algorithm is motivated by the observation that the PCA error can be evaluated separately for each column of X. As in Equation (6.1), let V be a matrix with r orthogonal columns. The reconstruction error of x_i is given by: $e_i = \|x_i - VV^T x_i\|^2 = \|x_i\|^2 - \|V^T x_i\|^2$, and the PCA error can be computed as: $E(S, r) = \sum_{i \notin S} e_i$.

The idea of Algorithm 6 is that if we know S we can calculate V using PCA, which gives optimal reduction to the reconstruction error of the columns not in S. Once V is known the best selection of k outliers from X is the columns with the k largest reconstruction errors. The PCA error reduction by this process improves the outlier selection. The complexity of the algorithm is $O(T_{\max} rmn)$, where $T_{\max}$ is the maximum number of iterations to convergence. Our experimental results show that the number of iterations is typically at most 3, and thus can be effectively viewed as a constant. Correctness proof will be given in the supplementary material of this paper.

6.6 Fractional Bounds on Sub-Optimality

The bounds that we derive in this section measure how close the computed result is relative to the optimal result. For example, it may be interesting to know that the error of the computed outlier set is within 10% of the optimal selection. Let E_k be the PCA error of the outlier set S_k produced by the Chunk-A^* algorithm for input size k. Let E_k^* be the smallest possible PCA error of an outlier set of size k. The bound B_k computed from (6.5) satisfies: $E_k \leq E_k^* + B_k$. Let b be the fractional bound: $b = (E_k - E_k^*)/E_k^*$. (It is meaningful only when $E_k^* \neq 0$). After each run of the Chunk-A^* algorithm, we can compute an upper bound on b as shown in (6.6):

$$b = \frac{E_k - E_k^*}{E_k^*} \leq \frac{B_k}{E_k^*} \leq \frac{B_k}{E_k - B_k} = b_0. \tag{6.6}$$

The value of b_0 is a nontrivial bound on b, and can be calculated since both B_k, E_k are known at termination. Now consider running Algorithm 6 after the Chunk-A^*, which gives a new outlier subset with the PCA error $E_k^1 \leq E_k$. We show that the fractional bound b_0 can be improved to the fractional bound b_1 as follow:

$$b = \frac{E_k^1 - E_k^*}{E_k^*} \leq \frac{E_k^1 - (E_k - B_k)}{E_k^*} \leq \frac{B_k - (E_k - E_k^1)}{E_k - B_k} = b_1. \tag{6.7}$$

Clearly, b_1 can be computed after the run of Algorithm 6 and it is a sharper bound on b than b_0.

6.7 Experimental Results

We evaluate the performance of the proposed algorithm on various synthetic and real datasets. The real datasets are publicly available. The comparison is with various algorithms that made their code available, including: R-graph (You et al., 2017); iSearch (Rahmani and Li, 2019a); GGD (Maunu et al., 2019a); FMS (Lerman and Maunu, 2018b); Cop (Rahmani and Atia, 2017); RMD (Goes et al., 2014); TME (Zhang, 2016a); OP (Xu et al., 2012a); MKF (Zhang et al., 2009); DHRPCA (Feng et al., 2012); TORP (Cherapanamjeri et al., 2017); R1PCA (Ding et al., 2006).

6.7.1 Experiments on Synthetic Datasets

We model our experiments after the detailed study described in a recent survey paper (Lerman and Maunu, 2018c). They use two synthetic data models: the Haystack model and the Blurryface model. Let U_* be the ground truth subspace. The error is computed as the squared principal angles between U_* and the recovered U.

The Haystack model. With fixed parameters $m = 200$, $n = 400$, and $r = 10$, inliers are drawn independently at random from $\mathbb{N}(0, U_*U_*^T/r)$, and the outliers are drawn independently at random from $\mathbb{N}(0, I/D)$. All points are perturbed by adding noise from i.i.d. $\mathbb{N}(0, 10^{-2}I)$. The entire dataset is centered at the end. One deficiency of the Haystack model is that the recovery of subspace can be easy for some algorithms, as mentioned in (Lerman and Maunu, 2018c).

The Blurryface model. The motivation for the Blurryface dataset is to generate data resembling real data. The Extended Yale face database B (Kuang-Chih Lee et al., 2005) was used. The first individual's face is used to obtain the 9-dimensional subspace U_*. The inliers are generated i.i.d. $\mathbb{N}(0, c_1U_*U_*^T/r)$ where c_1 is tuned to give inliers comparable magnitude.

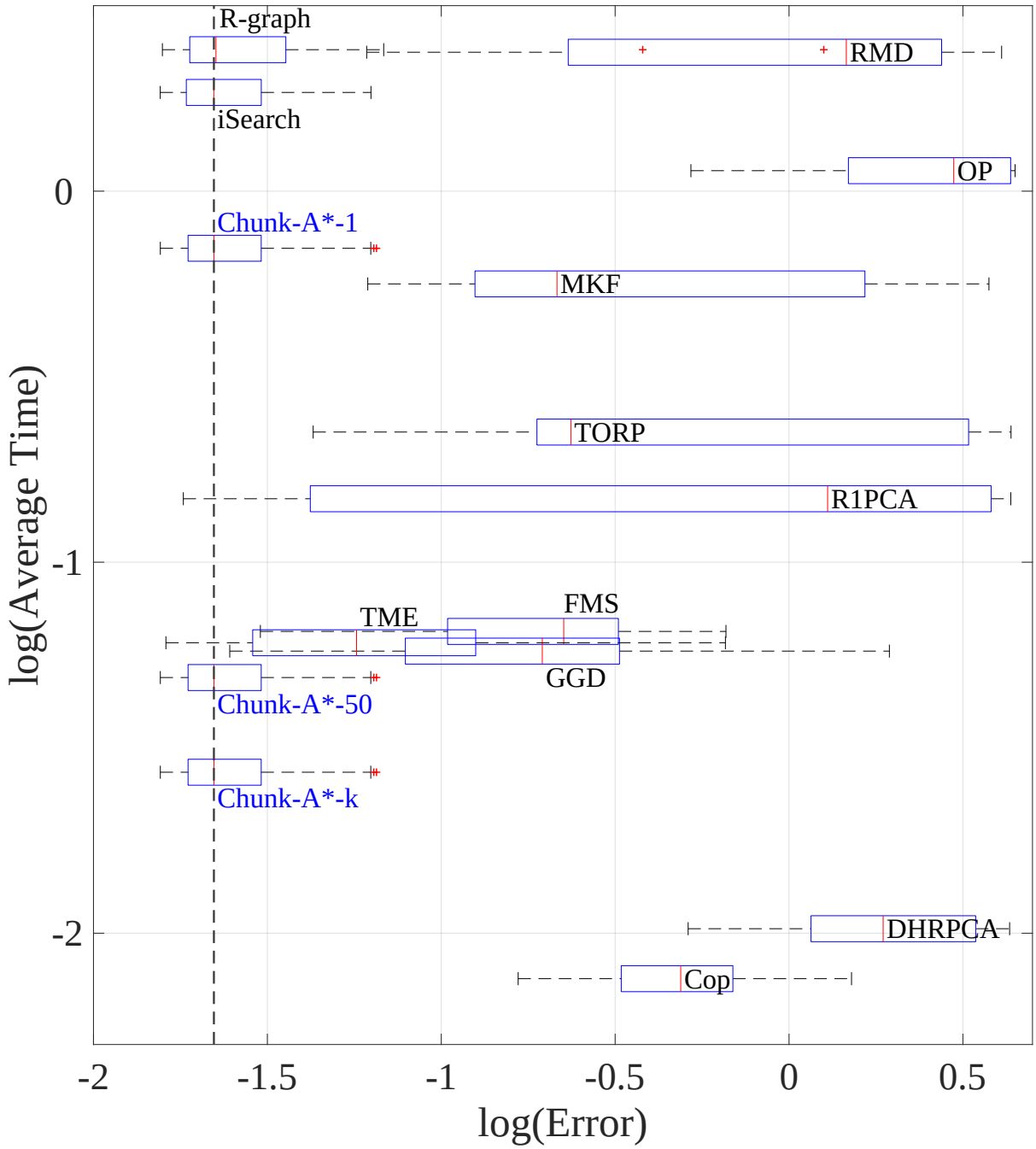

Figure 6.3: Comparison on the Haystack model. The horizontal axis is the boxplot of errors with different outlier percentages. The red vertical line in a box corresponds to the median of errors. The vertical axis is the average running time. The dashed vertical line corresponds to the median error of the ground truth. The closer to the dashed vertical line, the more accurate the algorithm is. The closer to the bottom, the faster the algorithm is.

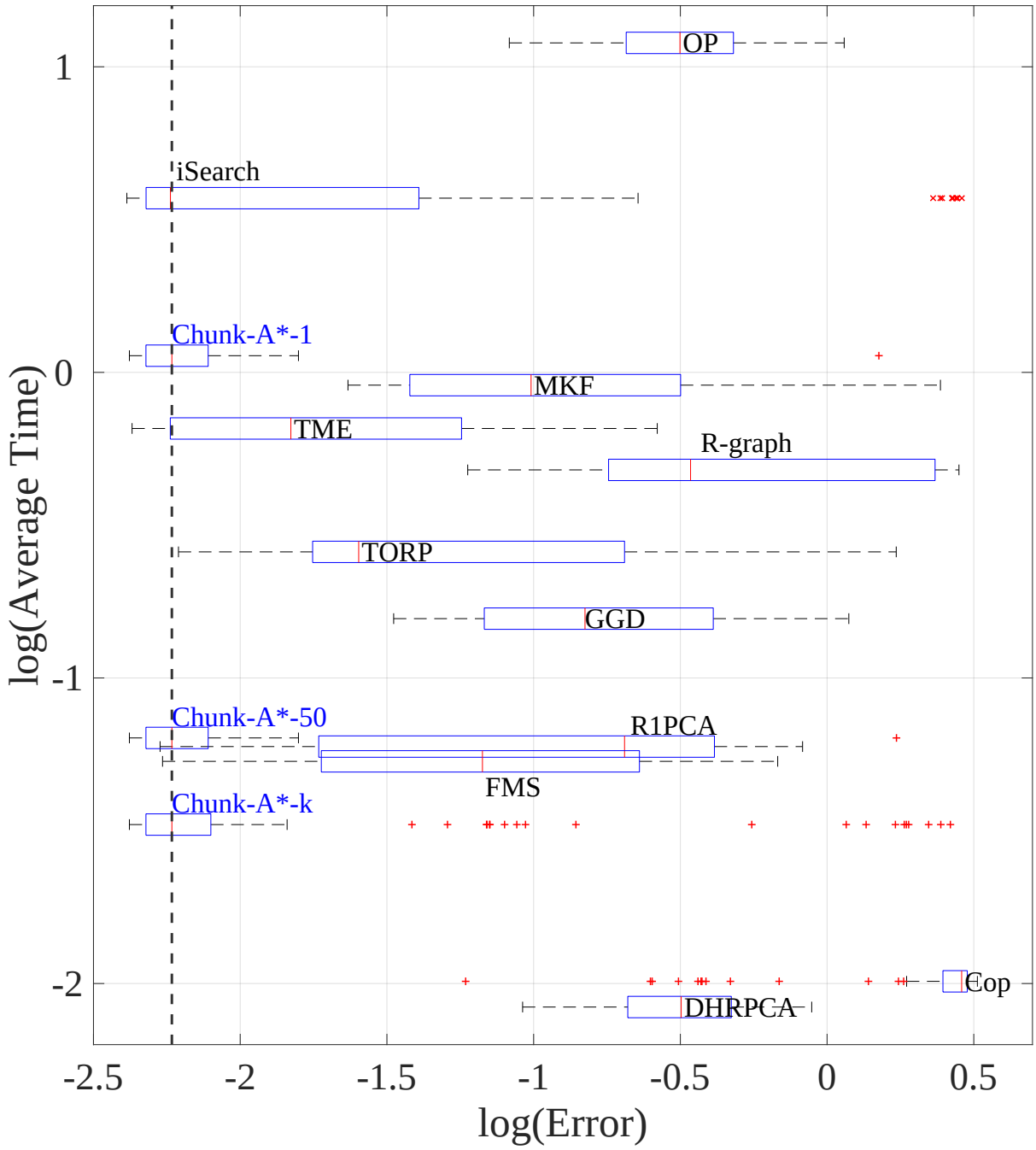

Figure 6.4: Comparison on the Blurryface model.

The outliers are sampled from other faces. The resulting dataset is of size m=400, n=500. See (Lerman and Maunu, 2018c) for details.

Table 6.2: Experiments with the suboptimal variant with different chunk sizes on TechTC01 dataset (29,261 × 163).

$r:k:\epsilon$		c=1	c=0.2∗k	c=0.5∗k	c=k
5:20:0.2	b_1	**0.12**	**0.12**	**0.12**	0.14
	error_0	**563,313**	**563,313**	**563,313**	583,633
	error_1	**563,313**	**563,313**	**563,313**	577,102
	time	67s	21s	18s	17s
5:60:0.2	b_1	**0.26**	0.31	0.31	0.50
	error_0	**110,054**	113,716	115,004	133,020
	error_1	**110,054**	113,716	113,716	130,433
	time	93s	18s	16s	16s
10:70:0.1	b_1	**0.15**	0.17	0.19	0.19
	error_0	**46,461**	47,213	48,115	48,148
	error_1	**46,461**	47,213	48,115	48,148
	time	41s	17s	19s	16s
10:90:0.1	b_1	**0.18**	0.20	0.22	0.21
	error_0	**19,673**	19,881	20,314	20,169
	error_1	**19,673**	19,881	20,314	20,169
	time	417s	18s	17s	17s

To investigate the effect of different outlier percentages on errors, 10 datasets were generated for each percentage value from 10% to 90%. The Chunk-A^* uses ϵ=∞ and several chunk sizes. The algorithm is marked as Chunk-A^* followed by the c value. Chunk-A^*-1 is Chunk-A^* with c=1, Chunk-A^*-50 is Chunk-A^* with c=50, and Chunk-A^*-k is Chunk-A^* with c=k, which is the largest allowable value of c. The number of outliers is given as input for the following algorithms: R-graph, iSearch, DHRPCA, Shah, Cop and Chunk-A^*. Parameters unspecified for other algorithms are the same as those used in (Lerman and Maunu, 2018c).

The results for the Haystack model are shown in Figure 6.3. Our Chunk-A^* is the fastest among those which achieve high accuracy. The results for the Blurryface model are shown in Figure 6.4. Our Chunk-A^* is both accurate and fast.

Table 6.3: Accuracy and bounds with different filter functions. The minimum errors and bounds are highlighted. "-" indicates no results after running for 5 minutes.

dataset	r:k	$f = l$ (optimal)	$f = l + 0.2u$	$f = l + 0.5u$	$f = l + 1.0u$	$f = u$	$f = l + u$ bound:b_0	$f = l + 0.5u$ bound:b_0	iSearch	R-graph	TME
spectf	5:5	**290,498**	**290,498**	**290,498**	**290,498**	**290,498**	0.29	**0.09**	293,043	299,100	303,907
267×45	7:7	**203,556**	**203,556**	**203,556**	**203,556**	**203,556**	0.40	**0.19**	217,235	212,274	218,210
vehicle	5:5	**35,908**	**35,908**	**35,908**	36,211	36,211	0.39	**0.38**	52,553	54,686	42,234
846×18	10:5	**1,212**	1,242	1,242	1,242	1,580	**0.05**	**0.05**	2,122	4,140	2,354
libras	1:3	**591.43**	**591.43**	**591.43**	**591.43**	**591.43**	**0.00**	**0.00**	**591.43**	598.72	**591.43**
360×90	20:7	-	-	**1.03**	**1.03**	1.07	0.59	**0.22**	1.13	1.15	1.15

Table 6.4: Greedy variant on big datasets. "-" indicates no results since the implementation does not support sparse data format. The common parts of the errors: $e_1 = 4.61\text{E}10$, $e_2 = 6.2\text{E}11$, and $e_3 = 3.1\text{E}11$.

dataset	$r_d : r : k$		$c = 1$	$c = 0.5{*}k$	$c = k$	iSearch	R-graph	TME
Day1:sparse	100:30:100	error	> 60min	**59,976**	60,413	-	-	-
$20,000 \times 3,231,957$		runtime		884s	607s	-	-	-
Sift:dense	70:15:100	error	e_1 + **5.95E6**	e_1 + 5.96E6	e_1 + 5.96E6	ArrayLimit	> 10min	e_1 + 6.08E6
$128 \times 1,000,000$		runtime	1523s	78s	65s			43s
YPMSD:dense	50:10:100	error	e_2 + **2.79E8**	e_2 + **2.79E8**	e_2 + 2.80E8	ArrayLimit	> 10min	e_2 + 3.02E8
$90 \times 515,345$		runtime	438s	22s	17s			8s
Covtype:dense	30:3:30	error	e_3 + **8.45E7**	e_3 + **8.45E7**	e_3 + **8.45E7**	ArrayLimit	> 10min	e_3 + 9.84E7
$54 \times 581,012$		runtime	69s	7s	6s			21s

6.7.2 Experiments on Real Datasets

We describe experiments with standard datasets that are commonly used for evaluating machine learning algorithms. Since the ground truth is unknown, the error is taken to be the PCA error on the inliers, as defined in Section 6.2.

Various filter functions: With different selections of filter functions, our algorithm produces optimal and suboptimal results. The results of different filter functions for various datasets with c=1 are shown in Table 6.3. Observe that: 1. With ϵ=0 (optimality guaranteed) the algorithm always produces the smallest values; 2. Larger ϵ values give solutions that are typically less accurate than the optimal; 3. Smaller ϵ values yield tighter bounds; 4. Our algorithm is significantly more accurate than other algorithms.

Bounds on sub-optimality: The suboptimal solutions given by our algorithm come with guarantees on how close the solutions are to the optima. Table 6.2 shows the bounds and

errors with different chunk sizes, where error_0 is the error after running Chunk-A^*; error_1 is the error after running Algorithm 6. Observe that: 1. With chunking ($c > 1$), the algorithm runs faster while the error and bound values become bigger. 2. Algorithm 6 improves the results when the chunk size is big (e.g., $c \geq 0.5{*}k$).

Experiments with big datasets: Experimental results with the greedy variant applied to big datasets are shown in Table 6.4. Without chunking (c=1), the algorithm is much slower. R-graph and iSearch cannot work with large datasets.

6.8 Concluding Remarks

Identifying outliers for Principal Component Analysis is an important problem in data analytics and machine learning. We propose a new algorithm for outlier detection that combines the "chunk recursive elimination" and the combinatorial search, similar to the classical A* search algorithm. We describe three variants of the algorithm. One variant is guaranteed to produce optimal solutions. Other variants are not optimal, but compare favorably with current alternatives. We also show how to compute sub-optimality bounds for this problem. Extensive experiments demonstrate the effectiveness of the proposed approach.

Acknowledgments

We would like to thank various authors who made their code available online. In particular, our experiments are closely modeled after the work described in (Lerman and Maunu, 2018c) which we found to be invaluable.

CHAPTER 7

ACCELERATED COMBINATORIAL SEARCH FOR OUTLIER DETECTION WITH PROVABLE BOUNDS ON SUB-OPTIMALITY: SUPPLEMENTARY MATERIAL

7.1 Proof of Lemmas in Section 6.3 and 6.4

In this section we give the technical proofs of lemmas in Section 6.3 and 6.4. Our proofs have similar structure to the proofs in (He et al., 2019). The tool we use is the interlacing property of eigenvalues.

7.1.1 Eigenvalue Inequalities

We need some nontrivial results from matrix theory. Let $X = (x_1 \dots x_n)$ be the $m \times n$ data matrix. Let V be $m \times r$ with orthogonal columns and $r \leq \min(m, n)$. Set $y_i = V^T x_i$ and consider the reconstruction errors:

$$e_i(r) = \|x_i - VV^T x_i\|^2, \quad E(r) = \sum_i e_i(r). \tag{7.1}$$

It is known (e.g., (Golub and Van-Loan, 2013; Jolliffe, 2002)) that the columns of V in (7.1) can be taken as the r eigenvectors corresponding to the r largest eigenvalues of the following matrix: $B = XX^T = \sum_{i=1}^{n} x_i x_i^T$. We refer to B as the matrix of second moments of X. Let $v_1, \dots, v_m$ be the eigenvectors of B and let $\lambda_1, \dots, \lambda_m$ be the corresponding eigenvalues arranged in decreasing order. Then it is known (e.g., (Golub and Van-Loan, 2013)) that:

$$E(r) = \sum_{i=r+1}^{m} \lambda_i. \tag{7.2}$$

Suppose the q outliers in a set $T = \{t_1, \dots t_q\}$ are removed from X. Denote the updated data matrix by X_T, the corresponding matrix of second moments by B_T, its eigenvalues by $\lambda_1(T), \dots \lambda_m(T)$, and the corresponding reconstruction error by $E_T(r)$. Observe that

$B_T = X_T X_T^T = \sum_{i=1}^{n} x_i x_i^T - \sum_{t=1}^{q} t_t t_t^T$ is a rank-q update of B. From the classical eigenvalue interlacing property (e.g., (Golub and Van-Loan, 2013; Thompson, 1976)):

$$\text{for each } i: \quad \lambda_{i+q} \leq \lambda_i(T) \leq \lambda_i.$$

These inequalities hold (in the case where B_T is known to be positive semidefinite) under the convention that $\lambda_i = 0$ for $i > m$. Summing up over i in the range $r+1 \leq i \leq m$ we get:

$$\text{for any outlier set } T: \quad \sum_{i=r+q+1}^{m} \lambda_i \leq \sum_{i=r+1}^{m} \lambda_i(T) \leq \sum_{i=r+1}^{m} \lambda_i. \tag{7.3}$$

7.1.2 Proof of Lemma 13 in Section 6.3.3

Lemma 13: For any subset S_i of size$(S_i) \leq k$:

$$l(S_i, r, k) \leq \ d(S_i, r, k) \ \leq u(S_i, r, k)$$

when size$(S_i) = k$ both inequalities become equalities.

Proof: Recall the definitions:

$$\begin{aligned}
d(S_i, r, k) &= \min_{\text{size}(S_i \cup T) = k} E(S_i \cup T, r), \\
l(S_i, r, k) &= E(S_i, r + k - k_i) = \sum_{j=r+k-k_i+1}^{m} \lambda_j, \\
u(S_i, r, k) &= E(S_i, r) = \sum_{j=r+1}^{m} \lambda_j,
\end{aligned}$$

where $k_i = \text{size}(S_i)$. To prove the left inequality it is enough to observe that the value of $k - k_i$ in the definition of l is exactly the value of q in (7.3). The right inequality follows from the right inequality in (7.3). When $k_i = k$ the values of $l()$ and $u()$ are equal, and $d()$ is "sandwiched" between them. ■

7.1.3 Proof of Lemmas in Section 6.4.1

Lemma 14: The value of l_i is monotonically increasing along any path.

Proof: Let S_i be the subset associated with node n_i. We need to show that if $S_j \supset S_i$ then $l_j = l(S_j, r, k)$ is greater than $l_i = l(S_i, r, k)$. Set $T = S_j \setminus S_i$. Then the lemma follows from the left inequality in (7.3). ■

Lemma 15: The value of u_i is monotonically decreasing along any path.

Proof: Let S_i be the subset associated with node n_i. We need to show that if $S_j \supset S_i$ then $u_j = u(S_j, r, k)$ is smaller than $u_i = u(S_i, r, k)$. As in the proof of Lemma 14, set $T = S_j \setminus S_i$. Then the lemma follows from the right inequality in (7.3). ■

Lemma 16: Consider the choice $f_i = u_i$. The f value of the super child is smaller than the f value of any of its siblings.

Proof: Let S_i be the subset associated with node n_i. Let U_i be the list of all subsets that can be obtained by adding a single data point to S_i. Let $\overline{n}_j$ be the super child of n_i. Let $\overline{S}_j$ be the subset associated with node $\overline{n}_j$. From the creation of the super node, the $\overline{S}_j$ is the union of the $c(S_i)$ subsets in U_i with the smallest f. This implies: 1. those $c(S_i)$ subsets have smaller f than any other children of n_i. 2. $\overline{S}_j$ is the superset of those $c(S_i)$ subsets. Then, the lemma follows from Lemma 15. ■

Lemma 17: Consider the choice $f_i = u_i$. Let n_i be the node picked at Line 2 of Algorithm 5. Let n_j be a child of n_i. The following three properties hold:

a. The size of S_j associated with node n_j is larger than the size of all other nodes currently in the fringe list.

b. The next node to be picked is a child of n_i.

c. If the chunk size $c > 1$, the next node to be picked is the super child of n_i.

Proof: The proof is by induction. Property **a** follows trivially from Property **b**. To prove Property **b** observe that from Lemma 15, u_i is monotonically decreasing (non-increasing)

along any path. Therefore, the f values of the children of n_i will be no greater than the f values of all the nodes currently in the fringe. Since ties are resolved in favor of a child, so that the child of n_i will be selected next. To prove Property **c**, we need to prove that the f value of the super node is smaller than the f value of any other children of n_i. This follows from Lemma 16. ■

Lemma 18: Suppose Theorem 12 is false. Then for any node n_z on the path from the root to n_* the following condition holds: $f_z < f_{**}$.

Proof: The falsehood of Theorem 12 can be written as follows: $e^{**} > e^* + \epsilon(u_{\max} - l_{**})$. Since both n_* and n_{**} are goal nodes their corresponding subsets S_* and S_{**} are both of size k. Lemma 13 implies: $e^{**} = l_{**} = u_{**}$ and $e^* = l_* = u_*$. Using this and some algebra it can be shown that an equivalent falsehood condition is: $l_{**} > l_* + \frac{\epsilon}{1+\epsilon}(u_{\max} - l_*)$. The lemma can now be proved as follows:

$$
\begin{aligned}
f_{**} \quad &= l_{**} + \epsilon u_{**} = (1+\epsilon) l_{**} && (c_1)\\
&> (1+\epsilon) l_* + \epsilon(u_{\max} - l_*) && (c_2)\\
&= l_* + \epsilon u_{\max} \geq l_z + \epsilon u_{\max} && (c_3)\\
&\geq l_z + \epsilon u_z = f_z
\end{aligned}
$$

c_1 : from the definition of f. c_2 : from the equivalent falsehood assumption. c_3 : from Lemma 14, $l_* > l_z$. ■

Lemma 19: Let S_* be an optimal outlier subset of size k. If the algorithm always uses chunk size $c(S_i)$ satisfying: $1 \leq c(S_i) \leq k - \text{size}(S_i)$ then during the run of the algorithm the fringe list F always contains a subset of S_*.

Proof: We use induction. The claim is clearly true initially when F contains the empty set. Suppose it is true when S_i is picked. We need to show that the claim holds after S_i is replaced with a super child and possibly other direct children. If S_i is not a subset of

S_* then from the inductive assumption there is another subset of S_* in F and we are done. The only case left is when S_i is a subset of S_*. There are $k-\text{size}(S_i)$ immediate children of S_i (of size size(S_i)+1) that are all distinct subsets of S_*. If $c(S_i) < k-\text{size}(S_i)$ then at least one of them is not in the super node and we are done. We are left with the case where $c(S_i) = k-\text{size}(S_i)$. If the super node contains one subset that is not a subset of S_i then there is one subset of S_* outside the super node and we are done. Otherwise the super node is the union of all the immediate children of S_i that are subsets of S_*. This means that the super node equals to S_* and we are done. ■

7.2 The Correctness Proof for Algorithm 6

Suppose that X is partitioned into a subset of outliers S and the reminder subset $\tilde{X} = X \backslash S$. The PCA error as given by Equation (6.1) in the main paper can be written as follows, where V is constrained to be of rank r with orthogonal columns:

$$\begin{aligned} E(S, r) &= \min_V ||\tilde{X} - VV^T\tilde{X}||_F^2 \\ &= \min_V \sum_{x_i \in S} e_i(V), \end{aligned}$$

where $e_i(V) = ||x_i - VV^T x_i||^2$. The error minimized by Algorithm 6 is $E(S, r)$ subject to the additional constraint $|S| = k$. From the first equality it is clear that Line 4 minimizes $E(S, r)$ with respect to V when S is fixed. From the second equality it is clear that lines 5-7 minimize $E(S, r)$ with respect to the selection in S when V is fixed. This shows that $E(S, r)$ cannot increase during the iterations. It cannot decrease forever since the number of size-k subsets is finite. Therefore the algorithm converges to a local minimum. ■

CHAPTER 8

A LOOKAHEAD ALGORITHM FOR ROBUST SUBSPACE RECOVERY

Authors – Guihong Wan, Haim Schweitzer

Department of Computer Science, EC 31

The University of Texas at Dallas

800 West Campbell Road

Richardson, Texas 75080-3021

The preliminary results are published as a student abstract in Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021.

The full version is submitted to the 21st IEEE International Conference on Data Mining, ICDM 2021.

Abstract

A common task in the analysis of experimental data is to compute an approximate embedding of the data in a low dimensional subspace. The standard algorithm for computing this subspace is the well-known Principal Component Analysis (PCA). The PCA can be extended to the case where some data points are viewed as "outliers" that can be ignored, allowing the remaining data points ("inliers") to be more tightly embedded. We develop a new algorithm that detects outliers so that they can be removed prior to applying the PCA. Such algorithms are typically called "Robust PCA" or "Robust Subspace Recovery" algorithms. The main idea is to rank each point by looking ahead and evaluating the change in the global PCA error if that point is to be considered an outlier. Our technical contribution is showing that this lookahead procedure can be implemented efficiently, producing an accurate algorithm with running time not much above the running time of standard PCA algorithms. Extensive experiments show that the proposed lookahead algorithm is much more accurate than the current state-of-the-art algorithms.

8.1 Introduction

Principal Component Analysis (PCA) is a common technique for embedding data in a low dimensional subspace. See, e.g., (Jolliffe, 2002; Burges, 2010; Cabral et al., 2013; Vidal et al., 2016; Gray, 2017). Until recently the task of computing PCA on large data was considered a challenge, but recent algorithmic developments that use randomization enable effective computation of PCA even for very large datasets. See, e.g., (Halko et al., 2011; Musco and Musco, 2015; Li et al., 2017; Clarkson and Woodruff, 2017).

Unfortunately, the subspace computed by PCA is known to be sensitive to outliers. It is also known that it can be made significantly more accurate if the algorithm is allowed to ignore a fraction of the data, considered to be outliers. Algorithms for outlier detection and

removal for PCA are sometimes called "Robust Principal Component Analysis (RPCA)", or, alternatively, "Robust Subspace Recovery (RSR)". It is known that the problem is NP-hard (e.g., (Hardt and Moitra, 2013)). For recent surveys and overviews that discuss many variants of these robust algorithms see (Vaswani and Narayanamurthy, 2018; Vaswani and Narayanamurthy, 2018; Lerman and Maunu, 2018c). The reference (Vaswani and Narayanamurthy, 2018) reviews techniques that consider outliers as partially corrupt observations, while the reference (Lerman and Maunu, 2018c) reviews techniques that consider each point as either an inlier or an outlier. In this paper we follow the same interpretation of outliers as in the reference (Lerman and Maunu, 2018c).

There is a large number of studies on outlier detection and removal specifically for computing robust PCA. See, e.g., (Lerman and Maunu, 2018c; Rousseeuw and Hubert, 2018). They can be categorized into several groups.

Robust Estimator. One way to obtain robust PCA is to replace the classical covariance or correlation matrix by a robust estimator (e.g., (Croux and Haesbroeck, 2000; Maronna et al., 2006; Zhang and Lerman, 2014a; Zhang, 2016a)).

Projection Pursuit. The projection pursuit techniques attempt to find directions sequentially in which each direction should be orthogonal to previously found directions and maximize a robust scale function. See, e.g., (Huber, 2004; Hubert and Engelen, 2004; Croux et al., 2007; Kwak, 2008; McCoy et al., 2011).

Least Absolute Deviations. The robustness is achieved by replacing the least squares formulation with the least absolute deviations. The absolute formulation has several attractive features, such as rotational invariance. However, it is NP-hard (Clarkson and Woodruff, 2015). Many methods have been developed to approximate the solution. See e.g., (McCoy et al., 2011; Xu et al., 2012a; Li and Haupt, 2015; Clarkson and Woodruff, 2015; Cherapanamjeri et al., 2017; Lerman and Maunu, 2018b).

Algorithm 7: The general framework of "Filtering Outliers" RPCA/RSR algorithms.

Input:

X: a data matrix of n data points.

r: the desired rank of the subspace.

k: an upper bound on the number of outliers.

Output:

V: a low-dimensional subspace.

μ: the column mean of the inliers.

1. Rank data points based on some measures;
$S \leftarrow k$ data points with lowest scores.

2. $V, \mu \leftarrow$ compute V and μ using the standard PCA algorithm for data points not in S.

Filtering Outliers. An important method to solve the robust subspace recovery problem is to first filter outliers and then fit a subspace to the remaining data by using classical PCA. Our algorithm belongs to this group.

For the general framework of "Filtering Outliers" robust subspace recovery algorithms, see Algorithm 7. It comes with two parameters: r and k. The desired rank of the low subspace r is chosen based on applications. For example, images of an individual's face with fixed pose under varying lighting conditions are known to lie in a subspace with rank at most 9 (see, e.g. (Basri and Jacobs, 2003)). This suggests $r = 9$ for that application. The upper bound on the number of outliers, denoted by k, can be bigger than the true number of outliers, but the remaining data points should be sufficiently big for the rank r estimate. We explicitly consider the inlier data mean as part of the recovered PCA model. Many previously proposed robust subspace algorithms (e.g., (Zhang et al., 2015a; Shah et al., 2018)) perform initial centering of the data, but do not update the center of the recovered inliers accordingly when the outliers are changed during the run of the algorithm. As we show experimentally this may lead to inaccurate outlier detection and wrong subspace recovery.

The key challenge of the the filtering approach to the RPCA/RSR problem is the detection of the outliers. Many studies view the data as coming from a fixed distribution, and

attempt to detect outliers as data points at the margins of the distribution (e.g., (Roberts, 1999; Scheirer et al., 2011; Hubert and Engelen, 2004; Xu et al., 2012b; Zhang et al., 2015a)). Another common approach is ranking, which associates a value with each point indicating how likely it is to be an outlier. These values are then used to identify outliers. For example, the value can be the reconstruction error as defined in (8.4), or how likely the point is to lie on a low-dimensional subspace shared with other points. See, e.g., (Lei Xu and Yuille, 1995; Chen and Lerman, 2009; Hubert and Engelen, 2004; Soltanolkotabi et al., 2012; You et al., 2017; Rahmani and Atia, 2017; Rahmani and Li, 2019b). Another approach that uses randomization is described in (Zhang et al., 2015b), where at each iteration a point is removed with probability proportional to its variance in the current direction. A deterministic version of the algorithm was developed in (Feng et al., 2012). In (Shah et al., 2018), the authors view outlier detection as a search problem, and apply a combinatorial search algorithm to solve it. The result is more accurate than other algorithms (with proper choice of parameters it is optimal), but the running time makes it impractical even for medium size datasets.

Our Approach

As in other studies that use ranking to filter outliers, the fundamental part in our approach is to assign a value to each point, indicating how likely it is to be an outlier. We differ from other algorithms by the method in which this value is computed. A natural measure is the distance of a point from the PCA model. This distance corresponds to the "reconstruction error" of that point (to be defined in Section 8.2), a common criterion for evaluating outliers. However, as we show next, the reconstruction error may not be a good choice in situations where the PCA model is inaccurate.

Consider the 5 points in the left panel of Figure 8.1, where the initial rank-1 PCA that was computed from all 5 points is shown as a green arrow. Even though we expect point 1 to be detected as an outlier, its reconstruction error (distance from the line) is 0.1, the smallest

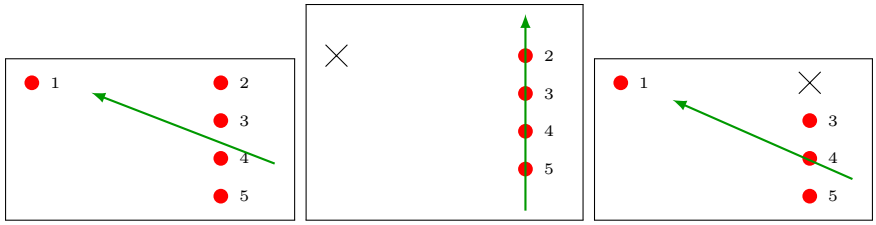

Figure 8.1: The lookahead idea. Left panel: rank-1 PCA of the entire data. Middle panel: rank-1 PCA of the data without point 1, the PCA error is 0. Right panel: rank-1 PCA of the data without point 2, the PCA error is 1.04.

among all 5 points. The largest reconstruction error is 2.2, for point 2. If the reconstruction error is used as the criterion for selecting outliers, point 2 will be detected as an outlier, which may not be desired. The approach we propose is to replace the search for a point that has large reconstruction error with the search for a point that, when removed, reduces the PCA error computed from all the remaining points most. In the middle panel of Figure 8.1, we show the PCA obtained if point 1 is viewed as an outlier and removed. In this case the reconstruction errors of all other points are 0, so that the PCA model error is 0. In the right panel, we show the PCA obtained if point 2 is removed. The PCA model error computed as the summation of the reconstruction errors of inliers is 1.04. Therefore, point 1 should be more likely to be an outlier than point 2, or any other point.

Clearly, performing this lookahead procedure for each point and evaluating its outlier likelihood as the PCA model error obtained "if" it is selected as an outlier is more powerful than relying on the immediate reconstruction error, but it appears to be much more expensive. Our technical contribution is showing that these lookahead errors can be computed very efficiently by exploiting a special matrix structure. Specifically, we show that the desired errors can be computed from certain eigenvalues, and that these eigenvalues can be computed efficiently by rank-1 modification of centered matrices.

Main Contributions

Our main contributions are summarized below:

- An algorithm for fast computation of eigenvalues of centered matrices by rank-1 modification.
- An algorithm for robust subspace recovery that uses the above fast computation of eigenvalues.
- Experimental evaluation with the above algorithm showing it to be typically more accurate than the current state of the art, with similar running time.

Paper Organization

The errors of modeling data by PCA and robust PCA are defined in Section 8.2. In particular, it is shown that the PCA error can be calculated as the sum of certain eigenvalues. Our lookahead algorithm is described in sections 8.3 and 8.5. The key idea for reducing the complexity of our approach is to exploit rank-1 updates of centered matrices. In Section 8.4 we prove that this relation exists. Experimental results are described in Section 8.6.

8.2 PCA Variants and Their Errors

8.2.1 Standard PCA

Let $X = (x_1 \dots x_n)$ be the data matrix of size $m \times n$, where n is the number of data points and m is the dimension of each point. For a given rank $r \leq \min\{m, n\}$, the PCA computes a linear mapping from the m dimensional x_i to the r dimensional w_i. The mapping is specified in terms of the matrix V with r orthogonal columns and the mean μ:

$$w_i = V^T(x_i - \mu), \quad x_i \approx V w_i + \mu, \quad \mu = \frac{1}{n}\sum_i x_i. \tag{8.1}$$

The reconstruction error for a point x_i, and the PCA model error (the reconstruction error for the entire data matrix X), are defined as follows:

Algorithm 8: The standard PCA algorithm.
Input:
X: a data matrix of size $m \times n$.
r: the desired rank of the subspace.
Output: V and μ.
1. $\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$.
2. $C = \sum_{i=1}^{n}(x_i - \mu)(x_i - \mu)^T$.
3. Compute the eigendecomposition of C, and create V from the r eigenvectors with the largest eigenvalues.

$$e_i = \|(x_i - \mu) - Vw_i\|^2,\ E(X) = \sum_{i=1}^{n} e_i. \tag{8.2}$$

The goal of the standard PCA is to compute V such that the PCA error $E(X)$ is minimized. It is known (e.g., (Jolliffe, 2002)) that the matrix V which minimizes $E(X)$ has as its columns the r eigenvectors corresponding to the r largest eigenvalues of the scaled covariance matrix C. This shows that the PCA can be computed by the simple algorithm shown in Algorithm 8. A straightforward implementation of this PCA algorithm has a running time of $O(m^2 n)$, where the dominant part is the computation of C in Step 2. Current state-of-the-art algorithms such as (Halko et al., 2011; Musco and Musco, 2015; Li et al., 2017; Clarkson and Woodruff, 2017) can reduce the running time to $O(rmn)$.

Our results rely heavily on a simple relationship between the PCA model error and the eigenvalues of C that we state as a theorem below.

Theorem 13. Let $E(X)$ be the PCA error for the data matrix X as defined in (8.2), and let C be the scaled covariance matrix. Let $\lambda_1(C) \geq \cdots \geq \lambda_m(C)$ be the eigenvalues of C. Then:

$$E(X) = \sum_{i=r+1}^{m} \lambda_i(C) = \text{trace}\{C\} - \sum_{i=1}^{r} \lambda_i(C). \tag{8.3}$$

Proof: Let X_c be the "centered" matrix obtained from X by subtracting μ from each column. Then: $C = X_c X_c^T$. Using the relationship between the Frobenius norm and the

trace we get the following expression for the error in (8.2):

$$\begin{aligned}
E(X) &= \|X_c - VV^T X_c\|_F^2 = \|(I - VV^T)X_c\|_F^2 \\
&= \text{trace}\{(I - VV^T)X_c X_c^T (I - VV^T)\} \\
&= \text{trace}\{(I - VV^T)C(I - VV^T)\} \\
&= \text{trace}\{C - VV^T C - CVV^T + VV^T CVV^T\} \\
&= \text{trace}\{C\} - \text{trace}\{VV^T C\} - \text{trace}\{CVV^T\} \\
&\quad + \text{trace}\{VV^T CVV^T\} \\
&= \text{trace}\{C\} - \text{trace}\{V^T CV\} \qquad \text{(note1)} \\
&= \text{trace}\{C\} - \sum_{i=1}^{r} \lambda_i(C) \\
&= \sum_{i=r+1}^{n} \lambda_i(C) \qquad \text{(note2)}
\end{aligned}$$

The simplification in the line marked with (note1) was obtained by using the cyclic invariant property of the trace which implies: trace$\{VV^T C\}$, trace$\{CVV^T\}$, and trace$\{VV^T CVV^T\}$, are all equal to trace$\{V^T CV\}$. The simplification in the line marked with (note2) was obtained using the fact that trace$\{C\} = \sum_{i=1}^{n} \lambda_i(C)$. ■

8.2.2 Robust PCA / Robust Subspace Recovery

Let S be the outlier subset, containing indices in the range $[1, n]$. The analogous formulas to (8.1) and (8.2) for the inliers (data points that are not in S) are:

$$\begin{aligned}
&\mu = \frac{1}{n - |S|} \sum_{i \notin S} x_i, \quad w_i = V^T(x_i - \mu), \\
&e_i = \|(x_i - \mu) - V w_i\|^2, \quad E(X) = \sum_{i \notin S} e_i.
\end{aligned} \tag{8.4}$$

Observe that it is easy to find the minimizer V for $E(X)$ when S is given, since it can be computed as the standard PCA of the inliers. It is also easy to compute S when V, μ

Algorithm 9: The KMeans-style RSR algorithm.
As a formula: $\{V, \mu, S\} = \mathrm{KM}(S, X, r)$.
Input:
S: the current subset of j outliers.
X: the $m \times n$ data matrix.
r: the desired number of principal components.
Output: V, μ, and a new subset S of j outliers.
Initialization: Set current error value to infinity.

1 **repeat**
2 Set old error value to the value of current error.
3 Compute V, μ as the rank-r PCA of the inliers.
4 Compute $e_i = ||(x_i - \mu) - VV^T(x_i - \mu)||^2$ for each column of X.
5 Replace the columns of S by the j columns of X with the largest e_i.
6 Set new error value = $\sum_{x_i \notin S} e_i$.
7 **until** *new error value = old error value*;

and the number of outliers are given. Suppose k is the number of outliers, then S is the index set of the k data points with largest e_i. Alternating these two steps in a "k-means" style converges quickly to a local minimum which unfortunately can be much worse than the global minimum. This algorithm, that we call the KMeans-style robust subspace recovery algorithm, is shown in Algorithm 9. A similar algorithm would get as input V, μ, k instead of S, and would start with Line 4 of Algorithm 9. The algorithm typically runs a small number of iterations (2-3) and its accuracy is not competitive with top robust subspace recovery algorithms.

8.2.3 Correctness Proof for the KMeans-style RSR Algorithm

The goal of the robust subspace recovery algorithm is to detect k outliers and to compute the subspace V, which is constrained to be of rank r with orthogonal columns. The robust

PCA error can be rewritten as follows:

$$E(X) = \min_{S,V} \sum_{i \notin S} e_i(V),$$

$$\text{where } e_i(V) = ||(x_i - \mu) - VV^T(x_i - \mu)||^2,$$

$$\text{and } \mu = \frac{1}{n - |S|} \sum_{i \notin S} x_i.$$

Thus, the error to be minimized is $E(X)$ subject to the additional constraint $|S|=k$. Clearly, Line 3 of Algorithm 9 minimizes $E(X)$ with respect to V when S is fixed, and lines 4-6 minimize $E(X)$ with respect to S when V is fixed. This shows that $E(X)$ cannot increase during the iterations. The algorithm cannot decrease the error forever, since the number of size-k subsets is finite. Therefore, the algorithm converges to a local minimum. ■

8.3 The Proposed Lookahead Algorithm

The lookahead algorithm can be considered as a filtering outlier algorithm according to the characterization in Section 8.1. We begin by describing it without specifying the filter values associated with data points. This is followed by specifying the filter values. An algorithm for computing the filter values efficiently is described in Section 8.5.

8.3.1 Top View of the Lookahead Algorithm

Let S_1, S_2 be two outlier sets such that $|S_1| = |S_2|$. (Both sets have the same number of outliers.) Suppose we have access to a function f such that:

$$f(S_1, X, r) \ > \ f(S_2, X, r) \quad \Rightarrow \quad S_1 \text{ “appears to be” better than } S_2.$$

Using f we propose an iterative algorithm for computing k outliers, with the details shown in Algorithm 10. In each iteration the algorithm runs the update algorithm for detecting more outliers and the KMeans-style RSR (Algorithm 9) for improving the current outlier detection.

Algorithm 10: Top view of the lookahead algorithm
Input:
X, the data matrix.
r, the desired PCA rank.
k, the desired number of outliers.
α, a ratio parameter satisfying: $0 \leq \alpha \leq 1$.
Output: V, μ, and a subset S of k outliers.
Initialization: $S = \emptyset$ (the empty set).
Updates: while $|S| < k$ do:
$S = \text{Update}(S, k, \alpha, X, r)$.
$\{V, \mu, S\} = \text{KM}(S, X, r)$.

Algorithm 11: The lookahead update algorithm.
As a formula: $S_{\text{output}} = \text{Update}(S, k, \alpha, X, r)$.
Input:
S, an outlier subset of j outliers. ($n-j$ inliers.)
k, the desired number of outliers.
α, a ratio parameter $0 \leq \alpha \leq 1$.
X: the $m \times n$ data matrix.
r: the desired PCA rank.
Output: A new outlier subset which contains more outliers than the input outlier subset.
1. For $i = 1...n-j$, create the child subset S_i by adding the inlier i to S, and compute $f_i = f(S_i, X, r)$.
2. Compute c from α by the following formula:

$$c = \alpha(k-j-1)+1 \tag{8.5}$$

3. Observe that each child subset is of size $j+1$.
Return the union of the c children with the largest f_i.

The update algorithm is shown as Algorithm 11. The input outlier subset S is of size j. Then there are $n-j$ inliers. For each inlier x_i, we compute the filter value f_i if that inlier is added into S as an additional outlier.

The formula (8.5) in Algorithm 11 is designed so that $c=1$ for $\alpha=0$, and $c=k-j$ for $\alpha=1$. Observe that $k-j$ is the remaining number of outliers that still need to be detected. Intermediate values of α give a linearly scaled value of c between these two extremes. The value of α affects both the accuracy and the running time of the algorithm. Increasing α

would in general result in a reduction in accuracy and a faster running time. The most accurate choice is α=0, which gives c=1. This requires k updates and $O(nk)$ filter value evaluations. The fastest choice is α=1, which gives c=k−j. This requires 1 update and $O(n)$ filter value evaluations. For a constant value of α, the number of updates can be shown to be $O(\log k)$, and the number of filter value evaluations is $O(n \log k)$.

The outlier subset returned by the algorithm is of size $j + c$, which is more than the size of the input outlier subset.

8.3.2 Correctness of Algorithm 10

We first observe that the algorithm terminates because the "Update" step always returns a bigger outlier set than its input outlier set, and the "KM" step does not change the number of outliers. To see that at termination $|S|$=k, observe that the formula (8.5) guarantees $|S| \leq k$. Therefore, iteratively increasing $|S|$ would guarantee that eventually $|S|$=k.

8.3.3 The Filter Values

A natural choice for f_i is the reconstruction error e_i as given in (8.4). However, as shown in the example in Figure 8.1, it may be very inaccurate. Instead, we propose to use the PCA model error after the point is removed. We refer to this error as the lookahead error.

Suppose the outlier set S is known. The lookahead error of an inlier point x_i with respect to S is the PCA error E, as defined in (8.4), of the child outlier set S_i, obtained by adding x_i to S, then:

$$f_i = f(S_i, X, r) = E(Z_i),$$

where the columns of Z_i are those in X but not in S_i, and $E(Z_i)$ is defined in (8.4). Notice that $E(Z_i)$ is computed from eigenvalues of the scaled covariance matrix. That is, the mean should be updated accordingly when the ouliers are changed.

A direct computation of the lookahead error as defined above requires that a standard PCA algorithm is applied to the inliers for each S_i. This is clearly impractical. We proceed to show how to calculate lookahead errors efficiently. That part is our main technical result.

8.4 Rank-1 Modifications

The efficient calculation of lookahead values by our algorithm is based on efficient eigenvalue estimation for the scaled covariance matrix. It relies on a rank-1 modification relation between the scaled covariance matrix at the parent set level and the scaled covariance matrix at its child set level. In this section we establish this relation.

The lookahead values needed for the update algorithm in Algorithm 11 require the PCA error of each child set S_i. Let $p = n - |S|$ be the number of the corresponding inliers at the parent set S. We use the following notations:

Z : the $m \times p$ matrix of the inliers. $Z = X \setminus S$.

B : $= ZZ^T$.

μ : the column mean of Z.

Z^c : the "centered" Z. μ is subtracted from columns.

C : $= Z^c(Z^c)^T$.

Suppose a child inlier set is constructed by removing x_i from Z. (Equivalently, x_i is added into S to obtain S_i.)

Z_i : the $m \times (p-1)$ matrix of the remaining inliers.

B_i : $= Z_i Z_i^T$.

μ_i : the column mean of Z_i.

Z_i^c : the "centered" Z_i. μ_i is subtracted from columns.

C_i : $= Z_i^c(Z_i^c)^T$.

We proceed to show that C_i can be obtained as a rank-1 modification of C. To the best of our knowledge, this was not previously observed.

Theorem 14. Define: $y_i = x_i - \mu$, and $\beta = \frac{p}{p-1}$. Then:

$$C_i = C - \beta y_i y_i^T. \tag{8.6}$$

Proof: The following relations are known:

$$1: \quad C = B - p\mu\mu^T, \quad 2: \quad C_i = B_i - (p-1)\mu_i\mu_i^T,$$
$$3: \quad B_i = B - x_i x_i^T, \quad 4: \quad \mu_i = \frac{p\mu - x_i}{p-1}.$$

For the relations 1 and 2, see, e.g., (Cadima and Jolliffe, 2009). The relations 3 and 4 are easy to verify.

From 1, we get: $B = C + p\mu\mu^T$. Substituting this value of B in 3 gives: $B_i = C + p\mu\mu^T - x_i x_i^T$. Substituting this value of B_i in 2 gives: $C_i = C + p\mu\mu^T - x_i x_i^T - (p-1)\mu_i\mu_i^T$. Substituting the value of μ_i from 4 into this equation gives:

$$\begin{aligned} C_i &= C + p\mu\mu^T - x_i x_i^T - (p-1)\frac{p\mu - x_i}{p-1}\frac{(p\mu - x_i)^T}{p-1} \\ &= C + \frac{p(p-1)\mu\mu^T - (p-1)x_i x_i^T - (p\mu - x_i)(p\mu - x_i)^T}{p-1} \\ &= C - \frac{p}{p-1}(x_i x_i^T + \mu\mu^T - \mu x_i^T - x_i\mu^T) \\ &= C - \beta y_i y_i^T. \end{aligned}$$

This expression gives the desired relation. ■

A similar result relates the traces of C and C_i:

Corollary (of Theorem 14):

$$\text{trace}\{C_i\} = \text{trace}\{C\} - \beta\|y_i\|^2 \tag{8.7}$$

Proof: Apply trace to both sides of (8.6) and observe that $\text{trace}\{y_i y_i^T\} = ||y_i||^2$. ■

8.5 Complexity of the Lookahead Algorithm

To complete the description of the lookahead algorithm as given in Section 8.3, we need to describe how to compute the filter values. We show that the complexity of computing filter values for $O(n)$ inliers is $O(rdn)$, where $d \leq \min\{m, n\}$ is the rank of the data matrix, and $r \leq d$ is the desired rank of the low dimensional subspace.

The computational tool that we use here is the well-known technique for computing eigenvalues of a matrix perturbed by rank-1 modification. See, e.g., (Bunch et al., 1978). It can be stated as follows:

Rank-1 modification: Let A be a symmetric matrix of rank d with a known eigendecomposition. Let y be a vector, and $\beta \geq 0$ a scalar. Suppose $\tilde{A} = A - \beta yy^T$. Then there is an algorithm that computes each eigenvalue of $\tilde{A}$ in $O(d)$ cost.

Using the same notation as in Section 8.4, the filter value that needs to be calculated is the PCA error for Z_i^c corresponding to each inlier x_i. From Theorem 13 it follows that this PCA error can be calculated from the eigenvalues of C_i. Each PCA error can be computed from the r largest eigenvalues (and a trace) where r is the desired PCA rank. Applying Theorem 14 and its corollary it follows that the cost of computing these r eigenvalues is $O(rd)$.

To further elaborate on the eigendecompositions performed by the lookahead algorithm, we discuss the three locations where eigenvectors and/or eigenvalues are calculated.

1. Root factorization of the data matrix X of size $m \times n$. Using current state-of-the-art algorithms (e.g., (Halko et al., 2011)), this factorization can be computed with d, a user specified rank parameter, and its running time is $O(dmn)$. If d is not specified, it is taken as $\min\{m, n\}$. We view this as an initial dimensionality reduction step.

2. Factorization at each parent subset S. At each parent level, we need the eigendecomposition computed from the corresponding inlier subset. Using the root factorization, this can be done in $O(kdm)$.

Table 8.1: Complexity of the lookahead algorithm.

α value	complexity
0	$O(dmn + rdnk)$
1	$O(dmn)$
$0 < \alpha < 1$	$O(dmn + rdn \log k)$

3. Factorization for each child subset S_i. As discussed in the beginning of this section, this takes $O(rd)$. For all inlier child subsets, it takes $O(rdn)$.

Let T be the number of parents that the lookahead algorithm expands. The total number of children becomes $O(Tn)$, and the total running time complexity of the lookahead algorithm is: $O(dmn + Tkdm + Trdn)$. It is the same as $O(dmn + Trdn)$ assuming n is very big. In Table 8.1, we show these results for the three choices of the ratio parameter α that were analyzed in Section 8.3.1.

The running time of the KMeans-style RSR algorithm (Algorithm 9) is $O(trdn)$, where t is the number of iterations for the algorithm to converge. Even though t is typically very small (2-3 in our experiments) we make sure the algorithm runs no more than 5 iterations. Thus, using it adds to the overall complexity a term of $O(Trdn)$ which can be ignored.

8.6 Experimental Results

The experiments that we describe follow closely the methodology used in the recent overview of the field (Lerman and Maunu, 2018c). We refer to this reference as the **overview**. The **overview** describes two artificial datasets to be used for comparing robust subspace recovery algorithms. We experimented with these datasets and added experiments with some standard datasets that are being used for evaluating machine learning algorithms. Experiments were performed on an iMac with Processor 4GHZ Quad-Core Inter i7 and Memory 32 GB.

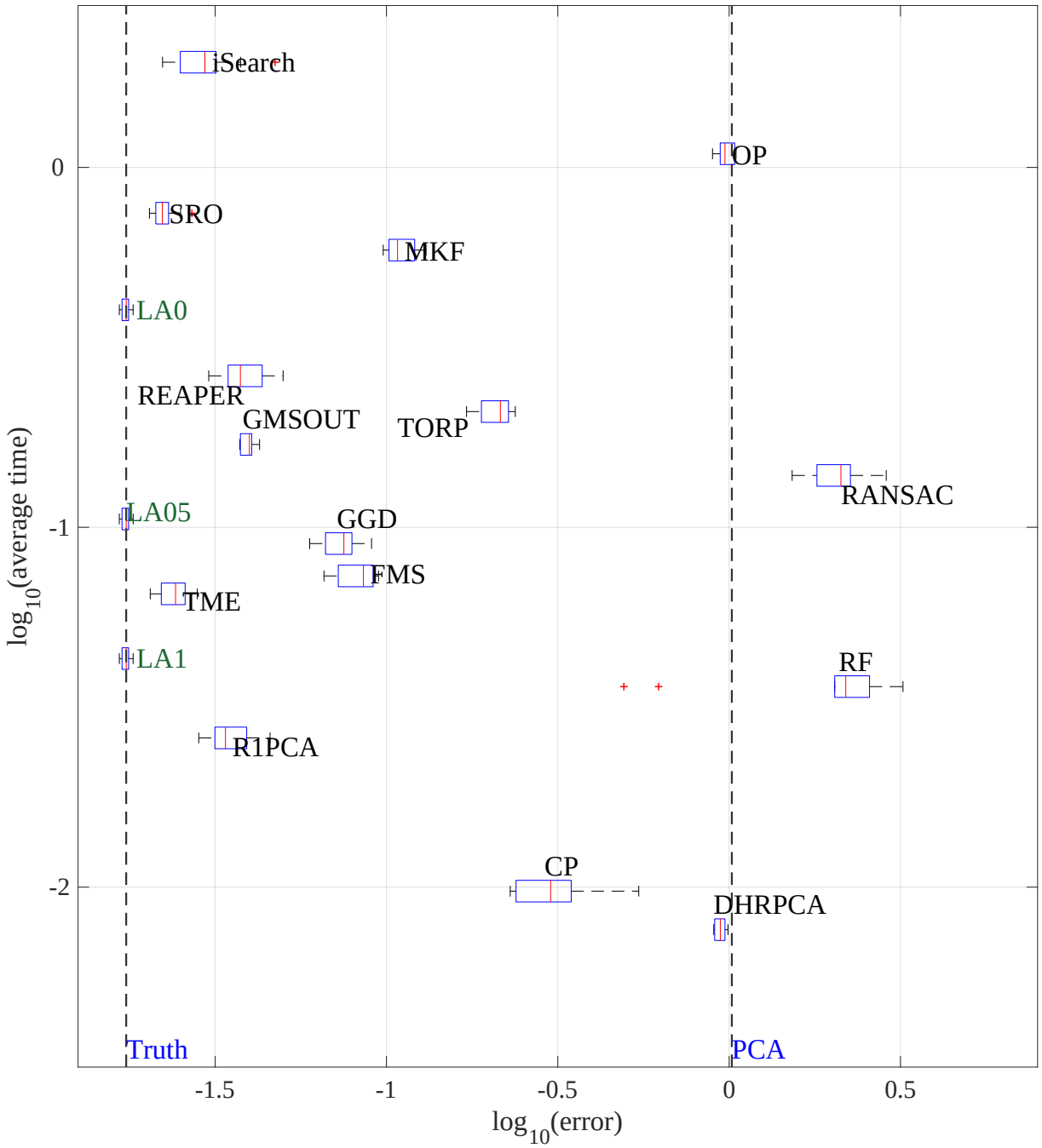

Figure 8.2: The boxplot diagram of the results for the Haystack experiments. Here the outlier fraction is 0.2, and the outlier mean is **0** (in all coordinates). Time values were averaged over 10 runs. The red line in the box is the median error over the 10 runs. The algorithm is more accurate when the box is small and close to the vertical line for Truth. The algorithm is faster when the box is closer to the bottom.

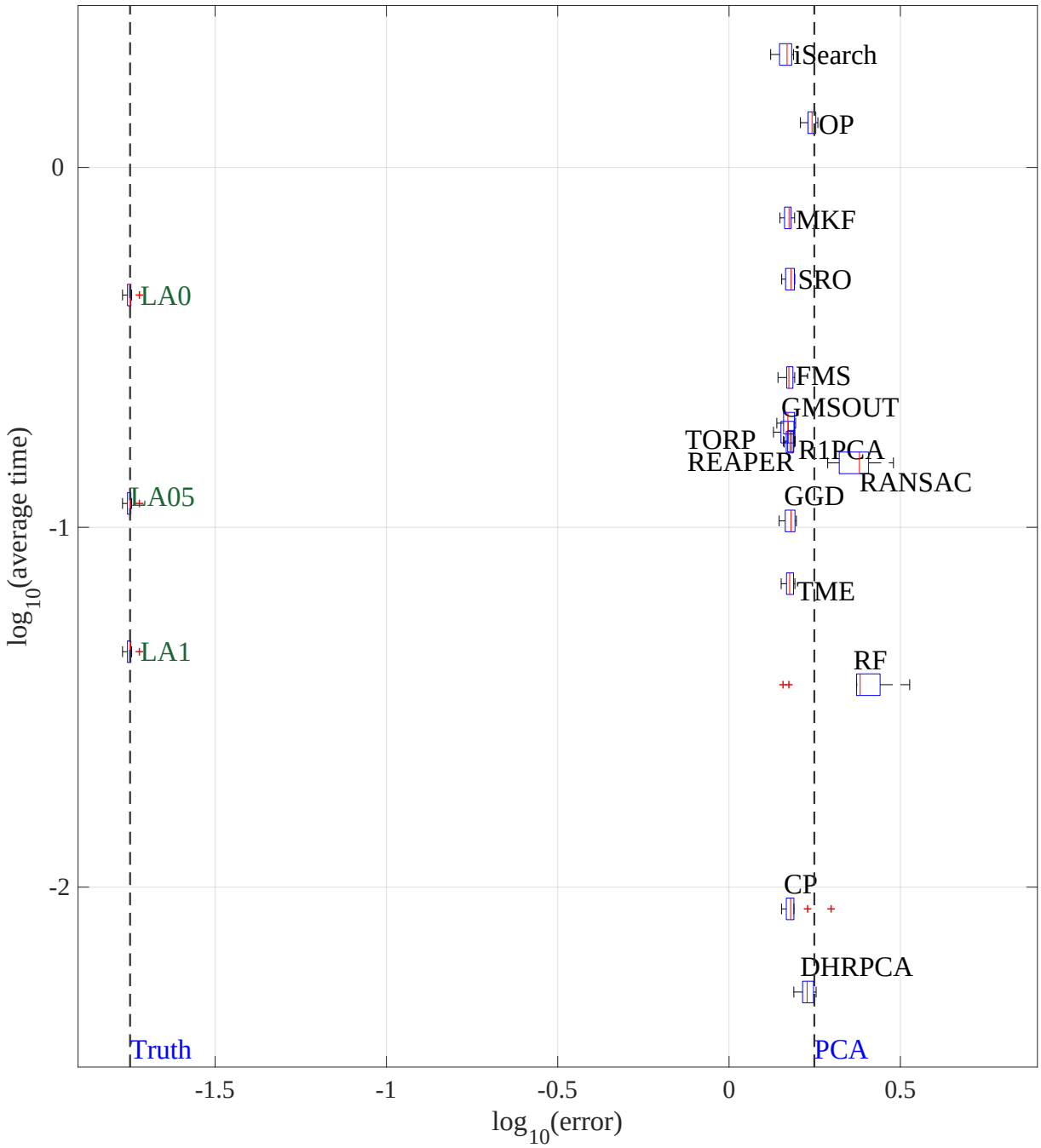

Figure 8.3: The boxplot diagram of the results for the Haystack experiments. Here the outlier fraction is 0.2, and the outlier mean is **0.1** (in all coordinates). Time values were averaged over 10 runs. The algorithm is more accurate when the box is small and close to the vertical line for Truth. The algorithm is faster when the box is closer to the bottom. There is a significant deterioration in accuracy of other algorithms.

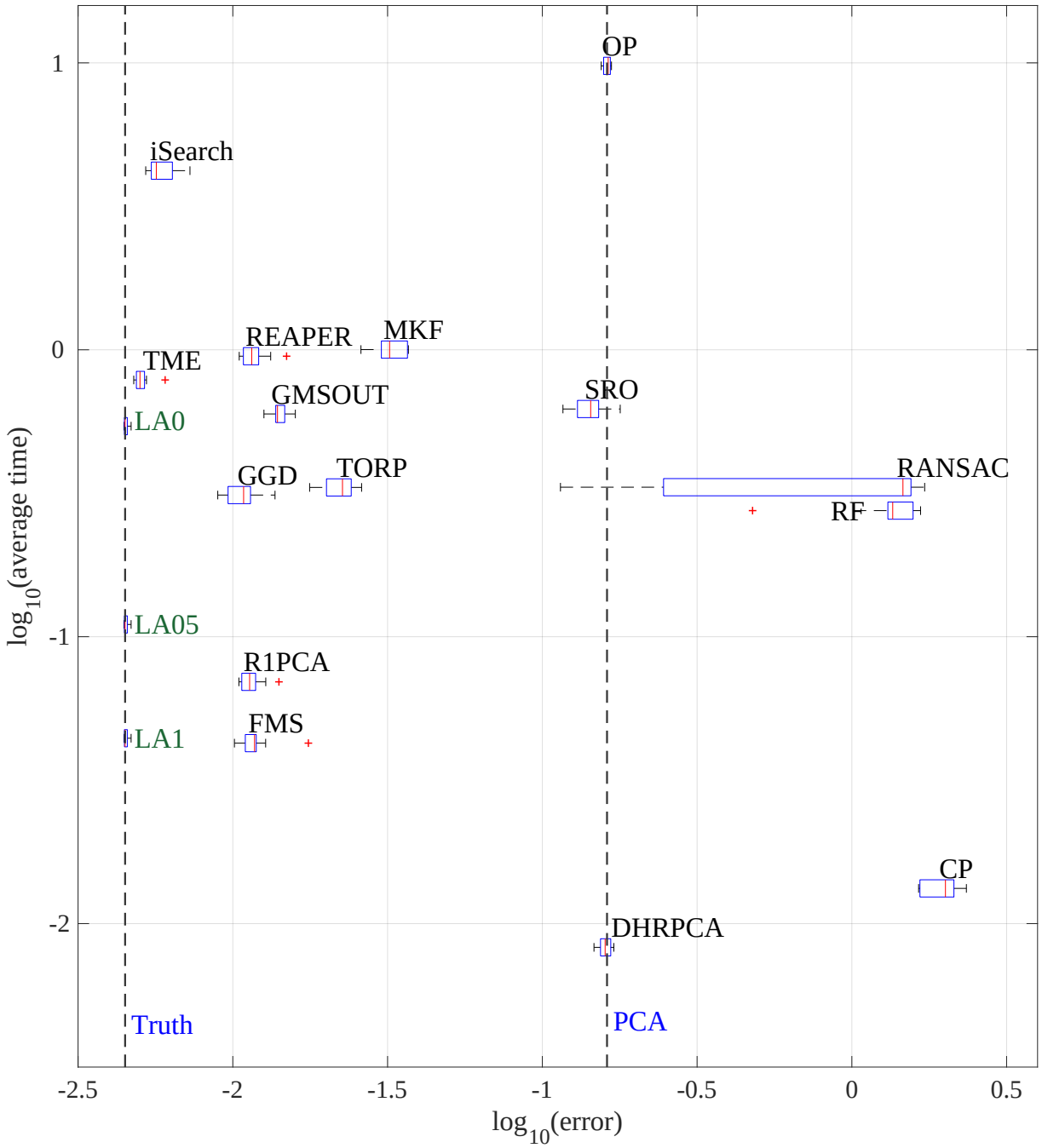

Figure 8.4: The boxplot diagram of the results for the Blurryface experiments. Outlier fraction: 0.2. The running time is averaged over 10 runs for each algorithm.

8.6.1 The Competitors

When describing the results we refer to our lookahead algorithm with the initials LA followed by the α value. Thus, LA0 means the algorithm with α=0, LA05 means the algorithm with α=0.5, and LA1 means the algorithm with α=1. The comparison was with other algorithms that made their code available and are listed below: SRO (You et al., 2017), iSearch (Rahmani and Li, 2019b), GGD (Maunu et al., 2019b), FMS (Lerman and Maunu, 2018a), CP (Rahmani and Atia, 2017), TME (Zhang, 2016b), OP (Xu et al., 2012b), MKF (Zhang et al., 2009), DHRPCA (Feng et al., 2012), REAPER (Lerman et al., 2015), TORP (Cherapanamjeri et al., 2017), R1PCA (Ding et al., 2006), GMSOUT (Zhang and Lerman, 2014b), RANSAC (Fischler and Bolles, 1981), RF (Hardt and Moitra, 2013). Additional information about these algorithms is available in the citations above. Most of them are also discussed in the **overview**. When using the artificial datasets, we add "Truth" as the ground truth result (when available), and "PCA" for the results obtained by the (non-robust) standard PCA algorithm.

8.6.2 The Boxplot Diagrams

Our goal is to compare our lookahead algorithm to the competitors in terms of both running time and accuracy. Following **overview** we display the results as a boxplot diagram which is a standard visualization tool in descriptive statistics (Wickham and Stryjewski, 2012). The position of an algorithm in the diagram shows its accuracy as the horizontal axis coordinate, and its running time as its vertical axis coordinate. Thus, the ground truth (which is known in some of our experiments) is located at the bottom left. In addition, one would expect the location of the standard (non-robust) PCA to be at the bottom right, since it is typically faster than the robust versions but not as accurate. The results of multiple experiments of the same algorithm on randomized data are shown as a box. Thus, the smaller the box the more reliable are the results. The red line in the box is the median value.

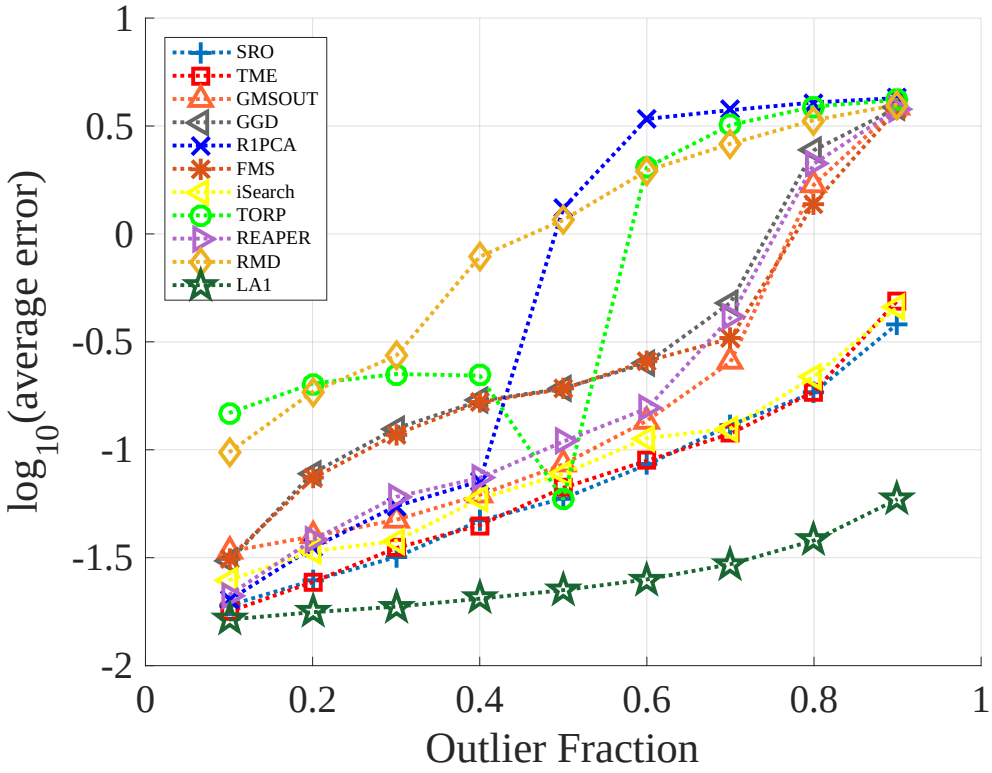

Figure 8.5: Error versus outlier fraction for the Haystack experiments. Outlier mean: **0**. The errors are averaged over 10 runs.

8.6.3 Experiments with the Haystack Model

The Haystack model is commonly used to evaluate subspace recovery algorithms. See, e.g., (Rahmani and Atia, 2017; Lerman and Maunu, 2018a,c; Rahmani and Li, 2019b). The model first selects a random subspace in r dimensions. The inliers are created by drawing from a zero mean Gaussian distribution of points on the model. The outliers are generated from a zero mean isotropic Gaussian distribution. Additional zero mean Gaussian noise is then added to both inliers and outliers.

Following **overview**, 400 points are generated in a 200 dimensional space with model rank r=10. We use the same Haystack model as in **overview** with one exception: the mean

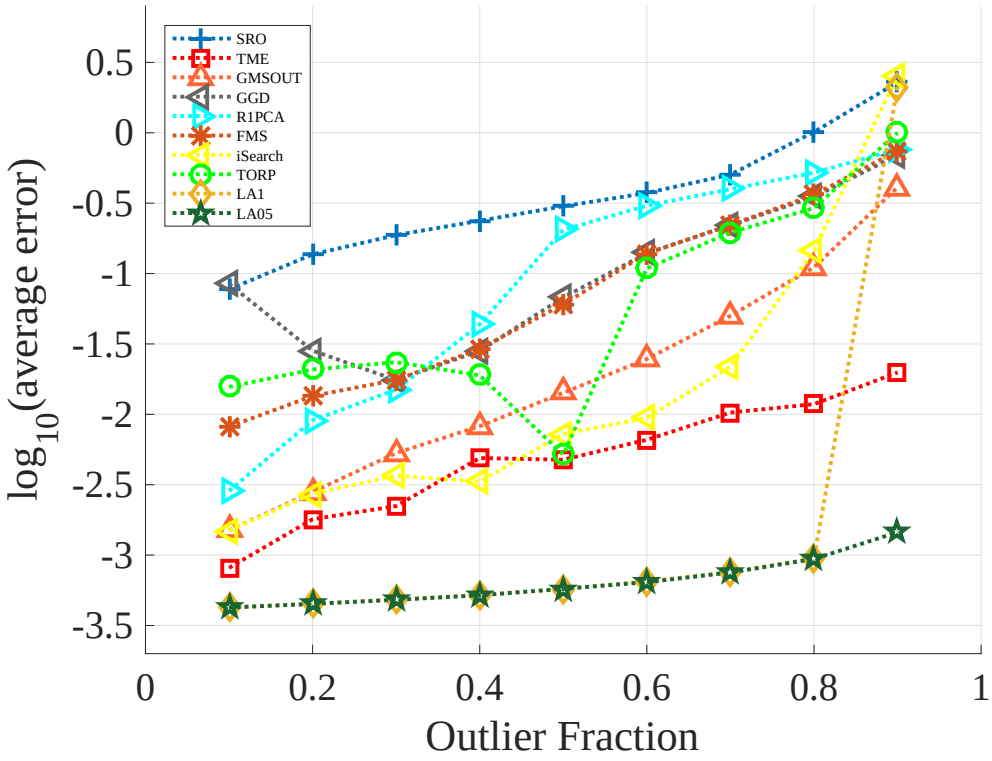

Figure 8.6: Error versus outlier fraction for Blurryface.

of outliers can be nonzero, and set to be (the same) constant in all coordinates. After all the inliers and the outliers are generated as explained above, the entire dataset is centered.

Since the ground truth is known, the algorithm performance can be measured by how accurate is the recovered subspace. Following **overview**, we take the error as the squared principal angle between the ground truth subspace and the recovered subspace.

Running the experiments we observed a huge decline in the accuracy of current state-of-the-art algorithms with the introduction of a nonzero mean outliers. The boxplot diagram of the experiments when the mean of outliers is **0** is shown in Figure 8.2. The results when the mean of outliers is **1** is shown in Figure 8.3. As explained earlier the vertical axis in the plot corresponds to speed, and the horizontal axis to accuracy. We observe that the lookahead variants are much more accurate than the other algorithms. Comparing Figure 8.2 and

Figure 8.3 (the mean of outliers changes from **0** to **1**), there is a significant deterioration in accuracy of all other algorithms that do not use lookahead. In fact, as shown in Figure 8.3 many have similar accuracy to the (non robust) standard PCA. One of the reasons is that when new outliers are detected, other algorithms do not update the mean of the remaining inliers. Since our algorithm is implemented by using the centered rank-1 modification shown in Theorem 14, when ouliers are updated, the mean of the inliers are updated accordingly. In terms of speed, as the increase of α value, the LA algorithm runs faster (closer to the bottom). LA0 is among the slowest, but LA1 is clearly above average. (The LA algorithms use dimensionality reduction with $d = 40$).

More information about how the accuracy changes with the increase of outlier percentage can be obtained from the plot in Figure 8.5. Observe the rapid increase in the error measurements which is very distinct even with the logarithmic scale of the plot. There is also an increase in the error of LA1, but its advantage over other algorithms clearly increases with the increase of outlier percentage.

8.6.4 Experiments with the Blurryface Model

The blurryface model is another artificial dataset that was proposed by **overview** to evaluate robust subspace recovery algorithms. The goal is to recover a 9-dimensional subspace corresponding to the manifold of a group of images under various illumination changes. Here the ground truth subspace is created from a particular group. The inliers are generated randomly and distributed in that subspace. The outliers are sampled from other groups. The resulting dataset has 500 points in a 400 dimensional space.

The boxplot diagram of the results for the experiments is shown in Figure 8.4. As in the previous boxplot diagram, the vertical axis corresponds to speed, and the horizontal axis to accuracy. We observe that the lookahead variants are again very accurate. while the accuracy of some other algorithms is even worse than the (non robust) standard PCA. (The LA algorithms use initial dimensionality reduction with $d = 40$).

Table 8.2: Comparison of the PCA error on real datasets. "-" indicates that no results were obtained after running for 30 minutes. "ArrayLimit" indicates that the algorithm threw out the "ArrayLimit" error.

$r:k$	$\alpha:0$	$\alpha:0.3$	$\alpha:0.5$	$\alpha:1$	iSearch	TME	SRO	FMS
Geographical Original of Music ($m=68$, $n=1,059$)								
10:200	13,567	**13,564**	**13,564**	13,569	14,206	14,164	14,454	14,216
10:700	**3,096**	3,097	3,103	3,114	4,232	3,546	3,401	3,517
YearPredictionMSD ($m=90$, $n=515,345$)								
10:200	**6.161E11**	**6.161E11**	**6.161E11**	**6.161E11**	ArrayLimit	6.210E11	-	6.208E11
10:1E4	-	**4.879E11**	**4.879E11**	**4.879E11**	ArrayLimit	5.126E11	-	5.125E11
Covtype ($m=54$, $n=581,012$)								
15:5E3	-	**292,488**	**292,488**	**292,488**	ArrayLimit	294,923	-	295,317
30:5E3	-	**31,738**	**31,738**	31,755	ArrayLimit	32,102	-	33,191

More information about how the accuracy changes with the increase of outlier percentage can be obtained from the plot in Figure 8.6. LA05 and LA1 are distinctly better than the other algorithms until the 80% point where the accuracy of LA1 deteriorates. When the ratio parameter is 1 and there are 90% outliers, LA1 cannot work well.

8.6.5 Experiments on Real Datasets

We ran additional experiments on datasets that are common in machine learning studies. Since the ground truth is not known for these datasets we report instead the PCA error as discussed in Section 8.2.

Table 8.2 shows results on several datasets that were obtained from the UCI repository. The algorithms being compared are the lookahead algorithm with several different parameters and the top performers among the other algorithms. In all tested cases, all variants of the lookahead algorithm outperform all other algorithms. As the increase of the α value, the lookahead algorithm generally produces solutions which are less accurate, but it runs faster. For example, the PCA error when α=1 is larger than the error when α=0, while when α=0 the lookahead algorithm may run for a long time.

Table 8.3: Comparison on big datasets. "NoSupport" indicates that the implementation does not support sparse data format. "-" indicates that no results were obtained after running for 40 minutes.

datasets	$r:k$		$\alpha:0.5$	$\alpha:0.8$	$\alpha:1$	iSearch	TME	SRO	FMS
Day1: sparse m=20,000 n=3,231,957 d=200	30:100	error	**106,335**	106,384	107,166	NoSupport	NoSupport	NoSupport	NoSupport
		runtime (min)	49	41	**28**				
	30:2E4	error	**482**	484	484	NoSupport	NoSupport	NoSupport	NoSupport
		runtime (min)	88	63	**32**				
Sift m=128 n=1,000,000 d=70	20:100	error (+3E10)	**3.99E8**	**3.99E8**	**3.99E8**	ArrayLimit	5.67E8	-	5.67E8
		runtime (min)	2.9	2.2	1.4		**0.7**		19
	20:5E5	error (+1E10)	**1.68E8**	**1.68E8**	**1.68E8**	ArrayLimit	9.05E8	-	9.54E8
		runtime (min)	26	13	2		**0.7**		19
CNAE m=856 n=1,080	10:100	error	**3,417**	**3,417**	3,418	4,074	3,457	3,459	3,484
		runtime (s)	27	19	**6**	123	139	22	7
	10:300	error	**2,065**	**2,065**	2,066	2,629	2,166	2,140	2,155
		runtime (s)	30	16	**6**	122	134	23	7

Table 8.3 shows results on several big datasets that are publicly available, and sometimes used in the evaluation of big data machine learning algorithms. The results indicate that there is little difference in accuracy between the the lookahead algorithm with different α values, but a huge difference in running time. The lookahead variants are clearly more accurate than the other algorithms. The TME is fast for the Sift dataset but it becomes very slow for the CNAE data where the dimension is high ($m = 856$).

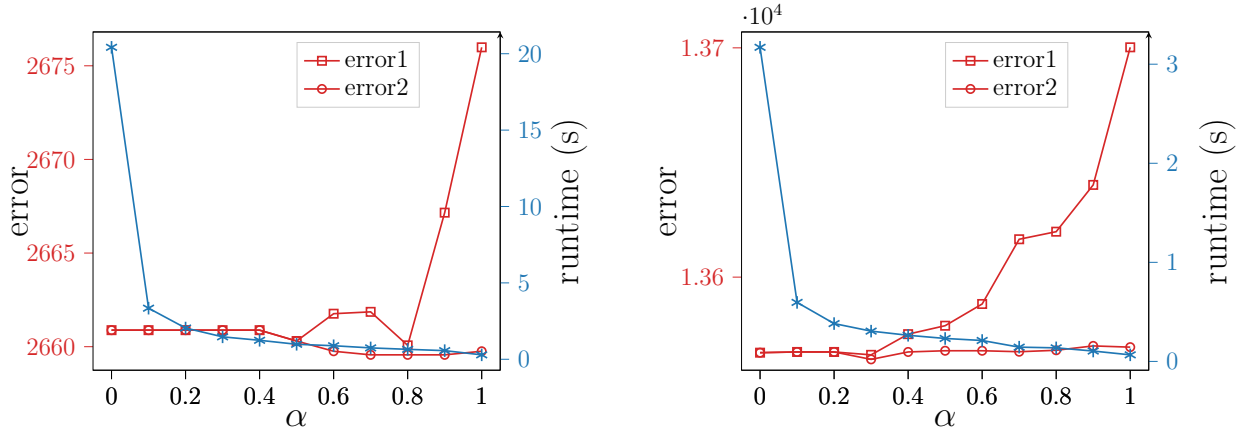

Figure 8.7: The change of error and running time against the increasing α. First plot: the CNAE dataset; Second plot: the Geographical Origins of Music dataset.

8.6.6 The Effect of the Ratio Parameter α

In this section, we investigated the effect of different values of the ratio parameter α on the accuracy and the running time. When the value of α is bigger, more outliers are selected in each iteration, and then a faster algorithm is expected.

The results are shown in Figure 8.7 for different datasets. The horizontal axis is for the values of parameter α. The left vertical axis corresponds to the errors and the right vertical axis to the running time. The values of "error1" are the PCA errors when the KMeans-style algorithm (see Algorithm 9) is not used to improve the outlier detection. The values of "error2" are the solution PCA errors given by the lookahead algorithm in Algorithm 10.

As expected, when values of the parameter α increase, the running time decreases. In particular, it drops sharply at the beginning. Observe that the solutions ("error2") remain accurate as the increase of α, and the KMeans-style algorithm improves the results especially when the values of α are big.

8.7 Concluding Remarks

PCA is known to be not robust to perturbation. Without taking outliers into consideration, the analysis can be misleading even if there is just a single outlier in the data. (See the example in Figure 8.1.) This makes RPCA/RSR techniques essential for many applications that rely on PCA.

In this paper we describe a lookahead algorithm that performs a one-step lookahead for estimating the likelihood that a point may be an outlier. Even though the idea is straightforward, it was not previously proposed. We believe that one of the reasons is that naive implementations of this idea are too expensive. The technical contribution of this paper is described in Section 8.4, where we show that the lookahead procedure can be implemented efficiently with rank-1 updates for eigenvalues. The key idea is to show that the lookahead

error can be computed from eigenvalues of the scaled covariance matrix (Theorem 13), and that these eigenvalues can be computed efficiently using rank-1 modification techniques. We believe that the key observation that certain scaled covariance matrices are related to each other by rank-1 modification (Theorem 14) is new. Extensive experimental results show that the proposed algorithm compares favorably with the current state-of-the-art algorithms.

CHAPTER 9

CONCLUSION AND FUTURE WORK

9.1 Conclusion

The works presented in this dissertation address three main problems: unsupervised column subset selection, supervised column subset selection, and outlier detection for PCA.

The first is the unsupervised column subset selection problem. When each column corresponds to a data feature, it is known as unsupervised feature selection. We study the hybrid of the unsupervised feature selection and feature extraction. This is an NP-hard problem as the unsupervised feature selection problem is NP-hard. We show that it is impossible to solve this problem by sequentially combining the solutions to feature selection and feature extraction. We introduce a combinatorial search approach to simultaneously find a given number of optimal selected features and a given number of best extracted features. Based on our knowledge this is the first optimal algorithm of the hybrid representation. We also provide faster variants of the algorithm, whose solutions come with bounds on sub-optimality.

The second is the supervised column subset selection problem where the target matrix is different from the data matrix, which is a generalization of the unsupervised variant. We study a common setting where several columns of the data matrix are selected to simultaneously approximate all the columns of another target matrix, which is an NP-hard problem. We propose two algorithms to solve this problem. The first algorithm is inspired by the (weighted) A^* algorithm. We show that with 0 assigned to a weight parameter the algorithm is guaranteed to be optimal, but its running time may be very slow. Other nonzero values give practical algorithms that are more accurate than the current state-of-the-art greedy algorithms. While there is a significant similarity between our algorithm and the classical theory of A^*, we are not aware of direct applications of A^* to the problem discussed here. The second algorithm is the Spectral Pursuit for X and Y (SPXY), which selects columns

capturing the spectral characteristics of the target matrix. What we found surprising is that it is possible to efficiently implement the algorithm with linear complexity w.r.t. the size of the two matrices. In addition to producing a solution, our algorithms give a bound on how far the solution is from the optimum. We show experimentally that our algorithms can outperform the current state-of-the-art methods.

The third problem is the outlier detection for PCA. Identifying outliers for PCA is an important problem in data analytics and machine learning. The problem is also NP-hard. We propose two algorithms to solve this problem. The first algorithm combines the "chunk recursive elimination" and the combinatorial search, similar to the classical A* algorithm. Three variants of the algorithm are described. One variant is guaranteed to produce optimal solutions. Other variants are not optimal, but compare favorably with current alternatives. We also show how to compute sub-optimality bounds for this problem. The second algorithm uses a lookahead procedure that performs a one-step lookahead for estimating the likelihood that a point may be an outlier. We show that it is possible to efficiently implement the lookahead strategy to achieve running time not much slower than the standard PCA. The key idea is to show that the lookahead error can be computed from eigenvalues, and that these eigenvalues can be computed efficiently using rank-one modification techniques. Extensive experiments demonstrate the effectiveness of the proposed algorithms.

9.2 Future Work

Anytime Variants. The main tools we used for addressing the selection problems discussed in this dissertation are the A^* and weighted A^* search algorithms. The A^* algorithm is provably optimal under certain conditions, but can be very slow. The weighted A^* algorithm is faster than the A^* algorithm, but gives non-optimal solutions. Another family of variants of the A^* search algorithm is anytime A^*. See, e.g., (Likhachev et al., 2004; Hansen and Zhou, 2007; Richter et al., 2010). The anytime A^* algorithms generate a fast,

non-optimal solution before gradually improve it. It is interesting to see whether the anytime A^* search algorithms can be applied for the column subset selection problem and the outlier detection for PCA problem.

Chunk Column Subset Selection. We successfully combined the chunking with the A^* search for the outlier detection problem in (Wan and Schweitzer, 2021a). We are interested in applying the chunking for the column subset selection problems. This application is not straightforward, since the columns in the data matrix may not be orthogonal to each other. We plan to investigate methods that use chunking for accelerating column subset selection.

Applications to Deep Learning. Some studies have introduced approaches that use column subset selection and robust PCA for various deep learning applications. For example, the unsupervised column subset selection is used to sample a small number of data points to be labeled for training neural networks (Zaeemzadeh et al., 2019; Joneidi et al., 2020). In (Lai et al., 2020), a robust subspace recovery layer is introduced to be embedded within an autoencoder for anomaly detection. We plan to study the performance of our techniques in similar applications.

REFERENCES

Aggarwal, C. C. (2016). *Outlier Analysis* (2nd ed.). Springer Publishing Company, Incorporated.

Amaldi, E. and V. Kann (1998, December). On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theoretical Computer Science 209*(1–2), 237–260.

Arai, H., C. Maung, and H. Schweitzer (2015). Optimal column subset selection by A-Star search. In *Proceedings of the 29th National Conference on Artificial Intelligence (AAAI'15)*, pp. 1079–1085.

Arai, H., C. Maung, K. Xu, and H. Schweitzer (2016). Unsupervised feature selection by heuristic search with provable bounds on suboptimality. In *Proceedings of the 30th National Conference on Artificial Intelligence (AAAI'16)*, pp. 666–672. AAAI Press.

Ayesha, S., M. K. Hanif, and R. Talib (2020). Overview and comparative study of dimensionality reduction techniques for high dimensional data. *Information Fusion 59*, 44–58.

Basri, R. and D. W. Jacobs (2003). Lambertian reflectance and linear subspaces. *IEEE transactions on pattern analysis and machine intelligence 25*(2), 218–233.

Belmerhnia, L., E.-H. Djermoune, and D. Brie (2014). Greedy methods for simultaneous sparse approximation. In *2014 22nd European Signal Processing Conference (EUSIPCO)*, pp. 1851–1855.

Bertsimas, D., A. King, and R. Mazumder (2016). Best subset selection via a modern optimization lens. *The Annals of Statistics 44*(2), 813–852.

Bertsimas, D., B. Van Parys, et al. (2020). Sparse high-dimensional regression: Exact scalable algorithms and phase transitions. *The Annals of Statistics 48*(1), 300–323.

Boutsidis, C., P. Drineas, and M. Magdon-Ismail (2013). Near-optimal coresets for least-squares regression. *IEEE Transactions on Information Theory 59*(10), 6880 – 6892.

Boutsidis, C., M. W. Mahoney, and P. Drineas (2009). An improved approximation algorithm for the column subset selection problem. In *SODA*, pp. 968–977.

Boutsidis, C., D. P. Woodruff, and P. Zhong (2016). Optimal principal component analysis in distributed and streaming models. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pp. 236–249.

Bringmann, K., P. Kolev, and D. P. Woodruff (2017). Approximation algorithms for ℓ_0-low rank approximation. In *NIPS'17*. Curran Associates, Inc.

Bunch, J. R., C. P. Nielsen, and D. C. Sorensen (1978). Rank-one modification of the symmetric eigenproblem. *Numer. Math. 31*, 31–48.

Burges, C. (2010, January). *Dimension Reduction: A Guided Tour*. Hanover, MA, USA: Now Publishers Inc.

Businger, P. and G. H. Golub (1965). Linear least squares solutions by Householder transformations. *Numer. Math. 7*, 269–276.

Cabral, R., F. D. L. Torre, J. P. Costeira, and A. Bernardino (2013, Dec). Unifying nuclear norm and bilinear factorization approaches for low-rank matrix decomposition. In *2013 IEEE International Conference on Computer Vision*, pp. 2488–2495.

Cadima, J. and I. Jolliffe (2009, 10). On relationships between uncentred and column-centred principal component analysis. *Pakistan Journal of Statistics 25*(4), 473–503.

Çivril, A. and M. Magdon-Ismail (2012). Column subset selection via sparse approximation of SVD. *Theoretical Computer Science 421*, 1–14.

Chen, G. and G. Lerman (2009). Spectral curvature clustering (scc). *International Journal of Computer Vision 81*(3), 317–330.

Chen, J. and X. Huo (2006). Theoretical results of sparse representations of multiple measurement vectors. *IEEE Transactions on Signal processing 54*(12), 4634–4643.

Chen, S., S. A. Billings, and W. Luo (1989). Orthogonal least squares methods and their application to non-linear system identification. *International Journal of control 50*(5), 1873–1896.

Cherapanamjeri, Y., P. Jain, and P. Netrapalli (2017). Thresholding based outlier robust PCA. In S. Kale and O. Shamir (Eds.), *Proceedings of the 30th Conference on Learning Theory, COLT 2017, Amsterdam, The Netherlands, 7-10 July 2017*, Volume 65 of *Proceedings of Machine Learning Research*, pp. 593–628. PMLR.

Chierichetti, F., S. Gollapudi, R. Kumar, S. Lattanzi, R. Panigrahy, and D. P. Woodruff (2017). Algorithms for ℓ_p low-rank approximation. In *Proceedings of the 34th International Conference on Machine Learning*, Volume 70, pp. 806–814. PMLR.

Clarkson, K. L. and D. P. Woodruff (2015). Input sparsity and hardness for robust subspace approximation. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pp. 310–329. IEEE.

Clarkson, K. L. and D. P. Woodruff (2017, January). Low-rank approximation and regression in input sparsity time. *Journal of the ACM 63*(6), 54:1–54:45.

Cotter, S. F., B. D. Rao, K. Engen, and K. Kreutz-Delgado (2005). Sparse solutions to linear inverse problems with multiple measurement vectors. *ASP 53*(7), 2477–2488.

Croux, C., P. Filzmoser, and M. R. Oliveira (2007). Algorithms for projection-pursuit robust principal component analysis. *Chemometrics and Intelligent Laboratory Systems 87*(2), 218–225.

Croux, C. and G. Haesbroeck (2000). Principal component analysis based on robust estimators of the covariance or correlation matrix: influence functions and efficiencies. *Biometrika 87*(3), 603–618.

Cui, Y. and L. Fan (2012). A novel supervised dimensionality reduction algorithm: Graph-based fisher analysis. *Pattern Recognition 45*(4), 1471–1481.

Cunningham, J. P. and Z. Ghahramani (2015). Linear dimensionality reduction: Survey, insights, and generalizations. *Journal of Machine Learning Research 16*(89), 2859–2900.

Daniel, J., W. Gragg, L. Kaufman, and G. Stewart (1976). Reorthogonalization and stable algorithms for updating the gram-schmidt qr factorization. *Mathematics of Computation 30*(136), 772–795.

Davis, G., S. Mallat, and M. Avellaneda (1997). Adaptive greedy approximations. *Constructive approximation 13*(1), 57–98.

Deshpande, A. and L. Rademacher (2010). Efficient volume sampling for row/column subset selection. In *FOCS*, pp. 329–338. IEEE Computer Society Press.

Deshpande, A., L. Rademacher, S. Vempala, and G. Wang (2006). Matrix approximation and projective clustering via volume sampling. *Theory of Computing 2*(12), 225–247.

Ding, C., D. Zhou, X. He, and H. Zha (2006). R1-pca: rotational invariant l 1-norm principal component analysis for robust subspace factorization. In *Proceedings of the 23rd international conference on Machine learning*, pp. 281–288.

Donoho, D. L. et al. (2000). High-dimensional data analysis: The curses and blessings of dimensionality. *AMS math challenges lecture 1*(2000), 32.

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

Espadoto, M., R. M. Martins, A. Kerren, N. S. Hirata, and A. C. Telea (2019). Towards a quantitative survey of dimension reduction techniques. *IEEE transactions on visualization and computer graphics*.

Feng, J., H. Xu, and S. Yan (2012). Robust pca in high-dimension: A deterministic approach. In *Proceedings of the 29th International Conference on Machine Learning*, pp. 1827–1834.

Fischler, M. A. and R. C. Bolles (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM 24*(6), 381–395.

Furnival, G. M. and R. W. Wilson (1974). Regressions by leaps and bounds. *Technometrics 16*(4), 499–511.

Gillis, N. and S. A. Vavasis (2018). On the complexity of robust pca and l1-norm low-rank matrix approximation. *Mathematics of Operations Research 43*(4), 1072–1084.

Goes, J., T. Zhang, R. Arora, and G. Lerman (2014). Robust stochastic principal component analysis. In *Artificial Intelligence and Statistics*, pp. 266–274.

Golub, G. H. and C. F. Van-Loan (2013). *Matrix Computations* (Fourth ed.). Baltimore: Johns Hopkins University Press.

Gray, V. (2017). *Principal Component Analysis: Methods, Applications and Technology*. Mathematics Research Developments. Nova Science Publishers, Incorporated.

Gu, M. and S. C. Eisenstat (1996). Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM J. Computing 17*(4), 848–869.

Guha, T. and R. K. Ward (2011). Learning sparse representations for human action recognition. *IEEE transactions on pattern analysis and machine intelligence 34*(8), 1576–1588.

Guruswami, V. and A. K. Sinop (2012). Optimal column-based low-rank matrix reconstruction. In Y. Rabani (Ed.), *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pp. 1207–1214. SIAM.

Guyon, I. and A. Elisseeff (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research 3*, 1157–1182.

Guyon, I., S. Gunn, M. Nikravesh, and L. A. Zadeh (2008). *Feature extraction: foundations and applications*, Volume 207. Springer.

Guyon, I., J. Weston, S. Barnhill, and V. Vapnik (2002). Gene selection for cancer classification using support vector machines. *Machine Learning 46*(1-3), 389–422.

Hadi, A. S., A. R. Imon, and M. Werner (2009). Detection of outliers. *Wiley Interdisciplinary Reviews: Computational Statistics 1*(1), 57–70.

Halko, N., P. Martinsson, Y. Shkolnisky, and M. Tygert (2011, October). An algorithm for the principal component analysis of large data sets. *SIAM Journal of Scientific Computing 33*(5), 2580–2594.

Halko, N., P. G. Martinsson, and J. A. Tropp (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review 53*(2), 217–288.

Hansen, E. A. and R. Zhou (2007). Anytime heuristic search. *Journal of Artificial Intelligence Research 28*, 267–297.

Hardt, M. and A. Moitra (2013). Algorithms and hardness for robust subspace recovery. In S. Shalev-Shwartz and I. Steinwart (Eds.), *COLT*, Volume 30 of *JMLR Workshop and Conference Proceedings*, pp. 354–375. JMLR.org.

Hart, P. E., N. J. Nilsson, and B. Raphael (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics 4*(2), 100–107.

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning* (second ed.)., Section 3.3.1. Springer.

He, B., S. Shah, C. Maung, G. Arnold, G. Wan, and H. Schweitzer (2019). Heuristic search algorithm for dimensionality reduction optimally combining feature selection and feature extraction. In *Proceedings of the 33rd National Conference on Artificial Intelligence (AAAI'19)*, California, pp. 2280–2287. AAAI Press.

Hill, Jr, R. and B. N. Parlett (1992). Refined interlacing properties. *SIAM journal on matrix analysis and applications 13*(1), 239–247.

Hodge, V. and J. Austin (2004, October). A survey of outlier detection methodologies. *Artificial intelligence review 22*(2), 85–126.

Hou, C., Y. Jiao, F. Nie, T. Luo, and Z. Zhou (2017). 2d feature selection by sparse matrix regression. *IEEE Transactions on Image Processing 26*(9), 4255–4268.

Huang, S., Y. Peng, C. Chang, K. Cheng, S. Huang, and B. Chen (2020). Restoration of images with high-density impulsive noise based on sparse approximation and ant-colony optimization. *IEEE Access 8*.

Huber, P. J. (2004). *Robust statistics*, Volume 523. John Wiley & Sons.

Hubert, M. and S. Engelen (2004, 02). Robust PCA and classification in biosciences. *Bioinformatics 20*(11), 1728–1736.

Jolliffe, I. T. (2002). *Principal Component Analysis* (second ed.). Springer-Verlag.

Joneidi, M., S. Vahidian, A. Esmaeili, W. Wang, N. Rahnavard, B. Lin, and M. Shah (2020). Select to better learn: Fast and accurate deep learning using data selection from nonlinear manifolds. In *CVPR'20*, pp. 7819–7829.

Khalid, S., T. Khalil, and S. Nasreen (2014). A survey of feature selection and feature extraction techniques in machine learning. In *2014 science and information conference*, pp. 372–378. IEEE.

Kneip, A. and P. Sarda (2011). Factor models and variable selection in high-dimensional regression analysis. *The Annals of Statistics 39*(5), 2410–2447.

Kokiopoulou, E., J. Chen, and Y. Saad (2011). Trace optimization and eigenproblems in dimension reduction methods. *Numerical Linear Algebra with Applications 18*(3), 565–602.

Kuang-Chih Lee, J. Ho, and D. J. Kriegman (2005). Acquiring linear subspaces for face recognition under variable lighting. *IEEE Transactions on Pattern Analysis and Machine Intelligence 27*(5), 684–698.

Kwak, N. (2008). Principal component analysis based on l1-norm maximization. *IEEE transactions on pattern analysis and machine intelligence 30*(9), 1672–1680.

Lai, C.-H., D. Zou, and G. Lerman (2020). Robust subspace recovery layer for unsupervised anomaly detection. In *Eighth International Conference on Learning Representations*.

Lei Xu and A. L. Yuille (1995). Robust principal component analysis by self-organizing rules based on statistical physics approach. *IEEE Transactions on Neural Networks 6*(1), 131–143.

Lerman, G. and T. Maunu (2018a). Fast, robust and non-convex subspace recovery. *Information and Inference: A Journal of the IMA 7*(2), 277–336.

Lerman, G. and T. Maunu (2018b). Fast, robust and non-convex subspace recovery. *Information and Inference: A Journal of the IMA 7*(2), 277–336.

Lerman, G. and T. Maunu (2018c). An overview of robust subspace recovery. *Proceedings of the IEEE 106*(8), 1380–1410.

Lerman, G., M. B. McCoy, J. A. Tropp, and T. Zhang (2015). Robust computation of linear models by convex relaxation. *Foundations of Computational Mathematics 15*(2), 363–410.

Li, H., G. C. Linderman, A. Szlam, K. P. Stanton, Y. Kluger, and M. Tygert (2017, January). Algorithm 971: An implementation of a randomized algorithm for principal component analysis. *ACM Transactions on Mathematical Software 43*(3), 28:1–28:14.

Li, X. and J. Haupt (2015). Identifying outliers in large matrices via randomized adaptive compressive sampling. *IEEE Transactions on Signal Processing 63*(7), 1792–1807.

Likhachev, M., G. J. Gordon, and S. Thrun (2004). Ara*: Anytime a* with provable bounds on sub-optimality. In *Advances in neural information processing systems*, pp. 767–774.

Mairal, J., F. Bach, J. Ponce, et al. (2014). Sparse modeling for image and vision processing. *Foundations and Trends® in Computer Graphics and Vision 8*(2-3), 85–283.

Malioutov, D., M. Cetin, and A. S. Willsky (2005). A sparse signal reconstruction perspective for source localization with sensor arrays. *IEEE transactions on signal processing 53*(8), 3010–3022.

Mallat, S. (1999). *A wavelet Tour of Signal Processing*. Academic Press.

Maronna, R., D. Martin, and V. Yohai (2006). *Robust Statistics: Theory and Methods*. Hoboken, NJ: Wiley.

Maung, C. and H. Schweitzer (2013). Pass-efficient unsupervised feature selection. In *Advances in Neural Information Processing Systems (NIPS)*, Volume 26, pp. 1628–1636.

Maung, C. and H. Schweitzer (2015, March). Improved greedy algorithms for sparse approximation of a matrix in terms of another matrix. *IEEE Transactions on Knowledge and Data Engineering 27*(3), 769–780.

Maunu, T., T. Zhang, and G. Lerman (2019a). A well-tempered landscape for non-convex robust subspace recovery. *J. Mach. Learn. Res. 20*(37), 1–59.

Maunu, T., T. Zhang, and G. Lerman (2019b). A well-tempered landscape for non-convex robust subspace recovery. *Journal of Machine Learning Research 20*(37), 1–59.

McCoy, M., J. A. Tropp, et al. (2011). Two proposals for robust pca using semidefinite programming. *Electronic Journal of Statistics 5*, 1123–1160.

Musco, C. and C. Musco (2015). Randomized block krylov methods for stronger and faster approximate singular value decomposition. In *NIPS'15*, Cambridge, MA, USA, pp. 1396–1404. MIT Press.

Natarajan, B. K. (1995). Sparse approximate solutions to linear systems. *SIAM Journal of Computing 25*(2), 227–234.

Neff, R. and A. Zakhor (1997). Very low bit-rate video coding based on matching pursuits. *IEEE Transactions on Circuits and Systems for Video Technology 7*(1), 158–171.

Pati, Y. C., R. Rezaiifar, and P. S. Krishnaprasad (1993, Nov). Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *The Twenty-Seventh Asilomar Conference on Signals, Systems and Computers*, Volume 1, pp. 40–44.

Paul, S., M. Magdon-Ismail, and P. Drineas (2015). Column selection via adaptive sampling. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 28*, pp. 406–414. Curran Associates, Inc.

Pearl, J. (1984). *Heuristics : intelligent search strategies for computer*. Reading, Massachusetts: Addison-Wesley.

Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 2*(11), 559–572.

Pohl, I. (1970). Heuristic search viewed as path finding in a graph. *Artificial intelligence 1*(3-4), 193–204.

Qian, C., Y. Yu, and Z. Zhou (2015). Subset selection by pareto optimization. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.), *Advances in Neural Information Processing Systems 28*, pp. 1774–1782. Curran Associates, Inc.

Qin, Z., J. Fan, Y. Liu, Y. Gao, and G. Y. Li (2018). Sparse representation for wireless communications: A compressive sensing approach. *IEEE Signal Processing Magazine 35*(3), 40–58.

Rahmani, M. and G. K. Atia (2017, Dec). Coherence pursuit: Fast, simple, and robust principal component analysis. *IEEE Transactions on Signal Processing 65*(23), 6260–6275.

Rahmani, M. and P. Li (2019a). Outlier detection and robust pca using a convex measure of innovation. In *Advances in Neural Information Processing Systems*, pp. 14223–14233.

Rahmani, M. and P. Li (2019b). Outlier detection and robust pca using a convex measure of innovation. In *NIPS'19*, pp. 14200–14210.

Richter, S., J. T. Thayer, and W. Ruml (2010). The joy of forgetting: Faster anytime search via restarting. In *Twentieth International Conference on Automated Planning and Scheduling*.

Roberts, S. J. (1999). Novelty detection using extreme value statistics. *IEE Proceedings-Vision, Image and Signal Processing 146*(3), 124–129.

Rousseeuw, P. J. and M. Hubert (2018). Anomaly detection by robust statistics. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8*(2), e1236.

Russell, S. and P. Norvig (2010). *Artificial Intelligence - A Modern Approach*. Pearson Education.

Scheirer, W. J., A. Rocha, R. J. Micheals, and T. E. Boult (2011). Meta-recognition: The theory and practice of recognition score analysis. *IEEE transactions on pattern analysis and machine intelligence 33*(8), 1689–1695.

Selesnick, I. (2017). Sparse regularization via convex analysis. *IEEE Transactions on Signal Processing 65*(17), 4481–4494.

Shah, S., B. He, C. Maung, and H. Schweitzer (2017). Computing robust principal components by A* search. In *IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 1042 – 1049.

Shah, S., B. He, C. Maung, and H. Schweitzer (2018, November). Computing robust principal components by A* search. *International Journal on Artificial Intelligence Tools 27*(7), 1860013.

Shah, S., B. He, K. Xu, C. Maung, and H. Schweitzer (2018). Solving generalized column subset selection with heuristic search. In *Proceedings of the 32nd National Conference on Artificial Intelligence (AAAI'18)*, pp. 8153–8154. AAAI Press.

Shitov, Y. (2017a, January). Column subset selection is np-complete. arXiv e-print (arXiv:1701.02764[math.CO]).

Shitov, Y. (2017b, January). Column subset selection is np-complete. arXiv e-print (arXiv:1701.02764[math.CO]).

Simonov, K., F. Fomin, P. Golovach, and F. Panolan (2019). Refined complexity of pca with outliers. In *International Conference on Machine Learning*, pp. 5818–5826. PMLR.

Solorio-Fernández, S., J. A. Carrasco-Ochoa, and J. F. Martínez-Trinidad (2020). A review of unsupervised feature selection methods. *Artificial Intelligence Review 53*(2), 907–948.

Soltanolkotabi, M., E. J. Candes, et al. (2012). A geometric analysis of subspace clustering with outliers. *The Annals of Statistics 40*(4), 2195–2238.

Song, Z., D. P. Woodruff, and P. Zhong (2017). Low rank approximation with entrywise ℓ_1-norm error. In *STOC'17*, New York, NY, USA, pp. 688–701. ACM.

Soussen, C., R. Gribonval, J. Idier, and C. Herzet (2013). Joint k-step analysis of orthogonal matching pursuit and orthogonal least squares. *IEEE Transactions on Information Theory 59*(5), 3158–3174.

Thayer, J. T. and W. Ruml (2008). Faster than weighted a*: An optimistic approach to bounded suboptimal search. In *Proceedings of the Eighteenth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS'08, pp. 355–362. AAAI Press.

Thompson, R. (1976). The behavior of eigenvalues and singular values under perturbations of restricted rank. *Linear Algebra and its Applications 13*(1-2), 69–78.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *J. Royal. Statist. Soc B. 58*(1), 267–288.

Tropp, J. A. (2004). Greed is good: algorithmic results for sparse approximation. *IEEE Transactions on Information Theory 50*(10), 2231–2242.

Tropp, J. A., A. C. Gilbert, and M. J. Strauss (2006). Algorithms for simultaneous sparse approximation. part I: Greedy pursuit. *Signal Processing 86*(3), 572–588.

Vaswani, N. and P. Narayanamurthy (2018). Static and dynamic robust pca and matrix completion: A review. *Proceedings of the IEEE 106*(8), 1359–1379.

Vaswani, N. and P. Narayanamurthy (2018). Static and dynamic robust pca and matrix completion: A review. *Proceedings of the IEEE 106*(8), 1359–1379.

Vepakomma, P., C. Tonde, A. Elgammal, et al. (2018). Supervised dimensionality reduction via distance correlation maximization. *Electronic Journal of Statistics 12*(1), 960–984.

Vidal, R., Y. Ma, and S. S. Sastry (2016). *Generalized Principal Component Analysis*. New York: Springer-Verlag.

Wan, G., C. Maung, C. Zhang, and H. Schweitzer (2020). Fast distance metrics in low-dimensional space for neighbor search problems. In *2020 IEEE International Conference on Data Mining, ICDM 2020, November 17-20, 2020*. IEEE.

Wan, G. and H. Schweitzer (2021a). Accelerated combinatorial search for outlier detection with provable bound on sub-optimality. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 35, pp. 12436–12444.

Wan, G. and H. Schweitzer (2021b). A fast algorithm for simultaneous sparse approximation. In *Advances in Knowledge Discovery and Data Mining (PAKDD'21)*, pp. 42–54. Springer.

Wan, G. and H. Schweitzer (2021c). Heuristic search for approximating one matrix in terms of another matrix. In *Proceedings of the Thirty International Joint Conference on Artificial Intelligence, IJCAI-21*. International Joint Conferences on Artificial Intelligence Organization.

Wan, G. and H. Schweitzer (2021d). A new robust subspace recovery algorithm (student abstract). In *Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 35, pp. 15911–15912.

Wang, H. (2012). Factor profiled sure independence screening. *Biometrika 99*(1), 15–28.

Wickham, H. and L. Stryjewski (2012). 40 years of boxplots. Technical report, had.co.nz.

Wright, J., Y. Ma, J. Mairal, G. Sapiro, T. S. Huang, and S. Yan (2010). Sparse representation for computer vision and pattern recognition. *Proceedings of the IEEE 98*(6), 1031–1044.

Xu, H., C. Caramanis, and S. Sanghavi (2010). Robust pca via outlier pursuit. In *Advances in Neural Information Processing Systems*, pp. 2496–2504.

Xu, H., C. Caramanis, and S. Sanghavi (2012a). Robust pca via outlier pursuit. *IEEE Transactions on Information Theory 58*(5), 3047–3064.

Xu, H., C. Caramanis, and S. Sanghavi (2012b). Robust pca via outlier pursuit. *IEEE Transactions on Information Theory 58*(5), 3047–3064.

Xu, X. and Z. Shi (2017). Multi-objective based spectral unmixing for hyperspectral images. *ISPRS Journal of Photogrammetry and Remote Sensing 124*, 54–69.

Xu, Y., Z. Li, J. Yang, and D. Zhang (2017). A survey of dictionary learning algorithms for face recognition. *IEEE access 5*, 8502–8514.

You, C., D. P. Robinson, and R. Vidal (2017). Provable self-representation based outlier detection in a union of subspaces. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3395–3404.

Zaeemzadeh, A., M. Joneidi, N. Rahnavard, and M. Shah (2019). Iterative projection and matching: Finding structure-preserving representatives and its application to computer vision. In *CVPR'19*.

Zhang, H., Z. Lin, C. Zhang, and E. Y. Chang (2015a). Exact recoverability of robust pca via outlier pursuit with tight recovery bounds. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pp. 3143–3149. AAAI Press.

Zhang, H., Z. Lin, C. Zhang, and E. Y. Chang (2015b). Exact recoverability of robust pca via outlier pursuit with tight recovery bounds. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Zhang, Q., Y. Liu, R. S. Blum, J. Han, and D. Tao (2018). Sparse representation based multi-sensor image fusion for multi-focus and multi-modality images: A review. *Information Fusion 40*, 57–75.

Zhang, T. (2009). Adaptive forward-backward greedy algorithm for sparse learning with linear models. In *Advances in Neural Information Processing Systems*, pp. 1921–1928.

Zhang, T. (2016a). Robust subspace recovery by tyler's m-estimator. *Information and Inference: A Journal of the IMA 5*(1), 1–21.

Zhang, T. (2016b). Robust subspace recovery by tyler's m-estimator. *Information and Inference: A Journal of the IMA 5*(1), 1–21.

Zhang, T. and G. Lerman (2014a). A novel m-estimator for robust pca. *The Journal of Machine Learning Research 15*(1), 749–808.

Zhang, T. and G. Lerman (2014b). A novel m-estimator for robust pca. *The Journal of Machine Learning Research 15*(1), 749–808.

Zhang, T., A. Szlam, and G. Lerman (2009). Median k-flats for hybrid linear modeling with many outliers. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pp. 234–241. IEEE.

Zhang, Z., Y. Xu, J. Yang, X. Li, and D. Zhang (2015). A survey of sparse representation: algorithms and applications. *IEEE access 3*, 490–530.

Zhu, F., B. Fan, X. Zhu, Y. Wang, S. Xiang, and C. Pan (2015). 10,000+ times accelerated robust subset selection. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, pp. 3217–3223.

Zhu, X., R. Hu, C. Lei, K. H. Thung, W. Zheng, and C. Wang (2019). Low-rank hypergraph feature selection for multi-output regression. *World Wide Web 22*(2), 517–531.

BIOGRAPHICAL SKETCH

Guihong Wan received the BS degree from South-Central University for Nationalities in 2010. She joined the Department of Computer Science at The University of Texas at Dallas as a graduate student in 2017. She has been working on her PhD under the supervision of Dr. Haim Schweitzer since Fall 2018. Her current research interests are in machine learning, computer vision, and various aspects of deep learning.

CURRICULUM VITAE

Guihong Wan

June 20, 2021

Contact Information:

Department of Computer Science
The University of Texas at Dallas
800 West Campbell Road
Richardson, TX 75080-3021, U.S.A.

Educational History:

BS, Electronic Information Engineering, South-Central University for Nationalities, 2010
MS, Computer Science, The University of Texas at Dallas, 2021
PhD, Computer Science, The University of Texas at Dallas, 2021

AI Inspired Algorithms for Several Combinatorial Optimization Problems in Data Science
PhD Dissertation
Department of Computer Science, The University of Texas at Dallas
Advisor: Dr. Haim Schweitzer

Publications:

- Guihong Wan and Haim Schweitzer, "Heuristic Search for Approximating One Matrix in Terms of Another Matrix," in Proceedings of the 30th International Joint Conference on Artificial Intelligence, IJCAI 2021.
- Guihong Wan and Haim Schweitzer, "Accelerated Combinatorial Search for Outlier Detection with Provable Bounds on Sub-Optimality," in Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021.
- Guihong Wan, "Edge Sparsification for Graphs via Meta-Learning," in IEEE 37th International Conference on Data Engineering, ICDE 2021.
- Guihong Wan and Haim Schweitzer, "A New Robust Subspace Recovery Algorithm (Student Abstract)," in Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021.
- Guihong Wan and Haim Schweitzer, "A Fast Algorithm for Simultaneous Sparse Approximation,". in Advances in Knowledge Discovery and Data Mining, PAKDD 2021.

- Guihong Wan, Crystal Maung, Chenxu Zhang, and Haim Schweitzer, "Fast Distance Metrics in Low-dimensional Space for Neighbor Search Problems," in IEEE 20th International Conference on Data Mining, ICDM 2020.
- Guihong Wan, Crystal Maung, and Haim Schweitzer, "Improving the Accuracy of Principal Component Analysis by the Maximum Entropy Method," in IEEE 31st International Conference on Tools with Artificial Intelligence, ICTAI 2019.
- Baokun He, Guihong Wan, and Haim Schweitzer, "A Bias Trick for Centered Robust Principal Component Analysis (Student Abstract)," in Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020.
- Baokun He, Swair Shah, Crystal Maung, Gordon Arnold, Guihong Wan, and Haim Schweitzer, "Heuristic Search Algorithm for Dimensionality Reduction Optimally Combining Feature Selection and Feature Extraction," in Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019.