ALGORITHMS FOR ROBUST DATA ANALYSIS

by

Baokun He

APPROVED BY SUPERVISORY COMMITTEE:

Haim Schweitzer, Chair

Farokh Bastani

Ding-Zhu Du

Weili Wu

ALGORITHMS FOR ROBUST DATA ANALYSIS

by

BAOKUN HE, 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

May 2020

ACKNOWLEDGMENTS

I would like to express my sincere gratitude and appreciation to my adviser, Dr. Haim Schweitzer, for his guidance and continuous support for my PhD study. It has been a great time working with him. He has been supportive and has given me the freedom to choose whatever is best for my career. He has not only provided insightful discussion on the research work but also has helped me improve my soft skills. I am so lucky to have him be my adviser.

I would like to thank Swair Shah, Crystal Maung, Ke Xu, Guihong Wan and Rong Jin for their contributions in the joint work and all the fun time we have spent together.

Thanks to my parents for their support and infinite love. Thanks to my husband, Shiyang Cheng, for cheering me up when I lost my confidence. I would not have made it without him. Especially, I would like to give special thanks to my grandma, Fengzhen Shi. She taught me the good habit of reading, taught me to be curious about new things, and taught me to never stop learning.

February 2020

ALGORITHMS FOR ROBUST DATA ANALYSIS

Baokun He, PhD
The University of Texas at Dallas, 2020

Supervising Professor: Haim Schweitzer, Chair

Data analysis plays an important role in making decisions, making predictions, and helping business operate. Unfortunately, in many situations the data is not reliable and robust analysis is needed to obtain stable results. This may be challenging when the data is high-dimensional. Transforming high-dimensional data into low-dimensional data is an important prior step in applications such as managing the data, performing efficient learning, retrieving information, and other data analytics tasks.

Feature selection and feature extraction are two classical techniques to achieve dimensionality reduction. Feature selection removes irrelevant and redundant features and keeps only the most important ones. Unlike feature selection, feature extraction generates the features as arbitrary functions of the data. They are typically more accurate than those obtained by feature selection. But the extracted features are harder to interpret than the selected features. Another disadvantage is that feature extraction is less robust than feature selection. In this dissertation we describe algorithms and methods to improve the robustness of feature selection and feature extraction.

We propose computing a hybrid low rank representation (HLR) of selected features and extracted features. The robustness of the HLR model comes from the selected features, and its accuracy comes from the extracted features. We develop an algorithm to solve this hybrid problem optimally by combinatorial search. We propose optimal, sub-optimal, and greedy

variants of the algorithm to solve this hybrid problem. The sub-optimal and the greedy variants come with the exact bounds on the representation accuracy.

Principal Component Analysis (PCA) is a widely used feature extraction algorithm. It is known to be sensitive to outliers that reduce its robustness. Robust Principal Component Analysis (RPCA), also known as Robust Subspace Recovery (RSR), is a classical approach to improve the robustness of the PCA by identifying and removing outliers. We develop a novel RPCA algorithm, which converts this problem into graph search. We show how to solve the graph search problem optimally by applying heuristic search techniques from AI. The results obtained by our algorithm are optimal in terms of accuracy. We also describe a sub-optimal variant that runs much faster than the optimal variant and produces a solution that is guaranteed to be near the optimal.

Outlier based RPCA removes outliers from the data and computes the principal components of the remaining data. The centered variant requires the center of non-outliers, which is unknown until after the outliers are determined. Not using an accurate center may lead to the detection of wrong outliers. We propose a method that can be used to improve the robustness of many currently known RPCA algorithms. Our method implicitly centers the non-outliers; it is implemented by appending a bias value to each data element. It can be used with "black box" RPCA algorithms since only their input needs to be augmented.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

High dimensional datasets are common in machine learning and statistical applications. The "curse of dimensionality" phenomenon makes the analysis of high dimensional data challenging. Dimensionality reduction is a collection of techniques that is designed to overcome this problem. Datasets can be labeled, unlabeled, or partially labeled; this leads to various dimensionality reduction methods of supervised, unsupervised and semi-supervised. Typically, a labeled dataset contains a set of data which are marked as labels or classes. An unlabeled dataset does not contain label or class information. Supervised dimensionality reduction methods compute a feature subset to predict the label information. Unsupervised dimensionality reduction methods exploit the pattern and structures of the data, such as data distribution, data variance, and separability. In the semi-supervised cases the goal is integrating a small amount of labeled data into unlabeled data as additional information to improve the performance of an unsupervised method.

A feature space is composed of a set of feature vectors. Each feature vector is a vector which represents the values of the feature over all samples. Feature selection and feature extraction are two common techniques used in order to reduce the number of dimensions of the feature space. Both of these techniques improve the training speed and generalization properties of the models, see e.g., (Guyon and Elisseeff, 2003; Kuhn and Johnson, 2013). If the reduced features come from the data itself, this process is called feature selection. As shown in Figure 1.1 feature extraction is different from feature selection. Feature extraction returns a set of features which are results of arbitrary functions of the data.

Compared to feature selection, feature extraction is more general and the arbitrary function may provide a better accuracy. But the feature space is problematic since there may be no physical meaning for better readability and interpretation (Krızek, 2008). Additionally, feature selection is more robust to the perturbation of values than feature extraction.

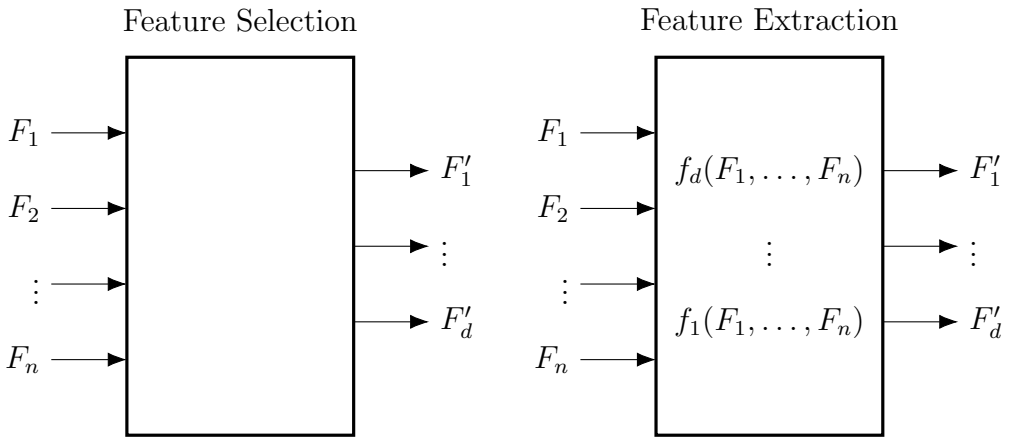


Figure 1.1: Feature Selection and Extraction

From a statistical point of view the analysis is based on the observations of the data. However, in many cases the data is not reliable. What we expected is a minor error in the model or data should cause only small error or no effect to the final result. Unfortunately, this is not always true. Thus, the "robust" procedure has been proposed (Huber, 2011). An ideal robust estimator provides a good fit to the data when the data contains outliers, as well as when the data is free of them (Maronna et al., 2019). To reduce the effect of outliers there are many studies of robust feature selection and robust feature extraction. In this dissertation we focus on unsupervised dimensionality reduction methods and how to achieve a robust estimator.

1.1 Our Contributions

We formulate a problem combining feature selection and extraction. We refer to it as Hybrid Low Rank (HLR) representation. It is easy to show that sequentially solving the selection and extraction sub-problems cannot return an optimal HLR solution. The key to achieve the optimal combination of selected features and extracted features is to obtain them simultaneously. We use a combinatorial graph search algorithm to solve this hybrid problem. To our knowledge this is the first optimal algorithm to solve this problem. We provide a way

to speed up this algorithm but sacrifice some accuracy. We proceed to show how to obtain bounds on how close the sub-optimal solution is to the optimal one.

Using the same combinatorial graph search framework we formulate outlier Robust Principal Component Analysis (RPCA) problem and come up with another set of heuristic functions which gets optimal outliers when the number of outliers is known. The results obtained by our algorithm are optimal, and more accurate than the current state of the art. This comes at the cost of running time, which is typically slower than the current state of the art. To accelerate the algorithm we describe a variant of the algorithm that runs much faster and produces a solution that is guaranteed to be near the optimal. We illustrate this by experiments both on synthetic and real-life datasets. This optimal RPCA algorithm is an uncentered RPCA which ignores the center of the data.

To keep the centering information and eliminate the effect of outliers on centering we obtain an optimal centered RPCA algorithm by extending the optimal RPCA algorithm with a "bias trick". Best to our knowledge this is the first optimal centered RPCA algorithm with respect to centering. It is proved that the bias trick can automatically get the center of the data. This centered RPCA algorithm compares favorably to the current state of the art methods in the experiments.

1.2 Structure of The Dissertation

In Chapter 2 we formally state feature selection and feature extraction problems and introduce the hybrid feature selection and extraction problem. We also discuss PCA, RPCA, robustness and outliers. Chapter 3 discusses the algorithm and analysis for the hybrid feature selection and extraction problem. Chapter 4 discusses our optimal uncentered RPCA algorithm. In Chapter 5 the bias trick is introduced which can be used to improve many RPCA algorithms, and our centered RPCA algorithm is discussed.

CHAPTER 2

BACKGROUND

There are many approaches of feature selection and extraction in unsupervised case. In this chapter we formulate feature selection, feature extraction, and hybrid feature selection and extraction problems. We lay emphasis on a widely used feature extraction method: PCA. We discuss its relationship with Singular Value Decomposition (SVD) and Eigenvalue Decomposition (ED). We discuss the impact of outlier on computing principal component and how to reduce the impact to solve the RPCA problem.

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 especially when there are billions features in the datasets. 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. Given a dataset $X \in \mathbb{R}^{m\times n}$ with m data points and n features, both feature selection and extraction are aiming to get a set of features $V \in \mathbb{R}^{m\times r}$ with $r \leq n$ from X, such that the reconstructed error from the set of features is minimized. As shown in (2.1), the reconstruction is computed by $X \approx VA$, where A is a $r \times n$ coefficients matrix and Θ is a matrix norm.

$$\min_{V,A} \Theta(X - VA) \quad \text{s.t. } V \in \mathbb{R}^{m\times r} \tag{2.1}$$

We note that the matrix VA is of rank r, so that X is approximated by a low rank matrix. Conversely, any rank r matrix can be expressed as the product of VA, and thus gives a compact representation in terms of r features.

To describe current and previous results we need the following notation. Let $E_{\text{FE}}, E_{\text{FS}}$ 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:

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

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: $1 \leq \frac{E_\alpha(S,X)}{E_{\text{FE}}} \leq p_\alpha$. 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 $p_\alpha = \sqrt{r+1}$. With this notation 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.

2.1 Feature Selection

In feature selection the V in (2.1) are constrained to be a subset of columns of X. We denote the subset as S such that a precise expression of the reconstructed error is:

$$\begin{aligned} &\min_{S,A} \Theta(X - SA) \\ &\text{s.t. } S \subset X, \text{ and } S \in \mathbb{R}^{m \times r} \end{aligned} \tag{2.2}$$

The notation $\subset$ is denoted as column subset. Since S is a column subset of X, feature selection is sometimes known as the Column Subset Selection Problem (CSSP). See, e.g. (Golub and Van-Loan, 2013).

Consider the sample data in Table 2.1. The features include height in centimeter, height in feet, age, and weight in pounds. The labels to be predicted are the Body Mass Index (BMI) and the weight in kilograms. The matrix notation of the data is shown in (2.3), where X is the data matrix, Y is the labels matrix.

$$X = \begin{pmatrix} 160 & 5.24 & 17 & 98 \\ 173 & 5.67 & 26 & 121 \\ 185 & 6.07 & 40 & 174 \end{pmatrix}, \quad Y = \begin{pmatrix} 17.4 & 44.45 \\ 18.4 & 54.9 \\ 23 & 78.9 \end{pmatrix} \tag{2.3}$$

Table 2.1: Toy data

Name	Height (cm) (feature)	Height (ft) (feature)	Age (feature)	Weight (lb) (feature)	BMI (label)	Weight (kilos) (label)
Alexa	160	5.24	17	98	17.4	44.45
Cortana	173	5.67	26	121	18.4	54.9
Siri	185	6.07	40	174	23	78.9

The unsupervised feature selection attempts to select features from the data matrix with no label information. In the toy example described above we would still have the second "Height" feature as redundant, but in the unsupervised case the "Age" feature cannot be discarded. Therefore, in this case we have the selection matrix as in (2.4) with zero approximation error.

$$X = \begin{pmatrix} 160 & 5.24 & 17 & 98 \\ 173 & 5.67 & 26 & 121 \\ 185 & 6.07 & 40 & 174 \end{pmatrix}, \quad S = \begin{pmatrix} 160 & 17 & 98 \\ 173 & 26 & 121 \\ 185 & 40 & 174 \end{pmatrix} \tag{2.4}$$

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). (Boutsidis et al., 2009) points out the NP-hardness of CSSP is an open problem. A further proof is given in (Çivril, 2014) which points out the CSSP is UG-hard. Recently the problem was proved NP-hard (Shitov, 2017), and therefore there are no optimal polynomial algorithms. 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

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

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).

2.2 Feature Extraction

Unlike feature selection, the columns of V returned by feature extraction has no constraint. The well-known algorithm for optimal feature extraction is shown in Figure 2.1. See, e.g., (Jolliffe, 2002; Li et al., 2017). Applications of this algorithm include the technique of 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 Figure 2.1 can be implemented efficiently even for large amounts of data, See, e.g., (Yu et al., 2017; Onuki and Tanaka, 2018). 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 and Vavasis,

2018; Song et al., 2017; Bringmann et al., 2017). For the more general case of entry-wise l_p norms see (Chierichetti et al., 2017).

2.3 Hybrid Feature Selection and Feature Extraction

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 extraction, the results obtained by feature selection are easy to interpret in terms of the underlying data (Drineas et al., 2008). Selected features generalize better than extracted features in machine learning tasks (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. In the other hand, feature extraction is easier to get the optimal solution than feature selection. Feature extraction has a better approximation accuracy.

There are studies on combining feature selection and extraction to get the benefits from both of the technologies. The problem is how approximate a dataset X with r_1 selected features and r_2 extracted features to solve the following optimization problem:

$$\begin{aligned} &\min_{S,V,A_1,A_2} \Theta(X - SA_1 - VA_2) \\ &\text{s.t.} \begin{cases} S \in \mathbb{R}^{m \times r_1}; S \subset X \\ V \in \mathbb{R}^{m \times r_2} \end{cases} \end{aligned} \tag{2.5}$$

Clearly, being a generalization of feature selection and extraction, this problem is at least as hard as feature selection which is known as NP-hard as we discussed in Section 2.1. It turns out that solving this problem by solving the sub-problems of feature selection and extraction one after the other does not return the optimal solution of this hybrid problem. In Chapter 3 we give an example of this. The solution to this problem in addition to being theoretically

interesting, problems related to it have surfaced in statistical literature. In (Kneip and Sarda, 2011; Wang, 2012) authors consider a factor analysis problem where the decomposition is into two matrices of the similar form as in (2.5). The solution approach taken opts for performing feature extraction followed by selection. But as we show in Chapter 3 this approach does not lead to the optimal solution.

2.4 Principal Component Analysis

As we mentioned in Section 2.2, Principal Component Analysis (PCA) is arguably the most widely used feature extraction technique. PCA solves a problem that fits a low-dimensional subspace $V \in \mathbb{R}^r$ to a set of data points $X \in \mathbb{R}^{m \times n}$. The basis of the low-dimensional subspace V is a set of orthonormal vectors $\{v_1, \cdots, v_r\}$. And each vector maximizes the variance of the data points in the direction of the low-dimensional subspace. The projection A on the low-dimensional space can be computed from $X \approx VA$. Thus, when the error of the approximation is small, one can use the r dimensional vectors a_i (ith column of A) as representatives of the m dimensional vectors x_i in the low-dimensional vector space spanned by V. See (Jolliffe, 1986) for many applications of the PCA technique.

Mathematically, PCA can be formulated either as a statistic model or a geometric model. A statistic model tries to learn the probability distribution of observed random data samples. A geometric model learns the overall geometric shape of the dataset with deterministic models such as subspaces, smooth manifolds, or topological spaces. The original formulation of PCA (Pearson, 1901; Hotelling, 1933) is proposed as a statistic model more than a century ago. Since there is an equivalence between statistic formulation and geometric formulation of PCA, in this dissertation we focus on the geometric view and show the relation between PCA and SVD.

From the geometrical view of PCA we find a subspace $V \subset \mathbb{R}^r$ of dimension r to embed the data points $X = \{x_1, x_2, \ldots, x_n\}$ (where $x_i \in \mathbb{R}^m$) such that the variance of the projected

data is maximized. Let $V = \{v_1, v_2, \ldots, v_r\}$ be the matrix of orthonormal basis of this low-dimensional subspace. Then the projection of point x_i in this subspace is $VV^T x_i$. The problem then can be written as:

$$\begin{aligned} &\max_V \sum_{i=1}^{n} \|VV^T x_i\|^2 \\ &\text{s.t. } V^T V = I_r \end{aligned} \tag{2.6}$$

where I_r is an identity matrix with size $r \times r$. We observe that $\|x\|^2 = \|x - VV^T x\|^2 + \|VV^T x\|^2$. As the data points are fixed so are their norms. So the problem in (2.6) is equivalent to,

$$\begin{aligned} &\min_V \sum_{i=1}^{n} \|x_i - VV^T x_i\|^2 \\ &\text{s.t. } V^T V = I_r \end{aligned} \tag{2.7}$$

In the matrix form it can be rewritten as follows,

$$\begin{aligned} &\min_V \|X - VV^T X\|_F^2 \\ &\text{s.t. } V^T V = I_r \end{aligned} \tag{2.8}$$

where $\|\,\cdot\,\|_F$ is the Frobenius norm. Setting $V^T X = A$ we get the following formulation

$$\begin{aligned} &\min_V \|X - VA\|_F^2 \\ &\text{s.t. } V^T V = I_r \end{aligned} \tag{2.9}$$

We observe that the geometric interpretation of PCA has the same form as the feature extraction problem in (2.1) using Frobenius norm. The Eckart-Young-Mirsky theorem says that SVD solves this problem optimally.

Theorem 2.1. *Eckart-Young-Mirsky. Let $X_r = \sum_{i=1}^{r} \sigma_i u_i v_i^T$ be the truncated SVD of X with $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$. Then, for any rank-$r$ matrix Y,*

$$\|X - X_r\| \leq \|X - Y\|$$

Eckart and Young gave a proof for this theorem for Frobenius norm (Eckart and Young, 1936). Mirsky gave a generalized proof which works for all unitary invariant matrix norms (Mirsky, 1960). Theorem 2.1 shows that the best rank r approximation to a matrix is its truncated SVD. If $X = U\Sigma V^T$ is the SVD of X, then setting $U_r\Sigma_r V_r^T$ as the approximation of X solves the problem in (2.7) (where V_r corresponds to the matrix formed by first r columns of V). The desired V in (2.7) is the matrix of first r right singular vectors V_r of X.

2.4.1 Centered PCA and Uncentered PCA

In the formula of PCA we consider the dataset X with mean zero. This is known as the centered PCA. In the computation of PCA this is achieved by subtracting the mean from the dataset. There is an uncentered variant which skipped the preliminary step of subtracting the mean from the dataset. And it has many applications like computer vision, climatology, astronomy, ecology, chemistry, etc. (Jolliffe, 2002; Cadima and Jolliffe, 2009). The resulting subspace of uncentered PCA is a lower dimensional subspace for embedding the uncentered data. Uncentered and centered PCA are closely related. See (Cadima and Jolliffe, 2009) for a discussion on uncentered PCA. The variant we address in Chapter 4 is the uncentered variant. One can still subtract the mean and apply our algorithm on centered data but the mean itself may be affected by outliers and a robust mean estimation should be used. In Chapter 5 we give a solution to get the centered PCA with optimal mean when there are outliers in the given dataset.

Figure 2.2 shows the centered PCA and uncentered PCA on an example dataset. In the centered variant the first principal vector is in the direction that maximizes the data variance. However, this is not the case in the uncentered PCA where one sometimes observes that the first principal vector is in the direction of the mean μ. Details can be found in Section 5.4. Let $X = (x_1, \ldots, x_n)$ be the matrix with m features. The uncentered PCA computes a matrix $V = (v_1, \ldots, v_r)$ of r "principal vectors" where the column vector v_j are mutually

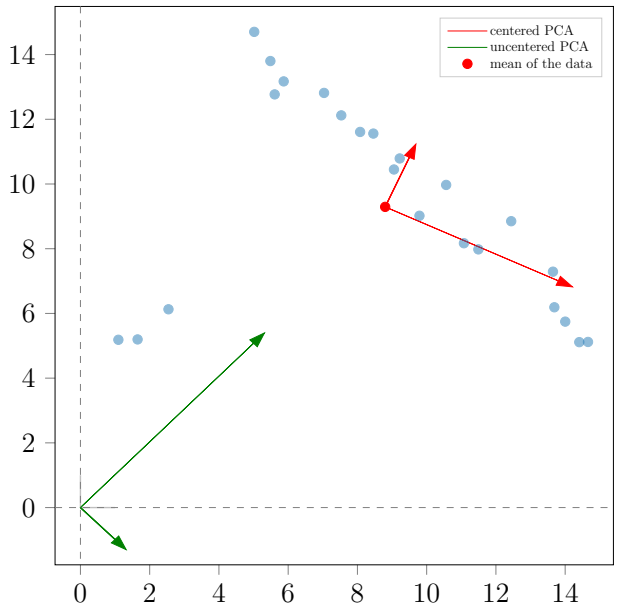


Figure 2.2: The principal vectors of centered and uncentered PCA. The data is shown as blue points with outliers.

orthogonal m dimensional vectors. The matrix V is computed to minimize the error in (2.9). It can be shown that the Euclidean error of the approximation in (2.9) is minimized for V computed as the r dominant eigenvectors of the matrix of second moments B.

$$\begin{aligned} B = XX^T &= \sum_i x_i x_i^T \\ &\approx V\Sigma U^T U \Sigma^T V^T \\ &= V\Sigma^2 V^T \end{aligned} \tag{2.10}$$

In the centered PCA case here one needs the mean μ of the dataset X. Each vector x_i can then be centered by mean subtraction: $\hat{x}_i = x_i - \mu$, and the matrix of second moments of the centralized vectors is the covariance matrix C. The matrix V of r principal vectors is

computed as the r dominant eigenvectors of the covariance matrix.

$$\begin{aligned} C &= \sum_i (x_i - \mu)(x_i - \mu)^T \\ &\approx V\Sigma U^T U \Sigma^T V^T \\ &= V\Sigma^2 V^T \end{aligned} \tag{2.11}$$

The V in centered PCA minimizes the error in (2.12), where $\mathbf{1}$ is a vector with size n whose entries are all 1s.

$$\begin{aligned} &\min_V \|X - \mu\mathbf{1}^T - VA\|_F^2 \\ &\text{s.t. } V^T V = I_r \end{aligned} \tag{2.12}$$

2.5 Robust Principal Component Analysis

As we discussed in Section 2.4 PCA attempts to model data by a low-dimensional subspace that captures the directions of maximum variance, but it is notoriously sensitive to corrupted data. People are more interested in finding a robust subspace which is also known as robust PCA (RPCA). The detection and removal of such corrupted data is a key component of robust variants of PCA.

Consider as an example the case where $m = 2$ and $r = 1$. In this case the columns of the $2 \times n$ matrix X can be visualized as 2-dimensional points, and the matrix V consists of a single column vector that specifies a 2-dimensional direction. This is illustrated in Figure 2.3. The red line is the classical PCA applying on the whole dataset. While the green line is obtained by applying RPCA. RPCA identifies the outliers in the data and fits a line of all the non-outliers. It is clear that RPCA has smaller approximation error of the non-outliers than PCA.

To address RPCA problem some approaches propose modifications to the PCA error function in order to minimize the influence of outliers. A common approach is to replace the l_2 norm in error with l_1 norm (Maronna et al., 2018; Press et al., 2007). This is clearly less

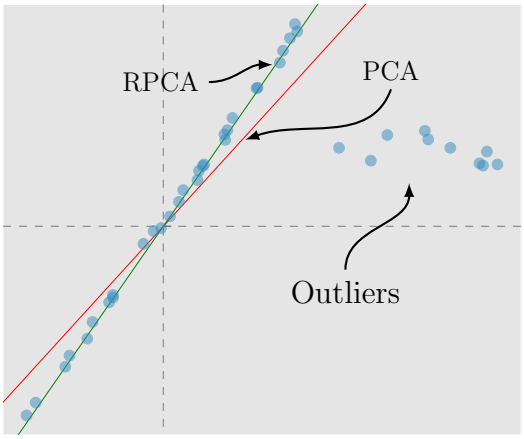


Figure 2.3: Robust Principal Components of points with outliers in 2D

effective than removing the outliers entirely where there is no influence of outliers on the PCA. There are two broad approaches which try to estimate the robust covariance matrix which can lead to robust principal components, M-estimator based approach (Maronna and Yohai, 2004; Campbell, 1980) and Projection Pursuit based approach (Li and Chen, 1985; Hubert et al., 2005). In these methods the computation does not remove the outliers but iteratively finds the solution which minimizes the impact of the outliers. M-estimator based methods are known to have problems dealing with high dimensional data (Maronna and Yohai, 2004). Another approach to robust PCA is via convex relaxation. In (Xu et al., 2010; Zhang et al., 2015) a convex relaxation of the RPCA problem is formulated and solved. As this approach does not directly solve the exact problem but rather solves a surrogate problem, it does not find the optimal solution to the original problem. It is also difficult to determine what is the sub-optimality bound of the achieved solution. In Chapter 4 we discuss our RPCA algorithm which gives optimal outliers and RPCA when the number of outliers is known.

2.5.1 Robustness and Outliers

Generally speaking there are two kinds of robustness. One is qualitative robustness, the other one is quantitative robustness (Huber, 2011; Daszykowski et al., 2007). The qualitative robustness aims to express the differences between two studied distributions. When either small changes in all of the observations or large changes in a few of them, the distribution has small change. The similarity between two distributions can be measured by means of the Prohorov distance (Daszykowski et al., 2007), Euclidean distance, etc. The quantitative robustness is used for describing how greatly a small change in the observation changes the estimator of distribution. This kind of robustness can be measured by breakdown point. The breakdown point is the smallest fraction of bad observations that may cause an estimator to take on arbitrarily large aberrant values. For example, the mean of a random variable has a breakdown point 0, while the median has a breakdown point 0.5 which is the maximum value of breakdown point. This means that the median is more robust than the mean.

There is no unanimous definition of outliers. In machine learning and statistics literature outliers are interpreted as the data points which appear to be inconsistent with the remainder of the dataset (Barnett and Lewis, 1974; Maronna et al., 2018). The interpretation of outliers changes depending upon the application domain (Chandola et al., 2009). Generally speaking, however, there are two kinds of outliers (Daszykowski et al., 2007). One is univariate. The other is multivariate. The univariate outliers are usually noise, e.g., the result of an experiment error. The multivariate outliers are harder to be distinguished than univariate outliers. The multivariate outliers usually have another pattern rather than the rest of the points. To identify multivariate outliers requires projection technique. In (Hubert et al., 2005) the authors introduce three different types of outliers with respect to projection and leverage score.

Hodge and Austin in (Hodge and Austin, 2004) provide a problem based classification of outlier detection methods,

Type 1 where we have no prior knowledge of which are the outlier and non-outlier data points. This is analogous to unsupervised learning.

Type 2 models both the outliers and the non-outlier data points, and requires pre-labeled outlier and non-outlier points to learn the distribution or model for both. This type is analogous to supervised learning.

Type 3 models only the non-outlier data points (or in some cases model just the outliers). This is analogous to semi-supervised learning where some of the non-outlier points are flagged *a priori*.

The work in (Chandola et al., 2009) provides a categorization of outlier detection methods based on the solution approach pursued. A class of outlier detection methods known as **spectral anomaly detection** uses a lower dimensional embedding to detect outliers under the assumption that in a low dimensional embedding of the data the outliers appear significantly different from the non-outlier points. PCA is one of the tools used for outlier detection (Jolliffe, 2011; Chandola et al., 2009). Outlier detection methods are called *robust* if the existence of outliers in the data used to compute the model for the non-outlier points does not distort this model in any way (Hodge and Austin, 2004; Rousseeuw and Leroy, 2005). As we saw in (2.7) the error of PCA is the sum of the l_2 norms of the approximation errors of individual data points the principal components are highly sensitive to outliers. This sensitivity is illustrated in Figure 2.3.

2.5.2 Two Variants

Recent studies have considered many robust variants of the PCA technique. See (Bouwmans et al., 2017) for a recent survey with nearly 500 references. There are two main camps: sparse-corruption and column-corruption. In the first one the corrupted data is entry-wise which is considered as a sparse matrix. As shown in (2.13) most studies set up the problem

(a) Original Image (b) Low Rank Image (c) Sparse Image

Figure 2.4: Example of RPCA for Background Subtraction

as that where the input dataset X is the sum of a low rank matrix L and a sparse matrix S, and convert the problem into an optimization problem. See e.g. (Candès et al., 2011; Chandrasekaran et al., 2011; Guo et al., 2014; Netrapalli et al., 2014; Peng et al., 2019).

$$X = L + S \tag{2.13}$$

The sparse-corruption variant of RPCA is widely use in computer vision such as subtracting background, eliminating noise. Figure 2.4 shows a background subtraction example on Waving Tree [1] dataset using a Non-Convex RPCA algorithm (Netrapalli et al., 2014).

Another group of RPCA is that the corruption is column of the data which is also coined outlier-based RPCA. This variant of RPCA reduces the impact of the column outliers (data points) in the dataset X. Let P, Q be the sets of non-outlier column indexes and outlier column indexes in X respectively. X_P and X_Q are the corresponding non-outliers and outliers in X. The outlier RPCA aims to get a robust low-dimensional subspace V of X_P. In a mathematical way the outlier-based RPCA minimizes the error in (2.14). Recall that Θ is a matrix norm, $V^T V = I_r$ and μ is the mean of X_P.

$$\Theta(X_P - \mu \mathbf{1}^T - VA) \tag{2.14}$$

[1] https://www.microsoft.com/en-us/download/details.aspx?id=54651

Figure 2.5: Outliers Detected by ROBPCA with $r = 3$ on Waving Tree Video

Figure 2.5 shows some of the frames from the Waving Tree dataset. Few frames where the person is present are outliers. The images with the red frame are the outliers detected by ROBPCA (Hubert et al., 2005) with respect to the first three principal vectors. In this dissertation we are focusing on the second variant of RPCA which is that the input matrix has column-corruption.

As shown in (2.14) the reconstruction error of column-corruption RPCA relates to the mean estimator μ. When collecting samples from datasets, it is hard to distinguish whether each sample is an outlier or not. In this case the RPCA is very sensitive to outliers because it minimizes the reconstruction error and few outliers with large errors dominate the reconstruction error. There are many studies addressing this problem. See e.g. (Xu et al., 2010, 2013; Hubert et al., 2005; Chen et al., 2016; Luo et al., 2016; Rahmani and Atia, 2017; Dong et al., 2019). The work in (Xu et al., 2010, 2013; Chen et al., 2016) considers the dataset having zero mean and the mean does not change after removing outliers. The studies in (Rahmani and Atia, 2017) use a coherence score to detect outliers and does not update the mean. Other studies in (Luo et al., 2016; Hubert et al., 2005; Dong et al., 2019) do not only get the robust low-dimension subspace but also achieve the robust mean estimator. In Chapter 4 and Chapter 5 we will have further discussion on several algorithms.

CHAPTER 3

HYBRID FEATURE SELECTION AND FEATURE EXTRACTION

In Chapter 2 we introduce two classical approaches to dimensionality reduction: 1. Approximating the data with a small number of features that exist in the data (feature selection, FS). 2. Approximating the data with a small number of arbitrary features (feature extraction, FE). In this chapter 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. This work was done in collaboration with Swair Shah, Crystal Maung, Guihong Wan and Gordon Arnold. The results were published in (Shah et al., 2018; He et al., 2019)

3.1 Problem being Addressed

As discussed in Chapter 2 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.

3.1.1 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:

$$\begin{aligned} X &\approx SA_1 + VA_2 \\ \text{error} &= \min_{S,V,A_1,A_2} \Theta(X - SA_1 - VA_2) \\ \text{s.t.} &\begin{cases} S \in \mathbb{R}^{m \times r_1}; S \subset X \\ V \in \mathbb{R}^{m \times r_2} \end{cases} \end{aligned} \tag{3.1}$$

where S consists of r_1 columns from X and the r_2 columns of V are unconstrained. We refer to the representation in (3.1) 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 3.2.1) 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.

The Main Contributions.

- A heuristic search algorithm for computing optimal and near optimal hybrid low rank (HLR).
- *A priori* and *a posteriori* bounds for these algorithms.
- 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 optimal feature selection algorithm for the spectral norm and for the nuclear norm.

3.2 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:

$$\begin{aligned} E_{\mathrm{FE}}(X, r) &= \min_{V,A} \Theta(X - VA) \\ &\quad \text{subject to } |V| = r \\ E_{\mathrm{FS}}(X, r) &= \min_{S,A} \Theta(X - SA) \\ &\quad \text{subject to } S \subset X,\ |S| = r \\ E_{\mathrm{HLR}}(X, r_1, r_2) &= \min_{S,A_1,V,A_2} \Theta(X - SA_1 - VA_2) \\ &\quad \text{subject to } S \subset X,\ |S| = r_1,\ |V| = r_2 \end{aligned} \tag{3.2}$$

From (3.2) it easily follows that with $r = r_1 + r_2$ we have:

$$E_{\mathrm{FE}}(X, r) \leq E_{\mathrm{HLR}}(X, r_1, r_2) \leq E_{\mathrm{FS}}(X, r) \tag{3.3}$$

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).

3.2.1 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 ($r_1 = 1$) and extract one feature ($r_2 = 1$). If optimal selection of one feature is applied to X_1, the best selection

(in Frobenius norm) is Column 3 (the error is $E_{\mathrm{FS}}(X_1, 1)$=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 $E_{\mathrm{HLR}}(X_1, 1, 1)$=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 $E_{\mathrm{HLR}}(X_2, 1, 1)$=18.8. This shows that optimal extraction followed by optimal selection does not guarantee the optimal HLR.

3.3 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).

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,

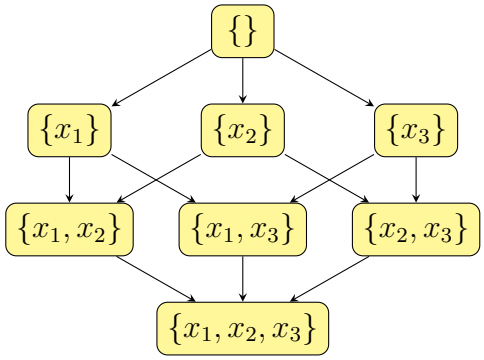


Figure 3.1: Example of the subsets graph

and we prove that the solution found by our algorithm comes with guaranteed bounds on its accuracy.

3.3.1 The Subsets 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 Figure 3.1. 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.

3.3.2 The Heuristic Search Algorithm

The algorithm in Figure 3.2 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

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 3.2: The best-first search algorithm

of the algorithm, when n_j is in the fringe). In our algorithm there is no such update. As explained in Section 3.3.1, 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.

3.3.3 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. We describe the

heuristic functions used to address this problem in (3.4).

$$
\begin{aligned}
E_{\mathrm{HLR}}(X, r_1, r_2) &= \min_{S, A_1, V, A_2} \Theta(X - SA_1 - VA_2) \\
&\quad \text{subject to} \;\; S \subset X, |S| = r_1, |V| = r_2 \\
d_i = d(n_i, r_1, r_2) &= \min_{S, A_1, V, A_2} \Theta(X - SA_1 - VA_2) \\
&\quad \text{subject to} \;\; S_i \subset S \subset X, |S| = r_1, |V| = r_2 \\
u_i = u(n_i, r_2) &= \min_{A_1, V, A_2} \Theta(X - S_i A_1 - VA_2) \\
&\quad \text{subject to} \;\; |V| = r_2 \\
f_i = f(n_i, r_1, r_2) &= \min_{A_1, V, A_2} \Theta(X - S_i A_1 - VA_2) \\
&\quad \text{subject to} \;\; |V| = r_1 + r_2 - k_i
\end{aligned}
\tag{3.4}
$$

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 $k_i = |S_i|$.

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 3.1.

Proposition 3.1. *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. ■

3.4 Unitarily Invariant Monotonic Functions

The FE, FS, and HLR were defined in (3.2) in terms of a general error criterion Θ. They are used in Section 3.3 to compute the heuristic functions f_i, u_i. The computation of f_i, u_i requires the solution of the following optimization problems: Given the matrices X and V compute efficiently:

$$\begin{aligned} V &= \arg\min_{V,A} \Theta(X - VA) \quad \text{subject to } |V| = r_2 \\ A &= \arg\min_{A} \Theta(X - VA) \end{aligned} \tag{3.5}$$

These optimization problems cannot be easily solved for arbitrary matrix norms. For example, if Θ is the l_1 entrywise matrix norm then the problem is known to be NP-hard (Gillis and Vavasis, 2018; Song et al., 2017). By contrast, as we point out here there is an efficient solution for a large family of error criteria, which include many commonly used norms.

Unitarily invariant matrix norms do not change their values if the matrix is rotated. Specifically, if Θ is a unitarily invariant norm then $\Theta(E) = \Theta(UEV^T)$ for any matrix E and orthogonal matrices U, V. It can be shown that a unitarily invariant norm can always be expressed as a monotonically non-decreasing function of its singular values. See e.g., (Marshall et al., 2011).

Thus, we have the following relation:

$$\Theta(A) = \theta(\sigma_1, \ldots, \sigma_m)$$

where $A \in \mathbb{R}^{m\times n}$, $\sigma_1 \geq \sigma_2 \ldots \geq \sigma_m$ are its singular values, and θ is monotonically increasing in all its arguments. We observe that the opposite is not true. There are monotonically increasing functions θ that do not correspond to norms. Here is a list of common error criteria and their description in terms of the function θ:

$$\begin{aligned}
&\text{Spectral:} && \theta = \sigma_1, \\
&\text{Frobenius:} && \theta = (\textstyle\sum_{j=1}^{m} \sigma_j^2)^{1/2}, \\
&\text{Nuclear:} && \theta = \textstyle\sum_{j=1}^{m} \sigma_j, \\
&\text{Schatten:} && \theta = (\textstyle\sum_{j=1}^{m} \sigma_j^p)^{1/p} \text{ for } p > 0
\end{aligned}$$

Clearly, these are all monotonically increasing functions. However, the Schatten p-norms are not matrix norms for $0 < p < 1$. See, e.g., (Tao, 2012). Also observe that both the Frobenius and the Nuclear norms are special cases of the Schatten p-norm but the Spectral norm is not.

Definition 3.1 (Unitarily Invariant Monotonic)**.** A matrix function Θ is called **unitarily invariant monotonic** if:

1. $\Theta(E) = \theta(\sigma_1, \ldots, \sigma_m)$, where σ_i are E singular values.

2. θ is nondecreasing in all of its arguments.

We are interested in unitarily invariant monotonic functions because of the properties given in Theorem 3.1.

Theorem 3.1. *Suppose $\Theta(E)$ is unitarily invariant monotonic. Let $X \in \mathbb{R}^{m\times n}$,$V \in \mathbb{R}^{m\times r}$, $A \in \mathbb{R}^{r\times n}$ be three matrices. Then:*

1. *If X and V are known then $A = V^+X$ minimizes $\Theta(X - VA)$, where V^+ is the pseudo-inverse of V.*

2. *If X is known then the matrix V consisting of the r eigenvectors corresponding to the largest eigenvalues of XX^T minimizes $\Theta(X - VA)$.*

Proof. Part 1 appears in the proof of Theorem B.7 in (Marshall et al., 2011). When Θ is a unitarily invariant norm Part 2 is the celebrated Eckart and Young theorem (Marshall et al., 2011). The extension to monotonic functions that are not norms is less well known. A proof can be found in Part b of Theorem 3.2 in (Mathar and Meyer, 1993). ■

Efficiently Computing f_i and u_i

Recall that:

$$f_i = f(n_i, r_1, r_2) = \min_{A_1, A_2, |V| = r_1 + r_2 - k_i} \Theta(X - S_i A_1 - V A_2)$$

where X and S_i are known. When Θ is unitarily invariant monotonic function the following procedure can be used to calculate f_i.

1. Compute $A_1 = \arg\min_{A_1} \Theta(X - S_i A_1)$. $\boxed{A_1 = S_i^+ X.}$

2. Compute $\boxed{X_1 = X - S_i A_1.}$

3. Compute $V = \arg\min_{|V| = r_1 + r_2 - k_i} \Theta(X_1 - V A_2)$ where $k_i = |S_i|$. The desired V is the $\boxed{\text{top } r_1 + r_2 - k_i \text{ eigenvectors of } X_1 X_1^T.}$

4. Compute $A_2 = \arg\min_{A_2} \Theta(X_1 - V A_2)$. $\boxed{A_2 = V^T X_1.}$

5. Compute $f_i = \Theta(X_1 - V A_2)$ from the singular values of $X_1 - V A_2$.

The calculation of u_i is essentially the same, where the only difference is the number of eigenvectors that are needed in Step 3.

The expensive part of the above procedure is the calculation of the eigenvectors (in Step 3) and singular values (in Step 5). These eigenvectors and singular values can be computed efficiently by observing the following. The eigensystem to be solved for the children of a

parent node (in Line 9 of the algorithm in Figure 3.2) is a rank-one update of the eigensystem calculated for the parent node. This can be implemented with specialized fast eigenvalue routines (Golub, 1973; Borges and Gragg, 1993).

3.5 The Three Variants of The Algorithm

Proposition 3.1 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 $\epsilon = \frac{1-\beta}{\beta}$, $\epsilon \geq 0$.

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

We prove in Theorem 3.2 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 3.3 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 3.4 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.

3.5.1 Proofs

Theorem 3.2. *With the choice $f'_i = u_i$, the algorithm terminates after examining r_1 nodes.*

Theorem 3.3. *With the choice $f'_i = f_i$, the algorithm terminates with an optimal solution. The optimal solution error is $E_{HLR}(X, r_1, r_2)$.*

Theorem 3.4. *Let n_* be an optimal solution node for the HLR. Let $e^* = E_{HLR}(X, r_1, r_2)$ 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:*

$$e^{**} \leq e^* + \epsilon(u_{\max} - f_{**}) \tag{3.6}$$

Lemma 3.1. *f_i is monotonically increasing along any path.*

Proof. 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:

$$\begin{aligned} f(n_j, r_1, r_2) &= \min_{|V_j|=r_1+r_2-k_i-1} \min_A \Theta(X - [S_i|x|V_j]A) \\ &\geq \min_{|V_i|=r_1+r_2-k_i} \min_A ([S_i|V_i]A) \\ &= f(n_i, r_1, r_2) \end{aligned}$$

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

Lemma 3.2. *The value of u_i is monotonically decreasing along any path.*

Proof. We need to show that if n_j is the child of n_i then $u_j \leq u_i$. From the definition in (3.4) 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 3.3. *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. The proof is by induction. Property **a.** follows trivially from Property **b.**. To prove Property **b.** observe that from Lemma 3.2, 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 3.4. *Suppose Theorem 3.4 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 3.4 can be written as follows: $e^{**} > e^* + \epsilon(u_{\max} - e^{**})$. Since both n_* and n_{**} are goal nodes Proposition 3.1 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 3.1 $f_* > f_z$. ■

Proof of Theorem 3.2:

Proof. The proof follows trivially from Lemma 3.3. ■

Proof of Theorem 3.3:

Proof. The proof follows as a corollary of Theorem 3.4 with $\epsilon = 0$. ■

Proof of Theorem 3.4:

Proof. If the theorem is false then from Lemma 3.4 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_{**}. ■

3.5.2 *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 3.1 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 3.1. 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 3.4 and Lemma 3.2 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.

3.5.3 Using *a posteriori* Bound 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 of this algorithm would either improve goal node or improve the *a posteriori* bound. Figure 3.3 describes the algorithm in detail.

3.6 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).

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 3.3: Optimistic Search Algorithm

There are many similarities between our model and the classical weighted A* graph search algorithm, see 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 , see 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 3.1 has a corresponding lemma in the classical theory, and our proof idea of Lemma 3.4 is similar (but not identical)

to the classical theory. However, there is no correspondence to our Proposition 3.1 (right hand side), Lemma 3.2, and Theorem 3.2. The bound obtained in Theorem 3.4 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.

3.7 Experimental Results

Running Time

Figure 3.4 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 3.1.1 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 3.1. 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

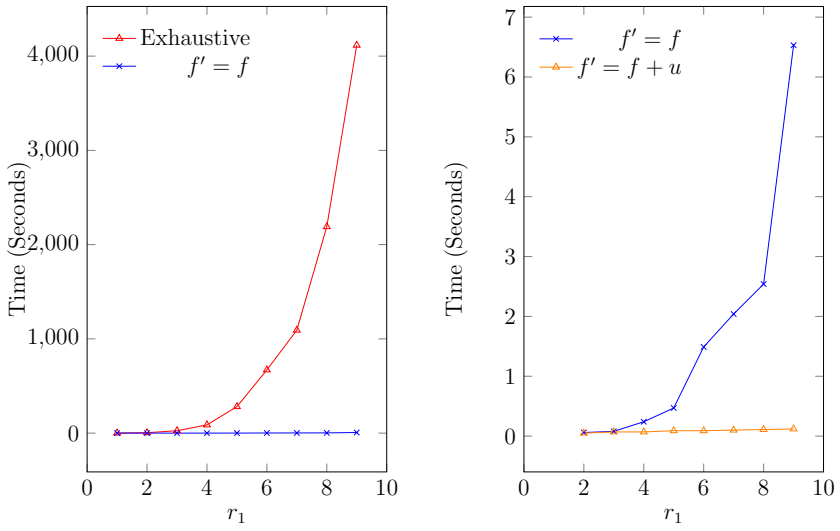


Figure 3.4: Running time of HLR on the dataset *vehicle*. x-axis shows r_1 and r_2=10 − r_1. Error criterion is the Schatten p-Norm with p=0.25.

Table 3.1: 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	183.42	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

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.

Table 3.2: Error of HLR with $r_1 + r_2 = 10$ on *vehicle* dataset

r_1	r_2	Schatten ($p = 0.5$)	Nuclear	Frobenius	Spectral
0	10	2987.72	414.46	169.57	99.75
2	8	2993.18	415.43	170.04	99.77
4	6	3014.01	418.66	171.52	100.38
6	4	3066.94	426.56	174.85	101.94
8	2	3138.24	437.36	178.44	102.51
10	0	3362.84	466.85	189.81	112.19

Comparison to Feature Selection and Extraction

Table 3.2 shows the accuracy comparison among the PCA, the HLR and the CSSP using $f' = f$ with different norm. In all cases we pick a total of $r_1+r_2 = 10$ columns to approximate the dataset X. One extreme is the case $\{r_1 = 0, r_2 = 10\}$, which corresponds to the PCA, and the other extreme $\{r_1 = 10, r_2 = 0\}$, is the CSSP. We can see that the PCA (feature extraction) gives the most accurate approximation of the data. The CSSP (feature selection) gives the least accurate results. The error of HLR is between those obtained by PCA and CSSP. This is a further example shown that by tuning the parameters r_1 and r_2 we can control the accuracy and robustness of the algorithm.

In Section 3.2.1 we show a simple example that an optimal combination of selected features and extracted features cannot be solved by sequentially obtained one and the other. Table 3.3 illustrates that the HLR cannot be solved by first performing the selection and then follow it by the PCA on two real datasets.

Minimizing Entry-wise l_0 and l_1 Norms

As discussed in Section 2.2 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,

Table 3.3: Errors of the HLR

Error Criterion	r_1	r_2	Optimal HLR	Optimal CSSP followed by PCA
vehicle dataset ($m = 846$, $n = 18$)				
Schatten ($p = 0.5$)	4	6	**3.014E2**	3.191E2
Nuclear	4	6	**4.187E2**	4.438E2
Frobenius	4	6	**1.715E2**	1.780E2
Spectral	4	6	**1.004E2**	1.028E2
Schatten ($p = 0.5$)	6	4	**3.067E2**	3.217E2
Nuclear	6	4	**4.266E2**	4.531E2
Frobenius	6	4	**1.748E2**	1.848E2
Spectral	6	4	**1.019E2**	1.095E2
Schatten ($p = 0.5$)	6	6	**1.258E2**	1.329E2
Nuclear	6	6	**2.292E2**	2.476E2
Frobenius	6	6	**1.063E2**	1.138E2
Spectral	6	6	**7.085E1**	7.317E1
spectf dataset ($m = 267$, $n = 45$)				
Schatten ($p = 0.5$)	2	3	**1.268E3**	1.299E3
Nuclear	2	3	**3.443E3**	3.547E3
Frobenius	2	3	**6.261E2**	6.528E2
Spectral	2	3	**2.035E2**	2.348E2
Schatten ($p = 0.5$)	3	2	**1.295E3**	1.318E3
Nuclear	3	2	**3.523E3**	3.670E3
Frobenius	3	2	**6.433E2**	6.841E2
Spectral	3	2	**2.070E2**	2.717E2
Schatten ($p = 0.5$)	3	3	**1.183E3**	1.201E3
Nuclear	3	3	**3.280E3**	3.399E3
Frobenius	3	3	**5.994E2**	6.283E2
Spectral	3	3	**1.918E2**	2.338E2

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 norms. Table 3.4 shows that for a fixed r, increasing r_1 indeed reduces the entry-wise l_0 and l_1 norms.

Table 3.4: 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

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 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 3.5. 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 3.5.3. The results are shown in Table 3.6. Observe that the solution error does not change, but the relative error bound is being reduced (slightly) with additional iterations.

Table 3.5: 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

Table 3.6: 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

3.8 Concluding Remarks

This chapter 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 listed, all with the same overall structure but each using different heuristics. *A priori* and *a posteriori* bounds are provided for suboptimal

and greedy variants. These two bounds show how close the results are to the optimal solution. We also shows how to use the *a posteriori* bound to improve the final output results.

The algorithm works with any unitarily invariant norm. By setting the extraction feature number to zero, the algorithm can be used to select the optimal features. This is the first optimal feature selection algorithm for the spectral norm and the nuclear norm.

CHAPTER 4

ROBUST PRINCIPAL COMPONENT ANALYSIS VIA OUTLIERS [1]

In Chapter 2 we discussed PCA, RPCA and the impact of outliers on RPCA. In this chapter we formulate RPCA problem in the same framework of heuristic search which is introduced in Chapter 3. Our formulation of Robust PCA gives a natural definition of outliers for PCA. We provide an optimal algorithm and a range of other faster variants with suboptimality guarantees. We compare it with the popular convex relaxation formulation. This work was done in collaboration with Swair Shah, Baokun He and Crystal Maung. The results were published in (Shah et al., 2017).

4.1 Problem Being Addressed

We consider the following optimization problem. The input is the matrix X of size $m \times n$, a number $k \leq n$ of outliers, and the desired number r of principal components. The output is the index subset P such that $|P| = n - k$, an orthogonal matrix V of size $m \times r$, and a matrix A of size $r \times (n - k)$. The output is computed to minimize the following error:

$$e(P, V, A) = \|X_P - VA\|_F^2 \tag{4.1}$$

In the above equation X_P is the matrix consisting of the columns of X with index values in P. Thus, the error in (4.1) evaluates how well the low dimensional model fits the non-outlier columns. The error is not affected by how accurate the model fits outlier vectors. It is easy to see that when Q, the set of k outlier columns is known, the values of $P, V.A$ that

[1] ©2017 IEEE. Portions adapted, with permission, from Shah, Swair, Baokun He, Crystal Maung, and Haim Schweitzer. "Computing Robust Principal Components by A* Search." In 2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI), pp. 1042-1049. IEEE, 2017.

minimize (4.1) can be easily calculated as follows:

$$\begin{aligned} &P(Q) = \{1, \ldots, n\} \setminus Q \quad (P \text{ is the complement of } Q) \\ &V(Q) \text{ is the matrix of the } r \text{ principal components of } X_{P(Q)}. \\ &A(Q) = \arg\min_A \|X_{P(Q)} - V(Q)A\|_F^2 = V^+ X_P \end{aligned} \tag{4.2}$$

where V^+ is the pseudoinverse of V. This shows that the error in (4.1) is a function of Q:

$$e(Q) = e(P(Q), V(Q), A(Q)) \tag{4.3}$$

Therefore, in this variant of robust PCA the challenge is to identify the outlier columns. Since there are $\binom{n}{k}$ possible choices of k column subsets, the exhaustive search approach of evaluating the errors of all such subsets and then selecting the subset with the smallest error is unacceptably slow. To the best of our knowledge this research is the first to propose an algorithm that is guaranteed to find the optimal solution to this problem, and runs significantly faster than exhaustive search.

4.2 Previous Approaches

Instead of removing the outliers, some robust estimation approaches attempt to minimize their influence on the predicted model. For example, in a line fitting task such as the one shown in Figure 2.3 one may attempt to evaluate the error in terms of absolute deviations (the l_1 norm) instead of squared deviations (the l_2 norm) as implied by the Frobenius norm that we use. See, e.g., (Maronna et al., 2006; Press et al., 2007).

A different technique of identifying outliers is based on "leverage scores". Let $X = V\Sigma U^T$ be the Singular Value Decomposition (SVD) of the matrix X. Then U is an $n \times m$ matrix with orthonormal columns. Let $u_1, \ldots, u_k$ be the columns of U corresponding to the largest singular values. Define the leverage score of Column i as:

$$\text{leverage\_score}(i) = \sum_{j=1}^{k} (u_j(i))^2 \quad \text{for } i = 1, \ldots, n \tag{4.4}$$

It is known that outlier columns have high leverage scores, so that the k largest leverage score values identify k outliers. See, e.g., (Hoaglin and Welsch, 1978; Chatterjee and Hadi, 1986).

Another approach to outlier detection for computing robust PCA is the Outlier Pursuit algorithm. It uses a relaxation of the optimization problem in (4.1) that leads to a convex optimization problem. See references (Xu et al., 2010; Zhang et al., 2015) for additional details. An experimental comparison between the outlier pursuit method and our approach is described in Section 4.6.

A recent approach to outlier detection for robust PCA is the "Coherence Pursuit", described in Reference (Rahmani and Atia, 2017). It is a very fast method that can be applied effectively to large datasets. Unfortunately, what they consider to be outliers is different from what is considered an outlier in previous work. They ignore vector lengths and their measure of coherence is related to the average Cosine of the angle between a vector and all other vectors. As such, their accuracy results cannot be compared with ours or the other methods reviewed here.

4.3 The Main Tools

Our algorithms are based on the classical A* algorithm with heuristics that require an efficient computation of eigenvalues. These tools are reviewed in this section.

A* algorithms for searching graphs.

A* search is a well known heuristic search algorithm for graphs. See, e.g., (Hart et al., 1968; Pearl, 1984; Russell and Norvig, 2010). In its standard formulation the algorithm computes a path between two given nodes, guided by a heuristic function h_i that can be computed at each node n_i. The algorithm expands nodes according to a criterion f_i, which is computed

from h_i by the following formula:

$$f_i = g_i + h_i$$

In the above formula g_i is the distance between the initial node and n_i, and the heuristic function h_i is problem specific. The optimality of A* depends on the particular properties of h_i. Consistency (see references above for the definition) of h_i guarantees optimality in a fast variant where each graph node is visited at most once. The weaker condition of admissibility (see references above for the definition) is sufficient to guarantee optimality in a slower variant that may visit the same node multiple times.

Weighted A* algorithms for searching graphs.

The only difference between the A* algorithm and the weighted A* algorithm is a slightly different method of combining the heuristics. Specifically, the algorithm expands nodes according to a criterion f_i' which is given by the following formula:

$$f_i' = f_i + \epsilon h_i = g_i + (1 + \epsilon)h_i$$

For $\epsilon > 0$ the weighted A* algorithm is not guaranteed to find an optimal solution, but it typically runs significantly faster than the A* algorithm. It is guaranteed to find a solution within $(1 + \epsilon)$ of the optimum. See, e.g., (Pearl, 1984).

Rank-one update eigenvalue calculations.

The heuristic evaluations in our algorithms use eigenvalues. Let X_0 be an $m \times k$ matrix, and let X_1 be an $m \times (k-1)$ matrix, constructed by removing the single column x from X_0. Suppose we are given the eigenvalues and the eigenvectors of the matrix $B_0 = X_0 X_0^T$. The heuristics in our algorithms require efficient calculations of the eigenvalues of $B_1 = X_1 X_1^T = B_0 - xx^T$.

This rank-one update problem has attracted a lot of attention. It was shown in (Golub, 1973) that the eigenvalues of B_1 are roots of a "secular" equation that can be constructed from the eigenvalues and the eigenvectors of B_0 and x. Studies of efficient numeric procedures for computing the roots of the secular equation include (Bini and Robol, 2014; Bunch et al., 1978; Melman, 1998; Borges and Gragg, 1993). In our experiments we use the Gragg method, as described in (Melman, 1998; Borges and Gragg, 1993).

4.4 Our Approach

As shown in Equation (4.3) it is possible to assign an error value to each column subset of X by considering that subset to be the outliers. We use the similar frame to the one used in Chapter 3. The main idea is to consider a graph that describes the relationship between these subsets, and then perform a graph search for a subset of size k that has the smallest error. The subset graph that we create is the same as the one used in (Arai et al., 2015, 2016). The search techniques that we propose are also similar to those described in these studies. Specifically, we use a variant of the A* algorithm to compute the optimal solution, and a variant of the weighted A* algorithm to compute a solution with guaranteed bounds on sub-optimality. The main difference between these previous studies and the current work is the heuristics that are being used. We propose heuristic functions that are specifically designed to compute the outliers, and provide proofs of optimality and sub-optimality for the algorithms that use these heuristic functions.

4.4.1 The Subset Graph

We use the column subset graph described in Figure 3.1 of Chapter 3 again. The only difference is that in this case the nodes in the graph contain subsets of data points instead of subsets of features. There is an edge from Subset Q_i to Subset Q_j if adding a single data point to Q_i creates Q_j. 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 $\{\} \to \{x_1\} \to \{x_1, x_3\}$ or by the path $\{\} \to \{x_3\} \to \{x_1, x_3\}$. 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.

4.4.2 The A* Algorithm

The algorithm in Figure 4.1 performs search for column subsets. It is a generic version of the A* algorithm that uses a closed nodes list (and thus visits each node at most once). The algorithm maintains two lists: the list L contains nodes to be examined, and the list C contains nodes which are marked as closed and do not need to be visited again. As we show in Section 4.4.3, by properly defining the function f' the algorithm finds the desired outliers.

4.4.3 Heuristic Functions

In this section we describe the heuristic functions that are needed by the A* algorithm in order to compute the outliers. In typical usage of A* it is enough to define the function h_i and then combine it with g_i to construct f_i. In our case there is no obvious choice for g_i and h_i. Instead, we provide a definition for g_i and f_i without using h_i. Since our definition of the heuristic functions is nonstandard, the optimality of the algorithm does not follow from the known theory and must be proved explicitly.

Recall that at the node i of the graph there is a column subset Q_i of k_i columns. We proceed to define the values of f_i and g_i. To simplify notation we write $|M|$ for the number

Input: X,k,r. **Output:** a column subset Q.
Each node n_i has a subset Q_i of size k_i. The algorithm maintains two lists: the fringe list L, and the closed nodes list C.
Initialization: Put a node with an empty subset into L.

```
while L is nonempty do
    Pick n_i with the smallest f'_i from L.
    if k_i = k then
        Stop and return n_i with Q_i as the solution subset.
    else
        Add n_i to C.
        for each child n_j of n_i do
            if n_j is not in C or L then
                Compute f'_j and put n_j in L.
            end
        end
    end
end
```

Figure 4.1: The generic A* algorithm for column subsets

of columns of a matrix M. Define:

$$\begin{aligned} P_i &= \{1, \ldots, n\} \setminus Q_i \quad (P_i \text{ is the complement of } Q_i) \\ g_i &= \min_{V,A} \|X_{P_i} - VA\|_F^2, \text{ subject to } |V| = r \\ f_i &= \min_{U,A} \|X_{P_i} - UA\|_F^2, \text{ subject to } |U| = r + k - k_i \end{aligned} \tag{4.5}$$

In the above equation X_{P_i} is the matrix created from the columns of X in the subset P_i. Using these definitions the value of f'_i is defined as:

$$f'_i = f_i + \epsilon g_i, \quad \epsilon \geq 0 \tag{4.6}$$

Clearly, the definition of the functions f_i and g_i in Equation (4.5) is different from the definition of these functions in the classical A*. We point out the properties that these functions share with the f_i, g_i that are used in the classical A* algorithm. These properties motivate our notation. The proof of these properties is given in Section 4.5.

- f_i is monotonically increasing along any path.
- If f_i is used as f'_i in the algorithm, the optimal (minimum error) solution is found.
- Defining f'_i as in (4.6) and selecting nodes with the smallest f'_i gives a solution with performance guarantees.
- At a goal node $f_i = g_i$.

There is, however, a major difference between our definition of g_i and the way it is defined in the classical theory of A* search. With the classical definition of g_i as the distance from the root to n_i, the value of g_i is monotonically increasing along any path. By contrast, we will show that the value of g_i in our algorithm is monotonically decreasing along any path. This also affects the values of f'_i that are used for the weighted A*. In the classical theory these values are monotonically increasing along any path, but in our case they are neither monotonically increasing nor monotonically decreasing. We proceed to show that using f'_i as defined in (4.6) guarantees sub-optimality. The optimality result for f will follow as the special case when $\epsilon = 0$.

4.5 Optimality and Suboptimality Theorems

We begin with the suboptimality proof for the case in which f' is used with $\epsilon \geq 0$. The optimality proof is obtained as a simple corollary when setting ϵ to 0.

Theorem 4.1. *The Suboptimality Theorem: Let n_* be an optimal solution node with the corresponding values: Q^*, P^*, V^*, A^*, that minimize the error in (4.3). Suppose the algorithm in Figure 4.1 terminates at a node n_{**} with the corresponding values: $Q^{**}, P^{**}, V^{**}, A^{**}$. Then:*

$$\|X_{P^{**}} - V^{**}A^{**}\|_F^2 \leq \|X_{P^*} - V^*A^*\|_F^2 + \epsilon g_{max} \tag{4.7}$$

where $g_{max} = \max_i g_i$, the value of g at the root (initial) node.

The proof will be given in terms of Lemma 4.4 stated below. Lemma 4.2 and Lemma 4.3 are used in the proof of Lemma 4.4. Lemma 4.1 is used in the proof of Lemma 4.2. In the proofs we use the following notation. We write $|M|$ for the number of columns of a matrix M, and $[M|x]$ for the matrix formed by appending a column x to the matrix M.

Lemma 4.1. *Let $[X|x]$ be the matrix formed by appending a column x to the matrix X, and let $[U|u]$ be the matrix formed by appending a column u to the matrix U. Then for any column vector x, and any $t \geq 0$:*

$$\min_{A,|U|=t} \|X - UA\|_F^2 \geq \min_{A',u,|U|=t} \|[X|x] - [U|u]A'\|_F^2$$

Proof. Let A_1, U_1 be the minimizers of the left hand side. Consider the assignment of values to the right hand side that does not necessarily minimizes it: $U = U_1$, $u = x$, and $A' = \begin{pmatrix} A_1 & 0 \\ 0 & 1 \end{pmatrix}$. Then manipulating the right hand side we get:

$$\begin{aligned} &\min_{A',u,|U|=t} \|[X|x] - [U|u]A'\|_F^2 \\ &\qquad \leq \|[X|x] - [U_1|x]A'\|_F^2 \\ &\qquad = \|[X|x] - [U_1A_1|x]\|_F^2 \\ &\qquad = \|[X - U_1A_1\|_F^2 \\ &\qquad = \min_{A,|U|=t} \|X - UA\|_F^2 \end{aligned}$$

■

Lemma 4.2. *f_i is monotonically increasing along any path.*

Proof. We need to show that if n_j is the child of n_i then $f_j \geq f_i$. Plugging in the definition of f_i as given in (4.5) we need to show:

$$\min_{A,|U|=r+k-k_j} \|X_{P_j} - UA\|_F^2 \geq \min_{A',|U'|=r+k-k_i} \|X_{P_i} - U'A'\|_F^2$$

Since n_j is the child of n_i the following two properties hold: $k_j = k_i + 1$, and $[X_{P_j}|x] = X_{P_i}$, where x is a column of X. Therefore, we need to prove:

$$\min_{A,|U|=r+k-k_i-1} \|X_{P_j} - UA\|_F^2 \geq \min_{A',|U'|=r+k-k_i} \|[X_{P_j}|x] - U'A'\|_F^2$$
$$= \min_{A',u,|U|=r+k-k_i-1} \|[X_{P_j}|x] - [U|u]A'\|_F^2$$

This follows from Lemma 4.1 with $t = r + k - k_i - 1$. ■

Lemma 4.3. *For every goal node n_i the following holds: $g_i = f_i$.*

Proof. At a goal node $k_i = k$, and the lemma follows trivially from (4.5). ■

Lemma 4.4. *Suppose Theorem 4.1 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 assumption that the theorem is false can be written as $g_{**} > g_* + \epsilon g_{\max}$. The claim can now be proved by the following chain of equalities / inequalities.

$$\begin{aligned}
f'_{**} &= f_{**} + \epsilon g_{**} = g_{**} + \epsilon g_{**} && \text{(from Lemma 4.3)} \\
&> g_* + \epsilon g_{\max} + \epsilon g_{**} && \text{(from the assumption)} \\
&= f_* + \epsilon g_{\max} + \epsilon g_{**} && \text{(from Lemma 4.3)} \\
&\geq f_z + \epsilon g_{\max} && \text{(from Lemma 4.2)} \\
&\geq f_z + \epsilon g_z \\
&= f'_z
\end{aligned}$$

■

Proof of The Suboptimality Theorem

Proof. If the theorem is false then from Lemma 4.4 it follows that all nodes on the path from the root to n_* 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 n_{**} is selected. But this means that n_* is selected as the solution and not n_{**}. ■

Clearly, The Suboptimality Theorem implies that running the algorithm with $\epsilon = 0$ produces an optimal solution. We state this result as "the optimality theorem."

Theorem 4.2. *The Optimality Theorem: Let n_* be an optimal solution node with the corresponding values Q^*, P^*, V^*, A^* that minimize the error in (4.3). Suppose the algorithm in Figure 4.1 is applied with $\epsilon = 0$ and the algorithm terminates at a node n_{**} with the corresponding values $Q^{**}, P^{**}, V^{**}, A^{**}$. Then:*

$$\|X_{P^{**}} - V^{**}A^{**}\|_F^2 = \|X_{P^*} - V^*A^*\|_F^2$$

We proceed to prove some additional results about g_i. They motivate a variant of our algorithm in which we take $f_i' = g_i$, or, equivalently, when the value of ϵ in Equation (4.6) approaches infinity.

Lemma 4.5. *The value of g_i is monotonically decreasing along any path.*

Proof. We need to show that if n_j is the child of n_i then $g_j \leq g_i$. Plugging in the definition of g_i as given in (4.5) we need to show:

$$\min_{A,|V|=r} \|X_{P_j} - VA\|_F^2 \leq \min_{A',|V'|=r} \|X_{P_i} - V'A'\|_F^2 = \min_{A',|V'|=r} \|[X_{P_j}|x] - V'A'\|_F^2$$

where x is the column removed from X_{P_i} to obtain X_{P_j}. The claim follows from the fact that the right hand side is the error in estimating the same matrix that appears in the left hand side plus an additional column vector. ■

Theorem 4.3. *The Dual Bound Theorem: Let d_i be the value of the best goal node below n_i. Then:*

1. *$f_i \leq d_i \leq g_i$ for every i:*

2. *At every goal node: $f_i = d_i = g_i$.*

Proof. 1 follows from Lemma 4.2 and Lemma 4.5. 2 follows from 1 and Lemma 4.3. ■

Running time.

The running time of the algorithm depends heavily on the calculation of the heuristic values f_i, g_i, as defined in (4.5). In this section we show that the computation involves estimating sums of eigenvalues, and can be done efficiently for the problem at hand. Recall that the function f_i is defined as follows:

$$f_i = \min_{U,A} \|X_{P_i} - UA\|_F^2, \text{ subject to } |U| = r + k - k_i$$

Without loss of generality we can assume that U is an orthogonal matrix, which gives the following formula for the minimizer: $A = U^T X_{P_i}$. With the definition $B_i = X_{P_i} X_{P_i}^T$, and exploiting the relationship between the Frobenius norm and the trace operator one can derive the following expression:

$$\begin{aligned} f_i &= \text{trace}\{(X_{P_i} - UU^T X_{Pi})(X_{P_i} - UU^T X_{Pi})^T\} \\ &= \text{trace}\{B_i\} - \text{trace}\{U^T B_i U\} \end{aligned}$$

These traces can be expressed in terms of the eigenvalues of B_i:

$$\text{trace}\{B_i\} = \sum_{t=1}^{m} \lambda_t, \quad \text{trace}\{U^T B_i U\} = \sum_{t=1}^{r+k-k_i} \lambda_t$$

Therefore:

$$f_i = \sum_{t=r+k-k_i+1}^{m} \lambda_t$$

The most expensive computation step of the algorithm is in Line 9, where the function f_j' needs to be computed for all the children of n_i. This requires computing eigenvalues of many matrices of the form: $B_j = X_{P_j} X_{P_j}^T = B_i - x_j x_j^T$ where x_j is a column of X. As discussed in Section 4.3 this can be done efficiently using rank-one update algorithms.

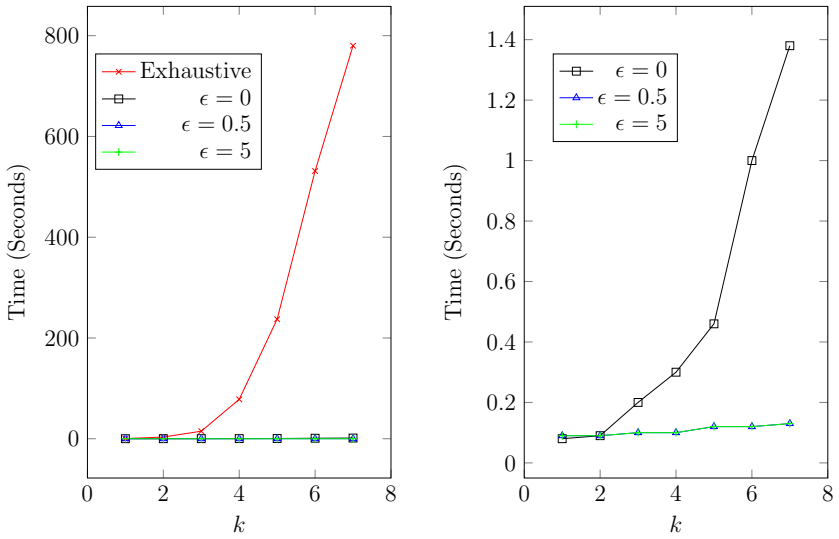


Figure 4.2: Run-time results on the dataset *vehicle* with $r = 3$.

4.6 Experimental Results

This section describes experimental results with our RPCA algorithms. The experiments were performed on various datasets and compared with results obtained with two competitors: the Outlier Pursuit algorithm as described in references (Xu et al., 2010; Zhang et al., 2015), and the Leverage Score algorithm as described in references (Hoaglin and Welsch, 1978; Chatterjee and Hadi, 1986). The code for the Outlier Pursuit algorithm was obtained from the authors. The data we use includes a simple toy example, datasets from the UCI Machine Learning repository (Frank and Asuncion, 2010), datasets of facial images taken from the Yale Face Database (Georghiades et al., 1997), and synthetic datasets where the number of outliers is known.

Running Time

Figure 4.2 shows running-time on the dataset *vehicle*. The left panel shows that the algorithm with $f_i' = f_i$ (optimal variant) is significantly faster than exhaustive search. The right panel shows that using $f_i + \epsilon g_i$ runs much faster than f_i.

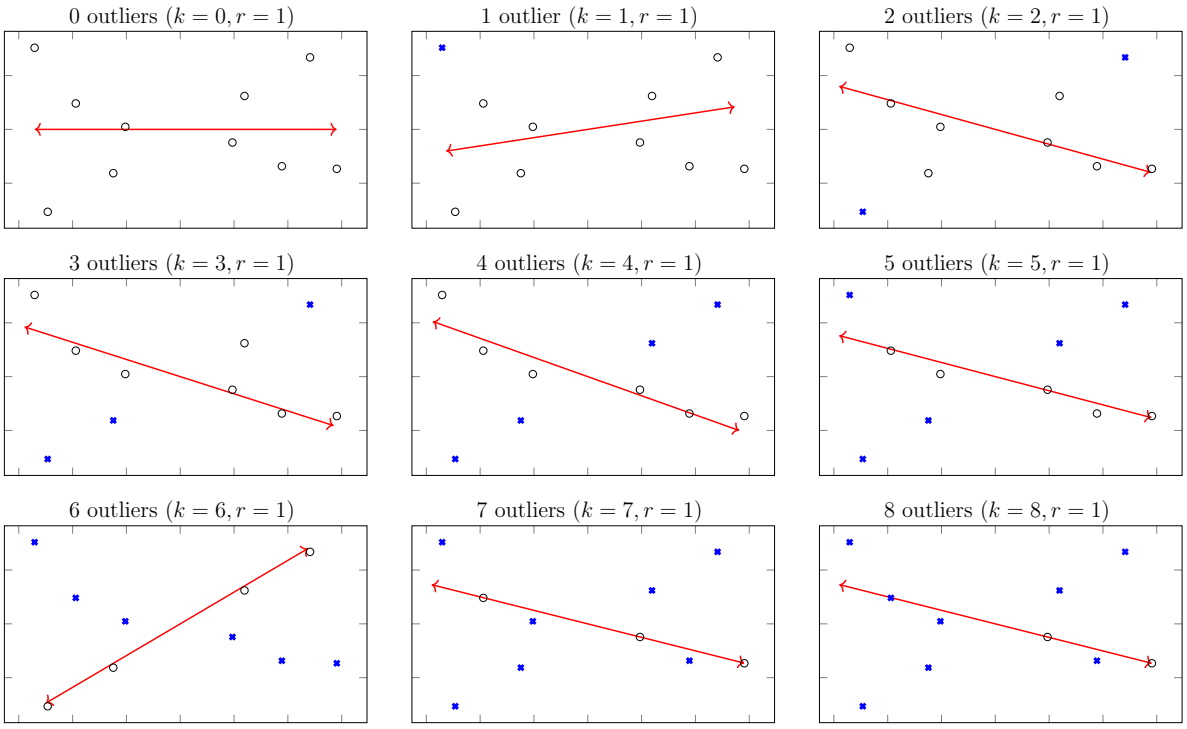


Figure 4.3: A toy example of data consisting of 10 points, and the first principal component is computed after optimally removing outliers. Data points are marked with an "o", and outliers with an "x".

A Toy Example

The first experiment that we describe is a toy example consisting of 10 points. It is intended to show that optimally identifying outliers involves combinatorial search, since each point may sometimes be considered an outlier depending on the total number of outlier points.

The results are shown in Figure 4.3. The data consists of 10 points which roughly show a line from the top left to the bottom right, and another line from the bottom left to the top right. As shown, without outlier removal, or even with the removal of one outlier, the principal component direction is some average between the two directions. The removal of 2,3,4,5 outliers allows the algorithm to determine the top-left to bottom-right direction as the principal component direction, since it has more points that the bottom-left to top-right direction. However, with 6 outliers the algorithm determines the principal component

Table 4.1: Reduction of average error with the increase in number of outliers

lung cancer dataset ($m = 27$ $n = 57$)		
k	$A_e(k)$ for $r = 2$	$A_e(k)$ for $r = 3$
0 (baseline)	15.438	13.217
1	15.133	12.828
2	14.845	12.455
4	14.213	11.736
6	13.580	10.982
8	12.836	10.081

direction to be the bottom-left to top-right direction, since the points in that direction fit a line very accurately. Further increasing the number of outliers gives different choices. In particular, when only two non-outlier points are left the principal component approximation error is 0.

Effectiveness

As discussed in Section 4.1 robust PCA improves the model accuracy for the non-outlier points. The experiment described in this section demonstrates this on a real dataset from the UCI repository. Table 4.1 shows the error averaged over non-outlier points as a function of k, the number of outliers. The average error $A_e(k)$ was computed with the following formula:

$$A_e(r,k) = \min_{A,|V|=r} \frac{\|X_P - VA\|_F^2}{(n-k)}$$

where the columns of X_P are the $n-k$ non-outlier points.

The results in Table 4.1 were computed with the optimal algorithm ($\epsilon = 0$). The values of $A_e(k)$ for $r = 2$ and $r = 3$ are shown for various values of k. The first row for $k = 0$ corresponds to the baseline error, when none of the outliers are removed. Clearly, $A_e(k)$ is reduced with the increase in k, giving an improved principal components data model.

Accuracy and Speed Comparison

In the experiment described in this subsection we investigate the effect of ϵ on the running time and on the accuracy of our algorithms. These results are compared to the results obtained by the Outlier Pursuit and the Leverage Score methods. The Outlier Pursuit algorithm takes as input the data matrix X and a parameter λ, see the discussion in Reference (Xu et al., 2010). It returns two matrices L and C, where the left singular vectors of L form the underlying subspace for non-outlier points and the columns of C contain the outliers. As described in Reference (Xu et al., 2010) we take the columns of C with the k largest l_2 norms as the outliers. The error that we report is the normalized error computed by:

$$\text{normalized error} = \min_{A,|V|=r} \frac{\|X_P - VA\|_F^2}{\|X\|_F^2} \tag{4.8}$$

where the columns of X_P are the $n-k$ non-outlier columns. The time is measured in seconds. When the algorithm running time exceeds 100 seconds we terminate the run and indicate it by a "$-$" in Table 4.2.

As shown in the table the optimal algorithm ($\epsilon = 0$) always has the least error, but it takes more time to run than the other algorithms. For example, on the dataset *vehicle* with the parameters $k = 10, r = 5$, Outlier Pursuit gets a result that is 5 times worse than the optimal result. However, the Outlier Pursuit algorithm takes 0.65 seconds, while our algorithm with optimal setting ($\epsilon = 0$) takes 9.07 seconds to produce the optimal solution.

The suboptimal algorithms are much faster than the optimal algorithm, and typically achieve an error close to the optimal. This error is typically much better than the result of Outlier Pursuit. For example, for the dataset *vehicle* with $k = 10, r = 2$, the Outlier Pursuit result has twice the error of the suboptimal algorithm with $\epsilon = 2$. For this case the suboptimal algorithm runs three times faster than Outlier Pursuit. In most of the cases that we have investigated the suboptimal algorithms beat Outlier Pursuit in terms of both accuracy and speed. We observe that the Leverage Score method is by far the fastest, as

Table 4.2: Accuracy and time for the optimal algorithm and the sub-optimal algorithm, compared to Outlier Pursuit (Xu et al., 2010) and the Leverage Score method. The time is measured in seconds, and the error is the normalized error. The minimum error is highlighted.

k	r	Optimal / time	Sub-optimal $\epsilon = 2$ / time	Sub-optimal $\epsilon = 5$ / time	Sub-optimal $\epsilon = 10$ / time	Outlier Pursuit / time	Leverage Score Method / time
vehicle dataset ($n = 18$, $m = 846$)							
5	2	**5.790E-04** / 1.01	5.910E-04 / 0.22	5.812E-04 / 0.22	**5.790E-04** / 0.22	8.098E-04 / 0.65	8.230E-04 / 0.01
5	3	**3.121E-04** / 0.69	3.442E-04 / 0.22	3.493E-04 / 0.22	3.493E-04 / 0.22	5.604E-04 / 0.64	5.871E-04 / 0.01
10	2	**1.227E-04** / 30.83	**1.227E-04** / 0.34	**1.227E-04** / 0.23	**1.227E-04** / 0.23	2.602E-04 / 0.65	4.138E-04 / 0.01
10	3	**5.820E-05** / 20.91	**5.820E-05** / 0.24	**5.820E-05** / 0.23	**5.820E-05** / 0.23	1.332E-04 / 0.65	2.506E-04 / 0.01
5	5	**9.842E-05** / 0.36	**9.842E-05** / 0.21	**9.842E-05** / 0.22	**9.842E-05** / 0.22	2.379E-04 / 0.64	2.509E-04 / 0.01
10	5	**8.550E-06** / 9.07	8.735E-06 / 0.22	8.735E-06 / 0.22	8.735E-06 / 0.22	4.898E-05 / 0.65	7.723E-05 / 0.01
spectf dataset ($n = 45$, $m = 267$)							
3	2	**1.042E-02** / 2.33	**1.042E-02** / 0.53	**1.042E-02** / 0.26	**1.042E-02** / 0.25	**1.042E-02** / 0.64	1.125E-02 / 0.01
4	2	**9.996E-03** / 84.01	**9.996E-03** / 0.77	**9.996E-03** / 0.26	**9.996E-03** / 0.26	1.013E-02 / 0.66	1.096E-02 / 0.01
10	3	-	6.128E-03 / 0.31	**6.113E-03** / 0.28	**6.113E-03** / 0.29	6.171E-03 / 0.65	7.340E-03 / 0.01
15	3	-	**4.768E-03** / 0.31	**4.768E-03** / 0.30	**4.768E-03** / 0.31	4.806E-03 / 0.65	6.201E-03 / 0.01
15	10	-	1.734E-03 / 0.32	1.734E-03 / 0.32	**1.713E-03** / 0.33	1.841E-03 / 0.65	2.239E-03 / 0.01
20	10	-	1.125E-03 / 0.34	**1.116E-03** / 0.34	**1.116E-03** / 0.34	1.150E-03 / 0.64	1.626E-03 / 0.01
libras dataset ($n = 90$, $m = 360$)							
4	3	-	-	**4.011E-02** / 93.76	**4.011E-02** / 0.83	4.166E-02 / 6.16	4.292E-02 / 0.01
10	3	-	-	**3.189E-02** / 1.28	**3.189E-02** / 0.92	3.550E-02 / 6.27	3.995E-02 / 0.01
10	4	-	-	**2.033E-02** / 47.23	**2.033E-02** / 0.92	2.198E-02 / 6.22	2.411E-02 / 0.01
15	4	-	-	-	**1.770E-02** / 1.02	2.009E-02 / 6.24	2.150E-02 / 0.01
15	10	-	-	**1.471E-03** / 1.06	**1.471E-03** / 0.98	1.769E-03 / 6.18	2.390E-03 / 0.01
20	10	-	-	**1.060E-03** / 1.11	**1.060E-03** / 1.11	1.469E-03 / 6.51	2.166E-03 / 0.01

it only requires one singular value decomposition. However, it typically has worse accuracy when compared to the other algorithms.

Experiments with Synthetic Dataset

In this experiment we use synthetic data generated with known parameters. We follow the procedure described in Reference (Xu et al., 2010) to generate datasets with a specified rank for the non-outlier points and a specified number of outliers. In all cases we took the number of outliers to be 15. The experiments compare the performance of our suboptimal algorithm with $\epsilon = 10$ to the performance of the Outlier Pursuit algorithm. The reported error is the normalized error as defined in Equation (4.8). The results are shown in Fig 4.4.

The results show that our suboptimal algorithm always finds the right outliers, and its error is always below the error of Outlier Pursuit. The advantage in accuracy of the suboptimal algorithm over the Outlier Pursuit is minimal when the data can be described

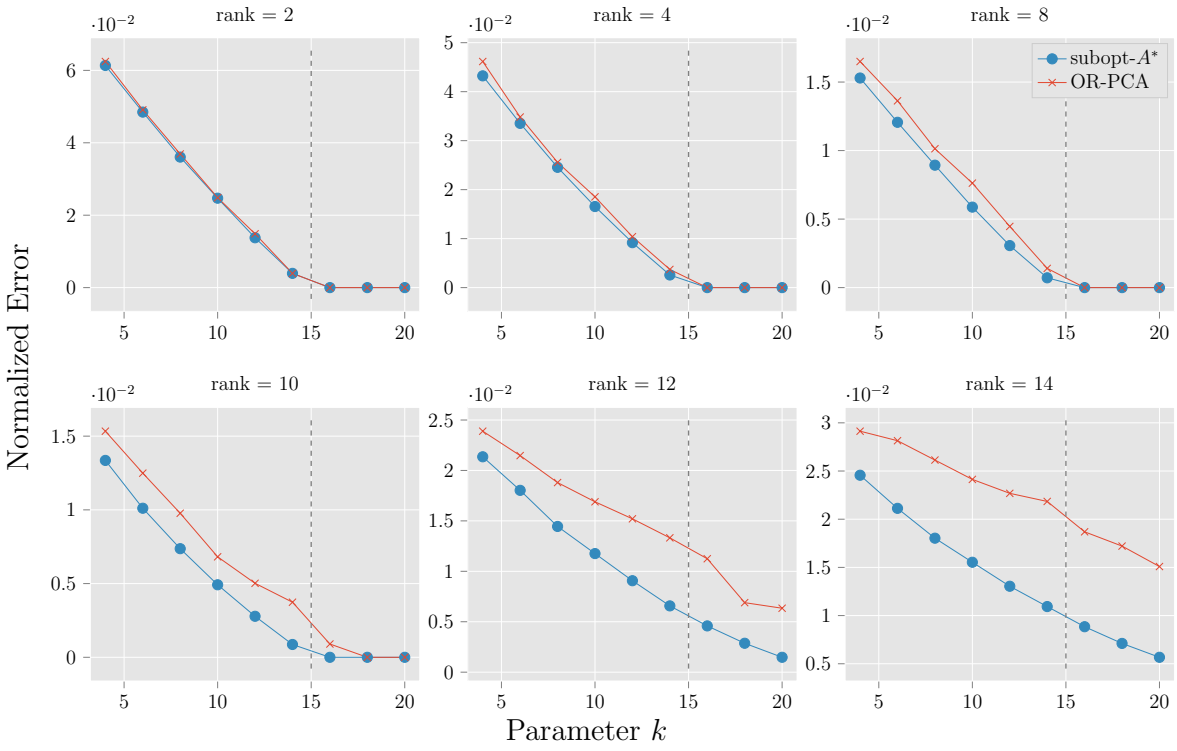


Figure 4.4: X is of shape 50×50. Each dataset has 15 outliers and rank of non-outlier points is displayed above each graph. The sub-optimal algorithm uses $\epsilon = 10$ for this experiment.

by a very low rank matrix, but becomes very significant when the data requires higher rank matrices. In all these experiments the suboptimal algorithm also had a significant running time advantage over the Outlier Pursuit. It ran at least three times faster than Outlier Pursuit in all the experiments.

Results on Datasets of Facial Images

The fourth experiment was performed on a dataset of image faces. The underlying assumption is that images of different orientations of the same person lie on a low dimensional subspace. Similarly, images of the same lie approximately on a low dimensional space (Murase and Nayar, 1995; Turk and Pentland, 1991). We use the Yale Face Database (Georghiades et al., 1997) to create 3 datasets, each with 12 face pictures. Out of those the non-outlier

Figure 4.5: Face Dataset 1. Results with $r = 2$ and $k = 4$.

Figure 4.6: Face Dataset 1. Top: Results for $r = 3$ and $k = 4$, Bottom : Results for $r = 3$ and $k = 6$

Figure 4.7: Face Dataset 2 (top) and Dataset 3 (bottom). Successful run on face graphs/rpca with $r = 3$. We use $k = 3$ in the top experiment and $k = 5$ in the bottom one.

columns are pictures of the same person under different poses, illumination and sometimes with and without glasses. The outlier pictures are pictures of other individuals. The task in this case is to detect the pictures of the different individuals as outliers. We use our algorithms with different values of k and r and show that we successfully detect the outliers.

The results are displayed in Fig 4.5, Fig 4.6, and Fig 4.7. Images with red borders are outliers detected by our algorithm, and the rest of the images are used to form the robust

principal components. Fig 4.5 shows the result on first dataset with $r = 2, k = 4$. We observe that our algorithm finds all the outliers. Fig 4.7 shows the successful detection of outliers on Dataset 2 (top) and on Dataset 3 (bottom) with the parameters $r = 3, k = 3$ and $r = 3, k = 5$ respectively. Note that the bottom image of Fig 4.7 has 4 outliers, and setting $k = 5$ picks outliers, thus even with overshooting the value of k does successfully recover the subspace underlying non-outlier points. This can also be seen in Fig 4.6. In the top image for $k = 4, r = 3$ we can see that two images out of four detected images are not outliers. Thus, we can still recover the linear subspace of non-outliers by overshooting parameter k. Setting $k = 6$ gives the result in the bottom image of Fig 4.6.

4.7 Concluding Remarks

Outlier RPCA is an important problem in data analytics and machine learning. We take a combinatorial approach to this problem and pose Outlier RPCA as a search problem on a graph representing the subsets of columns of the data matrix. We use the classical A^* algorithm to find the column subset of outliers of a given data matrix.

The A^* heuristic search approach finds the optimal solution while evaluating significantly fewer subsets than exhaustive search. For example, if exhaustive search is attempted to select 4 columns from the *libras* dataset, we estimate the running time to be about one month. This is solved in about five minutes using our optimal algorithm. The sub-optimal algorithm works much faster than the optimal algorithm and still manages to get much better accuracy than the current state of the art.

CHAPTER 5

THE BIAS TRICK FOR CENTERING PCA

In Section 2.5 we show that the PCA model is heavily influenced by outliers. There are many studies that compute robust principal components, attempting to avoid the effect of outliers. See e.g. (Zhang et al., 2015; Xu et al., 2010; Shah et al., 2017; Xu et al., 2013; Rahmani and Atia, 2017; Hubert and Engelen, 2004) and recently survey papers (Lerman and Maunu, 2018). We refer to these algorithms as RPCA algorithms. However, there appears to be a problem of all the RPCA algorithms we are familiar with: the way in which the data center is computed. The center to be used by RPCA should not be influenced by the (unknown) outliers. Current outlier based RPCA algorithms ignore centering the data or use heuristic methods to update the center of non-outliers. In this chapter we show an approach, "bias trick", that automatically centers the non-outliers. The bias trick appends a large bias value to each data element and unifies the uncentered and centered PCA variants. Due to the automatic centering property of the bias trick, converting an uncentered RPCA algorithm to a centered RPCA algorithm with the bias method is straightforward and does not require any code change to the algorithm itself. Using this bias trick we obtain the first RPCA algorithm that is optimal with respect to centering.

5.1 Problem Being Addressed

As we discussed in Section 2.4.1 there is little difference between the uncentered PCA and the centered PCA from a computational point of view. Since in most situations the data can be easily centered, most of the recently published fast algorithms for computing PCA ignore the centering of the data. See, e.g., (Li et al., 2017; Halko et al., 2011). However, the situation is very different for algorithms that attempt to compute an RPCA by identifying some data points as outliers to be removed. Figure 5.1 shows different variants on a simple dataset.

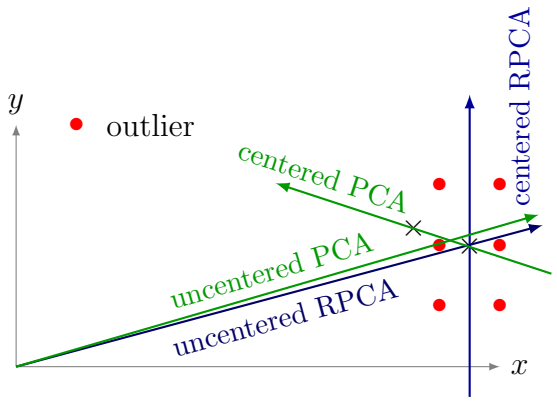


Figure 5.1: The direction of the dominant principal component for several PCA variants on a simple dataset of 7 points with one outlier. The PCA directions are computed from the entire data. The RPCA directions are computed from the 6 non-outliers. The data: (7,3),(7,2),(7,1),(8,3),(8,2),(8,1),(1,4).

The data consists of 7 points, and the direction of the dominant principal component of each variant is shown as an arrow. We can see that the single outlier has a huge effect on the centered variants, and a much smaller effect on the uncentered variants. The problem is that the centering should be applied only to the non-outliers, but they are unknown. Let $X = (x_1, \ldots, x_n)$ be the matrix with m dimensional columns. Let P, Q be the sets of non-outlier column indexes and outlier column indexes in X respectively. More precisely, the approximation error of RPCA in (2.14) can be written in Frobenius norm as shown in (5.1).

$$\mathrm{E}_{\mathrm{RPCA}} = \frac{1}{|X_P|} \sum_{i}^{n} ||(x_i - \mu_P) - VV^T(x_i - \mu_P)||^2, \quad x_i \in X_P \tag{5.1}$$

where μ_P is the mean of non-outlier points X_P, $|X_P|$ is the number of non-outlier data points in X_P.

Some previously proposed robust PCA algorithms perform initial centering of the data, but do not update the center based on the outliers. These include (Zhang et al., 2015; Xu et al., 2010; Shah et al., 2017). Other algorithms such as (Xu et al., 2013; Rahmani and Atia, 2017) do not explicitly center the data. The first assumes a probability distribution of the mean, and the second considers only directions of vectors which makes centering

unnecessary. Other algorithms such as (Hubert and Engelen, 2004) handle the centering as part of the algorithm, but not optimally. This review of the current state of the art suggests that optimal centering in RPCA is not fully solved.

5.1.1 Our Contributions

We propose a general method, the bias trick, which can be used to convert any algorithm that computes uncentered PCA into an algorithm that computes centered PCA with a "sufficient large" bias value. Details are shown in Section 5.3. Furthermore, the bias trick reduces the centered RPCA algorithm to uncentered RPCA algorithm. Once the bias is appended to the algorithm input, the outliers computed by the uncentered algorithm are those corresponding to centered RPCA. Therefore, this conversion retains favorable properties of the uncentered algorithm such as accuracy, sparsity, and efficiency.

Using the bias trick with the algorithm from Chapter 4 that computes optimal uncentered RPCA gives the first optimal centered RPCA algorithm. We implemented this algorithm and describe some experimental results, clearly showing improved performance over all competitors.

5.2 Relationship to Previous Work

The bias trick unifies uncentered PCA and centered PCA. Similar transformations are used to unify algorithms in other fields. In computer graphics one manipulates 2D points such as $p = (x, y)$. The rotation of p can be expressed as matrix multiplication but it is not possible to do the same for translation. Going to homogeneous coordinates by mapping each point p into the 3D vector $p_h = (x, y, 1)$ unifies translation and rotation, since it can be shown that both 2D translation and rotation can be expressed as 3×3 matrix multiplications of p_h. See, e.g., (Hughes et al., 2013).

Another example is from the study of neural networks. The standard "perceptron" can be viewed as a linear threshold unit with zero threshold, or, alternatively, with an unknown threshold value that must be learned. These two cases lead to two different training algorithms, but they can be unified by appending a constant bias of 1 to each data point. See, e.g., (Minsky and Papert, 1988).

The bias method that we describe is similar but not identical to these examples. In our case the appended bias must be large, in the sense that the approximation of the centered PCA of X is improved when the bias value is increased. We show that this gives a practical algorithm, since "reasonably large" bias values will give "sufficiently accurate" approximations. This statement is quantified in Section 5.4. To the best of our knowledge the observation that appending bias to the data unifies centered and uncentered PCA is new. While applying it is straightforward, our correctness proof as given in the appendix is quite elaborate.

5.3 The Bias Trick

Adding a large bias value to each data point transfers the data from r dimension into $r+1$ dimension. Figure 5.2 illustrates transferring a two dimensional data (on lower plane) to three dimension (on higher plane).

Let $\mathsf{PCA}()$ be an uncentered PCA algorithm. It gets as input the matrix X of size $m \times n$ and the number k of desired principal vectors. It returns the principal vectors as the matrix V of size $m \times k$, and k eigenvalues. To apply the bias trick and obtain the centered PCA we do the following:

1. Select a large value b. (See Section 5.4.)

2. Add b as an additional coordinate to each column of X, creating a new matrix X_b of size $(m+1) \times n$.

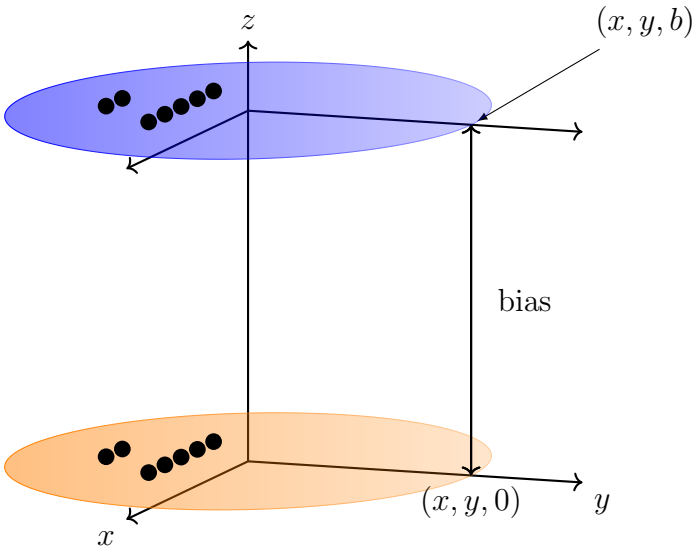


Figure 5.2: An illustration of the bias trick idea

3. Run PCA() on X_b to compute $k+1$ eigenvectors and eigenvalues. Each eigenvector is of size $(m+1)$.

4. Let $\lambda_1^b \ldots, \lambda_{k+1}^b$ be the eigenvalues computed in Step 3. Then the k eigenvalues of the centered PCA are approximately $\lambda_2^b \ldots \lambda_{k+1}^b$.

5. Let $u_1^b \ldots, u_{k+1}^b$ be the eigenvectors computed in Step 3. Let v_j be the jth eigenvector of the centered PCA. It is given approximately by the top m values of the u_{k+1}^b.

Figure 5.3 shows a simple example illustrating the bias trick, where the data is given by the matrix X, and we assume no outliers. The goal is to compute the centered PCA of X. We can center the data by subtracting the mean from each column first, producing the matrix X_c, and then computes the eigenvectors and eigenvalues of $X_c X_c^T$. The principal vectors computed in this way are the columns of the matrix V_c, and the corresponding eigenvalues are λ_c. Instead of removing mean from each data point, We can use the bias trick by append the bias value of 100 to each column of X, producing X_b. Its uncentered PCA is V_b, with the corresponding eigenvalues λ_b. It is clear that an accurate approximation (V_c and λ_C) of the centered PCA is embedded within V_b and λ_b.

X : data	X_c : centered data	X_b : data & bias
$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 0 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 & 1 \\ 2 & -1 & -1 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 3 & 0 & 0 \\ \mathbf{100} & \mathbf{100} & \mathbf{100} \end{pmatrix}$

$$V_c = \begin{pmatrix} -0.4718 & -0.8816 \\ 0.8816 & -0.4718 \end{pmatrix} \qquad \lambda_c = \begin{pmatrix} 7.6055 \\ 0.3944 \end{pmatrix}$$

$$V_b = \begin{pmatrix} 0.01999 & \mathbf{-0.4718} & \mathbf{-0.8814} \\ 0.0099 & \mathbf{0.8816} & \mathbf{-0.4717} \\ 0.9997 & 0.0006 & 0.0223 \end{pmatrix} \quad \lambda_b = \begin{pmatrix} 30015 \\ \mathbf{7.6055} \\ \mathbf{0.3942} \end{pmatrix}$$

Figure 5.3: An approximation to the centered PCA of X is embedded within the uncentered PCA of X_b.

Clearly, the bias trick is not an improvement over the standard centered PCA algorithm. It is more costly and less accurate. However, it has the advantage that it also works for centered RPCA where it does not require advanced knowledge of the outliers. Applying the bias trick for computing centered RPCA can be achieved by using RPCA() instead of PCA(), where RPCA() is any uncentered RPCA algorithm.

5.4 Correctness of The Bias Trick

In this section we prove the correctness of the bias trick. An important part can be traced back to (Cadima and Jolliffe, 2009). In that paper they prove the following result (as a corollary to their Theorem 2).

Theorem 5.1. *Theorem: (Cadima and Jolliffe): Let B be the matrix of second moments of the uncentered data, let μ be the data mean, and let C be the covariance matrix. If one of the eigenvectors of B is $\mu/\|\mu\|$ then all other eigenvector/eigenvalue pairs of B are also eigenvector/eigenvalue pairs of C.*

Notation

Let $X = (x_1 \dots x_n)$ be the data matrix, let $\mu = \frac{1}{n}\sum_i x_i$ be the data mean. The second moments matrix is given by:

$$B = \frac{1}{n}\sum_i x_i x_i^T \tag{5.2}$$

The covariance matrix is given by:

$$C = \frac{1}{n}\sum_i (x_i - \mu)(x_i - \mu)^T \tag{5.3}$$

Let λ_i, u_i be the eigenvalue/eigenvector pairs of C.

Create $X_b = (x_1^b, \dots, x_n^b)$ by adding a large bias b for each vector:

$$x_i^b = \begin{pmatrix} x_i \\ b \end{pmatrix} \tag{5.4}$$

X_b is $(m+1) \times n$. The column mean of X_b is:

$$\mu_b = \begin{pmatrix} \mu \\ b \end{pmatrix}. \tag{5.5}$$

The corresponding $(m+1) \times (m+1)$ matrix of second moments is:

$$B_b = \frac{1}{n}\sum_i^n x_i^b (x_i^b)^T = \begin{pmatrix} B & b\mu \\ b\mu^T & b^2 \end{pmatrix} \tag{5.6}$$

and the corresponding covariance matrix is:

$$C_b = \frac{1}{n}\sum_i^n (x_i^b - \mu_b)(x_i^b - \mu_b)^T = \begin{pmatrix} C & 0 \\ 0 & 0 \end{pmatrix} \tag{5.7}$$

Let λ_i^b, u_i^b be the eigenvalue/eigenvector pairs of B_b.

Define:

$$\begin{pmatrix} v_i \\ w_i \end{pmatrix} = u_i^b \tag{5.8}$$

where v_i is an m-vector and w_i is a scalar. The bias trick is useful since (as proved here) $u_i \approx v_{i+1}$ and $\lambda_i \approx \lambda^b_{i+1}$. Thus, the centered eigenvectors and eigenvalues are obtained from the uncentered and "biased" eigenvectors and eigenvalues.

To analyze the bias trick we need the notion of "approximation for sufficiently large values of the bias b". It is defined as follows:

Definition 5.1. We write $p \approx q$ if for any $\epsilon > 0$ there is b_ϵ such that $(p-q)^2 < \epsilon$ for all $b > b_\epsilon$. When p, q are vectors the squared error is replaced with squared norm, etc. We also say "p approximates q" if $p \approx q$.

Lemma 5.1. *For a sufficiently large value of b:*

$$\textit{Part 1. } w_1 \approx \frac{b}{\sqrt{b^2 + \|\mu\|^2}}$$
$$\textit{Part 2. } v_1 \approx \frac{\mu}{\sqrt{b^2 + \|\mu\|^2}}$$

Proof. From the Courant Fischer theorem (Golub and Van-Loan, 2013) the vector v_1 and the scalar w_1 minimize the following error:

$$\begin{aligned} E(v_1, w_1) &= \min_{a_i} \sum_i \| \begin{pmatrix} x_i \\ b \end{pmatrix} - a_i \begin{pmatrix} v_1 \\ w_1 \end{pmatrix} \|^2 \\ &= \min_{a_i} \sum_i \|x_i - a_i v_1\|^2 + (b - a_i w_1)^2 \end{aligned} \tag{5.9}$$

For sufficiently large value of b the rightmost term dominates the error and it is minimized by $a_i = \frac{b}{w_1}$. Substituting this in (5.9) gives:

$$E(v_1, w_1) = \sum_i \|x_i - \frac{b}{w_1} v_1\|^2 \tag{5.10}$$

Since v_1 and w_1 form an eigenvector they must satisfy:

$$|v_1|^2 + w_1^2 = 1$$

To minimize $E(v_1, w_1)$ subject to this constraint we use the method of Lagrange multipliers. The Lagrangian is:

$$L(v_1, w_1, \alpha) = \sum_i \|x_i - \frac{b}{w_1} v_1\|^2 + \alpha(|v_1|^2 + w_1^2 - 1) \tag{5.11}$$

Taking derivatives of (5.11) with respect to v_1 and equating to 0 gives:

$$(-b/w_1)(n\mu - \frac{nb}{w_1} v_1) + 2\alpha v_1 = 0 \tag{5.12}$$

Therefore, the vectors v_1 and μ are linearly dependent $v_1 = t\mu$. Substituting this in the constraint and solving for t we get:

$$t = \frac{\sqrt{1 - w_1^2}}{|\mu|}$$

So that:

$$v_1 \approx \frac{\sqrt{1 - w_1^2}}{|\mu|} \mu \tag{5.13}$$

To prove Part 1 we take derivatives of (5.11) with respect to w_1 and equate to 0. This gives:

$$2bnv_1^T \mu / w_1^2 - 2nb^2 |v_1|^2 / w_1^3 + 2\alpha w_1 = 0 \tag{5.14}$$

For sufficiently large b the right most term can be ignored. After multiplying by w_1^3 and simplifying this gives:

$$w_1 v_1^T \mu \approx b|v_1|^2 \tag{5.15}$$

Substituting the value of v_1 from (5.13) we get the following equation in w_1:

$$w_1 \|\mu\| \approx b\sqrt{1 - w_1^2} \tag{5.16}$$

Solving this equation for w_1 gives the formula in Part 1. Substituting the Part 1 expression for w_1 in (5.13) and simplifying gives the formula in Part 2. ■

Theorem 5.2. *For a sufficiently large value of b let λ_i^b, u_i^b be an eigenvalue/eigenvector pair of B_b, with $i > 1$. Suppose u_i^b is partitioned as follows: $u_i^b = \begin{pmatrix} v_i \\ w_i \end{pmatrix}$. Then $w_i \approx 0$ and λ_i, v_i are approximately eigenvalue/eigenvector pairs of C.*

Proof. From Lemma 5.1 it follows that

$$\begin{pmatrix} v_1 \\ b \end{pmatrix} \approx \frac{\mu_b}{\|\mu_b\|} \tag{5.17}$$

Since this approximately satisfies the condition of the Cadima and Jolliffe theorem stated above it follows that all other eigenvector/eigenvalue pairs of B_b are also approximately eigenvector/eigenvalue pairs of C_b. Let z be the $m+1$ vector:

$$z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} \tag{5.18}$$

where z_1 is an m-vector and z_2 is a scalar. From (5.7) it follows that

$$C_b z = \begin{pmatrix} C z_1 \\ 0 \end{pmatrix} \tag{5.19}$$

Therefore, if $C_b z = \lambda z$ then $z_2 = 0$ and $C z_1 = \lambda z_1$. ■

5.4.1 Estimating The Value of Bias

The proof of Lemma 5.1 requires that the eigenvector $u_1^b = \begin{pmatrix} v_1 \\ w_1 \end{pmatrix}$ corresponds to the largest eigenvalue of B_b. As shown, the other eigenvalues of B_b are the same as those of C_b. Therefore, we should have $\lambda(B_b) > \lambda(C_b)$, where $\lambda(B_b)$ is the largest eigenvalue of B_b and $\lambda(C_b)$ is the largest eigenvalue of C_b. Indeed, the proof of Lemma 5.1 shows that this is guaranteed for sufficiently large b, but we can derive a more accurate condition.

Lemma 5.2. *If* $v_1^T\mu \geq 0$ *and* $\|X\|_F \leq w_1 b$ *then* $\lambda(B_b) > \lambda(C_b)$.

Proof. Using $u_1^b = \begin{pmatrix} v_1 \\ w_1 \end{pmatrix}$ and (5.6) we have:

$$\lambda(B_b) = {u_1^b}^T B_b u_1^b = v_1^T B v_1 + w_1^2 b^2 + 2 w_1 b v_1^T \mu \tag{5.20}$$

Dropping the positive first term and the non-negative third term and using the assumptions of Lemma 5.2 gives:

$$\lambda(B_b) > w_1^2 b^2 \geq \|X\|_F^2 \tag{5.21}$$

The relations that we have are: $\|X\|_F^2 \geq \|X_c\|_F^2$, $\|X_c\|_F^2 \geq \lambda(B_c)$, and $\lambda(B_c) = \lambda(C)$. The first two are straightforward. In the third $\lambda(C)$ is the largest eigenvalue of C. From the proof of Theorem 5.2 we have $\lambda(C) = \lambda(C_b)$. Concatenating these relations gives the desired result $\lambda(B_b) > \lambda(C_b)$. ■

The first condition in Lemma 5.2 is likely to hold since according to Lemma 5.1 v_1, μ approach the same direction for large b. Therefore, we should select $b \geq \frac{1}{w_1}\|X\|_F = \gamma\|X\|_F$, which suggests that b should be measured in units of $\|X\|_F$.

5.5 Optimal Centered RPCA

Except for computing the centered PCA, one useful application of the bias trick is computing the outlier based centered RPCA model of a given data because the bias trick can automatically get the center of the non-outliers. We extend the algorithm in Figure 4.1 with the bias trick to get an optimal centered RPCA algorithm. To the best of our knowledge this is the first centered algorithm with guaranteed optimality. We refer to this algorithm as COPT that is described in Figure 5.4. The COPT algorithm in Figure 5.4 returns k outliers with respect to r principal vectors. The heuristic f_i' is the same as in (4.5).

Input: X, k, $r+1$, b. **Output:** a outlier indexes subset Q.
Each node n_i has a subset Q_i of size k_i.
The algorithm maintains two lists: the fringe list L, and the closed nodes list C.
Initialization:
For each x_i in X expand it with an extra value b.
Put a node with an empty subset into L.

```
1  while L is nonempty do
2  |  Pick n_i with the smallest f'_i from L.
3  |  if k_i = k then
4  |  |  Stop and return n_i with Q_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 and put n_j in L.
10 |  |  |  end
11 |  |  end
12 |  end
13 end
```

Figure 5.4: The Optimal Centering RPCA (COPT) Algorithm

5.6 Experimental Results

We do experiments with datasets from the UCI repository. In the first experiment we learn the effect of the bias value and give a practical value of $\gamma = 10$ ($b = 10\|X\|_F$) of the bias trick. We run all the experiments with $\gamma = 10$.

We show that the bias trick simplifies the centered RPCA algorithm to the uncentered RPCA algorithm. Results are shown on both toy datasets and real datasets.

We show our COPT algorithm returns smaller errors than other current state-of-the-art RPCA algorithms. A clear illustration is shown to demonstrate that our COPT algorithm returns more meaningful outliers than others.

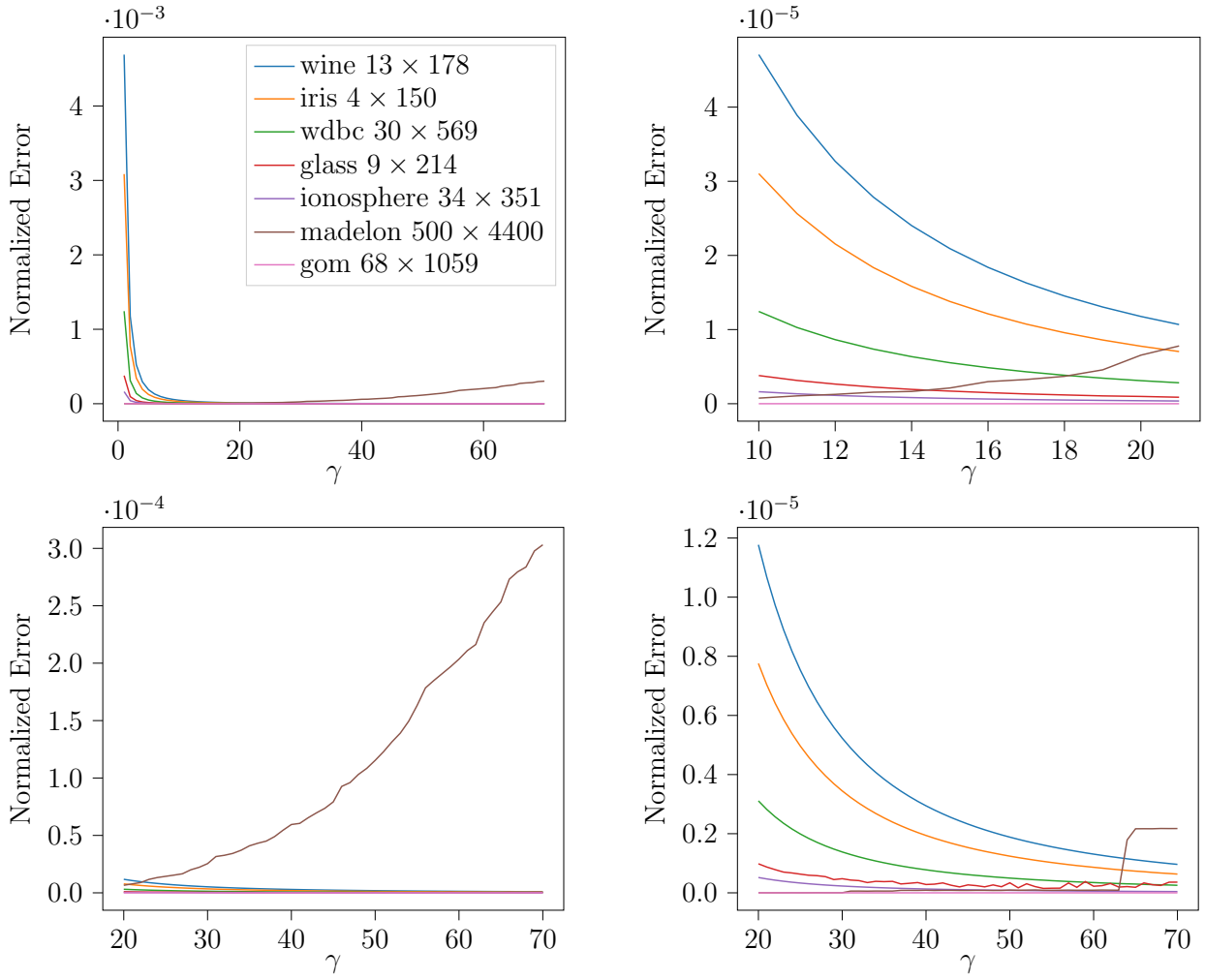


Figure 5.5: Error of approximate eigenvalues for different range of γ on various datasets from UC Irvine. Top two panels and bottom left panel: error of estimating all eigenvalues. Bottom right panel: error of estimating top 10 eigenvalues.

Effects of Different Bias Values

To determine the practicality of the bias method we study experimentally the relationship between γ and the accuracy. Clearly, selecting the bias value b too big may result in round off errors that reduce the accuracy.

Figure 5.5 shows the error of computing the eigenvalues as a function of γ on various datasets from UC Irvine repository. The datasets vary in size from the smallest *Iris* of size 4×150, to the largest *Madelon* of size 500×4400. The normalized error was computed

as follows:

$$\text{Normalized error} = \frac{\sum_i |\lambda_i - \tilde{\lambda_i}|}{\sum_i \lambda_i} \tag{5.22}$$

Based on these (and other) experiments shown in Figure 5.5 we recommend γ in the range $[10, 20]$, which typically gives relative errors in the range $[10^{-10}, 10^{-5}]$. The experiments described in Figure 5.5 show a decline in accuracy for increased γ only for the 500×4400 *Madelon* dataset. Still, the bias method does not break down and the only effect is a slight decrease in accuracy. In most cases we only use the first few eigenpairs. The bottom right panel in Figure 5.5 shows the normalized error of the first 10 eigenvalues. We can see for the large dataset *Madelon* the result is more stable. Our experiments show that using the standard Python software round off errors occur for $b > 10^7$. See Figure 5.6, it shows when $b > 10^7$ the results are not stable.

Combining the constraint $b < 10^7$ with our recommendation of selecting $b = 10\|X\|_F$ gives the following bound on the dataset: $\|X\|_F < 10^6$. The bias trick can still be used with matrices of this (and bigger) although it may not be possible to obtain very high accuracy. If one wishes to apply the bias method for larger datasets then there are several alternatives: 1. Scale the dataset. 2. Use higher accuracy eigen-decomposition routines. 3. Compute only the top few eigenpairs.

The Bias Trick with RPCA Algorithms

In this section we use the bias trick in conjunction with an RPCA algorithm viewed as a black box. Results are shown on both toy datasets and real datasets. Some algorithms were implemented by us, and for others we use the code made available by the authors. The algorithms are:

EU The "Exhaustive Uncentered" algorithm examines all non-outlier subsets of a specified size. The uncentered PCA error is computed for each subset, and the one with the

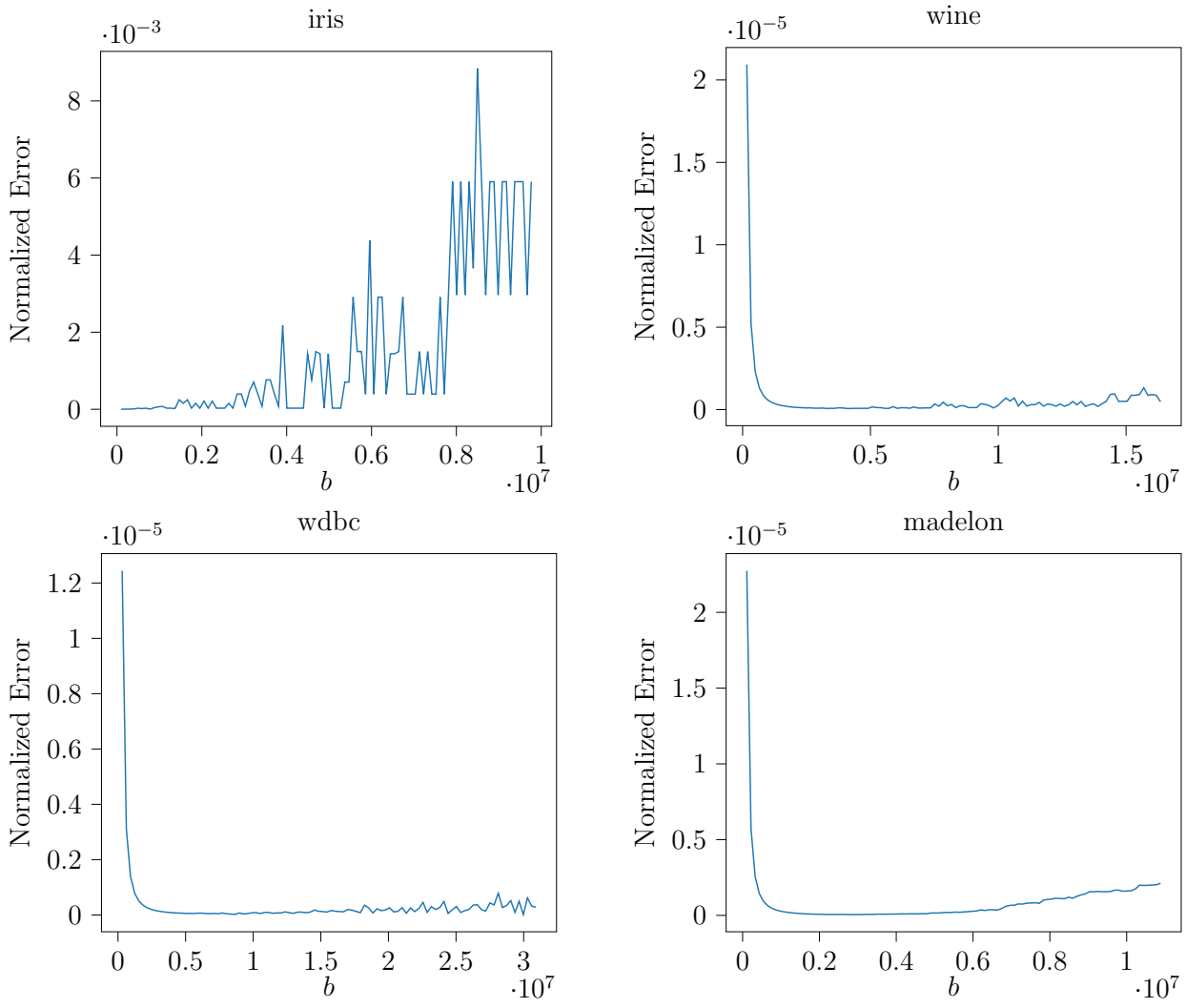


Figure 5.6: The bias trick returns unstable results when the value of $b > 10^7$

smallest error is returned. This computes optimal uncentered PCA but has exponential running time. (our implementation)

EC The "Exhaustive Centered" algorithm is identical to the EU, except that the centered PCA error is computed for each subset. This computes the optimal centered PCA but has exponential running time. (our implementation)

IU The "Iterative Uncentered" algorithm implements a "k-means" style iteration which iterates between outliers and principal vectors of uncentered PCA. (our implementation)

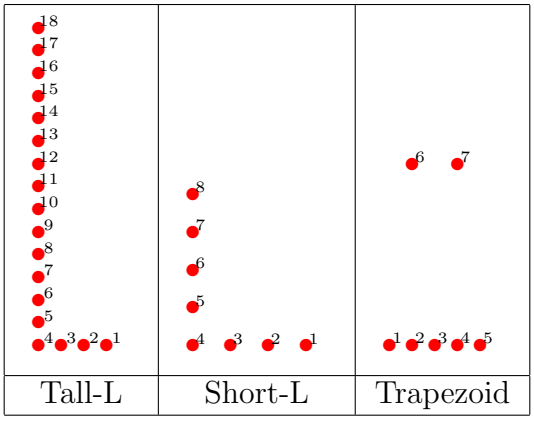


Figure 5.7: Toy datasets for testing robust centered PCA algorithms. The three outliers in the Tall-L dataset are points 1,2,3. The three outliers in the Short-L dataset are points 1,2,3. The two outliers in the Trapezoid dataset are points 6,7.

IC The "Iterative Centered" algorithm implements a "k-means" style iteration. The algorithm iterates between outliers and principal vectors of centered PCA. (our implementation)

HR-PCA The "High-dimensional Robust PCA" algorithm is from (Xu et al., 2013). (implementation from the authors)

CoP The "Coherence Pursuit" algorithm is from (Rahmani and Atia, 2017). (implementation from the authors)

Outlier-Pursuit The "Outlier Pursuit" algorithm is from (Xu et al., 2010). It does not take the number of outliers as input but computes it as part of the output. (implementation from the authors)

RobPCA The " Robust PCA" algorithm is from (Hubert and Engelen, 2004). (part of R software)

Toy Datasets

We created three toy datasets with two dimensional points as shown in Figure 5.7. The challenge is to compute the first principal component of centered RPCA for the specified

Table 5.1: Results of multiple RPCA algorithms on the toy datasets

	bias	Tall-L	Short-L	Trapezoid
EU	n	1,2,4 ✓̸	1,4,5 ✗	1,5 ✗
	y	1,2,3 ✓	1,2,3 ✓	6,7 ✓
EC	n	1,2,3 ✓	1,2,3 ✓	6,7 ✓
	y	n/a	n/a	n/a
IU	n	1,2,4 ✗	1,4,5 ✗	1,5 ✗
	y	1,2,3 ✓	1,4,5 ✗	1,5 ✗
IC	n	1,2,3 ✓	1,4,5 ✗	1,5 ✗
	y	n/a	n/a	n/a
HR-PCA	n	18,16,4 ✗	1,6,8 ✗	6,7 ✓
	y	2,17,18 ✗	3,6,7 ✗	7,3 ✗
CoP	n	1,9,10 ✗	1,4,5 ✗	1,5 ✗
	y	1,2,4 ✓̸	1,4,8 ✗	1,5 ✗
Outlier-Pursuit	n	1,2 ✓̸	1 ✗	none ✗
	y	1,2,3 ✓	1,8,4 ✗	1,5,6,7 ✓̸
RobPCA	n	1,2,3 ✓	1,2 ✓̸	6,7 ✓
	y	n/a	n/a	n/a

number of outliers. In all three cases the optimal solution has zero error. The easiest case is Tall-L (3 outliers), where centered PCA is similar to the optimal robust centered PCA. The Short-L dataset (3 outliers) is harder, since centered PCA does not give results similar to the optimal robust centered PCA. The Trapezoid dataset (2 outliers) is the hardest, as centered PCA is orthogonal to the optimal robust centered PCA.

The outliers computed by the various RPCA algorithms are shown in Table 5.1. To compute the outliers *without* the bias method we first center each dataset and then apply the algorithms with the rank parameter r=1, and the specified number of outliers. To compute the outliers *with* the bias method we append a bias to the data and then apply the algorithms with the rank parameter r=2, and the specified number of outliers.

We mark the optimal result with ✓, near optimal with ✓̸ , and the others with ✗. Some algorithms (EC, IC, RobPCA) center the data internally. This replaces the bias value with 0, and the bias method has no effect on the output (marked as n/a in the table). The result

Table 5.2: Average reconstruction error of multiple RPCA algorithms on real datasets.

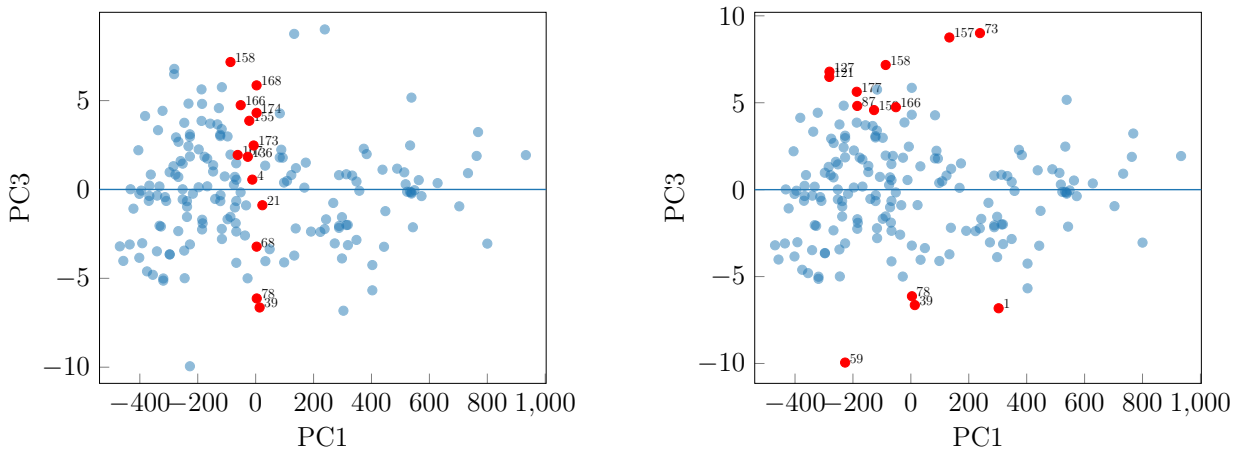
	bias	iris	wine	glass	wdbc	ionosphere	gom
$k:r$		11:1	13:2	7:4	25:3	8:3	15:4
HR-PCA	n	0.3393	17.2651	0.3173	95.6762	4.4468	35.8607
	y	0.3461	16.6891	0.3237	99.9206	4.4506	36.1500
CoP	n	0.3108	15.7866	0.1444	94.8265	4.2332	36.1052
	y	0.2585	13.0729	0.1861	74.9326	3.9871	33.8523
Outlier-Pursuit	n	0.2777	17.8383	0.1550	80.9527	3.9956	34.0282
	y	0.3407	17.6753	0.1550	81.3805	3.9956	34.0689

of EC is guaranteed to be optimal centered RPCA. When we apply EU on the bias input the optimal uncentered algorithm becomes optimal centered RPCA algorithm. Improvements were obtained for EU, IU, CoP, and Outlier-Pursuit(cannot specify the number of outliers). Only the results of HR-PCA became worse since its output is randomized and the results change in consecutive runs.

Real Datasets

The average reconstruction errors computed by the various RPCA algorithms on real datasets are shown in Table 5.2. Same as the experiments on toy datasets, to compute the outliers *without* the bias method we first center each dataset and then apply the algorithms with the rank parameter r, and the specified number of outliers. To compute the outliers *with* the bias method we append a bias to the data and then apply the algorithms with the rank parameter $r+1$, and the specified number of outliers.

Since some algorithms (EC, IC, RobPCA) center the data internally, we do not show the comparison here. Clear improvements were obtained for CoP. Almost all the cases CoP gets gains. The results of HR-PCA and Outlier-Pursuit are almost the same with and without bias. As we mentioned the HR-PCA randomly selects outliers and the results change in consecutive runs. Outlier-Pursuit decides the rank r and the number of outliers k internally.



(a) Only CoP, $E_{\text{rpca}} = 15.7866$ (b) CoP with the bias trick, $E_{\text{rpca}} = 13.0729$

Figure 5.8: Comparison of the location of outliers between initial centering and the bias trick. The experiment is applying CoP algorithm on *wine* dataset with $k = 13$ and $r = 2$.

Figure 5.8 shows that using the bias trick the algorithm CoP returns more meaningful outliers. Thirteen outliers were selected based on the first two principal vectors. The location of all the points in Figure 5.8 on the plane defined by the first and third principal vectors. The left panel is the location of outliers returned only by applying CoP algorithm. The right panel shows the location of outliers returned by applying CoP with the bias trick. It is clear that the outliers returned by applying the bias trick with CoP are further away from the plane composed by the first two eigenvectors (the horizontal line) than those returned by only applying CoP algorithm.

Optimal Centered RPCA

Tables 5.3 show reconstruction errors for different RPCA algorithms, using code provided by the authors. The errors are computed as shown in (5.1). We can see even we apply the greedy variant our COPT algorithm returns smaller errors in most cases.

Figures 5.9 compares the results of our COPT algorithm to the results of the other RPCA algorithms. Thirteen outliers were selected based on the first two principal vectors on *Wine*

Table 5.3: Error comparison among multiple RPCA algorithms. The smallest errors are highlighted. Results of COPT are obtained by greedy variant with $\epsilon = 1$

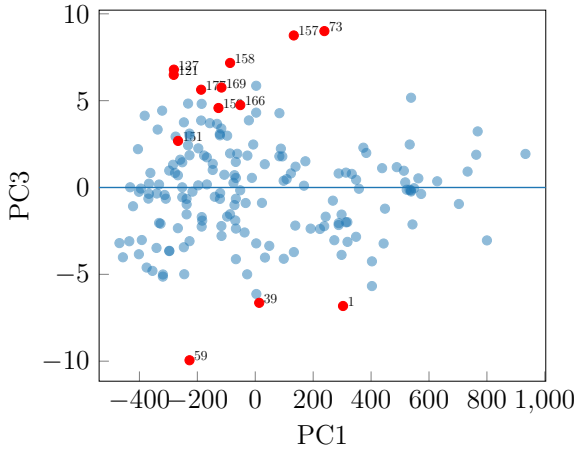
	iris	wine	glass	wdbc	ionosphere	gom
$k:r$	11:1	13:2	7:4	25:3	8:3	15:4
COPT	**0.2581**	**12.9881**	0.2026	**73.3576**	**3.9871**	**33.7604**
HR-PCA	0.3393	17.2651	0.3173	95.6762	4.4468	35.8607
CoP	0.3108	15.7866	**0.1444**	94.8265	4.2332	36.1052
Outlier_Pursuit	0.2777	17.8383	0.1550	80.9527	3.9956	34.0282
RobPCA	**0.2581**	13.6969	0.2537	90.7395	4.2453	35.5978

dataset. All the non-outlier points lie on the plane composed by the first two vectors. Meaningful outliers should be as far away as possible from this plane. The figure shows the location of all the points on the plane defined by the first and third principal vectors (computed by only non-outlier points). The horizontal lines represent the plane composed by the first two principal vectors. It clearly shows the outliers returned by COPT at the margins of the distribution, see Figure 5.9a. By contrast, in Figure 5.9b to Figure 5.9e the outliers are not the ones furthest away. This is a further evidence that our COPT compares favorably to the other RPCA algorithms.

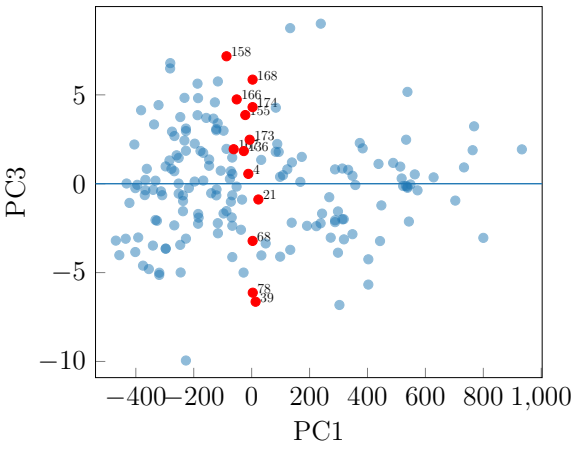
5.7 Concluding Remarks

Outlier based robust principal component analysis (PCA) removes outliers from the data and computes the principal components of the remaining data. This asks for getting center only of the non-outlier points. But it is hard to achieve since the outliers are unknown. In this chapter we show a bias trick which appends a large bias value to each data element and unifies the centered and the uncentered PCA variants.

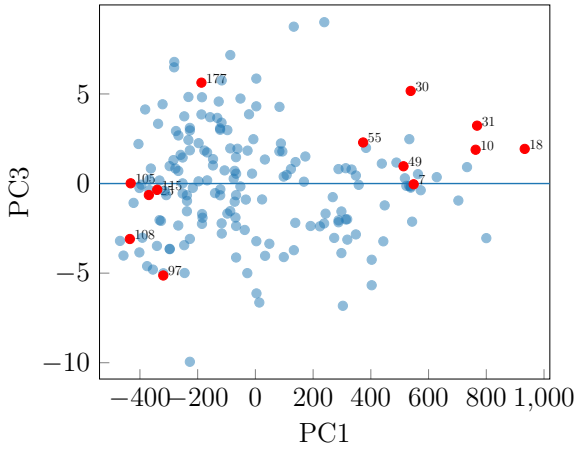
Furthermore the bias trick simplifies the centered RPCA problem to the uncentered RPCA problem. Given an algorithm, a dataset, the desired number of principal components r, and the number of outliers k, the bias method does not apply the algorithm directly to



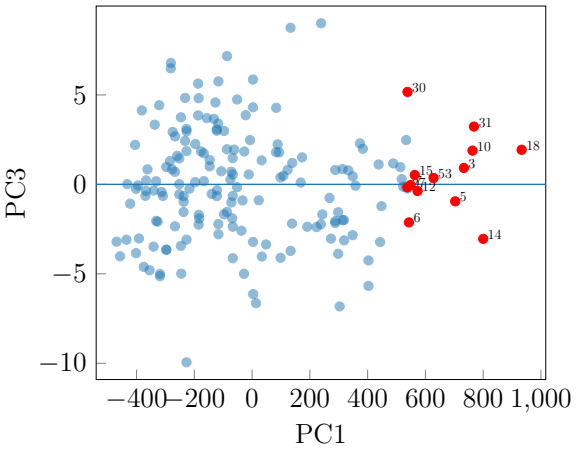
(a) COPT algorithm: $E_{\text{RPCA}} = 12.9881$.



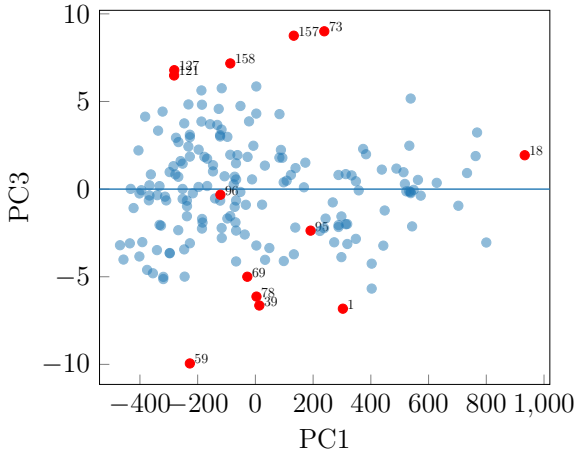
(b) CoP: $E_{\text{RPCA}} = 17.2651$.



(c) HR-PCA: $E_{\text{RPCA}} = 15.7866$.



(d) Outlier-Pursuit: $E_{\text{RPCA}} = 17.8383$.



(e) RobPCA: $E_{\text{RPCA}} = 13.6969$.

Figure 5.9: Positions of outliers selected by different RPCA algorithms on *wine* dataset with $k = 13$, $r = 2$. Red points are the outliers.

the data. Instead, a large (bias) value is added to each data element and the algorithm is applied to compute $r+1$ principal components. The algorithm identifies outliers in the modified data. The RPCA is obtained by computing centered PCA of the non-outliers. However, this does not guarantee to return the optimal centered RPCA and outliers with respect to center. By combining the bias trick and the algorithm from Chapter 4 we get the first optimal centered RPCA algorithm. From the experimental results we can see the sub-optimal variant of our algorithm still manages to get smaller error than the current state of the art.

CHAPTER 6

CONCLUSIONS

The work presented in this dissertation addresses three main problems. The three problems aim to do unsupervised robust data analysis. The first problem is the unsupervised hybrid feature selection and feature extraction. This is an NP-hard problem as the sub-problem of feature selection is NP-hard in the unsupervised case. We show that its not possible to solve this problem by sequentially combining the solutions to feature selection and feature extraction. We provide a heuristic search approach to simultaneously find a given number of optimal selected features and a fixed number of best extracted features. Based on our knowledge this is the first optimal algorithm of the hybrid expression. We also provide faster variants to this algorithm and provide bounds on the suboptimality. In additional we show how to use the suboptimality bounds to improve the results. This work gives an opportunity to balance the robustness and accuracy by tuning the number of selected features and the number of extracted features.

The second problem is the outlier RPCA. We discussed how this problem is very important not just because PCA tends to be very sensitive to outliers but also due to the fact that PCA itself is used as a technique to identify outliers. Thus PCA itself should be robust to the presence of outliers. We provide an optimal solution and faster variants with suboptimality bounds for the outlier robust PCA problem. One can control the trade-off between suboptimality and speed by controlling a parameter of the suboptimal algorithm. We compared it to the popular convex relaxation approach and showed better accuracy of the solution obtained by our algorithm. This approach also combines two very different areas of artificial intelligence research, namely A* search and PCA.

The third problem is a further study on the outlier RPCA. We discussed the importance of proper centering in RPCA. The center of RPCA should be only computed from the non-outliers. But this is hard because the outliers are unknown. We provide a method called

"bias trick" which can unify uncentered PCA and centered PCA. The bias trick asks for appending a large bias value to each data point. From the theorems and experiments we give a practical value to the bias. The bias trick can be used to compute centered RPCA since it automatically computes the center of the non-outliers. Given an RPCA algorithm, a dataset, the desired number of principal vectors r and the number of outliers k, appending the bias value to each data point and applying the algorithm to compute $r+1$ principal vectors. The RPCA algorithm identifies outliers in the modified data and obtains the centered PCA of the outliers. Based on the bias trick we give the first optimal centered RPCA algorithm (COPT). We compared it to the current state-of-the-art RPCA algorithms and showed smaller errors of the solution obtained by our algorithm.

REFERENCES

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. AAAI Press.

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.

Barnett, V. and T. Lewis (1974). *Outliers in statistical data*. Wiley.

Bini, D. A. and L. Robol (2014). Solving secular and polynomial equations: A multiprecision algorithm. *Journal of Computational and Applied Mathematics 272*, 276–292.

Borges, C. F. and W. B. Gragg (1993). A parallel divide and conquer algorithm for the generalized real symmetric definite tridiagonal eigenproblem. In L. Reichel, A. Ruttan, and R. S. Varga (Eds.), *Numerical Linear Algebra nad Scientific Computing*, de Gruyter, Berlin, pp. 11–29.

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

Bouwmans, T., A. Sobral, S. Javed, S. K. Jung, and E. Zahzah (2017). Decomposition into low-rank plus additive matrices for background/foreground separation: A review for a comparative evaluation with a large-scale dataset. *Computer Science Review 23*, 1–71.

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.

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

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

Campbell, N. A. (1980). Robust procedures in multivariate analysis i: Robust covariance estimation. *Applied statistics*, 231–237.

Candès, E. J., X. Li, Y. Ma, and J. Wright (2011). Robust principal component analysis? *Journal of the ACM (JACM) 58*(3), 11.

Chandola, V., A. Banerjee, and V. Kumar (2009). Anomaly detection: A survey. *ACM computing surveys (CSUR) 41*(3), 15.

Chandrasekaran, V., S. Sanghavi, P. A. Parrilo, and A. S. Willsky (2011). Rank-sparsity incoherence for matrix decomposition. *SIAM Journal on Optimization 21*(2), 572–596.

Chatterjee, S. and A. S. Hadi (1986). Influential observations, high leverage points, and outliers in linear regression. *Statistical Science 1*(3), 379–393.

Chen, Y., H. Xu, C. Caramanis, and S. Sanghavi (2016). Matrix completion with column manipulation: Near-optimal sample-robustness-rank tradeoffs. *IEEE Transactions on Information Theory 62*(1), 503–526.

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.

Çivril, A. (2014). Column subset selection problem is ug-hard. *Journal of Computer and System Sciences 80*(4), 849–859.

Daszykowski, M., K. Kaczmarek, Y. Vander Heyden, and B. Walczak (2007). Robust statistics in data analysis – a review: Basic concepts. *Chemometrics and intelligent laboratory systems 85*(2), 203–219.

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.

Dong, Y., S. Hopkins, and J. Li (2019). Quantum entropy scoring for fast robust mean estimation and improved outlier detection. In *Advances in Neural Information Processing Systems*, pp. 6065–6075.

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

Eckart, C. and G. Young (1936). The approximation of one matrix by another of lower rank. *Psychometrika 1*(3), 211–218.

Frank, A. and A. Asuncion (2010). UCI machine learning repository.

Georghiades, A., P. N. Belhumeur, and D. J. Kriegman (1997). Yale face database.

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.

Golub, G. H. (1973). Some modified matrix eigenvalue problems. *SIAM Review 15*(2), 318–334.

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

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

Guo, H., C. Qiu, and N. Vaswani (2014). An online algorithm for separating sparse and low-dimensional signal sequences from their sum. *IEEE Transactions on Signal Processing 62*(16), 4284–4297.

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.

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.

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

He, B., S. Shah, G. A. Crystal Maung, G. Wan, and H. Schweitzer (2019). Heuristic search algorithm for dimensionality reduction optimally combining feature selection and feature extraction. In *Thirty-Third AAAI Conference on Artificial Intelligence*.

Hoaglin, D. C. and R. E. Welsch (1978). The hat matrix in regression and anova. *The American Statistician 32*(1), 17–22.

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

Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of educational psychology 24*(6), 417.

Huber, P. J. (2011). *Robust statistics*. Springer.

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

Hubert, M., P. J. Rousseeuw, and K. Vanden Branden (2005). Robpca: a new approach to robust principal component analysis. *Technometrics 47*(1), 64–79.

Hughes, J. F., S. K. Feiner, J. D. Foley, K. Akeley, M. McGuire, A. v. Dam, and D. F. Sklar (2013). *Computer graphics: principles and practice*. Addison-Wesley.

Jolliffe, I. (2011). Principal component analysis. In *International encyclopedia of statistical science*, pp. 1094–1096. Springer.

Jolliffe, I. T. (1986). *Principal Component Analysis*. Springer-Verlag.

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

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

Krızek, P. (2008). *Feature selection: stability, algorithms, and evaluation*. Ph. D. thesis, PhD thesis, Czech Technical University in Prague, 2008. 6, 14, 36, 67, 93.

Kuhn, M. and K. Johnson (2013). *Applied predictive modeling*, Volume 26. Springer.

Lerman, G. and T. Maunu (2018). An overview of robust subspace recovery. *Proceedings of the IEEE 106*(8), 1380–1410.

Li, G. and Z. Chen (1985). Projection-pursuit approach to robust dispersion matrices and principal components: primary theory and monte carlo. *Journal of the American Statistical Association 80*(391), 759–766.

Li, H., G. C. Linderman, A. Szlam, K. P. Stanton, Y. Kluger, and M. Tygert (2017). Algorithm 971: An implementation of a randomized algorithm for principal component analysis. *ACM Transactions on Mathematical Software 43*(3), 28:1–28:14.

Luo, M., F. Nie, X. Chang, Y. Yang, A. Hauptmann, and Q. Zheng (2016). Avoiding optimal mean robust pca/2dpca with non-greedy l1-norm maximization. In *Proceedings of International Joint Conference on Artificial Intelligence*, pp. 1802–1808.

Maronna, R., D. Martin, and V. Yohai (2006). *Robust Statistics: Theory and Methods*. Hoboken, NJ: Wiley.

Maronna, R. A., R. D. Martin, V. J. Yohai, and M. Salibián-Barrera (2018). *Robust statistics: theory and methods (with R)*. Wiley.

Maronna, R. A., R. D. Martin, V. J. Yohai, and M. Salibián-Barrera (2019). *Robust statistics: theory and methods (with R)*. John Wiley & Sons.

Maronna, R. A. and V. J. Yohai (2004). Robust estimation of multivariate location and scatter. *Encyclopedia of Statistical Sciences 11*.

Marshall, A. W., I. Olkin, and B. C. Arnold (2011). *Inequalities: Theory of Majorization and Its Applications* (Second ed.). Springer.

Mathar, R. and R. Meyer (1993). Preorderings, monotone functions, and best rank r approximations with applications to classical MDS. *Journal of Statistical Planning and Inference 37*, 291–305.

Melman, A. (1998). Analysis of third-order methods for secular equations. *Mathematics of Computation 67*(221), 271–286.

Minsky, M. and S. Papert (1988). Perceptrons. In *Neurocomputing: foundations of research*, pp. 157–169. MIT Press.

Mirsky, L. (1960). Symmetric gauge functions and unitarily invariant norms. *The quarterly journal of mathematics 11*(1), 50–59.

Murase, H. and S. Nayar (1995). Visual learning and recognition of 3D objects from appearance. *International Journal of Computer Vision 14*, 5 – 24.

Netrapalli, P., U. Niranjan, S. Sanghavi, A. Anandkumar, and P. Jain (2014). Non-convex robust pca. In *Advances in Neural Information Processing Systems*, pp. 1107–1115.

Onuki, M. and Y. Tanaka (2018). Svd for very large matrices: An approach with polar decomposition and polynomial approximation. In *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 954–963. IEEE.

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.

Peng, C., C. Chen, Z. Kang, J. Li, and Q. Cheng (2019). Res-pca: A scalable approach to recovering low-rank matrices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7317–7325.

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery (2007). *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press.

Rahmani, M. and G. K. Atia (2017). Coherence pursuit: Fast, simple, and robust principal component analysis. *IEEE Transactions on Signal Processing 65*(23), 6260–6275.

Rousseeuw, P. J. and A. M. Leroy (2005). *Robust regression and outlier detection*, Volume 589. John wiley & sons.

Russell, S. and P. Norvig (2010). *Artificial Intelligence - A Modern Approach*. Pearson Education.

Shah, S., B. He, C. Maung, and H. Schweitzer (2017). Computing robust principal components by a* search. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pp. 1042–1049.

Shah, S., B. He, K. Xu, C. Maung, and H. Schweitzer (2018). Solving generalized column subset selection with heuristic search. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Shitov, Y. (2017). Column subset selection is np-complete. arXiv e-print (arXiv:1701.02764[math.CO]).

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.

Tao, T. (2012). *Topics in Random Matrix Theory*, Volume 132 of *Graduate studies in mathematics*. American Mathematical Society.

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.

Turk, M. A. and A. P. Pentland (1991). Face recognition using eigenfaces. In *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 586–591. IEEE.

Wang, H. (2012). Factor profiled sure independence screening. *Biometrika 99*(1), 15–28.

Xu, H., C. Caramanis, and S. Mannor (2013). Outlier-robust pca: The high-dimensional case. *IEEE Transactions on Information Theory 59*(1), 546–572.

Xu, H., C. Caramanis, and S. Sanghavi (2010). Robust pca via outlier pursuit. In *Advances in Neural Information Processing Systems*, pp. 2496–2504.

Yu, W., Y. Gu, J. Li, S. Liu, and Y. Li (2017). Single-pass pca of large high-dimensional data. *arXiv preprint arXiv:1704.07669*.

Zhang, H., Z. Lin, C. Zhang, and E. Y. Chang (2015). 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.

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.

BIOGRAPHICAL SKETCH

Baokun He was born in Hulunbuir, China. She went to Harbin Institute of Technology in 2005. She joined the Computer Science Department at The University of Texas at Dallas as a graduate student in 2015. She has been working on her PhD with Dr. Haim Schweitzer since 2017. Her main research interests are machine learning and statistics. She interned at Amazon in Seattle in the summer of 2018 and interned at Facebook in Seattle in the summer of 2019. She started to work as a research scientist at Facebook in February 2020.

CURRICULUM VITAE

Baokun He

February 5, 2020

Contact Information:

Department of Computer Science
The University of Texas at Dallas
800 W. Campbell Rd.
Richardson, TX 75080-3021, U.S.A.

Educational History:

B.S., Optoelectronic Information Engineering, Harbin Institute of Technology, 2009
M.S., Optical Engineering, China Academy of Space Technology, 2012
M.S., Computer Science, University of Texas at Dallas, 2019
Ph.D., Computer Science, University of Texas at Dallas, 2020

Algorithms for Robust Data Analysis
Ph.D. Dissertation
Computer Science Department, University of Texas at Dallas
Advisors: Dr. Haim Schweitzer

Employment History:

Research Scientist, Facebook Inc., Februray 2020 – Present
Software Engineer Intern for Machine Learning, Facebook Inc., May 2019 – August 2019
Research Scientist Intern, Amazon, May 2018 – August 2018

Publications:

- A Bias Trick for Centered Robust Principal Component Analysis, AAAI 2020
- Heuristic Search Algorithm for Dimensionality Reduction Optimally Combining Feature Selection and Feature Extraction, AAAI 2019
- Solving Generalized Column Subset Selection with Heuristic Search, AAAI 2018
- Computing Robust Principal Components by A* Search, IJAIT Vol.27 No. 07, 2018
- Computing Robust Principal Components by A* Search, ICTAI 2017