TOPICS IN STATISTICAL LEARNING WITH A FOCUS ON LARGE-SCALE DATA

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF STATISTICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Ya Le

August 2018

© Copyright by Ya Le 2018
All Rights Reserved

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Trevor Hastie) Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Bradley Efron)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Jonathan Taylor)

Approved for the Stanford University Committee on Graduate Studies

Preface

The widespread of modern information technologies to all spheres of society leads to a dramatic increase of data flow, including formation of "big data" phenomenon. Big data are data on a massive scale in terms of volume, intensity, and complexity that exceed the ability of traditional statistical methods and standard tools. When the size of the data becomes extremely large, it may be too long to run the computing task, and even infeasible to store all of the data on a single computer. Therefore, it is necessary to turn to distributed architectures and scalable statistical methods.

Big data vary in shape and call for different approaches. One type of big data is the tall data, i.e., a very large number of samples but not too many features. Chapter 1 describes a general communication-efficient algorithm for distributed statistical learning on this type of big data. Our algorithm distributes the samples uniformly to multiple machines, and uses a common reference data to improve the performance of local estimates. Our algorithm enables potentially much faster analysis, at a small cost to statistical performance. The results in this chapter are joint work with Trevor Hastie and Edgar Dobriban, and will appear in a future publication.

Another type of big data is the wide data, i.e., too many features but a limited number of samples. It is also called high-dimensional data, to which many classical statistical methods are not applicable. Chapter 2 — based on Le and Hastie (2014) [1] — discusses a method of dimensionality reduction for high-dimensional classification. Our method partitions features into independent communities and splits the original classification problem into separate smaller ones. It enables parallel computing and produces more interpretable results.

For unsupervised learning methods like principle component analysis and clustering, the key challenges are choosing the optimal tuning parameter and evaluating method performance. Chapter 3 proposes a general cross-validation approach for unsupervised learning methods. This approach randomly partitions the data matrix into K unstructured folds. For each fold, it fits a matrix completion algorithm to the rest $K-1$ folds and evaluates the prediction on the hold-out fold. Our approach provides a unified framework for parameter tuning in unsupervised learning, and shows strong performance in practice. The results in this chapter are joint work with Trevor Hastie, and will appear in a future publication.

Acknowledgments

I would like to thank many people who made this journey happy, inspiring and rewarding.

First and foremost, I would like to thank my advisor Trevor Hastie. Trevor is an amazing advisor, mentor and friend. I've learned a lot from his deep insights in statistics and his passion about developing scientific and practically useful methodologies. Trevor is very generous with his time and is always there to help. I am deeply grateful for all the inspiring discussions we had and the constructive feedbacks I received. Trevor has also been very supportive and understanding throughout my Ph.D.. He cares me both academically and personally. During my hardest and most emotional time, he told me that he has a daughter and he understands, which means a lot to me.

Next, I'm very grateful to my committee members, Bradley Efron, Jonathan Taylor, Robert Tibshirani and Scott Delp. These project wouldn't be possible without their valuable guidance and input. I'm also very grateful for the great faculty and staff at Department of Statistics.

I would also like to thank the members of Statistical Learning Group and Stanford Mobilize Center for the interesting and inspiring research discussions. I appreciate the opportunity to present my work to Robert Tibshirani, Jonathan Taylor and Scott Delp. Their challenging and insightful questions helped me become a better researcher. I have been very fortunate to collaborate with Eni Halilaj. Eni is a great scholar and a wonderful friend. I have benefited not only from her solid knowledge in bioengineering, but to a greater extent from her attitude towards work and collaboration.

I wouldn't have such a wonderful time without the company of my fellow Ph.D. students and friends. I would like to thank everyone I've met during my Ph.D.. Special thanks to my academic siblings Qingyuan Zhao, Jason Lee, Junyang Qian and Rakesh Achanta, my officemates Minyong Lee, Keli Liu and Stefan Wager, and my cohort Xiaoying Tian, Yuming Kuang, Bai Jiang, Jiyao Kou, Scott Powers, Charles Zheng, Kinjal Basu, Naftali Harris, Lucas Janson, Katelyn Gao, Julia Fukuyama, Kevin Guu, Edgar Dobriban, Amir Sepehri, Cyrus DiCiccio, Jackson Gorham, Ruojun Huang, Haben Michael.

Finally, I'm thankful to my parents for their unconditional love and support. Because of them, no matter where I am and what I am doing, I always feel encouraged and loved.

Contents

List of Tables

List of Figures

Chapter 1

A General Algorithm for Distributed Learning

1.1 Introduction

In the modern era, due to the explosion in size and complexity of datasets, a critical challenge in statistics and machine learning is to design efficient algorithms for large-scale problems. There are two major barriers for large-scale problems: 1) the data can be too big to be stored in a single computer's memory; and 2) the computing task can take too long to get the results. Therefore, it is commonplace to distribute the data across multiple machines and run data analysis in parallel.

Since communication can be prohibitively expensive when the dimensionality of the data is high, a statistical literature on "one-shot" or "embarrassingly parallel" distributed approaches has begun to emerge [2, 3, 4]. This literature has focused on distributed procedures which obtain local estimates in parallel on local machines, and only use one round of communication to send them to a center node and then combine them to get a final estimate. Within this context, the simplest algorithm is the *average mixture* (AVGM) algorithm [5, 6, 7, 8], which distributes the data uniformly, get local estimates and combine them by averaging.

AVGM is appealing as it is simple, general and communication-efficient. In classical setting where the feature dimension p is fixed, Zhang et al. [5] show that the mean-squared error of AVGM matches the rate of the global estimate when $B \leq n$, where B is the number of machines and n is the number of samples on each local machine. However, AVGM suffers from several drawbacks. First, each local machine must have at least $\Omega(\sqrt{N})$ samples to achieve the minimax rate of convergence, where N is the total sample size. This is a restrictive assumption because it requires B to be much smaller than $\sqrt{N}$. Second, AVGM is not consistent in high-dimensional setting. Rosenblatt and Nadler [9] show that the mean-squared error of AVGM decays slower than that of the global estimate when $p/n \to \kappa \in (0, 1)$. Lee et al. [4] improve on AVGM in high-dimensional sparse linear regression setting, but the proposed estimate is only applicable to lasso. Third, AVGM can perform poorly when the estimate is nonlinear [10].

There is also a flurry of research on distributed optimization [10, 11, 12]. For example, Boyd et al. [11] review the alternating direction method of multipliers (ADMM) and show that it is well suited to distributed convex optimization; Jordan et al. [10] propose the communication-efficient surrogate likelihood (CSL) framework which uses a surrogate to the global likelihood function to form the maximum likelihood estimator. However, these methods are not friendly to black-box algorithms. Suppose we have a black-box algorithm that we wanted to apply to the total N samples. In the distributed setting, both ADMM and CSL require us to know what is inside the black box, such as the loss function or the likelihood, and need to modify the black-box algorithm accordingly. Therefore, one has to rebuild a distributed version for each black-box algorithm. This greatly increases the workload and restricts the range of applications of such methods.

In this chapter, we propose a novel distributed algorithm, which we refer to as the *reference average mixture* (RAM) algorithm. At a high level, RAM distributes samples uniformly among B machines or processors, uses a common reference dataset that is shared across all machines to encourage B local estimates to be the same, and then combines the local estimates by averaging. There are several major benefits of this algorithm:

1. Communication-efficient. The communication cost in each iteration is $O(\min(p, m) \times B)$, where m is the size of the reference dataset.

2. General and friendly to black-box algorithms. RAM can be applied to any black-box algorithm without the need to know or modify what is inside the black box.

3. Asynchronous. In each iteration, the communication step of getting the average predictions on the reference data can be asynchronous in practice.

As for the theoretical properties of RAM, we prove the convergence of RAM for a class of convex optimization problems. We refer to the estimate by fitting all N samples on a single machine as the *global estimate*. We also prove that RAM converges to the global estimate in the general convex optimization setting when $p = 1$, and in the ridge regression setting for general $p \geq 1$. Our numerical experiments show that RAM provides substantial improvement over AVGM, and can achieve a performance comparable to that of the global estimate.

The outline of this chapter is as follows. Section 2 describes the problem and setting of this chapter. Section 3 presents the RAM algorithm. Section 4 discusses the properties of the RAM algorithm, including algorithm convergence and stability. In section 5, we show through simulations how RAM works in practice and how RAM are affected by various factors. We extends RAM to allow for adaptive tuning parameter in section 6. Section 7 applies our method to a real massive dataset.

1.2 Problem

Let $X \in \mathbb{R}^p$ denote a random input vector, and $Y \in \mathbb{R}$ a random output variable, with joint distribution $P(X, Y)$. We seek a function $f(X)$ for predicting Y given values of the input X. Let $L(X, Y, f)$ be a loss function that penalizes errors in prediction, and assume that f is in some collection of functions F. This leads us to a criterion for choosing f,

$$f^* = \arg\min_{f \in F} \mathbb{E}_P[L(X, Y, f)]. \tag{1.1}$$

In practice, the population distribution P is unknown to us, but we have access to a collection of N i.i.d. samples $S = \{(x_i, y_i)\}_{i=1}^N$ from the distribution P. Thus, we can estimate f^* by minimizing the penalized empirical risk

$$\hat{f} = \arg\min_{f \in F} \frac{1}{N} \sum_{i=1}^{N} L(x_i, y_i, f) + g(f), \tag{1.2}$$

where $g(f)$ is some penalty on f. If $g(f) = 0$, then $\hat{f}$ is the estimate by empirical risk minimization.

In the distributed setting, we are given a dataset of $N = nB$ i.i.d. samples from $P(X, Y)$, which we divide them uniformly amongst B processors. Denote $\hat{f}_G$ as the *global estimate*, the one we would get if all N samples are fitted on a single machine. The goal of this chapter is to recover $\hat{f}_G$ in the distributed setting.

1.3 Method

1.3.1 Motivation

Denote the dataset on machine b as S_b, for $b = 1, \ldots, B$. Then the local estimate on machine b is

$$\hat{f}_b = \arg\min_{f \in F} \frac{1}{|S_b|} \sum_{(x,y) \in S_b} L(x, y, f) + g(f). \tag{1.3}$$

Note that the global estimator $\hat{f}_G$ is also the solution to the following constrained optimization problem

$$\min_{f_1,\dots,f_B} \sum_{b=1}^{B} \frac{|S_b|}{N} \left(\frac{1}{|S_b|} \sum_{(x,y)\in S_b} L(x,y,f_b) + g(f_b) \right)$$
$$s.t.\ \ f_1 = f_2 = \cdots = f_B.$$

The term inside the parentheses is exactly the local penalized loss on machine b. This suggests that we may recover the global estimate by encouraging the local estimates to be the same.

One natural idea to encourage local estimates to be the same is to update $\hat{f}_b$ by adding a penalty on the distance between the local estimate $\hat{f}_b$ and the average estimate $\bar{f} = \sum_{b=1}^{B} \hat{f}_b / B$,

$$\hat{f}_b^{\text{new}} = \arg\min_{f} \frac{1}{|S_b|} \sum_{(x,y)\in S_b} L(x,y,f) + g(f) + \rho \cdot D(f,\bar{f}), \tag{1.4}$$

where $D(f,\bar{f})$ is some distance function, and ρ is a tuning parameter. If $\rho = 0$, then there is no penalty and $\hat{f}_b^{\text{new}} = \hat{f}_b$. If $\rho = \infty$, then all the local estimates are forced to be the average estimate $\hat{f}_b^{\text{new}} = \bar{f}$. However, for a general distance function D, this update is not friendly to black-box algorithms because the distance penalty changes the structure of the optimization problem. Thus, the original black-box algorithm used to solve (1.2) is invalid of solving (1.4). Therefore, we need to maintain the structure of (1.2) when designing the distance function D.

In our method, we design $D(f,\bar{f})$ to be the average loss of f on a reference dataset for which the output variable Y is set to be the average prediction on X, i.e., $\bar{f}(X)$. Specifically, let $A = \{a_i\}_{i=1}^{m}$ be a set of feature samples, where $a_i \in \mathbb{R}^p$. The reference dataset, A, can be either specified by the user or a random subsample of the dataset $\{x_i\}_{i=1}^{N}$. Then we define

$D(f, \bar{f})$ as

$$D(f, \bar{f}) = \frac{1}{|A|} \sum_{a \in A} L(a, \bar{f}(a), f). \tag{1.5}$$

Note that D not only measures the distance between f and $\bar{f}$ on the reference dataset A, but also has been designed to maintain the structure of the original problem. Therefore we can solve (1.4) using the original black-box algorithm, simply by updating the input dataset as $S_b \cup \{(a_i, \bar{f}(a_i)\}_{i=1}^m$.

1.3.2 Algorithm

We propose an iterative distributed algorithm, called the *reference average mixture* (RAM) algorithm. The algorithm is listed in Algorithm 1.1.

Algorithm 1.1 The Reference Average Mixture (RAM) Algorithm

for $b = 1$ to B **do**

  Initialize $\hat{f}_b^{(0)}$ as the local estimate on machine b

$$\hat{f}_b^{(0)} = \arg\min_f \frac{1}{|S_b|} \sum_{(x,y) \in S_b} L(x, y, f) + g(f)$$

end for

while not converged **do**

  $\bar{f}^{(k)} = \sum_{b=1}^{B} |S_b| \hat{f}_b^{(k)} / N$

  for $b = 1$ to B **do**

$$\hat{f}_b^{(k+1)} = \arg\min_f \frac{1}{|S_b|} \sum_{(x,y) \in S_b} L(x, y, f) + g(f) + \rho \cdot \frac{1}{|A|} \sum_{a \in A} L(a, \bar{f}^{(k)}(a), f)$$

  end for

end while

Return: $\bar{f}^{(k)}$

RAM has several important features:

1. Communication-efficient. In each iteration, the only communication step is to get the average predictions on the reference dataset, i.e., $\{\bar{f}^{(k)}(a_i)\}_{i=1}^m$. This requires each local machine b to pass its predictions $\{\hat{f}_b^{(k)}(a_i)\}_{i=1}^m$ to the master machine. The master

machine then takes the average of the predictions and sends $\{\bar{f}^{(k)}(a_i)\}_{i=1}^m$ back to every local machine. The communication cost in each iteration is therefore $O(mB)$. If $f \in F$ can be characterized by a p-dimensional vector β (for example, linear functions $f(x) = x^T\beta$ can be characterized by the parameter β), then the communication cost in each iteration is $O(\min(p, m) \times B)$. This is because we can either pass the estimated parameter $\hat{\beta}_b^{(k)}$ or the predictions on the reference dataset.

2. Friendly to black-box algorithms. Let Ψ denote a black-box algorithm that takes the data $S = \{(x_i, y_i)\}_{i=1}^N$ as input and outputs the global estimate $\hat{f}_G = \Psi(S)$. Then one can run RAM using Ψ, without the need to know or make changes to what is inside Ψ. This is because both the initialization step and the updating step can be achieved by only changing the input dataset to Ψ. Specifically, the initialization step is $\hat{f}_b^{(0)} = \Psi(S_b)$, and the updating step is $\hat{f}_b^{(k+1)} = \Psi(S_b \cup \{(a_i, \bar{f}^{(k)}(a_i)\}_{i=1}^m)$ with observation weights $(\underbrace{\frac{|S_b|+m}{|S_b|}, \ldots, \frac{|S_b|+m}{|S_b|}}_{|S_b|}, \underbrace{\frac{\rho(|S_b|+m)}{m}, \ldots, \frac{\rho(|S_b|+m)}{m}}_{m})$.

3. General. RAM is a general distributed algorithm that can work with any black-box algorithm Ψ. Since RAM only involves running Ψ with updated local data, it is still applicable even if Ψ is a general algorithm that does not necessarily solve a penalized risk minimization problem.

4. Asynchronous. The communication step of getting the average predictions on the reference data can actually be asynchronous in practice. Specifically, the master machine stores the most up-to-date predictions of B local machines on the reference data. When machine b completes the updating step, it sends its new predictions to the master machine. The master machine updates the predictions of machine b, computes the average predictions, and sends them back to machine b so that machine b can proceed to the next iteration. In this way, the master machine computes the average predictions using the most up-to-date local predictions, without the necessity of waiting

all B machines to finish in each iteration.

1.4 Method Properties

In this section, we examine three key properties of RAM. First, we investigate the convergence behavior of RAM - whether it converges and what the convergence rate is. Second, we study the quality of RAM by comparing the RAM estimate with the global estimate. Third, we discuss when RAM works well and when not.

1.4.1 Does RAM converge?

To examine the convergence property, we consider a general class of convex optimization problems. Here we assume that the prediction function $f \in F$ can be characterized by a p-dimensional vector $\beta \in D$, i.e. $f(\cdot) = f(\cdot;\beta)$. In this subsection, we show that for a given $\rho > 0$, our algorithm converges to an estimate $\hat{\beta}(\rho)$. In the next subsection we'll show that as $\rho \to \infty$, the associated estimate $\hat{\beta}(\rho)$ converges to the global estimate $\hat{\beta}_G$.

We denote the original loss on machine b as

$$L_b(\beta) \triangleq \frac{1}{|S_b|} \sum_{(x_i,y_i)\in S_b} L(x_i, y_i, \beta) + g(\beta) \tag{1.6}$$

and the loss on the reference data as

$$\tilde{L}(\beta, \hat{\beta}^{(k)}) \triangleq \frac{1}{|A|} \sum_{a\in A} L(a, b(\hat{\beta}^{(k)}), \beta), \tag{1.7}$$

where $\hat{\beta}^{(k)} = (\hat{\beta}_1^{(k)}, \ldots, \hat{\beta}_B^{(k)}) \in \mathbb{R}^{Bp}$ and $b(\hat{\beta}^{(k)}) = \sum_{b=1}^{B} f(a; \hat{\beta}_b^{(k)})/B$ is the average prediction of $\hat{\beta}^{(k)}$ at reference sample a. Therefore, at iteration $k+1$, the optimization problem faced by machine b is

$$\hat{\beta}_b^{(k+1)} = \arg\min_{\beta\in D} L_b(\beta) + \rho \cdot \tilde{L}(\beta, \hat{\beta}^{(k)}). \tag{1.8}$$

We provide a sufficient condition for the convergence of RAM.

THEOREM 1.1. *Given the dataset* $S = \{(x_i, y_i)\}_{i=1}^N$ *and the reference dataset* $A = \{a_i\}_{i=1}^m$, *suppose there exist* $M_0, M_1 \succeq 0$ *and* $C_1 > 0$ *such that*

- $\nabla^2 L_b(\beta) \succeq M_0$ *for* $b = 1, 2, \ldots, B$ *and any* β;
- $\frac{\partial^2}{\partial \beta^2} \tilde{L}(\beta, \gamma) \succeq M_1$ *for any* β, γ;
- $\max_{b=1,\ldots,B} \|\frac{\partial^2}{\partial \beta \partial \gamma_b} \tilde{L}(\beta, \gamma)\|_2 \leq C_1$ *for any* β, γ, *where* $\gamma = (\gamma_1, \ldots, \gamma_B)$.

Then, we have

$$\|\hat{\beta}^{(k+1)} - \hat{\beta}^{(k)}\|_2 \leq C \|\hat{\beta}^{(k)} - \hat{\beta}^{(k-1)}\|_2 \tag{1.9}$$

where

$$C = \sqrt{B} C_1 \|(M_0/\rho + M_1)^{-1}\|_2. \tag{1.10}$$

Proof. First, note that

$$\nabla_\beta L_b(\hat{\beta}^{(k+1)}) + \rho \frac{\partial}{\partial \beta} \tilde{L}(\hat{\beta}_b^{(k+1)}, \hat{\beta}^{(k)}) = 0. \tag{1.11}$$

This defines $\hat{\beta}_b^{(k+1)}$ as an implicit function of $\hat{\beta}^{(k)}$:

$$\hat{\beta}_b^{(k+1)} = T(\hat{\beta}^{(k)}). \tag{1.12}$$

Using the implicit differentiation, we have

$$\nabla T(\gamma) = (\nabla^2 L_b(\beta) + \rho \frac{\partial^2}{\partial \beta^2} \tilde{L}(\beta, \gamma))^{-1} \left(-\rho \frac{\partial^2}{\partial \beta \partial \gamma_1} \tilde{L}(\beta, \gamma), \ldots, -\rho \frac{\partial^2}{\partial \beta \partial \gamma_B} \tilde{L}(\beta, \gamma) \right) \tag{1.13}$$

where $\beta = T(\gamma)$. Therefore

$$\|\nabla T(\gamma)\|_2 \leq C_1 \|(M_0/\rho + M_1)^{-1}\|_2. \tag{1.14}$$

By the mean value theorem,

$$\begin{aligned}
\|\hat{\beta}_b^{(k+1)} - \hat{\beta}_b^{(k)}\|_2^2 &= \|\nabla T(\xi)(\hat{\beta}^{(k)} - \hat{\beta}^{(k-1)})\|_2^2 && (1.15)\\
&\leq \sum_b C_1 \|(M_0/\rho + M_1)^{-1}\|_2^2 \|\hat{\beta}_b^{(k)} - \hat{\beta}_b^{(k-1)}\|_2^2 && (1.16)\\
&= C_1 \|(M_0/\rho + M_1)^{-1}\|_2^2 \|\hat{\beta}^{(k)} - \hat{\beta}^{(k-1)}\|_2^2. && (1.17)
\end{aligned}$$

Therefore, we have

$$\|\hat{\beta}^{(k+1)} - \hat{\beta}^{(k)}\|_2 \leq C \|\hat{\beta}^{(k)} - \hat{\beta}^{(k-1)}\|_2 \tag{1.18}$$

where

$$C = \sqrt{B} C_1 \|(M_0/\rho + M_1)^{-1}\|_2. \tag{1.19}$$

□

The contraction factor C increases as B or ρ increases. This suggests that RAM converges slower when B or ρ is larger. It can be shown by simple algebra that linear regression with convex penalty, such as lasso and elastic net, has a contraction factor $C < 1$ for any ρ, S and A.

1.4.2 What does RAM converge to?

For a given ρ, let $\hat{\beta}(\rho) = (\hat{\beta}_1(\rho), \hat{\beta}_2(\rho), \ldots, \hat{\beta}_B(\rho))$ denote the RAM estimate upon convergence. Next, we show that $\hat{\beta}(\rho)$ converges to the global estimate when $\rho \to \infty$. To begin with, we consider the special case where $p = 1$.

ASSUMPTION 1.1. $L(x, y, \beta)$ and $g(\beta)$ are strongly convex in β with parameter c_0 and twice continuously differentiable.

This assumption is usually satisfied with a broad class of optimization problems, including ridge regression, elastic net, and generalized linear models among many others. Although the strong convexity may not be necessary in some application settings to ensure the convergence behavior, the convex structure makes the convergence behavior tractable.

THEOREM 1.2. *Under Assumption 1.1 and $p = 1$, as $\rho \to \infty$, the RAM estimate converges to the global estimate. That is,*

$$\lim_{\rho \to \infty} \hat{\beta}_b(\rho) = \hat{\beta}_G, \text{ for } b = 1, \ldots, B. \tag{1.20}$$

In addition, there exists a constant $C > 0$ such that

$$|\hat{\beta}_b(\rho) - \hat{\beta}_G| \leq \frac{C}{\rho}. \tag{1.21}$$

Proof. For a given ρ, the RAM estimate $\hat{\beta}(\rho) = (\hat{\beta}_1(\rho), \hat{\beta}_2(\rho), \ldots, \hat{\beta}_B(\rho))$. Within this proof, we set $\gamma = (\gamma_1, \ldots, \gamma_B) \triangleq \hat{\beta}(\rho)$ for the ease of notation. Denote

$$h(\beta) \triangleq \frac{1}{m} \sum_{i=1}^{m} L(a_i, b_i(\gamma), \beta). \tag{1.22}$$

Note that the reference data is chosen such that

$$h'(\gamma_1) + h'(\gamma_2) + \ldots + h'(\gamma_B) = 0. \tag{1.23}$$

and we denote α as the minimum of h. On machine b, since γ is the limit, we have by the first-order-condition

$$L_b'(\gamma_b) + \rho h'(\gamma_b) = 0. \tag{1.24}$$

By summing the above equations up for over j, we have

$$\sum_{j=1}^{B} L_j'(\gamma_j) = 0. \tag{1.25}$$

The global optimum β_0 satisfies

$$\sum_{j=1}^{B} L_j'(\beta_0) = 0. \tag{1.26}$$

Hence, we must have $\beta_0 \in [\min_j \gamma_j, \max_j \gamma_j]$ because

$$\sum_{j=1}^{B} L_j'(\min_j \gamma_j) \leq 0 \tag{1.27}$$

and

$$\sum_{j=1}^{B} L_j'(\max_j \gamma_j) \geq 0. \tag{1.28}$$

For any γ_i and γ_j, we have

$$\rho(h'(\gamma_i) - h'(\gamma_j)) = (L_j'(\gamma_j) - L_i'(\gamma_i)). \tag{1.29}$$

By mean value theorem, there exists some $\xi \in [\min_j \gamma_j, \max_j \gamma_j]$ such that

$$|\gamma_i - \gamma_j| \leq \frac{1}{\rho} \frac{|L_j'(\gamma_j) - L_i'(\gamma_i)|}{c_0}, \tag{1.30}$$

where $c_0 = \min_{y \in [\min_j \hat{\beta}_j^{(0)}, \max_j \hat{\beta}_j^{(0)}]} h''(y) > 0$ due to the strictly convexity of $\tilde{L}$. At the same time, there exists a $c_1 > 0$ such that

$$|L_j'(\gamma_j) - L_i'(\gamma_i)| \leq c_1 \tag{1.31}$$

for all i, j, because γ_j lie in the fixed compact interval $[\min_j \hat{\beta}_j^{(0)}, \max_j \hat{\beta}_j^{(0)}]$. Hence, we have

$$|\frac{1}{B} \sum_{j=1}^{B} \gamma_j - \beta_0| \leq \max_{i,j} |\gamma_i - \gamma_j| \leq \frac{1}{\rho} \frac{c_1}{c_0}. \tag{1.32}$$

Note that c_0, c_1 do not depend on ρ. Hence the convergence to global optimum is proved by setting ρ to ∞. $\square$

Now, we consider the general case where $p \geq 1$. To show what RAM converges to for general p, we use ridge regression as a benchmark case, where the closed-form solutions can be exploited to illustrate the convergence behaviors. In this case, we have

$$L_b(\beta) = \frac{1}{|S_b|} \sum_{(x_i,y_i)\in S_b} (y_i - x_i^\top \beta)^2 + \lambda \|\beta\|_2^2. \tag{1.33}$$

THEOREM 1.3. *In the setting of ridge regression, for any data $\{(x_i, y_i)\}_{i=1}^N$, reference data A and given $\rho > 0$, RAM converges to $\hat{\beta}(\rho)$ where*

$$\hat{\beta}(\rho) = \left(\sum_{b=1}^{B} \frac{|S_b|}{N} Q_b \right)^{-1} \sum_{b=1}^{B} \frac{|S_b|}{N} Q_b \hat{\beta}_b^{(0)} \tag{1.34}$$

and

$$Q_b = \left(\frac{1}{|S_b|} \sum_{(x_i,y_i)\in S_b} x_i x_i^\top + \frac{\rho}{|A|} \sum_{a\in A} a_i a_i^\top + \lambda I_p \right)^{-1} \left(\frac{1}{|S_b|} \sum_{(x_i,y_i)\in S_b} x_i x_i^\top + \lambda I_p \right). \tag{1.35}$$

Proof. At iteration k, the estimate on machine b is

$$\hat{\beta}_b^{(k)} = \left(\frac{1}{|S_b|} \sum_{(x_i,y_i)\in S_b} x_i x_i^\top + \frac{\rho}{|A|} \sum_{a\in A} a_i a_i^\top + \lambda I_p \right)^{-1} \left(\frac{1}{|S_b|} \sum_{(x_i,y_i)\in S_b} x_i y_i + \frac{\rho}{|A|} \sum_{a\in A} a_i a_i^\top \hat{\beta}^{(k-1)} \right), \tag{1.36}$$

where $\hat{\beta}^{(k)} = \sum_{b=1}^{B} \frac{|S_b|}{N} \hat{\beta}_b^{(k)}$ represents the RAM estimate at iteration k.

Let

$$Q_b = \left(\frac{1}{|S_b|} \sum_{(x_i,y_i)\in S_b} x_i x_i^\top + \frac{\rho}{|A|} \sum_{a\in A} a_i a_i^\top + \lambda I_p \right)^{-1} \left(\frac{1}{|S_b|} \sum_{(x_i,y_i)\in S_b} x_i x_i^\top + \lambda I_p \right), \tag{1.37}$$

then we get

$$\hat{\beta}_b^{(k)} = Q_b \hat{\beta}_b^{(0)} + (I - Q_b) \hat{\beta}^{(k-1)}. \tag{1.38}$$

Let $\bar{Q} = \sum_{b=1}^{B} \frac{|S_b|}{N} Q_b$. The above equation leads to

$$\begin{aligned}
\hat{\beta}^{(k)} &= \sum_{b=1}^{B} \frac{|S_b|}{N} Q_b \hat{\beta}_b^{(0)} + (I - \bar{Q})\hat{\beta}^{(k-1)} && (1.39)\\
&= \sum_{b=1}^{B} \frac{|S_b|}{N} Q_b \hat{\beta}_b^{(0)} + (I - \bar{Q}) \left(\sum_{b=1}^{B} \frac{|S_b|}{N} Q_b \hat{\beta}_b^{(0)} \right) + (I - \bar{Q})^2 \hat{\beta}^{(k-2)} && (1.40)\\
&\cdots && (1.41)\\
&= \left(I + (I - \bar{Q}) + \ldots + (I - \bar{Q})^{k-1} \right) \left(\sum_{b=1}^{B} \frac{|S_b|}{N} Q_b \hat{\beta}_b^{(0)} \right) + (I - \bar{Q})^k \hat{\beta}^{(0)} && (1.42)\\
&= \bar{Q}^{-1} \left(I - (I - \bar{Q})^k \right) \left(\sum_{b=1}^{B} \frac{|S_b|}{N} Q_b \hat{\beta}_b^{(0)} \right) + (I - \bar{Q})^k \hat{\beta}^{(0)}. && (1.43)
\end{aligned}$$

We claim that all the eigenvalues of $I - \bar{Q}$ are in $[0, 1)$. Then as $k \to \infty$, $I - \bar{Q} \to 0$. Thus

$$\hat{\beta}(\rho) = \lim_{k \to \infty} \hat{\beta}^{(k)} = \bar{Q} \left(\sum_{b=1}^{B} \frac{|S_b|}{N} Q_b \hat{\beta}_b^{(0)} \right). \quad (1.44)$$

To prove the claim, it suffices to show that the eigenvalues of $I - Q_b$ are in $[0, 1)$ for all $b = 1, \ldots, B$. Let η be an eigenvalue of $I - Q_b$ for any given b. Then there exists an nonzero vector u such that $(I - Q_b)u = \eta u$, i.e.,

$$\left(\frac{1}{|S_b|} \sum_{(x_i, y_i) \in S_b} x_i x_i^\top + \frac{\rho}{|A|} \sum_{a \in A} a_i a_i^\top + \lambda I_p \right)^{-1} \left(\frac{\rho}{|A|} \sum_{a \in A} a_i a_i^\top \right) u = \eta u. \quad (1.45)$$

Therefore we have

$$\begin{aligned}
\left(\frac{\rho}{|A|} \sum_{a \in A} a_i a_i^\top \right) u &= \eta \left(\frac{1}{|S_b|} \sum_{(x_i, y_i) \in S_b} x_i x_i^\top + \frac{\rho}{|A|} \sum_{a \in A} a_i a_i^\top + \lambda I_p \right) u, && (1.46)\\
(1 - \eta) \left(\frac{\rho}{|A|} \sum_{a \in A} a_i a_i^\top \right) u &= \eta \left(\frac{1}{|S_b|} \sum_{(x_i, y_i) \in S_b} x_i x_i^\top + \lambda I_p \right) u. && (1.47)
\end{aligned}$$

Multiplying both sides with $u^\top$, we get

$$(1-\eta)u^\top \left(\frac{\rho}{|A|}\sum_{a\in A} a_i a_i^\top\right) u = \eta u^\top \left(\frac{1}{|S_b|}\sum_{(x_i,y_i)\in S_b} x_i x_i^\top + \frac{\rho}{|A|}\sum_{a\in A} a_i a_i^\top + \lambda I_p\right) u. \quad (1.48)$$

Note that $u^\top \left(\frac{\rho}{|A|}\sum_{a\in A} a_i a_i^\top\right) u \geq 0$ and $u^\top \left(\frac{1}{|S_b|}\sum_{(x_i,y_i)\in S_b} x_i x_i^\top + \frac{\rho}{|A|}\sum_{a\in A} a_i a_i^\top + \lambda I_p\right) u > 0$. Thus $\eta \in [0, 1)$. □

THEOREM 1.4. *In the setting of ridge regression, for any data $\{(x_i, y_i)\}_{i=1}^N$ and reference data A, as $\rho \to \infty$, the RAM estimate converges to the global estimate, i.e.,*

$$\lim_{\rho\to\infty} \hat{\beta}(\rho) = \hat{\beta}_G. \quad (1.49)$$

Proof. According to Theorem 1.3, the RAM estimate for a given ρ is

$$\hat{\beta}(\rho) = \left(\sum_{b=1}^{B} \frac{|S_b|}{N} Q_b\right)^{-1} \sum_{b=1}^{B} \frac{|S_b|}{N} Q_b \hat{\beta}_b^{(0)}. \quad (1.50)$$

Note that as $\rho \to \infty$,

$$\begin{aligned} &\rho Q_b && (1.51)\\ =\ & \left(\frac{1}{\rho|S_b|}\sum_{(x_i,y_i)\in S_b} x_i x_i^\top + \frac{1}{|A|}\sum_{a\in A} a_i a_i^\top + \frac{\lambda}{\rho} I_p\right)^{-1} \left(\frac{1}{|S_b|}\sum_{(x_i,y_i)\in S_b} x_i x_i^\top + \lambda I_p\right) && (1.52)\\ \to\ & \left(\frac{1}{|A|}\sum_{a\in A} a_i a_i^\top\right)^{-1} \left(\frac{1}{|S_b|}\sum_{(x_i,y_i)\in S_b} x_i x_i^\top + \lambda I_p\right). && (1.53) \end{aligned}$$

Therefore

$$\begin{aligned}
\hat{\beta}(\rho) &= \left(\sum_{b=1}^{B} \frac{|S_b|}{N}\rho Q_b\right)^{-1} \sum_{b=1}^{B} \frac{|S_b|}{N}\rho Q_b \hat{\beta}_b^{(0)} \\
&\to \left(\left(\frac{1}{|A|}\sum_{a\in A} a_i a_i^\top\right)^{-1}\left(\frac{1}{N}\sum_{i=1}^{N} x_i x_i^\top + \lambda I_p\right)\right)^{-1}\left(\frac{1}{|A|}\sum_{a\in A} a_i a_i^\top\right)^{-1}\left(\frac{1}{N}\sum_{i=1}^{N} x_i^\top y_i\right) \\
&= \hat{\beta}_G.
\end{aligned}$$

□

1.4.3 When does RAM work?

Now we discuss when RAM works well and when not, and explain the reason behind.

When $B = 1$, it's expected that RAM is stable, i.e.,

$$\hat{f}^{(k)} = \hat{f}^{(0)}, \forall k \geq 0. \tag{1.54}$$

Here we abbreviate the subscript that represents the machine index as there is only one machine.

Since $\hat{f}^{(1)}$ is the solution to

$$\min_f \frac{1}{|S|}\sum_{(x,y)\in S} L(x,y,f) + g(f) + \rho \cdot \frac{1}{|A|}\sum_{a\in A} L(a, \hat{f}^{(0)}(a), f) \tag{1.55}$$

and $\hat{f}^{(0)}$ is the solution to

$$\min_f \frac{1}{|S|}\sum_{(x,y)\in S} L(x,y,f) + g(f), \tag{1.56}$$

stability of RAM requires that $\hat{f}^{(0)}$ is also the solution to

$$\min_f \frac{1}{|A|} \sum_{a \in A} L(a, \hat{f}^{(0)}(a), f). \tag{1.57}$$

The above condition holds in the cases where the data loss and the regularization penalty are separated. For example, for generalized linear models with elastic net penalty, L is the negative log likelihood and g is the elastic net penalty, and $\hat{f}^{(0)}$ satisfies the above condition. However, for algorithms where regularization or smoothing cannot be clearly represented or separated from data loss, the above condition fails. For example, for random forest and gradient boosted trees, we cannot separate the data loss and the regularization/smoothing. If we use L to represent the "internal objective function" that the algorithm tries to optimize, then the solution to (1.57) will be more regularized or smoothed compared to $\hat{f}^{(0)}$.

We illustrate this fact with a simulation where there is only one machine, i.e. $B = 1$, and the feature dimension $p = 1$. The feature vector x_i is sampled from $\mathcal{N}(0, I_p)$ and the response y_i is generated from a linear model $y_i = x_i^T \beta_0 + \epsilon_i$. The noise ϵ_i is sampled i.i.d. from $\mathcal{N}(0, \sigma^2)$, where σ is set such that the signal-to-noise ratio $Var(x_i^T \beta_0)/Var(\epsilon_i)$ is 0.5. The sample size is 50 and 100 for the random forest and the gradient boosted trees simulation respectively.

Figure 1.1 shows the simulation result of RAM with random forest as the black-box algorithm. The left figure suggests that RAM is not stable in this case, and the test R^2 decreases as it runs. The right figure presents the scatter plots of fitted $\hat{y}_i$ v.s. x_i at four different iterations of RAM when $\rho = 10$. From these plots we can see that the RAM fit gets smoother as iteration increases.

Figure 1.2 shows the simulation result of RAM with gradient boosted trees as the black-box algorithm. The left figure suggests that RAM is not stable in this case, although the test R^2 only decreases when the tuning parameter ρ is very large. The right figure presents

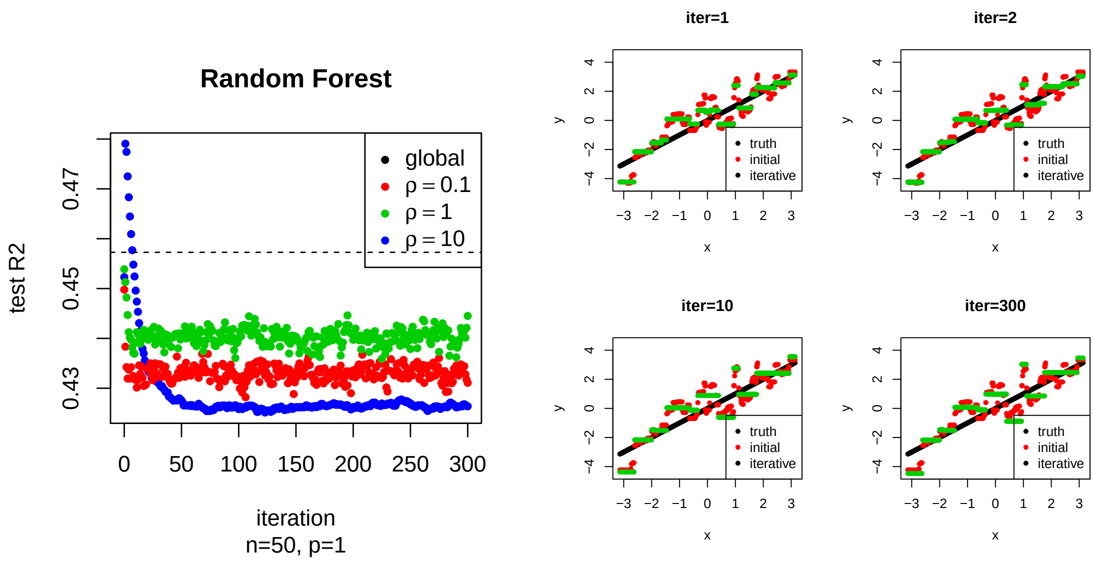
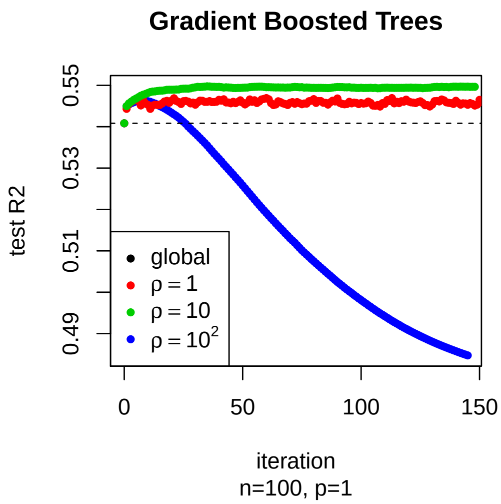


Figure 1.1: RAM with random forest as the black-box algorithm, where $B = 1$, $p = 1$ and $N = 50$. The left figure shows the test R^2 of RAM versus the number of iterations. The black dashed line indicates the test performance of the global estimate. The right figure represents the scatter plots of RAM fits at different iterations when $\rho = 10$. The black points are noiseless response $x_i^T\beta_0$ versus x_i, the red points are the global fits (i.e., RAM fits at iteration 0) and the green points are the RAM fits at the current iteration. This simulation shows that RAM is not stable with random forest, as the RAM fit gets smoother as iteration increases.

the scatter plots of fitted $\hat{y}_i$ v.s. x_i at four different iterations of RAM when $\rho = 100$. From these plots we can see that the RAM fit gets more shrunken as iteration increases.

1.5 Numerical Experiments

In this section, we study the performance of RAM and compare it with AVGM and the global estimate in several simulated examples. Note that the estimate at iteration 0 of RAM is equivalent to AVGM. The results show that RAM can significantly improve the performance over AVGM, and achieve a performance comparable to that of the global estimate. Most of the performance boost of RAM occurs within the first few iterations, so in practice we can



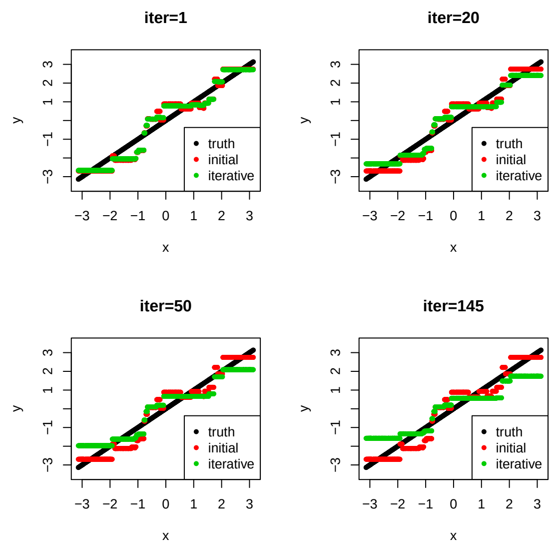


Figure 1.2: RAM with random forest as the black-box algorithm, where $B = 1$, $p = 1$ and $N = 100$. The left figure shows the test R^2 of RAM versus the number of iterations. The black dashed line indicates the test performance of the global estimate. The right figure represents the scatter plots of RAM fits at different iterations when $\rho = 100$. The black points are noiseless response $x_i^T \beta_0$ versus x_i, the red points are the global fits (i.e., RAM fits at iteration 0) and the green points are the RAM fits at the current iteration. This simulation shows that RAM is not stable with gradient boosted trees when ρ is very large, as the RAM fit gets more shrunken as iteration increases.

only run RAM for a few iterations instead of running it until convergence.

We consider two common machine learning methods: logistic regression for classification, and lasso for regression. For our simulated experiments, we fix the total number of training samples to be $N = 10^5$, and vary the number of parallel splits B, the number of reference samples m and the tuning parameter ρ. In all cases, the feature vector x_i is sampled from $\mathcal{N}(0, I_p)$. All estimates are evaluated and compared on a test dataset containing 10^5 samples.

For classification experiments, we choose logistic regression as the black-box algorithm. The metric of test performance is test accuracy, i.e., the fraction of test samples with correct class predictions. The feature dimension is set to be $p = 180$. The response y_i is sampled from $Bernoulli(p_i)$, where $\text{logit}(p_i) = x_i^T \beta_0 + \epsilon_i$. Here $\beta_0 \in \mathbb{R}^p$ is a sparse vector with $p/5$ nonzero entries, each equal to $\sqrt{2\log(p)/n}$, where $n = N/B$. The noise ϵ_i is sampled i.i.d.

from $\mathcal{N}(0, \sigma^2)$, where σ is set such that the signal-to-noise ratio $Var(x_i^T \beta_0)/Var(\epsilon_i)$ is 0.5.

For regression experiments, we choose lasso as the black-box algorithm. The metric of test performance is test R^2, i.e., the proportion of variance explained in the test dataset. The feature dimension is set to be $p = 600$. The response y_i is generated from a linear model, $y_i = x_i^T \beta_0 + \epsilon_i$. Here β_0 and ϵ_i are generated in the same way as in classification experiments.

1.5.1 Effect of tuning parameter ρ

First, we consider the effect of varying the tuning parameter ρ. We test $\rho = 0.1, 1, 5$ while fixing the number of machines $B = 50$ and the reference data size $m = p$. In Figure 1.3, we plot the test performance of RAM versus the number of iterations in the RAM algorithm, with different ρ values. Remember that the performance at iteration 0 is equivalent to that of AVGM. For the lasso case, we run our algorithm for a fixed grid of λ values, where λ is the tuning parameter of the l_1 penalty in lasso, and plot the maximum test R^2 at each iteration in the right figure.

From Figure 1.3, we can see that 1) RAM significantly improves the performance compared to AVGM, and can almost achieve the performance of the global estimate; 2) RAM has most of the performance boost in the first few iterations, so in practice we can run RAM for only a few iterations instead of running it until convergence; 3) with larger ρ, RAM takes longer to converge but also has a better performance upon convergence. The third point is expected, as in the extreme case $\rho = 0$, RAM converges immediately and returns the AVGM estimate. The third point also agrees with our theorem on the convergence rate of RAM.

1.5.2 Effect of reference data size m

Next, we vary the reference data size m. We test $m = p/4, p/2, p, 3p/2$ while fixing the number of machines $B = 50$ and the tuning parameter $\rho = 1$. In Figure 1.4, we plot the test

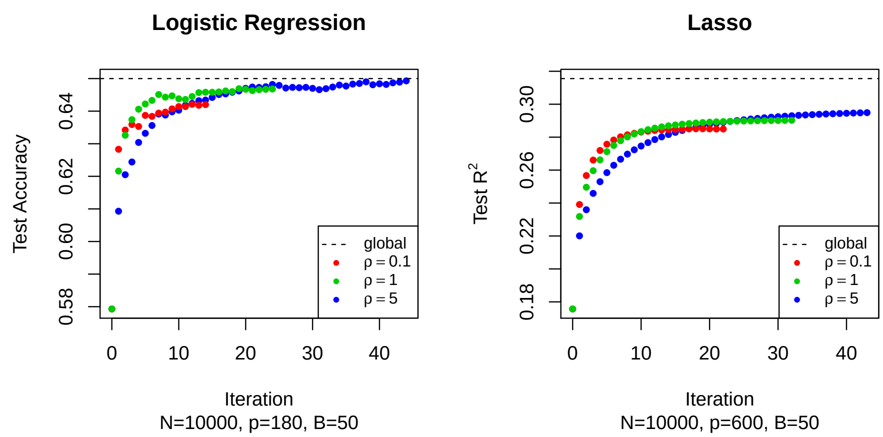


Figure 1.3: Comparing RAM, AVGM and the global estimate with varying tuning parameter ρ. Here $N = 10^5$, $B = 50$, $m = p$, $p = 180$ for the left figure and $p = 600$ for the right figure. The left figure shows the test accuracy of RAM with logistic regression as the black-box solver versus the number of iterations. The right figure shows the test R^2 of RAM with lasso as the black-box solver versus the number of iterations. The black dashed line indicates the test performance of the global estimate. Note that the performance of RAM at iteration 0 is equivalent to that of AVGM.

performance of RAM versus the number of iterations in the RAM algorithm, with different m values. Figure 1.4 shows that with larger m, RAM takes longer to reach the plateau but achieves a better performance. This makes sense as larger m means that it's harder to bring consensus among local estimates on the reference dataset, but it also has a stronger force to make the local estimates close to each other. In addition, when $m \geq p$, increasing m helps little in improving the performance.

1.5.3 Effect of number of machines B

Finally, we test the effect of the number of machines B. We set $B = 10, 25, 50$ while fixing the tuning parameter $\rho = 1$ and the reference data size $m = p$. In Figure 1.5, we plot the test performance of RAM versus the number of iterations in the RAM algorithm, with different B values. Figure 1.5 shows that with larger B, AVGM has a poorer performance while RAM

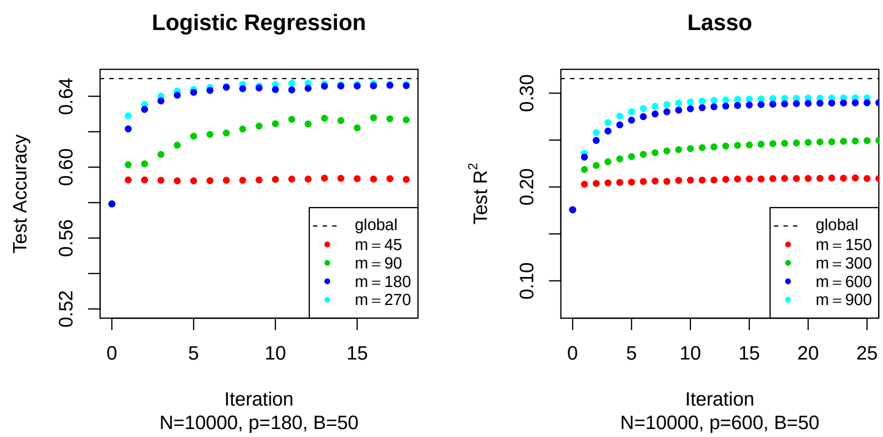


Figure 1.4: Comparing RAM, AVGM and the global estimate with varying reference data size m. Here $N = 10^5$, $B = 50$, $\rho = 1$, $p = 180$ for the left figure and $p = 600$ for the right figure. The left figure shows the test accuracy of RAM with logistic regression as the black-box solver versus the number of iterations. The right figure shows the test R^2 of RAM with lasso as the black-box solver versus the number of iterations. The black dashed line indicates the test performance of the global estimate. Note that the performance of RAM at iteration 0 is equivalent to that of AVGM.

achieves a better performance. The fact that AVGM performances worse when B is larger agrees with Theorem 1 in [5]. The reason why RAM performs worse when B is smaller could be that we need to use larger ρ to achieve the same level of performance when B is smaller. When B is smaller, the difference among local estimates is likely to be smaller, and thus the average loss on the reference data is also smaller. Therefore, it needs a larger ρ to have the same level of penalty in (1.4). Also, it takes longer to converge when B is larger, which agrees with our theorem on the convergence rate.

1.5.4 Lasso path convergence

For the regression experiments with lasso as the black-box solver, we show that the test performance of RAM converges to that of the global estimate uniformly over the entire grid of λ values, as ρ increases or m increases. Here we set the number of machines $B = 50$

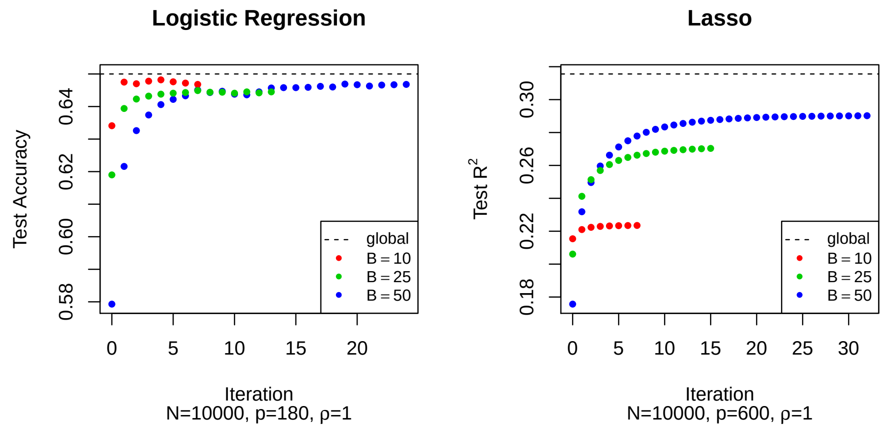


Figure 1.5: Comparing RAM, AVGM and the global estimate with varying number of machines B. Here $N = 10^5$, $\rho = 1$, $m = p$, $p = 180$ for the left figure and $p = 600$ for the right figure. The left figure shows the test accuracy of RAM with logistic regression as the black-box solver versus the number of iterations. The right figure shows the test R^2 of RAM with lasso as the black-box solver versus the number of iterations. The black dashed line indicates the test performance of the global estimate. Note that the performance of RAM at iteration 0 is equivalent to that of AVGM.

. In Figure 1.6, we plot the test performance of RAM versus λ, the tuning parameter of the l_1 penalty in lasso. In the left figure, we vary the tuning parameter $\rho = 0.1, 1, 5$ while fixing the reference data size $m = p$. In the right figure, we vary the reference data size $m = p/4, p/2, p, 3p/2$ while fixing the tuning parameter $\rho = 1$. Figure 1.6 shows that AVGM performs very differently from the global estimate as λ varies, while the RAM estimate performs similarly to the global estimate over the entire path.

1.6 Extension

In RAM algorithm, there is a tuning parameter ρ representing the magnitude of penalty on reference data. We show in Section 1.4.2 that the RAM estimate converges to the global solution as $\rho \to \infty$. In this section, we want to extend RAM with an adaptive tuning

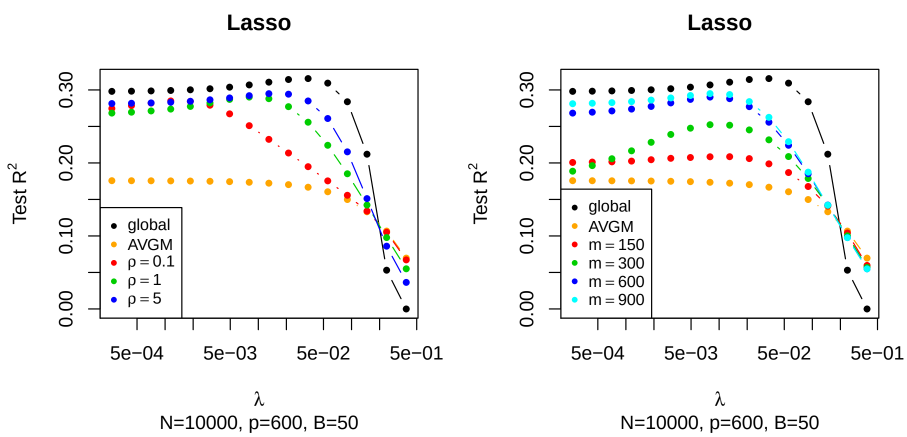


Figure 1.6: Comparing the test performance of RAM, AVGM and the global estimate as a function of lasso penalty parameter λ, with varying tuning parameter ρ and reference data size m. Here $N = 10^5$, $p = 600$, $m = p$ for the left figure and $\rho = 1$ for the right figure. Both figures show the test R^2 of RAM with lasso as the black-box solver versus the lasso penalty parameter λ.

parameter ρ_k, where ρ_k depends on the iteration k. In ridge regression setting, we prove that RAM converges to the global estimate when $\rho_k \to \infty$ and $\rho_{k+1} - \rho_k \to 0$ as $k \to \infty$. We illustrate through simulations that this adaptive approach also performs well with lasso and generalized linear models [1].

The adaptive RAM has two important advantages. First, it converges to the global estimate directly, without the need of running RAM multiple times with increasing ρ. Second, it requires no tuning, so it is easier to use in practice and much more computationally efficient.

THEOREM 1.5. *Consider the RAM algorithm for solving an OLS linear regression problem. Set the regularization parameter at the k-th iteration as ρ_k, such that*

1. $\rho_k \to \infty$ *and* ρ_k *is increasing,*

2. $\rho_{k+1} - \rho_k \to 0.$

[1] This section is an on-going work with Edgar Dobriban.

Let $\hat\beta^k$ be the value on the central machine at the k-th iteration, and let $\hat\beta_G$ be the global OLS solution. Then, the RAM iterates converge to the global solution

$$|\hat\beta^k - \hat\beta_G|_2 \le \frac{C}{\rho_k},$$

for some constant C not depending on k. The same holds for ridge regression, as we can rewrite ridge regression as an OLS problem with enlarged data.

Proof. By Theorem 1.3, the RAM estimate with fixed ρ at iteration $k+1$ is

$$\hat\beta^{(k+1)}(\rho) = \bar Q(\rho)\hat\beta(\rho) + (I - \bar Q(\rho))\hat\beta^{(k)}(\rho). \tag{1.58}$$

where $\bar Q(\rho) = \sum_b |S_b| Q_b(\rho)/N$ and

$$Q_b(\rho) = \left(\frac{1}{|S_b|}\sum_{(x_i,y_i)\in S_b} x_i x_i^\top + \frac{\rho}{|A|}\sum_{a_i\in A} a_i a_i^\top\right)^{-1}\left(\frac{1}{|S_b|}\sum_{(x_i,y_i)\in S_b} x_i x_i^\top\right)$$

.

Thus $\hat\beta^{(k+1)}(\rho) - \hat\beta(\rho) = [I - \bar Q(\rho)]\cdot[\hat\beta^{(k)}(\rho) - \hat\beta(\rho)]$, so

$$\hat\beta^{(k+1)}(\rho) - \hat\beta_G = \bar Q(\rho)[\hat\beta(\rho) - \hat\beta_G] + [I - \bar Q(\rho)]\cdot[\hat\beta^{(k)} - \hat\beta_G].$$

Choose now $\rho = \rho_k$ and let $\varepsilon_k = |\hat\beta^{(k)}(\rho_k) - \hat\beta_G|$. Then we have

$$\varepsilon_{k+1} \le |\bar Q(\rho_{k+1})|\cdot C_0/\rho_{k+1} + |I - \bar Q(\rho_{k+1})|\cdot\varepsilon_k$$

Further we need a bound on the operator norm: $|I - \bar Q(\rho)|$. We claim that $|I - \bar Q(\rho)| \le \rho/(C_1+\rho)$ and $|\bar Q(\rho)| \le C_2/(C_2+\rho)$ for ρ large. Overall, with $C' = C_0C_2$

$$\varepsilon_{k+1} \le \frac{C'}{\rho_{k+1}(C_2+\rho_{k+1})} + \frac{\rho_{k+1}}{C_1+\rho_{k+1}}\cdot\varepsilon_k$$

Now we need to set ρ_k such that $\varepsilon_k \to 0$. First, to get $r_{k+1} = (C_1 + \rho_{k+1})\varepsilon_{k+1}$ non-increasing, we need

$$(C_1 + \rho_{k+1})\varepsilon_{k+1} \leq \frac{C'(C_1 + \rho_{k+1})}{\rho_{k+1}(C_2 + \rho_{k+1})} + \rho_{k+1} \cdot \varepsilon_k \leq (C_1 + \rho_k)\varepsilon_k.$$

For ρ_k large enough, we have $\frac{(C_1+\rho_{k+1})}{(C_2+\rho_{k+1})} \leq C_3$ for some constant C_3, and absorbing this constant into C', we get that the last inequality is equivalent to

$$C' \leq \rho_{k+1}(C_1 + \rho_k - \rho_{k+1})\varepsilon_k$$

Take ρ_k such that

1. $\rho_k \to \infty$ in an increasing way,

2. $\rho_{k+1} - \rho_k \to 0$.

Then for large enough k, the inequality is equivalent to $C'/C'' \cdot 1/\rho_{k+1} \leq \varepsilon_k$.

Thus, for each k, there are two possibilities: either $r_{k+1} \leq r_k$, or $\varepsilon_k \leq C'/C'' \cdot 1/\rho_{k+1}$.

If the second inequality occurs infinitely often, then because the sequence r_k is non-increasing on the other indices, we get that $\varepsilon_k = O(1/\rho_k)$. Therefore, it remains to consider the case where $\varepsilon_k \geq C'/C'' \cdot 1/\rho_{k+1}$ above some index k_0. In that case, the sequence r_k is decreasing above that index, showing that $\varepsilon_{k+1} \leq C'''/(C + \rho_{k+1})$ for some constant, so that the sequence converges. At least in the second case, this also shows that $\varepsilon_k = \Theta(1/\rho_{k+1})$, at an exact rate. In the first case, $\varepsilon_k = O(1/\rho_k)$.

Now we prove our claim that on the operator norm bounds of $|I - \bar{Q}(\rho)|$. Recall $\bar{Q}(\rho) = \sum_b W_b \frac{|S_b|}{N}(\rho)$. Now, $|\bar{Q}(\rho)| \leq \sum_b |S_b||Q_b(\rho)|/N$, so it is enough to bound $|Q_b(\rho)|$. This matrix is of the form $(V + \rho U)^{-1}V$, for symmetric positive definite U, V. While the matrix is not symmetric, it is similar to the symmetric matrix $V^{1/2}(V + \rho U)^{-1}V^{1/2} = (I + \rho V^{-1/2}UV^{-1/2})^{-1}$. Since this matrix is symmetric and positive definite, its singular values are equal to its eigenvalues. Therefore, $|Q_b(\rho)| = [1 + \rho\lambda_p(V^{-1/2}UV^{-1/2})]^{-1}$.

Similarly, $|I - \bar{Q}(\rho)| \le \sum_b |S_b||I - Q_b(\rho)|/N$, so it is enough to bound $|I - Q_b(\rho)|$. This matrix is of the form $I - (V + \rho U)^{-1}V = \rho(V + \rho U)^{-1}U$, for symmetric positive definite U, V, which agree with the ones in the previous example. As before, we get, $|I - \bar{Q}(\rho)| = [1 + \rho\lambda_p(U^{-1/2}VU^{-1/2})]^{-1}$. □

To illustrate how this adaptive RAM works, we run RAM with $\rho_k = k^{0.8}$ at iteration k. The simulation setting is the same as in Section 1.5.1. In Figure 1.7, we plot the test performance of the adaptive RAM versus the number of iterations, and compare it with the RAM with fixed $\rho = 5$. Remember that the performance at iteration 0 is equivalent to that of AVGM.

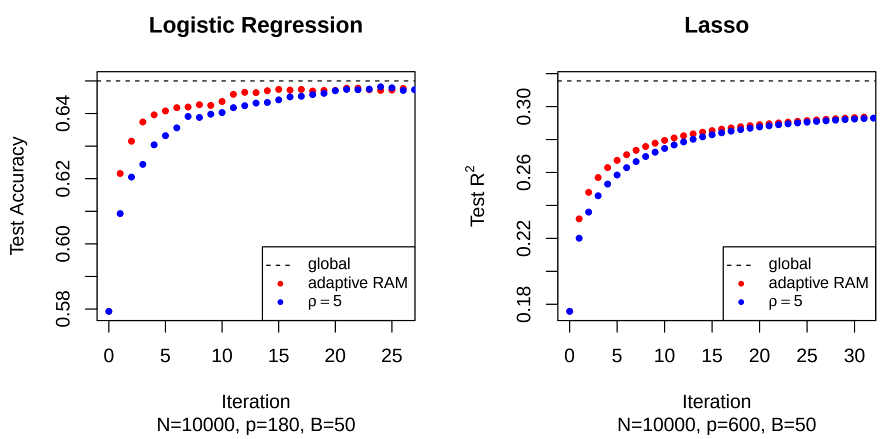


Figure 1.7: Comparing RAM estimate with adaptive ρ_k, RAM estimate with fixed $\rho = 5$ and the global estimate. Here $N = 10^5$, $B = 50$, $m = p$, $p = 180$ for the left figure and $p = 600$ for the right figure. The left figure shows the test accuracy of RAM with logistic regression as the black-box solver versus the number of iterations. The right figure shows the test R^2 of RAM with lasso as the black-box solver versus the number of iterations. The black dashed line indicates the test performance of the global estimate. Note that the performance of RAM at iteration 0 is equivalent to that of AVGM.

From Figure 1.7, we can see that the adaptive RAM can achieve the same performance as the RAM estimate in Figure 1.3 with large ρ, but with fewer iterations. Moreover, the adaptive RAM only takes one run and does not require parameter tuning. This shows that the adaptive RAM performs well in practice, even though we do not have theoretical guarantees in the setting of lasso and generalized linear models.

1.7 A Real Data Example

In this section, we apply our method to a real data example. The data contains 1586088 observations of beer ratings, and 738665 variables which are sparse document features. The goal is to predict the beer rating based on the document features. Since the feature dimension is high, we choose lasso as our prediction method. We divide the data randomly and equally into the training dataset and the test dataset. On the training dataset, we implement both the global algorithm, i.e. lasso on the entire training dataset, and the adaptive RAM. For RAM, we randomly split the data into 64 folds and apply the adaptive RAM with $\rho_k = 100k^{0.9}$. The reference data A is a $p \times p$ identity matrix. Even though the dimension of A is large, it is very sparse thus it requires little space for storage and does not increase the computational time of local lasso estimates much. We only run RAM for a few iterations, as RAM often benefits most from the first few iterations. We then apply the global estimate and the adaptive RAM estimate to the test dataset, and compare their R^2.

The results are in Figure 1.8. The left figure shows the test performance of the adaptive RAM versus the number of iterations in the RAM algorithm. We run our algorithm for a fixed grid of λ values, where λ is the tuning parameter of the l_1 penalty in lasso, and plot the maximum test R^2 at each iteration. The right figure compares the test performance of RAM, AVGM and the global estimate as a function of lasso penalty parameter λ.

From Figure 1.8, we see that the test R^2 of RAM increases rapidly, with around 26% improvement over AVGM, and can almost achieve the global performance in just four iterations. The RAM test R^2 as a function of λ looks quite differently from that of the global

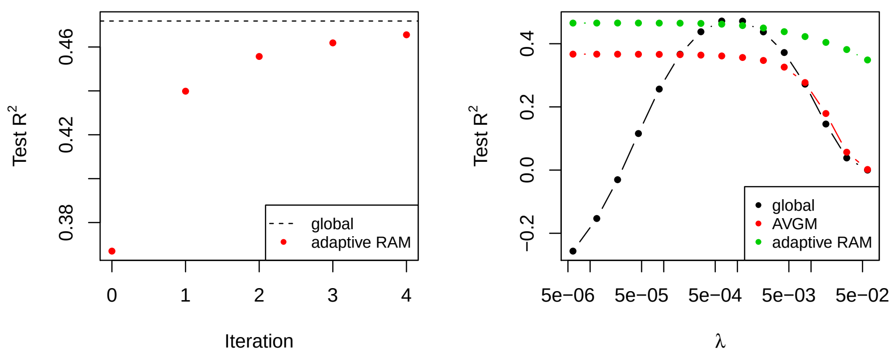


Figure 1.8: The adaptive RAM estimate with $\rho_k = 100k^{0.9}$. The left figure shows the test accuracy of RAM with logistic regression as the black-box solver versus the number of iterations. The black dashed line indicates the test performance of the global estimate. Note that the performance of RAM at iteration 0 is equivalent to that of AVGM. The right figure compares the test performance of RAM, AVGM and the global estimate as a function of lasso penalty parameter λ.

estimate. This is because we did not run RAM until convergence. But our interest is the best R^2 over a grid of λ values instead of the entire curve. The adaptive RAM works very well in improving the best R^2.

As for computational efficiency, for a given λ, running lasso on the entire training data took 43 minutes, while the adaptive RAM took only around 30 secs per iteration.

Through the beer data example, we see that RAM can greatly improve the performance in just a few iterations, and dramatically reduce the computational time by parallel computing.

1.8 Conclusion

We propose a general-purpose algorithm for distributed learning on large-scale data, called the reference average mixture (RAM) algorithm. RAM is communication-efficient, applicable to black-box algorithms, and can recover the global estimator, i.e., the estimator you would get if you could fit the entire data on a single machine, for a class of convex problems.

RAM is an iterative algorithm that distributes the N data samples uniformly to multiple machines, and employs a common reference dataset shared by all machines to improve the performance of local estimates. Larger weight on the reference data results in better performance of RAM algorithm but slower convergence. Therefore, we propose an extension, called adaptive RAM, which uses a sequence of increasing weights on the reference data to achieve better performance with faster convergence.

RAM can have most of the performance boost in the first few iterations, so it does not have to run until convergence in practice. RAM works well when the loss function can be represented as the sum of the data loss and the regularization penalty. For example, it doesn't work well with local regression, random forest and gradient boosted trees, but it works well with ridge regression, Lasso regression, generalized additive models with smoothing spline penalties, and deep learning.

Chapter 2

Community Bayes for High-Dimensional Classification

2.1 Introduction

In the generic classification problem, the outcome of interest G falls into K unordered classes, which for convenience we denote by $\{1, 2, \ldots, K\}$. Our goal is to build a rule for predicting the class label of an item based on p measurements of features $X \in \mathbb{R}^p$. The training set consists of the class labels and features for n items. This is an important practical problem with applications in many fields.

Nonlinear classifiers are favored tools when there are some strong interactions and linear boundaries cannot separate the classes. However, due to the large number of interaction terms, they are often poorly posed when the sample size n_k is not considerably larger than p for any class k, and clearly ill-posed if $n_k \leq p$. One approach is to ignore any interactions by assuming conditional independence of the features, which is called the naive Bayes approach. Naive Bayes classifiers often outperform far more sophisticated alternatives when the dimension p of the feature space is high , but the conditional independence assumption may be too rigid in the presence of strong interactions.

Another approach is to employ a regularization method [13, 14]. When data are from Gaussian distributions, quadratic discriminant analysis (QDA) is often favored. QDA can be derived as the maximum likelihood method for Gaussian populations with different means and different covariance matrices. Although there already exist a number of proposals to regularize linear discriminant analysis (LDA) [15, 16, 17, 18, 19, 20, 21, 22], few methods have been developed to regularize QDA. [23] suggests applying a ridge penalty to within-class covariance matrices, and [24] propose to add a ridge penalty and a ridge fusion penalty to the log-likelihood. For both methods, no elements of the resulting within-class precision matrices will be zero, leading to dense interaction terms in the discriminant functions and difficulties in interpretation. Assuming sparse conditions on both unknown means and covariance matrices, [25] propose to construct mean and covariance estimators by thresholding. [26] assume block-diagonal structure for covariance matrices and estimate each block by l_1 regularization. The blocks are assumed to be of the same size, and are formed based on two sample t statistics. However, both methods can only apply to two-class classification.

In this chapter we propose a method of interaction selection for nonlinear classifiers, for both Gaussian data and non-Gaussian data. This method relaxes the strong independence assumption of naive Bayes, but still allows us to regularize nonlinear classifiers and produce more interpretable results.

In the Gaussian case, we develop a class of rules spanning the range between QDA and naive Bayes by maximizing the penalized log likelihood. A *group lasso* penalty [27] is applied to the (i, j)th element across all K within-class precision matrices, which forces the zeros in the K estimated precision matrices to occur in the same places. This shared pattern of sparsity results in a sparse estimate of interaction terms, making the classifier easy to interpret. We refer to this classification method as *Sparse Quadratic Discriminant Analysis* (SQDA).

When the data are from an unknown general distribution, we don't know the data log likelihood and thus cannot apply the idea of SQDA, i.e., maximizing a penalized log likelihood to get sparse estimates of interactions. However, the estimated precision matrices of SQDA are block diagonal under suitable ordering of the features, and the blocks can be derived by simple thresholding rules on the within-class sample covariance matrices [28]. This implies that the features can be partitioned into several communities that are mutually independent within each class. If we can estimate such communities, then we will also get a sparse estimate of interactions, as features in different communities have no interactions. This feature partition idea also appears in [29, 30, 31].

Therefore, we propose the *community Bayes* model, an interaction selection method that identifies and makes use of conditional independent communities of features. Furthermore, it is a general approach applicable to unknown data distributions and any likelihood-based classifiers. Specifically, the community Bayes works by partitioning features into several communities, solving separate classification problems for each community, and combining the community-wise results into one final prediction. We show that this approach can improve the accuracy and interpretability of the corresponding classifier.

The chapter is organized as follows. In section 2.2 we discuss sparse quadratic discriminant analysis and the sample covariance thresholding rules. In section 2.3 we discuss the community Bayes idea. Simulated and real data examples appear in Section 2.4, concluding remarks in Section 2.5 and the proofs are gathered in the Appendix.

2.2 Community Bayes for Gaussian Data via Sparse Quadratic Discriminant Analysis

In this section, we discuss how to do interaction selection for Gaussian data. When the features follow a Gaussian distribution in each class, it's natural to use quadratic discriminant analysis for the classification task. We propose a method, called Sparse Quadratic Discriminant Analysis (SQDA), that can give sparse estimates of interactions and produce interpretable models. The connected components in the resulting precision matrices correspond exactly to those obtained from simple thresholding rules on the within-class sample covariance matrices [28]. This allows us to decompose the original classification problem into several independent sub-problems and leads to a massive reduction in computational complexity.

Suppose we have training data $(x_i, g_i) \in \mathbb{R}^p \times K$, $i = 1, 2, \ldots, n$, where x_i's are observations with measurements on a set of p features and g_i's are class labels. We assume that the n_k observations within the kth class are identically distributed as $N(\mu_k, \mathbf{\Sigma}_k)$, and the $n = \sum_k n_k$ observations are independent. We let π_k denote the prior for class k, and $\mathbf{\Theta}^{(k)} = \mathbf{\Sigma}_k^{-1}$ denote the precision matrix for class k. Then the quadratic discriminant analysis estimates μ, π, Σ by maximizing the log likelihood

$$\sum_{k=1}^{K} \left[\frac{n_k}{2} \log\det \mathbf{\Theta}^{(k)} - \frac{1}{2} \sum_{g_i = k} (x_i - \mu_k)^T \mathbf{\Theta}^{(k)} (x_i - \mu_k) + n_k \log \pi_k \right] \tag{2.1}$$

and assign sample x_i to the class k whose quadratic discriminant function valued at x_i

$$\delta_k(x_i) = \frac{1}{2} \log\det \mathbf{\Theta}_k - \frac{1}{2} (x_i - \mu_k)^T \mathbf{\Theta}_k (x_i - \mu_k) + \log \pi_k. \tag{2.2}$$

is largest among all classes.

Note that the interaction terms in (2.2) are $\{x_{ij} \mathbf{\Theta}_{jj'}^{(k)} x_{ij'}, j \neq j', k = 1, \ldots, K\}$. We assume that the true interaction terms are sparse, which means that the precision matrices have a similar pattern of sparsity across all classes.

Let $\theta_{jj'} = (\theta_{jj'}^{(1)}, \ldots, \theta_{jj'}^{(K)})$ denote the vector of the (j, j')th element across all K precision matrices. We propose to estimate $\mu, \pi, \mathbf{\Theta}$ by maximizing the penalized log likelihood

$$\sum_{k=1}^{K} \left[\frac{n_k}{2} \log\det \mathbf{\Theta}^{(k)} - \frac{1}{2} \sum_{g_i = k} (x_i - \mu_k)^T \mathbf{\Theta}^{(k)} (x_i - \mu_k) + n_k \log \pi_k \right] - \frac{\lambda}{2} \sum_{j \neq j'} \|\theta_{jj'}\|_2. \tag{2.3}$$

The last term is a *group lasso* penalty, applied to the (j, j')th element of each precision matrix, which forces a similar pattern of sparsity across all K precision matrices. Hence, we refer to this classification method as *Sparse Quadratic Discriminant Analysis* (SQDA).

Let $\mathbf{\Theta} = (\mathbf{\Theta}^{(1)}, \ldots, \mathbf{\Theta}^{(K)})$. Solving (2.3) gives us the estimates

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{g_i = k} x_i \tag{2.4}$$

$$\hat{\pi}_k = \frac{n_k}{n} \tag{2.5}$$

$$\hat{\mathbf{\Theta}} = \operatorname{argmax}_{\mathbf{\Theta}^{(k)} \succ 0} \sum_{k=1}^{K} n_k \left(\log\det\, \mathbf{\Theta}^{(k)} - \operatorname{tr}(\mathbf{S}^{(k)}\mathbf{\Theta}^{(k)}) \right) - \lambda \sum_{j \neq j'} \|\theta_{jj'}\|_2, \tag{2.6}$$

where $\mathbf{S}^{(k)} = \frac{1}{n_k} \sum_{g_i = k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T$ is the sample covariance matrix for class k.

This model has several important features:

1. λ is a nonnegative tuning parameter controlling the simultaneous shrinkage of all precision matrices toward diagonal matrices. The value $\lambda = 0$ gives rise to QDA, whereas $\lambda = \infty$ yields the naive Bayes classifier. Values between these limits represent degrees of regularization less severe than naive Bayes. Since it is often the case that even small amounts of regularization can largely eliminate quite drastic instability, smaller values of λ (smaller than ∞) have the potential of superior performance when some features have strong partial correlations.

2. The group lasso penalty forces $\hat{\mathbf{\Theta}}^{(1)}, \ldots, \hat{\mathbf{\Theta}}^{(K)}$ to share the locations of the nonzero elements, leading to a sparse estimate of interactions. Therefore, our method produces more interpretable results.

3. The optimization problem (2.6) can be separated into independent sub-problems of the same form by simple screening rules on $\mathbf{S}^{(1)}, \ldots, \mathbf{S}^{(K)}$. This leads to a potentially massive reduction in computational complexity.

The convex optimization problem (2.6) can be quickly solved by R package JGL [28]. Not for the purpose of classification as here, [28] propose to jointly estimate multiple related

Gaussian graphical models by maximizing the group graphical lasso

$$\max_{\mathbf{\Theta}^{(k)}\succ 0} \sum_{k=1}^{K} n_k \left(\log\det\mathbf{\Theta}^{(k)} - \mathrm{tr}(\mathbf{S}^{(k)}\mathbf{\Theta}^{(k)})\right) - \lambda_1 \sum_{j\neq j'} \|\theta_{jj'}\|_1 - \lambda_2 \sum_{j\neq j'} \|\theta_{jj'}\|_2. \quad (2.7)$$

They use the additional lasso penalty to further encourage sparsity within $(\theta_{jj'}^{(1)}, \ldots, \theta_{jj'}^{(K)})$. However, for the purpose of classification, our goal is to identify interaction terms – that is, we are interested in whether $(\theta_{jj'}^{(1)}, \ldots, \theta_{jj'}^{(K)})$ is a zero vector instead of whether each element $\theta_{jj'}^{(k)}$ is zero. Therefore, we only use the group lasso penalty to estimate precision matrices.

2.2.1 An illustrative example

We illustrate our method with a handwritten digit recognition example. Here we focus on the sub-task of distinguishing handwritten 3s and 8s. We use the same data as [32], who normalized binary images for size and orientation, resulting in 8-bit, 16×16 gray scale images. There are 658 threes and 542 eights in our training set, and 166 test samples for each. Figure 2.1 shows a random selection of 3s and 8s.

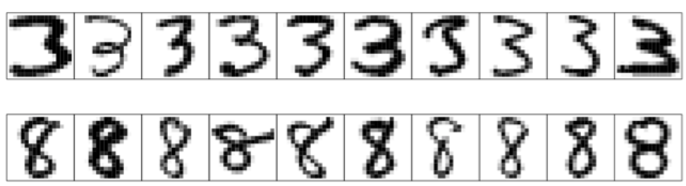


Figure 2.1: *Examples of digitized handwritten 3s and 8s. Each image is a 8 bit,* 16×16 *gray scale version of the original binary image.*

Because of their spatial arrangement the features are highly correlated and some kind of smoothing or filtering always helps. We filtered the data by replacing each non-overlapping 2×2 pixel block with its average. This reduces the dimension of the feature space from 256 to 64.

We compare our method with QDA, naive Bayes and a variant of RDA which we refer to as diagonal regularized discriminant analysis (DRDA). The estimated covariance matrix of

DRDA for class k is

$$\hat{\mathbf{\Sigma}}^{(k)}(\lambda) = (1-\lambda)\mathbf{S}^{(k)} + \lambda \text{diag}(\mathbf{S}^{(k)}), \text{ with } \lambda \in [0,1]. \tag{2.8}$$

The tuning parameters are selected by 5-fold cross validation. Table 2.1 shows the misclassification errors of each method and the corresponding standardized tuning parameters $s = P(\hat{\mathbf{\Theta}}(\lambda))/P(\hat{\mathbf{\Theta}}(0))$, where $P(\mathbf{\Theta}) = \Sigma_{j \neq j'} ||\theta_{jj'}^{(1)}, \ldots, \theta_{jj'}^{(K)}||_2$. The results show that SQDA outperforms DRDA, QDA and naive Bayes. In Figure 2.2, we see that the test error of SQDA decreases dramatically as the model deviates from naive Bayes and achieves its minimum at a small value of s, while the test error of DRDA keeps decreasing as the model ranges from naive Bayes to QDA. This is expected since at small values of s, SQDA only includes important interaction terms and shrinks other noisy terms to 0, but DRDA has all interactions in the model once deviating from naive Bayes. To better display the estimated precision matrices, we standardize the estimates to have unit diagonal (the standardized precision matrix is equal to the partial correlation matrix up to the sign of off-diagonal entries). Figure 2.3 displays the standardized sample precision matrices, and the standardized SQDA and DRDA estimates. From Figure 2.3, we can see that SQDA only includes interactions within a diagonal band.

Table 2.1: *Digit classification results of 3s and 8s. Tuning parameters are selected by 5-fold cross validation.*

	SQDA	DRDA	QDA	Naive Bayes
Test error	0.042	0.063	0.063	0.160
Training error	0.024	0.024	0.022	0.119
CV error	0.028	0.043	–	–
Standardized tuning parameter	0.016	0.760	–	–

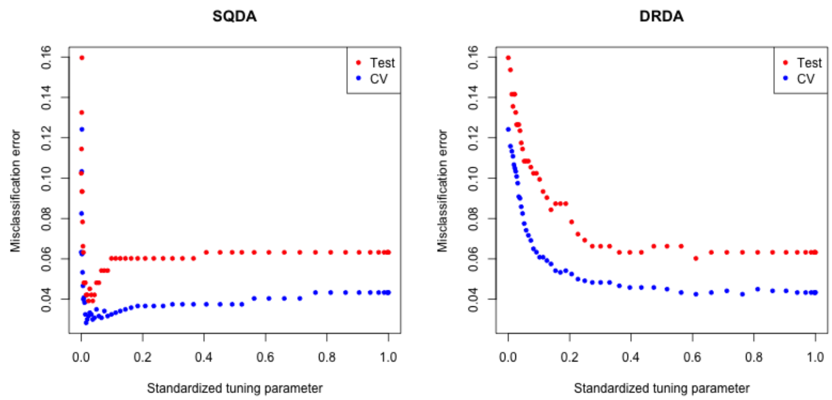


Figure 2.2: *The 5-fold cross-validation errors (blue) and the test errors (red) of SQDA and DRDA on 3s and 8s.*

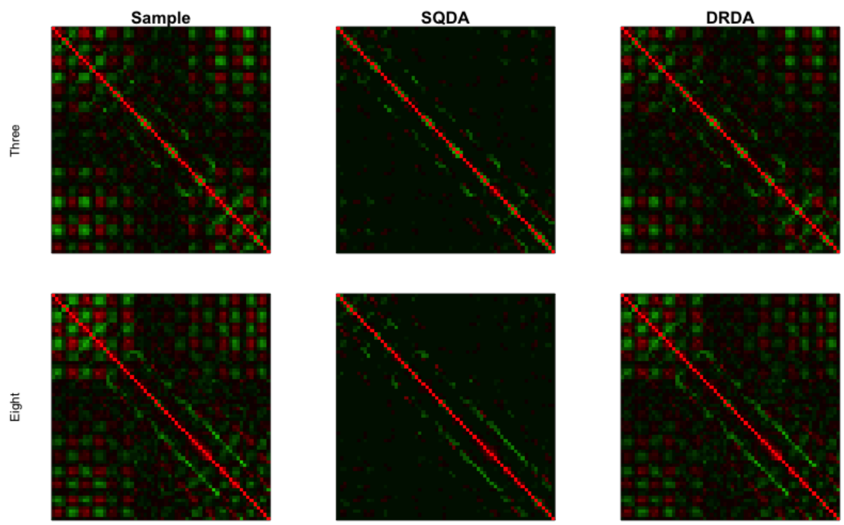


Figure 2.3: *Heat maps of the sample precision matrices and the estimated precision matrices of SQDA and DRDA. Estimates are standardized to have unit diagonal. The first line corresponds to the precision matrix of 3s and the second line corresponds to that of 8s.*

2.2.2 Independent communities of features

[28] establish a connection between the solution to (2.6) and the sample covariance matrices. By simple thresholding rules on the sample covariance matrices $\mathbf{S}^{(1)}, \ldots, \mathbf{S}^{(K)}$, the features can be divided into several independent communities, and thus the optimization problem (2.6) can be separated into several sub-problems of the same form, which reduces the dimensionality

of the original problem and leads to huge speed improvements.

Specifically, let $\hat{\mathbf{\Theta}} = (\hat{\mathbf{\Theta}}^{(1)}, \dots, \hat{\mathbf{\Theta}}^{(K)})$ denote the solution to (2.6) with regularization parameter λ, and define $\tilde{\mathbf{S}}_{jj'} = ||(n_1 \mathbf{S}^{(1)}_{jj'}, \dots, n_K \mathbf{S}^{(K)}_{jj'})||_2$. We can perform a thresholding on the entries of $\tilde{\mathbf{S}}$ and obtain a graph $\mathrm{G}^{(\lambda)} = (\mathcal{V}, \mathbf{E}^{(\lambda)})$ defined by $\mathcal{V} = \{1, 2, \dots, p\}$ and

$$\mathbf{E}^{(\lambda)}_{jj'} = \begin{cases} 1 & \text{if } \tilde{\mathbf{S}}_{jj'} > \lambda, \ i \neq j; \\ 0 & \text{otherwise.} \end{cases} \tag{2.9}$$

Suppose $\mathrm{G}^{(\lambda)}$ admits a decomposition into $k(\lambda)$ connected components

$$\mathrm{G}^{(\lambda)} = \cup_{l=1}^{k(\lambda)} \mathrm{G}^{(\lambda)}_l, \tag{2.10}$$

where $\mathrm{G}^{(\lambda)}_l = (\mathcal{V}^{(\lambda)}_l, \mathbf{E}^{(\lambda)}_l)$ are the components of the graph $\mathbf{G}^{(\lambda)}$. Then, the features can be reordered in such a way that each $\hat{\mathbf{\Theta}}^{(k)}$ is block diagonal

$$\hat{\mathbf{\Theta}}^{(k)} = \begin{pmatrix} \hat{\mathbf{\Theta}}^{(k)}_1 & 0 & \cdots & 0 \\ 0 & \hat{\mathbf{\Theta}}^{(k)}_2 & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \hat{\mathbf{\Theta}}^{(k)}_{k(\lambda)} \end{pmatrix}, \tag{2.11}$$

where the different components represent blocks of indices given by $\mathcal{V}^{(\lambda)}_l, \ l = 1, \dots, k(\lambda)$.

This result allows us to quickly find out the block diagonal structure of the estimated precision matrices by simple screening rules on $\tilde{\mathbf{S}}$. Then one can simply solve the optimization problem (2.6) on the features within each block separately, making problem (2.6) feasible for certain values of λ although it may be impossible to operate on the $p \times p$ variables $\mathbf{\Theta}^{(1)}, \dots, \mathbf{\Theta}^{(K)}$ on a single machine.

Note that the block diagonal structure of the estimated precision matrices also implies that the features can be partitioned into $k(\lambda)$ communities $\{\mathcal{V}^{(\lambda)}_l, l = 1, \dots, k(\lambda)\}$ that are mutually independent within each class. This important property motives us to propose a more general method for interaction selection, which is discussed in the next section.

2.3 Community Bayes for General Data

When the data are from an unknown general distribution, we don't know the data log likelihood and thus cannot apply the idea of SQDA, i.e., maximizing a penalized log likelihood to get sparse estimates of interactions. However, the estimated precision matrices of SQDA are block diagonal under suitable ordering of the features, which implies that the features can be partitioned into several communities that are mutually independent within each class. If we can estimate such communities, then we will also get a sparse estimate of interactions, as features in different communities have no interactions. This can greatly reduce the complexity of the classification problem.

In this section, we generalize this idea to general data distributions and other classification models, and refer to it as the *community Bayes* model. The main idea is to partition the features into several independent communities, and separate the original classification problem into several independent sub-problems. A related work in the regression setting is [30]. The authors propose to split the lasso problem into smaller ones by estimating the connected components of the sample covariance matrix. Their approach involves only one covariance matrix and works specifically for the lasso.

2.3.1 Main idea

We let $X \in \mathbb{R}^p$ denote the feature vector and $G \in \{1, \ldots, K\}$ denote the class variable. Suppose the feature set $\mathcal{V} = \{1, \ldots, p\}$ admits a partition $\mathcal{V} = \cup_{l=1}^{L} \mathcal{V}_l$ such that $X_{\mathcal{V}_1}, \ldots, X_{\mathcal{V}_L}$ are mutually independent conditional on $G = k$ for $k = 1, \ldots, K$, where $X_{\mathcal{V}_l}$ is a subset of

X containing the features in community $\mathcal{V}_l$. Then the posterior probability has the form

$$\begin{aligned}\log p(G=k|X) &= \log p(X|G=k) + \log p(G=k) - \log p(X) && (2.12)\\ &= \sum_{l=1}^{L} \log p(X_{\mathcal{V}_l}|G=k) + \log p(G=k) - \log p(X) && (2.13)\\ &= \sum_{l=1}^{L} \log p(G=k|X_{\mathcal{V}_l}) + (1-L)\log p(G=k) + C(X) && (2.14)\end{aligned}$$

where $C(X) = \sum_{l=1}^{L} \log p(X_{\mathcal{V}_l}) - \log p(X)$ only depends on X and serves as a normalization term.

The equation (2.14) implies an interesting result: we can fit the classification model on each community separately, and combine the resultant posteriors into the posterior to the original problem by simply adjusting the intercept and normalizing it. This result has three important consequences:

1. The global problem completely separates into L smaller tractable sub-problems of the same form, making it possible to solve an otherwise infeasible large-scale problem. Moreover, the modular structure lends it naturally to parallel computation. That is, one can solve these sub-problems independently on separate machines.

2. This idea is quite general and can be applied to any likelihood-based classifiers, including discriminant analysis, multinomial logistic regression, generalized additive models, classification trees, etc.

3. When using a classification model with interaction terms, (2.14) doesn't involve interactions across different communities. Therefore, it has fewer degrees of freedom and thus smaller variance than the global problem.

The key part is how to find conditionally independent communities. In SQDA, we use simple thresholding rules on sample covariance matrices. Moreover, [33] discover an interesting connection between thresholding on a similarity matrix and hierarchical agglomerative clustering. Therefore, to find conditionally independent communities for general data distributions, we propose to use the Spearman's rho, a robust nonparametric rank-based statistics, to directly estimate the unknown correlation matrices, and then apply hierarchical agglomerative clustering to the estimated correlation matrices to get the estimated communities. In the next section, we will show that this procedure consistently identify conditionally independent communities when the data are from a nonparanormal family.

The community Bayes algorithm is summarized in Algorithm 2.1.

Algorithm 2.1 The community Bayes algorithm

1: Compute $\hat{\mathbf{R}}^{(k)}$, the estimated correlation matrix based on the Spearman's rho for each class k.
2: Perform average, single or complete linkage clustering with similarity matrix $\tilde{\mathbf{R}}$, where $\tilde{\mathbf{R}}_{ij} = ||(n_1\hat{\mathbf{R}}^{(1)}_{ij}, \ldots, n_K\hat{\mathbf{R}}^{(K)}_{ij})||_2$, and cut the dendrogram at level τ to produce vertex-partition $\mathcal{V} = \cup_{l=1}^{L} \hat{\mathcal{V}}_l$.
3: For each community $l = 1, \ldots, L$, estimate $\log p(G = k|X_{\hat{\mathcal{V}}_l})$ using a classification method of choice.
4: Pick τ and any other tuning parameters by cross-validation.

2.3.2 Community estimation

In this section we address the key part of the community Bayes model: how to find conditionally independent communities. We propose to derive the communities by applying average, single or complete linkage clustering to the correlation matrices, which are estimated based on nonparametric rank-based statistics. In the case where data are from a nonparanormal family, we prove that given knowledge of L, this procedure consistently identify conditionally independent communities.

We assume that $X = (X_1, \ldots, X_p)^T|G = k$ follows a nonparanormal distribution $NPN(\mu^{(k)}, \mathbf{\Sigma}^{(k)}, f^{(k)})$ [34] and $\mathbf{\Sigma}^{(k)}$ is nonsingular. That is, there exists a set of univariate strictly increasing transformations $f^{(k)} = \{f_j^{(k)}\}_{j=1}^p$ such that

$$Z^{(k)} := f^{(k)}(X)|G = k \sim N(\mu^{(k)}, \mathbf{\Sigma}^{(k)}), \tag{2.15}$$

where $f^{(k)}(X) = (f_1^{(k)}(X_1), \ldots, f_p^{(k)}(X_p))^T$. Notice that the transformation functions $f^{(k)}$ can be different for different classes. To make the model identifiable, $f^{(k)}$ preserves the population mean and standard deviations: $\mathbb{E}(X_j|G = k) = E(f_j^{(k)}(X_j)|G = k) = \mu_j^{(k)}$, $\mathrm{Var}(X_j|G = k) = \mathrm{Var}(f_j^{(k)}(X_j)|G = k) = {\sigma_j^{(k)}}^2$. [34] prove that the nonparanormal distribution is a Gaussian copula when the transformation functions are monotone and differentiable.

Let $\mathbf{R}^{(k)}$ denote the correlation matrix of the Gaussian distribution $Z^{(k)}$, and define

$$\mathcal{E}_{ij} = \begin{cases} 1 & \text{if } (\mathbf{R}_{ij}^{(1)}, \ldots, \mathbf{R}_{ij}^{(K)}) \neq \vec{\mathbf{0}},\ i \neq j; \\ 0 & \text{otherwise.} \end{cases} \tag{2.16}$$

Suppose the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ admits a decomposition into L connected components $\mathcal{G} = \cup_{l=1}^L \mathcal{G}_l$. Then the vertex-partition $\mathcal{V} = \cup_{l=1}^L \mathcal{V}_l$ induced by this decomposition gives us exactly the conditionally independent communities we need. This is because

$$\mathcal{G}_l \text{ and } \mathcal{G}_{l'} \text{ are disconnected} \Leftrightarrow Z_{\mathcal{V}_l}^{(k)} \perp\!\!\!\perp Z_{\mathcal{V}_{l'}}^{(k)},\ \forall k \Leftrightarrow X_{\mathcal{V}_l} \perp\!\!\!\perp X_{\mathcal{V}_{l'}}|G = k,\ \forall k. \tag{2.17}$$

Let $(x_i, g_i) \in \mathbb{R}^p \times K,\ i = 1, \ldots, n$ be the training data where $x_i = (x_{i1}, \ldots, x_{ip})^T$. We estimate the correlation matrices using Nonparanormal SKEPTIC [35], which exploits the Spearman's rho and Kendall's tau to directly estimate the unknown correlation matrices. Since the estimated correlation matrices based on the Spearman's rho and Kendall's tau have similar theoretical performance, we only adopt the ones based on the Spearman's rho here.

In specific, let $\hat{\rho}_{jj'}^{(k)}$ be the Spearman's rho between features j and j' based on n_k samples in class k, i.e., $\{x_i|g_i = k, i = 1, \ldots, n\}$. Then the estimated correlation matrix for class k is

$$\hat{\mathbf{R}}^{(k)}_{jj'} = \begin{cases} 2\sin(\frac{\pi}{6}\hat{\rho}^{(k)}_{jj'}) & j \neq j'; \\ 1 & j = j'. \end{cases} \tag{2.18}$$

Notice that the graph defined by $\mathbf{R}^{(k)}$ has the same vertex-partition as that defined by $(\mathbf{R}^{(k)})^{-1}$. Inspired by the exact thresholding result of SQDA, we define $\tilde{\mathbf{R}}_{jj'} = ||(n_1\hat{\mathbf{R}}^{(1)}_{jj'}, \ldots, n_K\hat{\mathbf{R}}^{(K)}_{jj'})||_2$ and perform exact thresholding on the entries of $\tilde{\mathbf{R}}$ at a certain level τ, where τ is estimated by cross-validation. The resultant vertex-partition yields an estimate of the conditionally independent communities. Furthermore, there is an interesting connection to hierarchical clustering. Specifically, the vertex-partition induced by thresholding matrix $\tilde{\mathbf{R}}$ corresponds to the subtrees when we apply single linkage agglomerative clustering to $\tilde{\mathbf{R}}$ and then cut the dendrogram at level τ [33]. Single linkage clustering tends to produce trailing clusters in which individual features are merged one at a time. However, Theorem 2.1 shows that given knowledge of the true number of communities L, application of single, average or complete linkage agglomerative clustering on $\tilde{\mathbf{R}}$ consistently estimates the vertex-partition of $\mathcal{G}$.

THEOREM 2.1. *Assume that $\mathcal{G}$ has L connected components and $\min_k n_k \geq \frac{21}{\log p} + 2$. Define $\tilde{\mathbf{R}}^0_{ij} = ||(n_1\mathbf{R}^{(1)}_{ij}, \ldots, n_K\mathbf{R}^{(K)}_{ij})||_2$ and let*

$$\min_{i,j\in\mathcal{V}_l; l=1,\ldots,L} \tilde{\mathbf{R}}^0_{ij} \geq 16\pi K\sqrt{n_k \log p}, \text{ for } k = 1, \ldots, K. \tag{2.19}$$

Then the estimated vertex-partition $\mathcal{V} = \cup_{l=1}^L \hat{\mathcal{V}}_l$ resulting from performing SLC, ALC, or CLC with similarity matrix $\tilde{\mathbf{R}}$ satisfies $P(\exists l : \hat{\mathcal{V}}_l \neq \mathcal{V}_l) \leq \frac{K}{p^2}$.

Proof. Let $a = \min_{i,j\in\mathcal{V}_l;l=1,\ldots,L} \tilde{\mathbf{R}}^0_{ij}$. First note that $\{\max_{ij} \left|\tilde{\mathbf{R}}_{ij} - \tilde{\mathbf{R}}^0_{ij}\right| < \frac{a}{2}\} \Rightarrow \{\hat{\mathcal{V}}_l =$

$\mathcal{V}_l, \forall l\}$ [33]. Then

$$\begin{aligned} P\left(\exists l, \hat{\mathcal{V}}_l \neq \mathcal{V}_l\right) &\leq P\left(\max_{ij}\left|\tilde{\mathbf{R}}_{ij} - \tilde{\mathbf{R}}^0_{ij}\right| \geq \frac{a}{2}\right) \\ &\leq P\left(\sum_{k=1}^{K}\max_{ij} n_k \left||\hat{\mathbf{R}}^{(k)}_{ij}| - |\mathbf{R}^{(k)}_{ij}|\right| \geq \frac{a}{2}\right) \\ &\leq \sum_{k=1}^{K} P\left(\max_{ij} n_k \left||\hat{\mathbf{R}}^{(k)}_{ij}| - |\mathbf{R}^{(k)}_{ij}|\right| \geq \frac{a}{2K}\right) \\ &\leq \sum_{k=1}^{K} P\left(\max_{ij}\left|\hat{\mathbf{R}}^{(k)}_{ij} - \mathbf{R}^{(k)}_{ij}\right| \geq \frac{a}{2n_k K}\right) \end{aligned}$$

The second inequality is because $\left|\sqrt{a_1^2 + \ldots + a_m^2} - \sqrt{b_1^2 + \ldots + b_m^2}\right| \leq |a_1 - b_1| + \ldots + |a_m - b_m|$.

Since $a \geq 16\pi K\sqrt{n_k \log p}$, we have $P(\max_{ij}\left|\hat{\mathbf{R}}^{(k)}_{ij} - \mathbf{R}^{(k)}_{ij}\right| \geq \frac{a}{2n_k K}) \leq \frac{1}{p^2}$ by Theorem 4.1 of [35]. Therefore $P(\exists l, \hat{\mathcal{V}}_l \neq \mathcal{V}_l) \leq \frac{K}{p^2}$. □

A sufficient condition for (2.19) is

$$\min_{i,j\in\mathcal{V}_l; l=1,\ldots,L} ||(\mathbf{R}^{(1)}_{ij}, \ldots, \mathbf{R}^{(K)}_{ij})||_2 \geq 16\pi K\sqrt{\frac{\log p}{n_{max}}\frac{n_{max}}{n_{min}}}, \tag{2.20}$$

where $n_{min} = \min_k n_k$ and $n_{max} = \max_k n_k$. Therefore, Theorem 2.1 establishes the consistency of identification of conditionally independent communities by performing hierarchical clustering using SLC, ALC or CLC, provided that $n_{max} = \Omega(\log p)$, $n_{max}/n_{min} = O(1)$ as $n_k, p \to \infty$, and provided that no within-community element of $\mathbf{R}^{(k)}$ is too small in absolute value for all k.

2.4 Examples

2.4.1 Sparse quadratic discriminant analysis

In this section we study the performance of SQDA in several simulated examples and a real data example. The results show that SQDA achieves a lower misclassification error and better interpretability compared to naive bayes, QDA and a variant of RDA. Since RDA proposed by [23] has two tuning parameters, resulting in an unfair comparison, we use a version of RDA that shrinks $\hat{\mathbf{\Sigma}}^{(k)}$ towards its diagonal:

$$\hat{\mathbf{\Sigma}}^{(k)}(\lambda) = (1-\lambda)\mathbf{S}^{(k)} + \lambda \text{diag}(\mathbf{S}^{(k)}), \text{ with } \lambda \in [0,1] \tag{2.21}$$

Note that $\lambda = 0$ corresponds to QDA and $\lambda = 1$ corresponds to the naive Bayes classifier. For any $\lambda < 1$, the estimated covariance matrices are dense. We refer to this version of RDA as diagonal regularized discriminant analysis (DRDA) in the rest of this chapter.

We report the test misclassification errors and its corresponding standardized tuning parameters of these four classifiers. Let $P(\mathbf{\Theta}) = \sum_{i \neq j} ||(\theta_{ij}^{(1)}, \ldots, \theta_{ij}^{(K)})||_2$. The standardized tuning parameter is defined as $s = P(\hat{\mathbf{\Theta}}(\lambda))/P(\hat{\mathbf{\Theta}}(0))$, with $s = 0$ corresponding to the naive Bayes classifier and $s = 1$ corresponding to QDA.

Simulated examples

The data generated in each experiment consists of a training set, a validation set to tune the parameters, and a test set to evaluate the performance of our chosen model. Following the notation of [36], we denote ./././. the number of observations in the training, validation and test sets respectively. For every data set, the tuning parameter minimizing the validation misclassification error is chosen to compute the test misclassification error.

Each experiment was replicated 50 times. In all cases the population class conditional distributions were normal, the number of classes was $K = 2$, and the prior probability of each class was taken to be equal. The mean for class k was taken to have Euclidean norm $\sqrt{\mathrm{tr}(\mathbf{\Sigma}^{(k)})/p}$ and the two means were orthogonal to each other. We consider three different models for precision matrices and in all cases the K precision matrices are from the same model:

- Model 1. Full model: $\theta_{ij}^{(k)} = 1$ if $i = j$ and $\theta_{ij}^{(k)} = \rho_k$ otherwise.
- Model 2. Decreasing model: $\theta_{ij}^{(k)} = \rho_k^{|i-j|}$.
- Model 3. Block diagonal model with block size q: $\theta_{ij}^{(k)} = 1$ if $i = j$, $\theta_{ij}^{(k)} = \rho_k$ if $i \neq j$ and $i, j \leq q$ and $\theta_{ij}^{(k)} = 0$ otherwise, where $0 < q < p$.

Table 2.2 and 3.1, summarizing the results for each situation, present the average test misclassification errors over the 50 replications. The quantities in parentheses are the standard deviations of the respective quantities over the 50 replications.

Example 1: Dense interaction terms

We study the performance of SQDA on Model 1 and Model 2 with $p = 8$, $\rho_1 = 0$, and $\rho_2 = 0.8$. We generated 50/50/10000 observations for each class. Table 2.2 summarizes the results.

In this example, both situations have full interaction terms and should favor DRDA. The results show that SQDA and DRDA have similar performance, and both of them give lower misclassification errors and smaller standard deviations than QDA or naive Bayes. The model-selection procedure behaves quite reasonably, choosing large values of the standardized tuning parameter s for both SQDA and DRDA.

Example 2: Sparse interaction terms

We study the performance of SQDA on Model 3 with $\rho_1 = 0$, $\rho_2 = 0.8$ and $q = 4$. We performed experiments for $(p, n) = (8, 50), (20, 200), (40, 800), (100, 1500)$, and for each p we

Table 2.2: *Misclassification errors and selected tuning parameters for simulated example 1. The values are averages over 50 replications, with the standard errors in parentheses.*

	Full	**Decreasing**
Misclassification error		
SQDA	0.129(0.021)	0.103(0.011)
DRDA	0.129(0.020)	0.104(0.013)
QDA	0.238(0.028)	0.170(0.031)
Naive Bayes	0.179(0.026)	0.153(0.032)
Average standardized tuning parameter		
SQDA	0.828(0.258)	0.817(0.237)
DRDA	0.835(0.302)	0.781(0.314)

Table 2.3: *Misclassification errors and selected tuning parameters for simulated example 2. The values are averages over 50 replications, with the standard errors in parentheses.*

	$p = 8$, $n = 50$	$p = 20$, $n = 200$	$p = 40$, $n = 800$	$p = 100$, $n = 1500$
Misclassification error				
SQDA	0.079(0.010)	0.091(0.007)	0.086(0.006)	0.153(0.007)
DRDA	0.088(0.013)	0.109(0.009)	0.108(0.007)	0.188(0.009)
QDA	0.092(0.011)	0.114(0.006)	0.127(0.002)	0.230(0.025)
Naive Bayes	0.107(0.014)	0.121(0.004)	0.114(0.004)	0.215(0.035)
Average standardized tuning parameter				
SQDA	0.529(0.309)	0.190(0.046)	0.109(0.015)	0.024(0.004)
DRDA	0.688(0.314)	0.293(0.345)	0.194(0.230)	0.073(0.059)

generated $n/n/10000$ observations for each class. Table 3.1 summarizes the results.

In this example, all situations have sparse interaction terms and should favor SQDA. Moreover, the sparsity level increases as p increases. As conjectured, SQDA strongly dominates with lower misclassification errors at all dimensionalities. The standardized tuning parameter values for DRDA are uniformly larger than those for SQDA. This is expected because in order to capture the same amount of interactions DRDA needs to include more noises than SQDA.

Real data example: vowel recognition data

This data consists of training and test data with 10 predictors and 11 classes. We obtained the data from the benchmark collection maintained by Scott Fahlman at Carnegie Mellon University. The data was contributed by Anthony Robinson [37]. The classes correspond to 11 vowel sounds, each contained in 11 different words. Here are the words, preceded by the symbols that represent them:

Vowel	Word	Vowel	Word	Vowel	Word
i	heed	a:	hard	U	hood
I	hid	Y	hud	u:	who'd
E	head	O	hod	3:	heard
A	had	C:	hoard		

The word was uttered once by each of the 15 speakers. Four male and four female speakers were used to train the models, and the other four male and three female speakers were used for testing the performance.

This paragraph is technical and describes how the analog speech signals were transformed into a 10-dimensional feature vector. The speech signals were low-pass filtered at 4.7kHz and then digitized to 12 bits with a 10kHz sampling rate. Twelfth-order linear predictive analysis was carried out on six 512-sample Hamming windowed segments from the steady part of the vowel. The reflection coefficients were used to calculate 10 log-area parameters, giving a 10-dimensional input space. Each speaker thus yielded six frames of speech from 11 vowels. This gave 528 frames from the eight speakers used to train the models and 462 frames from the seven speakers used to test the models.

We implement our method on a subset of data, which consists of four vowels "Y", "O", "U" and "u:". These four vowels are selected because their sample precision matrices are approximately sparse, and SQDA usually performs well on such dataset. Figure 2.4 displays the standardized sample precision matrices, and the standardized SQDA and DRDA estimates. In addition to DRDA, QDA and naive Bayes, we also compare our method with three other classifiers, namely the Support Vector Machine (SVM), k-nearest neighborhood (kNN) and Random Forest (RF), in terms of misclassification rate. The SVM, kNN, and Random Forest were implemented by the R packages of "e1071", "class" and "randomForest" with default settings, respectively. The results of these classification procedures are shown in Table 2.4.

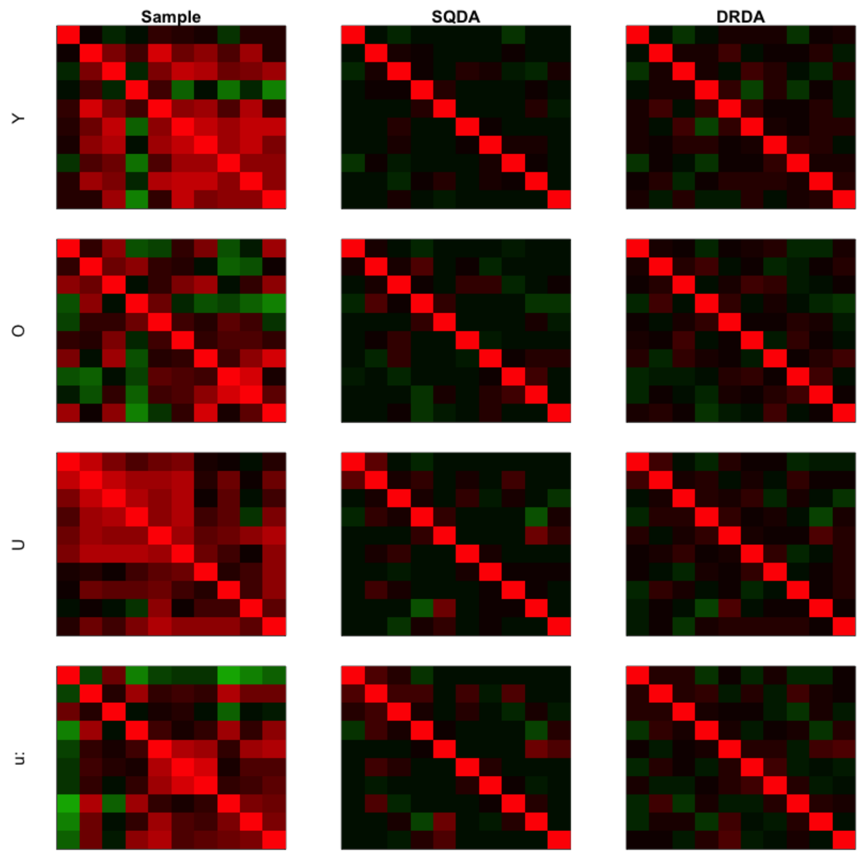


Figure 2.4: *Heat maps of the sample precision matrices and the estimated precision matrices of SQDA and DRDA on vowel data. Estimates are standardized to have unit diagonal.*

The SQDA performs significantly better than the other six classifiers. Compared with the second winner DRDA, the SQDA has a relative gain of (19.7% - 17.2%)/19.7% = 12.7%.

Table 2.4: *Vowel speech classification results. Tuning parameters are selected by 5-fold cross validation.*

	SQDA	DRDA	QDA	Naive Bayes	SVM	kNN	Random Forest
Test error	0.172	0.197	0.351	0.304	0.220	0.274	0.278
CV error	0.239	0.260	–	–	–	–	–

2.4.2 Community Bayes

In this section we study the performance of the community Bayes model using logistic regression as an example. We refer to the classifier that combines the community Bayes algorithm with logistic regression as community logistic regression. We compare the performance of logistic regression (LR) and community logistic regression (CLR) in several simulated examples and a real data example, and the results show that CLR has better accuracy and smaller variance in predictions.

Simulated examples

The data generated in each experiment consists of a training set, a validation set to tune the parameters, and a test set to evaluate the performance of our chosen model. Each experiment was replicated 50 times. In all cases the distributions of $Z^{(k)}$ were normal, the number of classes was $K = 3$, and the prior probability of each class was taken to be equal. The mean for class k was taken to have Euclidean norm $\sqrt{\mathrm{tr}(\mathbf{\Sigma}^{(k)})/(p/2)}$ and the three means were orthogonal to each other. We consider three models for covariance matrices and in all cases the K covariance matrices are the same:

- Model 1. Full model: $\mathbf{\Sigma}_{ij} = 1$ if $i = j$ and $\mathbf{\Sigma}_{ij} = \rho$ otherwise.
- Model 2. Decreasing model: $\mathbf{\Sigma}_{ij} = \rho^{|i-j|}$.
- Model 3. Block diagonal model with q blocks: $\mathbf{\Sigma} = \mathrm{diag}(\mathbf{\Sigma}_1, \dots, \mathbf{\Sigma}_q)$. $\mathbf{\Sigma}_i$ is of size $\lfloor \frac{p}{q} \rfloor \times \lfloor \frac{p}{q} \rfloor$ or $(\lfloor \frac{p}{q} \rfloor + 1) \times (\lfloor \frac{p}{q} \rfloor + 1)$ and is from Model 1.

For simplicity, the transformation functions for all dimensions were the same $f_1^{(k)} = \ldots =$

$f_p^{(k)} = f^{(k)}$. Define $g^{(k)} = (f^{(k)})^{-1}$. In addition to the identity transformation, two different transformations $g^{(k)}$ were employed as in [34]: the symmetric power transformation and the Gaussian CDF transformation. See [34] for the definitions.

We compare the performance of LR and CLR on the above models with $p = 16$, $\rho = 0.5$ and $q = 2, 4, 8$. We generated 20/20/10000 observations for each class and use average linkage clustering to estimate the communities. Two examples were studied:

- Example 1: The transformations $g^{(k)}$ for all three classes are the identity transformation. The population class conditional distributions are normal with different means and common covariance matrix. The logits are linear in this example.

- Example 2: The transformations $g^{(k)}$ for class 1,2 and 3 are the identity transformation, the symmetric power transformation with $\alpha = 3$, and the Gaussian CDF transformation with $\mu_{g0} = 0$ and $\sigma_{g0} = 1$ respectively. α, μ_{g0} and σ_{g0} are defined in [34]. The power and CDF transformations map a univariate normal distribution into a highly skewed and a bi-modal distribution respectively. The logits are nonlinear in this example.

Table 2.5 summarizes the test misclassification errors of these two methods and the estimated numbers of communities. CLR performs well in all cases with lower test errors and smaller standard errors than LR, including the ones where the covariance matrices don't have a block structure. The community Bayes algorithm introduces a more significant improvement when the class conditional distributions are not normal.

Real data example: email spam

The data for this example consists of information from 4601 email messages, in a study to screen email for "spam". The true outcome `email` or `spam` is available, along with the relative frequencies of 57 of the most commonly occurring words and punctuation marks in

Table 2.5: *Misclassification errors and estimated numbers of communities for the two simulated examples. The values are averages over 50 replications, with the standard errors in parentheses.*

	Full	**Decreasing**		**Block**	
			$q = 2$	$q = 4$	$q = 8$
Example 1					
CLR test error	0.253(0.041)	0.221(0.032)	0.227(0.033)	0.203(0.035)	0.262(0.030)
LR test error	0.274(0.040)	0.286(0.036)	0.271(0.039)	0.278(0.041)	0.325(0.036)
Number of communities	5.220(4.292)	6.040(4.262)	4.700(3.882)	5.320(3.395)	7.780(3.935)
Example 2					
CLR test error	0.221(0.049)	0.227(0.040)	0.208(0.041)	0.203(0.036)	0.239(0.031)
LR test error	0.255(0.061)	0.282(0.062)	0.263(0.062)	0.266(0.063)	0.309(0.057)
Number of communites	7.480(4.253)	7.320(3.977)	6.940(4.533)	6.220(3.627)	9.160(3.782)

the email message. Since most of the spam predictors have a very long-tailed distribution, we log-transformed each variable (actually $\log(x + 0.1)$) before fitting the models.

We compare CLR with LR and four other commonly used classifiers, namely SVM, kNN, Random Forest and Boosting. Same as in section 2.4.1, the SVM, kNN, Random Forest and Boosting were implemented by the R packages of "e1071", "class", "randomForest" and "ada" with default settings, respectively. The range of the number of communities for CLR were fixed to be between 1 and 20, and the optimal number of communities were selected by 5-fold cross validation. We randomly chose 1000 samples from this data, and split them into equally sized training and test sets. We repeated the random sampling and splitting 20 times. The mean misclassification percentage of each method is listed in Table 2.6 with standard error in parenthesis.

Table 2.6: *Email spam classification results. The values are test misclassification errors averaged over 20 replications, with standard errors in parentheses.*

CLR	LR	SVM	kNN	Random Forest	Boosting
0.068(0.019)	0.087(0.026)	0.070(0.011)	0.106(0.017)	0.071(0.010)	0.075(0.012)

The mean test error rate for LR is 8.7%. By comparison, CLR has a mean test error rate of 6.8%, yielding a 21.8% improvement. Figure 2.5 shows the test error and the cross-validation error of CLR over the range of the number of communities for one replication. It corresponds to LR when the number of communities is 1. The estimated number of communities in this replication is 6, and these communities are listed below. Most features in community 1 are negatively correlated with spam, while most features in community 2 are positively correlated.

- `hp, hpl, george, lab, labs, telnet, technology, direct, original, pm, cs, re, edu, conference, 650, 857, 415, 85, 1999, ch;, ch(, ch[, CAPAVE, CAPMAX, CAPTOT.`
- `over, remove, internet, order, free, money, credit, business, email, mail, receive, will, people, report, address, addresses, make, all, our, you, your, 000, ch!, ch$.`
- `font, ch#.`
- `data, project.`
- `parts, meeting, table.`
- `3d.`

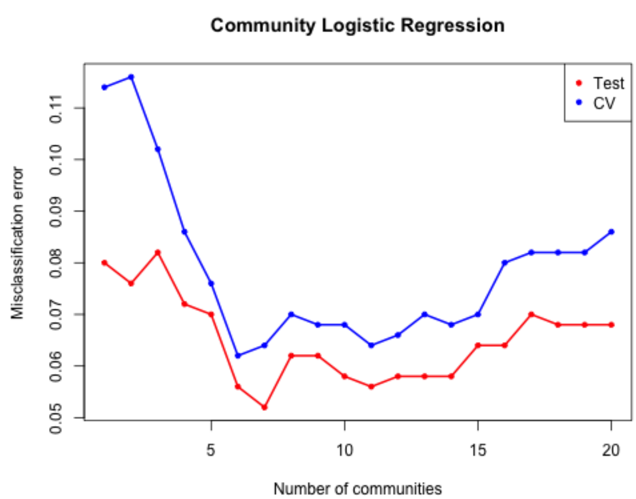


Figure 2.5: *The 5-fold cross-validation errors (blue) and the test errors (red) of CLR on the spam data.*

2.5 Conclusions

In this chapter we have proposed sparse quadratic discriminant analysis, a classifier ranging between QDA and naive Bayes through a path of sparse graphical models. By allowing interaction terms into the model, this classifier relaxes the strict and often unreasonable assumption of naive Bayes. Moreover, the resulting estimates of interactions are sparse and easier to interpret compared to other existing classifiers that regularize QDA.

Motivated by the connection between the estimated precision matrices and single linkage clustering with the sample covariance matrices, we present the community Bayes model, a simple procedure applicable to non-Gaussian data and any likelihood-based classifiers. By exploiting the block-diagonal structure of the estimated correlation matrices, we reduce the problem into several sub-problems that can be solved independently. Simulated and real data examples show that the community Bayes model can improve accuracy and reduce variance in predictions.

Chapter 3

Cross-Validation for Unsupervised Learning

3.1 Introduction

In unsupervised learning, we have a set of n observations $(x_1, x_2, \dots, x_n)$ of a random p-dimensional vector X having joint density $P(X)$. The goal of unsupervised learning is to directly infer the properties of $P(X)$. Many estimation problems can be cast as unsupervised learning. One problem is to identify low-dimensional manifolds within the X-space that represent high data density. This discovers the association among the variables and attempts to represent them as functions of a smaller set of "latent" variables. Methods fall in this category include principal component analysis, multidimensional scaling, self-organizing maps and principal curves. Another common problem is to find multiple regions of the X-space that contain models of $P(X)$. This provides information about whether or not $P(X)$ can be represented by a mixture of simpler densities. Cluster analysis and mixture modeling are examples in this category.

Many unsupervised learning problems involve model selection. Many manifold learning methods require choosing a bandwidth or a kernel. In principal component analysis, we need to determine the number of components to keep. In cluster analysis, one need to choose the number of clusters in the data. With supervised learning, there is a clear measure of model performance, usually as the expected loss over the joint distribution $P(X, Y)$. This measure can be used to evaluate the effectiveness of different models and conduct model selection. However, due to the lack of the supervisor Y, it is very difficult to evaluate unsupervised learning methods. Model selection is often done in an ad-hoc manner in these contexts, such as rules of thumb and manual inspection. Such informal selection rules are problematic as the model effectiveness is a matter of opinion and different people may come to different conclusions.

We would like to have a general and well-specified procedure for model selection in unsupervised learning. *Cross-validation* (CV) is probably the simplest and most widely used method for model evaluation in supervised setting, as it is nonparametric and can work in a variety of contexts. This method directly estimates the generalization error. It divides the data into a training set and a test set, fits a model to the training set, and evaluates the model error on the test set. It repeats the above procedure multiple times over different partitions, and then averages the model error over all replicates.

However, cross-validation is not directly applicable in unsupervised learning. When evaluating the model error on the test set, cross-validation uses X_{test} to predict Y_{test}, and compare the predictions with Y_{test}. But in unsupervised learning, there is no Y and thus there is nothing to predict. Therefore, this requires us to consider more general way of partitioning the data.

Several methods have been proposed to extend cross-validation to unsupervised learning. Wold [38] suggests to partition the data by leaving out random elements of the data matrix X as the test set. He treats the test set as the missing data, and uses an algorithm called NIPALS to work with the incomplete data. However, his estimate of the missing entries is dependent on the missing entries themselves, which makes the cross-validation error biased downwards. In the field of collaborative filtering and matrix completion, a similar idea has also been used for model selection and parameter tuning – leaving out random observed matrix elements as the test set and fitting the targeted matrix completion algorithm using the remaining observed elements [39, 40, 41, 42, 43]. However, there is no study experimenting and justifying this approach, and it has not been applied to unsupervised learning problems with complete data. Gabriel [44] proposes another way to partition the data. He randomly selects one variable as the response Y, and then apply the leave-one-out cross-validation. The limitation of Gabriel's approach is that each test set, also called holdout, is of size 1×1, which makes it very expensive for large problems. Owen and Perry [45] generalizes this approach to $r \times s$ holdouts and call it *bi-cross-validation* (BCV). Specifically, suppose the holdout set is the upper left submatrix, then X is partitioned as follows

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \tag{3.1}$$

where $A \in \mathbb{R}^{r\times s}$, $B \in \mathbb{R}^{r\times(p-s)}$, $C \in \mathbb{R}^{(n-r)\times s}$ and $D \in \mathbb{R}^{(n-r)\times(p-s)}$. Their estimate on the holdout A relies on a truncated SVD of D. This makes it a natural approach for selecting the number of components in principal component analysis and nonnegative matrix factorization, but it is not clear how to apply it to other unsupervised learning problems. Moreover, the validity of the estimate requires assuming X and D have the same rank. This assumption is clearly violated when $\min(n-r, p-s) < \text{rank}(X)$, so their approach is not applicable when X has a large rank.

In this section, we develop a cross-validation procedure that is generally applicable to unsupervised learning methods. Here we focus on the scattered random holdout proposed by Wold [38]. The K-fold cross-validation approach on the data matrix X is as follows. We randomly partition the observed elements into K folds, where each fold is a set of scattered matrix elements. For each of the K folds, we leave it out as the hold-out set, fit to the remaining $K-1$ folds using some matrix completion algorithm, and evaluate its performance on the hold-out set. In general, the matrix completion algorithm used for estimating the hold-out set can be an expectation-maximization (EM) adaption of the unsupervised learning algorithm that we are considering.

The outline of this section is as follows. Section 2 describes the assumptions and notations that we will use in this chapter. Section 3 presents the cross-validation approach. Section 4 shows how to apply our approach to matrix denoising and principle component analysis, and provides a computationally efficient approximation for leave-one-out cross-validation. In section 5, we extends our approach to incomplete data matrix X, and applies it to matrix completion problems. We looks at cross-validation for clustering analysis in section 6.

3.2 Assumptions and Notations

Suppose we have noisy data matrix X about an $n \times p$ matrix X^0 of interest,

$$X = X^0 + E, \tag{3.2}$$

where we call X^0 the signal part and E the noise part. Without loss of generality, we assume $n \leq p$ and define the aspect ratio as $\beta = \frac{n}{p}$. In this chapter, we are concerned with situations where the estimand has some structure, namely, X^0 has low rank. Specifically, let

$$X^0 = \sqrt{p} U D V^T, \tag{3.3}$$

where $U \in \mathbb{R}^{n \times k_0}$, $V \in \mathbb{R}^{p \times k_0}$, and $D = \mathrm{diag}(d_1, d_2, \ldots, d_{k_0})$, with $U^T U = I_{k_0}$, $V^T V = I_{k_0}$, and $d_1 \geq d_2 \geq \cdots \geq d_{k_0} > 0$.

Let $\Omega \subset \{1, \ldots, n\} \times \{1, \ldots, p\}$ denote the indices of observed entries. We use the symbol $*$ to denote a missing value, and let $\mathbb{R}_* = \mathbb{R} \cup \{*\}$. Define a matrix $P_\Omega(X) \in \mathbb{R}_*^{n \times p}$ with elements

$$P_\Omega(X)(i,j) = \begin{cases} X_{ij} & \text{if } (i,j) \in \Omega \\ * & \text{otherwise,} \end{cases} \tag{3.4}$$

which is a projection of the matrix $X_{n\times p}$ onto the observed entries. Similarly $P_\Omega^\perp(X)$ has elements

$$P_\Omega^\perp(X)(i,j) = \begin{cases} X_{ij} & \text{if } (i,j) \notin \Omega \\ * & \text{otherwise.} \end{cases} \tag{3.5}$$

Finally, we define

$$||P_\Omega(X)||_F^2 = \sum_{(i,j)\in\Omega} X_{ij}^2. \tag{3.6}$$

This notation allows us to describe matrices with missing entries.

We denote the estimate of X^0 based on observations $P_\Omega(X)$ as $\hat{X}$. There are two types of errors that we are interested in. The first one is the squared Frobenius *model error*

$$\mathrm{ME}(\hat{X}) = ||X^0 - \hat{X}||_F^2. \tag{3.7}$$

Another one is the *prediction error*. We let E' be a matrix independent of E but having the same distribution conditionally on X^0. We set $X' = X^0 + E'$ and define the prediction error

$$\mathrm{PE}(\hat{X}) = \mathbb{E}||X' - \hat{X}||_F^2. \tag{3.8}$$

If E is independent of X^0, then

$$\begin{aligned} \mathrm{PE}(\hat{X}) &= \mathbb{E}||X^0 - \hat{X} + E'||_F^2 \\ &= \mathbb{E}[\mathrm{ME}(\hat{X})] + \mathbb{E}||E||_F^2. \end{aligned}$$

The prediction error is thus equal to the sum of the expected model error and an irreducible error term.

When the estimator $\hat{X}(\lambda)$ is dependent on a tuning parameter λ, we have $\mathrm{PE}(\lambda) =$

$\mathrm{PE}(\hat{X}(\lambda))$ and denote its minimizer as

$$\lambda^*_{\mathrm{PE}} = \arg\min_{\lambda} \mathrm{PE}(\lambda). \tag{3.9}$$

We denote the soft-thresholding operator on X as

$$S_\lambda(X) \equiv U D_\lambda V^T, \tag{3.10}$$

with $D_\lambda = \mathrm{diag}[(d_1 - \lambda)_+, \dots, (d_r - \lambda)_+]$, UDV^T is the *singular value decomposition* (SVD) of X, $D = \mathrm{diag}[d_1, \dots, d_r]$ and $t_+ = \max(t, 0)$.

3.3 Cross-Validation Approach

Let $f(\cdot)$ denote some unsupervised learning algorithm and $\hat{X} = f(P_\Omega(X))$ denote the estimate of X^0 based on $P_\Omega(X)$. For certain unsupervised learning algorithms such as principal component analysis and k-means clustering, $f(\cdot)$ can only be applied to complete matrices, i.e. $\Omega = \{1, \dots, n\} \times \{1, \dots, p\}$. In this section we describe our hold-out approach for getting a CV estimate of $\mathrm{PE}(\hat{X})$. It's possible to get an estimate of $\mathrm{ME}(\hat{X})$ from the estimate of $\mathrm{PE}(\hat{X})$ by substracting an estimate of the irreducible error. For now, though, we choose to focus just on estimating $\mathrm{PE}(\hat{X})$.

The K-fold CV approach on the observed matrix $P_\Omega(X)$ is as follows. We randomly partition the observed elements Ω into K folds. For each of the K folds, we leave it out as the hold-out set, fit to the remaining $K-1$ folds, and evaluate its performance on the hold-out set.

Specifically, denote the K folds as $\{\Omega_1, \dots, \Omega_K\}$ and $\Omega = \cup_{k=1}^K \Omega_k$. For each $k = 1, \dots, K$, we fit to the other $K-1$ folds to get an estimate of X,

$$\hat{X}^{(i)} = \tilde{f}\left(P_{\Omega \setminus \Omega_k}(X)\right), \tag{3.11}$$

and compute its error in estimating the ith fold

$$\mathrm{err}_i = \|P_{\Omega_i}(X - \hat{X}^{(k)})\|_F^2. \tag{3.12}$$

Here $\tilde{f}(\cdot)$ is a matrix completion algorithm that behaves similarly to $f(\cdot)$. If $f(\cdot)$ can work with incomplete matrices, then $\tilde{f}(\cdot) = f(\cdot)$. Otherwise, we need to find an approximation to $f(\cdot)$ that can be applied to incomplete matrices. More details on $\tilde{f}(\cdot)$ are discussed later in this section.

Then the CV error is

$$\mathrm{err}_{\mathrm{CV}} = \sum_{i=1}^{K} \mathrm{err}_k. \tag{3.13}$$

Let $\hat{S}$ be a matrix filled with CV estimates, i.e., $\hat{S}_{ij} = \hat{X}_{ij}^{(k)}$ for any $(i,j) \in \Omega_k$ and for each fold Ω_k. Then the CV error can be also expressed as

$$\mathrm{err}_{\mathrm{CV}} = \|P_{\Omega}(X - \hat{S})\|_F^2. \tag{3.14}$$

If we assume that the elements of the noise matrix E are independent, then X_{ij} and $\hat{S}_{ij}$ are independent for each entry $(i,j) \in \Omega$. Thus we can use $\mathrm{err}_{\mathrm{CV}}$ as an estimate of the prediction error $\mathrm{PE}(\hat{X})$, i.e.,

$$\hat{\mathrm{PE}}(\hat{X}) = \mathrm{err}_{\mathrm{CV}}. \tag{3.15}$$

The CV approach is summarized in Algorithm 3.1.

The key problem in our CV approach is step 3 in Algorithm 3.1 – treating the hold-out set as missing entries and imputing them with partially observed X. This requires us to solve a matrix completion problem. If $f(\cdot)$ can only work with complete matrices, then we need to propose a matrix completion algorithm $\tilde{f}(\cdot)$ that approximates $f(\cdot)$ well. There is a rich literature on matrix completion [46, 47, 48, 49, 50, 51]. Using which matrix completion algorithm depends on the unsupervised learning problem we are solving. In general, we can

Algorithm 3.1 Cross-Validation Approach

Randomly partition the observed elements Ω into K folds $\{\Omega_1, \dots, \Omega_K\}$
for $k = 1$ to K **do**
    Leave out Ω_k as the hold-out set, fit to the other $K-1$ folds, and get the estimate

$$\hat{X}^{(i)} = \tilde{f}\left(P_{\Omega \setminus \Omega_k}(X)\right)$$

    Compute the error in estimating the kth fold

$$\mathrm{err}_i = \|P_{\Omega_i}(X - \hat{X}^{(k)})\|_F^2$$

end for
Compute the cross-validation error

$$\mathrm{err}_{\mathrm{CV}} = \sum_{i=1}^{K} \mathrm{err}_k$$

Return: $\mathrm{err}_{\mathrm{CV}}$

use an expectation-maximization (EM) procedure with $f(\cdot)$. This procedure first initializes the missing entries using column means, and then iteratively estimates the missing entries by applying $f(\cdot)$ to the imputed matrix. This algorithm is summarized in Algorithm 3.2.

Algorithm 3.2 Matrix Completion Algorithm $\tilde{f}(\cdot)$

Input: $P_\Omega(X)$
Initialize the missing entries $P_{\Omega^\perp}(X)$ with column means and denote the imputed matrix as $\tilde{X}$
while not converged **do**
    Compute the estimate $\hat{X} \leftarrow f(\tilde{X})$
    Update the missing entries with the current estimate

$$\tilde{X} \leftarrow P_\Omega(X) + P_{\Omega^\perp}(\hat{X}) \tag{3.16}$$

end while
Return: $\hat{X}$

In summary, we propose a general and data-driven CV approach. This CV approach can give us an estimate of the prediction error, and guide us to perform model selection

3.4 Cross-Validation for Matrix Denoising

When the observed data matrix X is complete, i.e., $\Omega = \{1, \ldots, n\} \times \{1, \ldots, p\}$, the problem of estimating the low-rank signal X^0 from noisy data X is often refered as *matrix denoising*. In this section we discuss how to apply our CV approach for matrix denoising and show through extensive experiments that the CV approach can estimate both the prediction error and the optimal rank well.

The most popular estimation technique for matrix denoising is *Principal Component Analysis* (PCA), or *Truncated Singular Value Decomposition* (TSVD) [52, 53, 54]. Write

$$X = \sum_{i=1}^{n} \hat{d}_i \hat{u}_i \hat{v}_i^T \tag{3.17}$$

for the SVD of the data matrix X, where $\hat{u}_i \in \mathbb{R}^n$ and $\hat{v}_i \in \mathbb{R}^p$, $i = 1, \ldots, n$ are the left and right singular vectors of X corresponding to the singular value $\hat{d}_i$. The PCA estimator with rank k is

$$\hat{X}(k) = \sum_{i=1}^{k} \hat{d}_i \hat{u}_i \hat{v}_i^T. \tag{3.18}$$

This is the best approximation of rank k to the data in the least square sense, and therefore maximum likelihood estimator when E has independent Gaussian entries.

In this section, we focus on estimating the prediction error and the optimal rank k of the PCA estimator with our CV approach. Note that the PCA estimator doesn't apply to incomplete matrices, thus we need a matrix completion algorithm for our CV approach. The matrix completion algorithm should approximate PCA well. In particular, when there are no missing entries, it should give the same estimate as PCA does.

Here we propose to use HARD-IMPUTE and SOFT-IMPUTE+. The latter is a post-processing on SOFT-IMPUTE [46, 48].

- HARD-IMPUTE is an EM-algorithm that iteratively estimates the missing values of $P_\Omega(X)$ by the values from the first k terms of the SVD of the completed matrix. See

Algorithm 3.3 for more details.

Algorithm 3.3 HARD-IMPUTE [46, 48]

1: Set the rank k and initialize $\hat{X}_k = 0$
2: **while** not converged **do**
3: Fill in the missing entries $\hat{X}^{\text{full}} \leftarrow P_\Omega(X) + P_{\Omega^\perp}(\hat{X})$
4: Compute SVD $\hat{X}^{\text{full}} = \sum_{i=1}^{n} \hat{d}_i \hat{u}_i \hat{v}_i^T$
5: Update the estimate $\hat{X}_k \leftarrow \sum_{i=1}^{k} \hat{d}_i \hat{u}_i \hat{v}_i^T$
6: **end while**
7: **Return:** $\hat{X}_k$

- SOFT-IMPUTE computes the nuclear-norm regularization solution

$$\hat{X}_\lambda = \arg\min_Z \frac{1}{2} ||P_\Omega(X - Z)||_{F,}^2 + \lambda ||Z||_*. \tag{3.19}$$

This algorithm iteratively replaces the missing elements with those obtained from a soft-thresholded SVD. See Algorithm 3.4 for more details.

Algorithm 3.4 SOFT-IMPUTE [46, 48]

1: Set the regularization parameter λ and initialize $\hat{X}_\lambda = 0$
2: **while** not converged **do**
3: Fill in the missing entries $\hat{X}^{\text{full}} \leftarrow P_\Omega(X) + P_{\Omega^\perp}(\hat{X})$
4: Update the estimate $\hat{X}_\lambda \leftarrow S_\lambda(\hat{X}^{\text{full}})$
5: **end while**
6: **Return:** $\hat{X}_\lambda$

- SOFT-IMPUTE+ is a post-pocessed version $\hat{X}_\lambda^u$, which "unshrinks" the singular values of the matrix $\hat{X}_\lambda$. Let $\hat{X}_\lambda = \sum_{i=1}^{k} \hat{d}_i \hat{u}_i \hat{v}_i^T$, where k is the rank of $\hat{X}_\lambda$. Then $\hat{X}_\lambda^u$ is obtained by

$$\begin{aligned} \hat{\alpha} &= \text{argmin}_{\alpha_i \geq 0, i=1,\ldots,k} ||P_\Omega(X - \sum_{i=1}^{k} \alpha_i \hat{u}_i \hat{v}_i^T)||_F^2 \\ \hat{X}_\lambda^u &= \sum_{i=1}^{k} \hat{\alpha}_i \hat{u}_i \hat{v}_i^T. \end{aligned}$$

The estimation can be done via ordinary least squares. If the least squares solutions $\hat{\alpha}$ do not meet the positivity constraints, then the negative sign can be absorbed into the corresponding singular vector.

3.4.1 Prediction error estimation

First we discuss the behavior of CV with HARD-IMPUTE and SOFT-IMPUTE+ as prediction error estimators. Experiment results show that both HARD-IMPUTE and SOFT-IMPUTE+ cross-validations estimate PE(k) well, while SOFT-IMPUTE+ is more robust in scenarios where the matrix denoising problem is challenging. Both methods estimate the predictor error better when the matrix dimension p is larger and when the number of folds K is larger. But the marginal gain decreases as K increases. In practice we find that 10-fold cross-validation is sufficent to approximate PE(k) well.

Effect of matrix dimension and the number of CV folds

To examine the effect of matrix dimension p and the number of CV folds, we run a set of simulation examples with $\beta = 1$, $k_0 = 6$ and $D = \text{diag}(10, 9, 8, 7, 6, 5)$. The dimension p for each example is 20, 50, 100, and 200 respectively. E is a matrix of i.i.d. Gaussian noise with $E_{ij} \sim N(0, \sigma^2)$. Here we set $\sigma = 1$. U and V are random orthonormal matrices independent of E. We implement both algorithms with number of folds $K = 5, 10, 40, 100$. For better illustration, we plot the estimated root mean prediction error, which is $\sqrt{\text{PE}(k)/np}$. Figure 3.1 shows the results.

Empirical results show that both HARD-IMPUTE and SOFT-IMPUTE+ cross-validations estimate PE(k) well when $k \leq k^*_{\text{PE}}$. When $k > k^*_{\text{PE}}$, SOFT-IMPUTE+ does a better job at estimating PE(k), but tends to underestimate. HARD-IMPUTE, on the other hand, increases steeply for k past the minimizer.

For a fixed true rank k_0 and a fixed aspect ration $\beta = n/p$, both HARD-IMPUTE and SOFT-IMPUTE+ estimate the predictor error better when the matrix dimension p is larger.

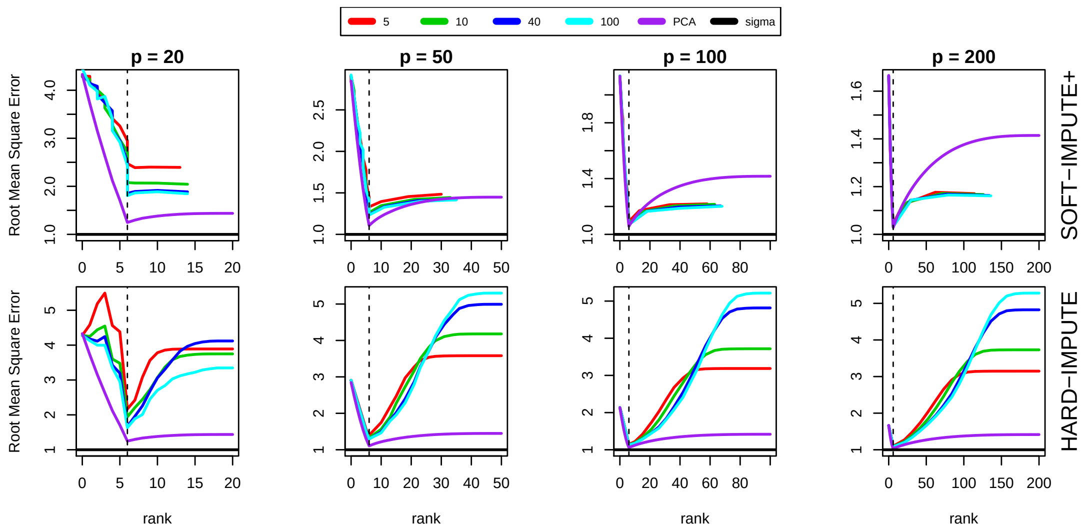


Figure 3.1: Estimated root mean prediction errors with SOFT-IMPUTE+ and HARD-IMPUTE cross-validations. The true prediction error is in purple, the noise level is in black, and the CV curves with fold $K = 5, 10, 40$ and 100 are in red, green, blue and light blue respectively. The dotted vertical line indicates the true rank k_0. Both HARD-IMPUTE and SOFT-IMPUTE+ cross-validations estimate PE(k) well when $k \leq k^*_{\text{PE}}$. When $k > k^*_{\text{PE}}$, SOFT-IMPUTE+ does a better job at estimating PE(k), but tends to underestimate. Both HARD-IMPUTE and SOFT-IMPUTE+ estimate the predictor error better when the matrix dimension p is larger and when the number of folds K is larger. But the marginal gain decreases as K increases.

For a given X, even though the performances of both algorithms are better when the number of folds K is larger, the marginal gain decreases as K increases. In practice we find that 10-fold cross-validation is sufficent to approximate PE(k) well.

Robustness in challenging matrix denoising settings

Compared to SOFT-IMPUTE+ cross-validation, HARD-IMPUTE cross-validation is more unstable when the underlying matrix completion problem is hard.

We examine the following four settings.

(a) Same non-zero singular values: rank k_0 is 6, n is 20, and the singular values are all 5.

(b) Large rank k_0: rank k_0 is 9, n is 20, and the singular values are $(10, 9, 8, 7, 6, 5, 4, 3, 2)$.

(c) Low signal-to-noise ratio: rank k_0 is 6 and n is 20. We scale the singular values such that $||\sqrt{p}UDV^T||_F^2 = \mathbb{E}||E||_F^2$. The singular values are $0.237 * (10, 9, 8, 7, 6, 5)$.

(d) Small aspect ratio β: rank k_0 is 1, n is 4, and the singular values are $(10, 9)$.

For all settings, E is a matrix of i.i.d. Gaussian noise with $E_{ij} \sim N(0, \sigma^2)$, and $\sigma = 1$. U and V are random orthonormal matrices independent of E. The dimension p is 20 and the number of folds K is 10.

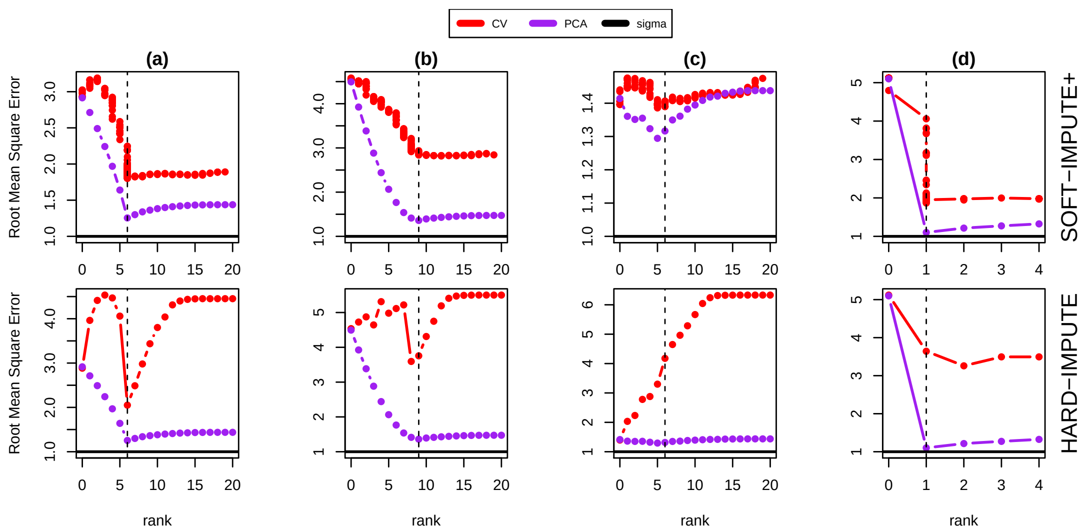


Figure 3.2: Estimated root mean prediction errors with SOFT-IMPUTE+ and HARD-IMPUTE cross-validations. The true prediction error is in purple, the noise level is in black, and the CV curves are in red. The dotted vertical line indicates the true rank k_0. The four settings are (a) same non-zero singular values, (b) large rank k_0, (c) low signal-to-noise ratio, and (d) small the aspect ratio β. In all scenarios SOFT-IMPUTE+ cross-validation is more robust than HARD-IMPUTE cross-validation.

The simulation results are shown in Figure 3.2. The results suggest that SOFT-IMPUTE+ cross-validation is more robust than HARD-IMPUTE cross-validation. This may be due to the fact that HARD-IMPUTE can only converge to a local minimum.

3.4.2 Rank Estimation

Second we see how well our CV approach works at estimating the optimal rank, especially as compared to other rank-selection methods. We generate data in the same manner as [55], and record how far off the minimizer of $\hat{\mathrm{PE}}(k)$ is from the minimizer of $\mathrm{PE}(k)$.

Each simulation generates a data matrix $X = \sqrt{p}UDV^T + E$. We fix the dimensions and number of generating signals as $n = 50$, $p = 100$ and $k^0 = 6$. We set $D = \mathrm{diag}(10, 9, 8, 7, 6, 5)$. We generate U, V and E independently of each other, and the elements of E are uncorrelated with each other.

We consider two types of factors. For "Gaussian" factors, we put the elements of U distributed i.i.d. with $U_{11} \sim N(0, \frac{1}{n})$ and the elements of V i.i.d. with $V_{11} \sim N(0, \frac{1}{p})$. For "Sparse" factors we use sparsity parameter $s = 10\%$ and set

$$\begin{aligned}
\mathbb{P}(U_{11} &= 0) = 1 - s, \\
\mathbb{P}(U_{11} = -\frac{1}{\sqrt{sn}}) &= \mathbb{P}(U_{11} = \frac{1}{\sqrt{sn}}) = \frac{s}{2}.
\end{aligned}$$

Similarly, we put

$$\begin{aligned}
\mathbb{P}(V_{11} &= 0) = 1 - s, \\
\mathbb{P}(V_{11} = -\frac{1}{\sqrt{sp}}) &= \mathbb{P}(V_{11} = \frac{1}{\sqrt{sp}}) = \frac{s}{2}.
\end{aligned}$$

The scalings in both cases are chosen so that $\mathbb{E}[U^T U] = \mathbb{E}[V^T V] = I_{k_0}$. Gaussian factors are uniformly spread out in the observations and variables, while sparse factors are only observable in a small percentage of the matrix entries (about 1%).

We use three types of noise. For "white" noise, we generate the elements of E i.i.d. with $E_{11} \sim N(0, 1)$. For "heavy" noise, we use i.i.d. elements with $E_{11} \sim \sigma_\nu^{-1} t_\nu$, and $\nu = 3$. Here t_ν is a t random variable with ν degrees of freedom, and $\sigma_\nu = \sqrt{v/(v-2)}$ is chosen so that $\mathbb{E}[E_{11}^2] = 1$. Heavy noise is so-called because it has a heavy tail. Lastly, for "colored" noise, we first generate $\sigma_1^2, \ldots, \sigma_n^2 \sim \text{Inverse} - \chi^2(\nu_1)$ and $\tau_1^2, \ldots, \tau_p^2 \sim \text{Inverse} - \chi^2(\nu_2)$ independently, with $\nu_1 = \nu_2 = 3$. Then we generate the elements of E independently as $E_{ij} \sim c_{\nu_1,\nu_2}^{-1} \cdot N(0, \sigma_i^2 + \tau_j^2)$, where $c_{\nu_1,\nu_2} = \sqrt{1/(v_1 - 2) + 1/(\nu_2 - 2)}$. Again, c_{ν_1,ν_2} is chosen so that $\mathbb{E}[E_{ij}^2] = 1$. Colored noise simulates heteroscedasticity.

Both HARD-IMPUTE and SOFT-IMPUTE+ cross-validations are implemented with 5 folds. We compare their performance with those of eight other methods. They are as follows:

1. BCV: Owen and Perry's (2,2)-fold Bi-Cross-Validation estimator [45].
2. AIC: Rao & Edelman's AIC-based estimator [56].
3. BIC_1, BIC_2 and BIC_3: Bai & Ng's BIC-based estimators [57].
4. F: Faber & Kowalski's modification of Malinowski's F-test, with a significance level of 0.05 [58, 59].
5. MDL: Wax & Kailaith's estimator based on the minimum description length principle [60].
6. UIP: Kritchman & Nadler's estimator based on Roy's union-intersection principle and their background noise estimator, with a significance level of 0.001 [61, 62].

In addition, an approximation to HARD-IMPUTE, called "Approx" in Table 3.1, is implemented and compared with HARD-IMPUTE and SOFT-IMUTE+. We refer readers to Section 3.4.3 for details on this approximation approach.

Table 3.1 summarize the results of 100 replicates. The results of comparison methods are obtained from [55]. For the Gaussian factors, HARD-IMPUTE cross-validation seems to be the clear winner. For the Sparse factors, SOFT-IMPUTE+ cross-validation performs very well with white noise and heavy noise, and HARD-IMPUTE performs much better than others with colored noise. HARD-IMPUTE cross-validation tends to underestimate ranks, while SOFT-IMPUTE+ tends to overestimate. This is because SOFT-IMPUTE+

only unshrinks singular values, and its singular vector estimations are the same as those of SOFT-IMPUTE.

Table 3.1: Rank Estimation with Gaussian factors and Sparse factors. The values are the average difference between the estimated rank and the true minimizer of PE(k) for 100 replicates of Gaussian or Sparse factors with various types of noise. The values in parentheses are the corresponding standard deviations. For the Gaussian factors, HARD-IMPUTE cross-validation and its approximation are the clear winner. For the Sparse factors, SOFT-IMPUTE+ cross-validation performs very well with white noise and heavy noise, and HARD-IMPUTE performs much better than others with colored noise. HARD-IMPUTE cross-validation tends to underestimate ranks, while SOFT-IMPUTE+ tends to overestimate. HARD-IMPUTE approximation beats HARD-IMPUTE in that it estimates the rank more accurately and more robustly. It is also beats SOFT-IMPUTE+ in most cases except the setting of sparse factors and white noise.

Method	**HardImpute**	**SoftImpute+**	**Approx**	**BCV**	**AIC**	**BIC$_1$**	**BIC$_2$**	**BIC$_3$**	F	**MDL**	**UIP**
	Gaussian Factors										
White Noise	.00	.00	.00	.47	-.01	-.75	-1.33	.01	.01	-.56	.01
	(.00)	(.00)	(.00)	(.70)	(.22)	(.72)	(.87)	(.10)	(.10)	(.61)	(.10)
Colored Noise	-.24	1.26	.23	2..44	7.46	2.06	1.31	7.05	3.99	2.10	4.81
	(.81)	(1.32)	(.63)	(1.78)	(.88)	(2.22)	(1.91)	(1.80)	(2.02)	(2.18)	(1.92)
Heavy Noise	-.03	.53	.00	1.26	2.65	.18	-.42	1.41	1.10	.25	1.49
	(.17)	(.67)	(.00)	(1.13)	(1.58)	(1.29)	(1.33)	(1.59)	(1.35)	(1.27)	(1.42)
	Sparse Factors										
White Noise	-1.16	.02	-.13	.17	-.12	-1.03	-1.54	.07	.15	-.86	.28
	(1.74)	(.14)	(.35)	(1.11)	(.50)	(.87)	(1.11)	(.41)	(.36)	(.79)	(.47)
Colored Noise	-1.66	2.02	.20	2.32	7.69	2.29	1.52	7.35	4.69	2.37	5.36
	(1.99)	(1.83)	(.85)	(2.32)	(.71)	(2.28)	(2.11)	(1.37)	(1.95)	(2.20)	(1.77)
Heavy Noise	-1.32	.71	.10	1.37	3.44	.20	-.59	1.89	1.68	.32	2.14
	(1.48)	(.78)	(.40)	(1.32)	(1.97)	(1.44)	(1.38)	(1.91)	(1.73)	(1.40)	(1.83)

3.4.3 Cross-Validation Approximation

The matrix completion algorithms HARD-IMPUTE and SOFT-IMPUTE+ are computationally expensive. This makes the cross-validation strategy even more expensive in practice, as it needs to run the matrix completion algorithm multiple times. In this section, we propose an approximation to these matrix completion algorithms, which dramatically speeds up the process. Given the SVD of X, this method only takes $O(n)$ to impute $\hat{X}_{ij}$. Experiments on matrix denoising show that although this approximation overestimates the prediction error, it performs very well in finding the true rank.

Approximation to HARD-IMPUTE and SOFT-IMPUTE

HARD-IMPUTE iteratively updates the truncated SVD with the current estimate of the missing values, while our approximation only does one-step update of the truncated SVD. Specifically, let the SVD of X be $X = UDV^T$, and denote $A^{(k)} = U^{(k)}(D^{(k)})^{1/2}$ and $B^{(k)} = V^{(k)}(D^{(k)})^{1/2}$, where $U^{(k)}, D^{(k)}, V^{(k)}$ are the best rank-k approximations to U, D, V. Then $(A^{(k)}, B^{(k)})$ is the solution to

$$\min_{\text{rank}(A)=\text{rank}(B)=k} \|X - AB^T\|_F^2. \tag{3.20}$$

Suppose X_{ij} is left-out, then we update the ith row of $A^{(k)}$ and the jth row of $B^{(k)}$ with X_{ij} missing, i.e.,

$$\hat{\alpha} = \arg\min_{\alpha} \|X[i, -j] - B^{(k)}_{-j}\alpha\|_2^2, \tag{3.21}$$

$$\hat{\beta} = \arg\min_{\beta} \|X[-i, j] - A^{(k)}_{-i}\beta\|_2^2, \tag{3.22}$$

where $X[i, -j]$ denotes the ith row of X with entry X_{ij} removed, $B^{(k)}_{-j}$ is $B^{(k)}$ with the jth row removed, and similarly for $X[-i, j]$ and $A^{(k)}_{-i}$. Then we estimate X_{ij} by multiplying the updated ith row of $A^{(k)}$ and the jth row of $B^{(k)}$,

$$\hat{X}_{ij} = \hat{\alpha}^T \hat{\beta}. \tag{3.23}$$

We can generalize this approach and get an approximation to SOFT-IMPUTE as well. SOFT-IMPUTE iteratively solves the nuclear norm regularization problem with the current estimate of the missing values,

$$\hat{X}^{\text{new}} = \arg\min_{Z} \frac{1}{2} ||P_\Omega(X) + P_{\Omega^\perp}(\hat{X}) - Z||_{F,}^2 + \lambda ||Z||_*. \tag{3.24}$$

Hastie et al. [48] show that $\hat{X}^{\text{new}} = \hat{A}\hat{B}^T$, where $\hat{A}, \hat{B}$ are the solution to

$$\min_{A,B} \frac{1}{2} \|P_\Omega(X) + P_{\Omega^\perp}(\hat{X}) - AB^T\|_F^2 + \frac{\lambda}{2}(\|A\|_F^2 + \|B\|_F^2), \tag{3.25}$$

as long as we use enough columns on A and B. This motives us to approximate SOFT-IMPUTE by solving (3.25) and updating A, B once.

Specifically, let $A^{(\lambda)}, B^{(\lambda)}$ minimizes $\frac{1}{2}||X - AB^T||_F^2 + \frac{\lambda}{2}(||A||_F^2 + ||B||_F^2)$. Then $A^{(\lambda)} = US_\lambda(D)^{1/2}$ and $B^{(\lambda)} = VS_\lambda(D)^{1/2}$. Suppose X_{ij} is left-out, then we update the ith row of $A^{(\lambda)}$ and the jth row of $B^{(\lambda)}$ with X_{ij} missing, i.e.,

$$\hat{\alpha} = \arg\min_{\alpha} \frac{1}{2} \|X[i, -j] - B_{-j}^{(\lambda)} \alpha\|_2^2 + \frac{1}{2} \|\alpha\|_2^2, \tag{3.26}$$

$$\hat{\beta} = \arg\min_{\beta} \|X[-i, j] - A_{-i}^{(\lambda)} \beta\|_2^2 + \frac{1}{2} \|\beta\|_2^2. \tag{3.27}$$

Then the estimate of the missing value is $\hat{X}_{ij} = \hat{\alpha}\hat{\beta}^T$.

Due to the special structure of $A^{(k)}$ and $B^{(k)}$, we can simplify the expression of $\hat{X}_{ij}$ such that the computational cost is $O(n)$.

Proposition 3.4.1. *For approximation to HARD-IMPUTE,*

$$\hat{X}_{ij} = \left(U_i^{(k)T} D^{(k)} + \left(X_{ij}^{(k)} - X_{ij}(1+2c) \right) \frac{V_j^{(k)T}}{1+c} \right) \left(D^{(k)} \right)^{-1} \left(D^{(k)} V_j^{(k)} + \left(X_{ij}^{(k)} - X_{ij}(1+2d) \right) \frac{U_i^{(k)}}{1+d} \right), \tag{3.28}$$

where $c = V_j^{(k)T} V_j^{(k)}$, $d = U_i^{(k)T} U_i^{(k)}$, $X_{ij}^{(k)}$ is the (i,j) element of the best rank-k approximation $X^{(k)} = U^{(k)} D^{(k)} V^{(k)T}$, and $U_i^{(k)}$ is the ith row of $U^{(k)}$ (as a column vector) and similarly for $V_j^{(k)}$.

For approximation to SOFT-IMPUTE,

$$\hat{X}_{ij} = \left(U_i^T D S_\lambda(D)^{1/2} + \left(X_{ij}^{(\lambda)} - X_{ij}(1+2c) \right) \frac{B_j^{(\lambda)T}}{1+c} \right) \tilde{D}^{-2} \left(S_\lambda(D)^{1/2} D V_j + \left(X_{ij}^{(\lambda)} - X_{ij}(1+2d) \right) \frac{A_i^{(\lambda)}}{1+d} \right), \tag{3.29}$$

where $\tilde{D} = \lambda + S_\lambda(D)$, $c = B_j^{(\lambda)T} \tilde{D}^{-1} B_j^{(\lambda)}$, $d = A_i^{(\lambda)T} \tilde{D}^{-1} A_i^{(\lambda)}$, $X_{ij}^{(\lambda)}$ is the (i,j) element of $X^{(\lambda)} = U^{(\lambda)} D^{(\lambda)} V^{(\lambda)T}$, and $A_i^{(\lambda)}$ is the ith row of $A^{(\lambda)}$ (as a column vector) and similarly for $B_j^{(\lambda)}$.

Proof. For approximation to HARD-IMPUTE,

$$\hat{X}_{ij} = X[i,-j]^T V_{-j}^{(k)} \left((V_{-j}^{(k)})^T V_{-j}^{(k)} \right)^{-1} (D^{(k)})^{-1} \left((U_{-i}^{(k)})^T U_{-i}^{(k)} \right)^{-1} (U_{-i}^{(k)})^T X[-i,j] \tag{3.30}$$

$$\begin{aligned}
(V_{-j}^{(k)T} V_{-j}^{(k)})^{-1} &= (V^{(k)T} V^{(k)} - V_j^{(k)} V_j^{(k)T})^{-1} \\
&= (I - V_j^{(k)} V_j^{(k)T})^{-1} \\
&= I + \frac{V_j^{(k)} V_j^{(k)T}}{1 + V_j^{(k)T} V_j^{(k)}}
\end{aligned}$$

Therefore

$$\begin{aligned}
\hat{X}_{ij} &= U_i^T D V_{-j}^T V_{-j}^{(k)} (I + \frac{V_j^{(k)} V_j^{(k)T}}{1 + V_j^{(k)T} V_j^{(k)}})(D^{(k)})^{-1}(I + \frac{U_i^{(k)} U_i^{(k)T}}{1 + U_i^{(k)T} U_i^{(k)}}) U_{-i}^{(k)T} U_{-i} D V_j \\
&= (U_i^{(k)T} D^{(k)} - X_{ij} V_j^{(k)T})(I + \frac{V_j^{(k)} V_j^{(k)T}}{1 + V_j^{(k)T} V_j^{(k)}})(D^{(k)})^{-1}(I + \frac{U_i^{(k)} U_i^{(k)T}}{1 + U_i^{(k)T} U_i^{(k)}})(D^{(k)} V_j^{(k)} - X_{ij} U_i^{(k)}) \\
&= \left(U_i^{(k)T} D^{(k)} + \frac{1}{1 + V_j^{(k)T} V_j^{(k)}} \left(X_{ij}^{(k)} V_j^{(k)T} - X_{ij} V_j^{(k)T} (1 + 2 V_j^{(k)T} V_j^{(k)}) \right) \right) (D^{(k)})^{-1} \\
&\quad \left(D^{(k)} V_j^{(k)} + \frac{1}{1 + U_i^{(k)T} U_i^{(k)}} \left(X_{ij}^{(k)} U_i^{(k)} - X_{ij} U_i^{(k)} (1 + 2 U_i^{(k)T} U_i^{(k)}) \right) \right)
\end{aligned}$$

For approximation to SOFT-IMPUTE, we will use $A = A^{(\lambda)}$ and $B = B^{(\lambda)}$ for ease of notation. Then

$$\hat{\beta} = (A_{-i}^T A_{-i} + \lambda I)^{-1} A_{-i}^T X[-i, j]. \tag{3.31}$$

Thus

$$\hat{X}_{ij} = X[i, -j]^T B_{-j} (B_{-j}^T B_{-j} + \lambda I)^{-1} (A_{-i}^T A_{-i} + \lambda I)^{-1} A_{-i}^T X[-i, j] \tag{3.32}$$

According to the Sherman-Morrison-Woodbury formula,

$$(A - uv^T)^{-1} = A^{-1} + \frac{A^{-1} u v^T A^{-1}}{1 + v^T A^{-1} u} \tag{3.33}$$

So

$$\begin{aligned}
(A_{-i}^T A_{-i} + \lambda I)^{-1} &= (S_\lambda(D) + \lambda - A_i A_i^T)^{-1} \\
&= (\lambda + S_\lambda(D))^{-1} [\lambda + S_\lambda(D) + \frac{A_i A_i^T}{1 + A_i^T (\lambda + S_\lambda(D))^{-1} A_i}] (\lambda + S_\lambda(D))^{-1}.
\end{aligned}$$

Let $\tilde{D} = \lambda + S_\lambda(D)$. Then

$$\hat{\beta} = \tilde{D}^{-1} [\tilde{D} + \frac{A_i A_i^T}{1 + A_i^T \tilde{D}^{-1} A_i}] \tilde{D}^{-1} S_\lambda(D)^{1/2} (I - U_i U_i^T) D V_j \tag{3.34}$$

Therefore

$$
\begin{aligned}
\hat{X}_{ij} =&(U_i^T D - X_{ij} V_j^T) S_\lambda(D)^{1/2} \tilde{D}^{-1} [\tilde{D} + \frac{B_j B_j^T}{1 + B_j^T \tilde{D}^{-1} B_j}] \cdot \tilde{D}^{-2} \\
&\cdot [\tilde{D} + \frac{A_i A_i^T}{1 + A_i^T \tilde{D}^{-1} A_i}] \tilde{D}^{-1} S_\lambda(D)^{1/2} (DV_j - U_i X_{ij}) \\
=& \left(U_i^T D S_\lambda(D)^{1/2} + \left(X_{ij}^{(\lambda)} - X_{ij}(1+2c) \right) \frac{B_j^T}{1+c} \right) \cdot \tilde{D}^{-2} \\
&\cdot \left(S_\lambda(D)^{1/2} D V_j + \left(X_{ij}^{(\lambda)} - X_{ij}(1+2d) \right) \frac{A_i}{1+d} \right)
\end{aligned}
$$

□

Given the SVD of X, computing $\hat{X}_{ij}$ takes only $O(k)$ for HARD-IMPUTE approximation and $O(n)$ for SOFT-IMPUTE approximation. Therefore, the approximations dramatically lowers the computational complexity.

Experiments

To illustrate how the approximation works in practice, we implement it in the same simulation setting as in Section 3.4.2 and compare its performance with HARD-IMPUTE and SOFT-IMPUTE+. Since SOFT-IMPUTE tends to overestimate the true rank, the approximation to SOFT-IMPUTE is not considered.

Figure 3.3 shows the true PCA prediction error and the estimated error via cross-validation with HARD-IMPUTE, SOFT-IMPUTE+ and HARD-IMPUTE approximation. The first row is the setting of Gaussian factors, while the second row is sparse factors. The three columns correspond to the setting of white, colored and heavy noise. The figure shows that SOFT-IMPUTE+ best estimates the prediction error. HARD-IMUTE overestimates the prediction error by a large margin and also has very large variance. HARD-IMPUTE approximation partially solves the overestimation issue of HARD-IMPUTE, and has much smaller variance than HARD-IMPUTE. Therefore in practice HARD-IMPUTE approximation is favorable to

HARD-IMPUTE.

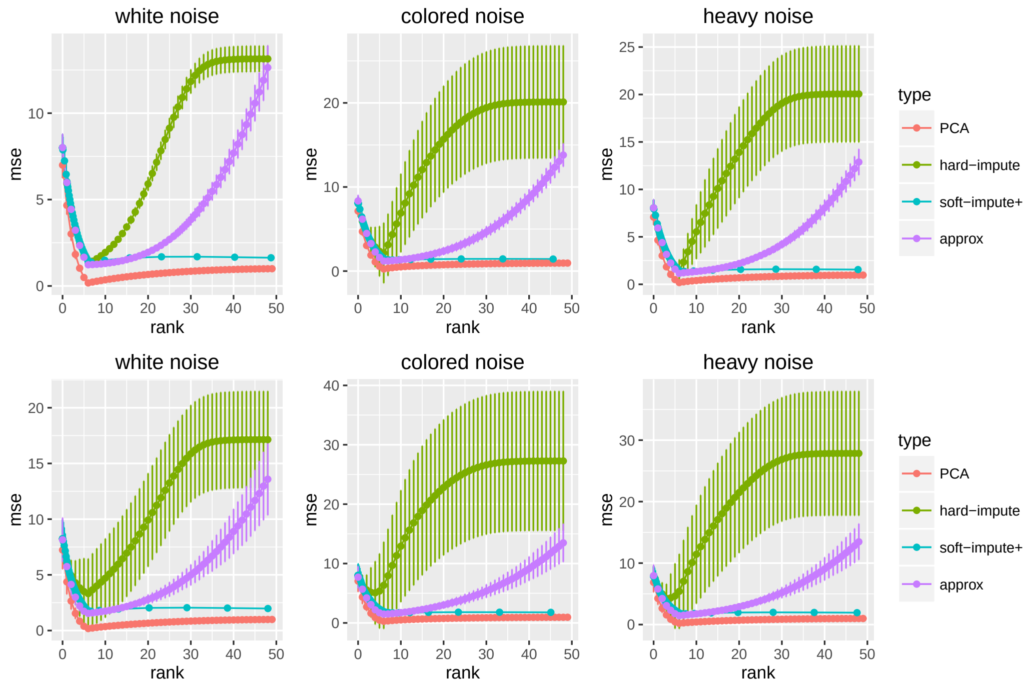


Figure 3.3: Estimated mean prediction errors by cross-validation with HARD-IMPUTE, SOFT-IMPUTE+ and HARD-IMPUTE approximation. The true prediction error, i.e. PCA prediction error, is in red. The first row is Gaussian factors, while the second row is sparse factors. SOFT-IMPUTE+ best estimates the prediction error. HARD-IMUTE overestimates the prediction error by a large margin and also has very large variance. HARD-IMPUTE approximation partially solves the overestimation issue of HARD-IMPUTE, and has much smaller variance than HARD-IMPUTE.

As for estimating the rank of the signal matrix, Table 3.1 demonstrates that HARD-IMPUTE approximation beats HARD-IMPUTE in that it estimates the rank more accurately and more robustly. It is also beats SOFT-IMPUTE+ in most cases except the setting of sparse factors and white noise.

3.5 Cross-Validation for Matrix Completion

When the observed data matrix X is incomplete, i.e., $\Omega \neq \{1, \ldots, n\} \times \{1, \ldots, p\}$, the problem of estimating the low-rank signal X^0 from the incomplete data X (with or without noise) is often refered as *matrix completion*.

Besides the matrix completion algorithms SOFT-IMPUTE, SOFT-IMPUTE+ and HARD-IMPUTE mentioned in the last section, there is another algorithm called OptSpace [47], which minimizes the cost function $F(U, V)$, defined by

$$\begin{aligned} F(U,V) &= \min_{S \in \mathbb{R}^{k\times k}} \mathcal{F}(U,V,S), \\ \mathcal{F}(U,V,S) &= \frac{1}{2} \sum_{(i,j)\in\Omega} (X_{ij} - (USV^T)_{ij})^2. \end{aligned}$$

Here $U \in \mathbb{R}^{n\times k}, V \in \mathbb{R}^{p\times k}$ are orthogonal matrices, and rank k is the tuning parameter.

In this section we examine the behavior of cross-validations with SOFT-IMPUTE, SOFT-IMPUTE+, HARD-IMPUTE and OptSpace as prediction error estimators. Experiment results show that cross-validations with all four matrix completion algorithms estimate their corresponding PE(λ) well, except that CV with HARD-IMPUTE is not stable. All four matrix completion algorithms estimate the predictor error better when the matrix dimension p is larger and when the number of folds K is larger, although the marginal gain decreases as K increases. Moreover, SOFT-IMPUTE and SOFT-IMPUTE+ cross-validations are more robust than HARD-IMPUTE and OptSpace cross-validations in challenging matrix completion settings.

3.5.1 Effect of matrix dimension and the number of CV folds

To examine the effect of matrix dimension p and the number of CV folds, we run a set of simulation examples with $\beta = 1$, $k_0 = 6$, $|\Omega|/np = 0.9$, and $D = \text{diag}(10, 9, 8, 7, 6, 5)$. The dimension p for each example is 20, 50 and 100 respectively. E is a matrix of i.i.d. Gaussian noise with $E_{ij} \sim N(0, \sigma^2)$. Here we set $\sigma = 1$. U and V are random orthonormal matrices independent of E. We implement both algorithms with number of folds $K = 5, 10, 40, 100$. For better illustration, we plot the estimated root mean prediction error, which is $\sqrt{\text{PE}(k)/|\Omega|}$. Figure 3.4 shows the results.

Empirical results show that cross-validations with all four matrix completion algorithms estimate their corresponding PE(λ) well, except that CV with HARD-IMPUTE is not stable. As the results of $p = 20$ shows, HARD-IMPUTE CV estimates the prediction well for folds $K = 5, 40, 100$ but the CV error curve goes wildly for $K = 10$.

For a fixed true rank k_0 and a fixed aspect ration $\beta = n/p$, cross-validations with all four matrix completion algorithms estimate the predictor error better when the matrix dimension p is larger. For a given X, even though the performances of both algorithms are better when the number of folds K is larger, the marginal gain decreases as K increases. In practice we find that 10-fold cross-validation is sufficent to approximate PE(λ) well.

3.5.2 Robustness in challenging matrix completion settings

Compared to SOFT-IMPUTE and SOFT-IMPUTE+ cross-validation, HARD-IMPUTE and OptSpace cross-validations are more unstable when the underlying matrix completion problem is hard.

We examine the following five settings.

(a) Same non-zero singular values: rank k_0 is 6, n is 20, and the singular values are all 5.

(b) Large rank k_0: rank k_0 is 9, n is 20, and the singular values are $(10, 9, 8, 7, 6, 5, 4, 3, 2)$.

(c) Low signal-to-noise ratio: rank k_0 is 6 and n is 20. We scale the singular values such that $||\sqrt{n}UDV^T||_F^2 = \mathbb{E}||E||_F^2$. The singular values are $0.237 * (10, 9, 8, 7, 6, 5)$.

(d) Small aspect ratio β: rank k_0 is 1, n is 4, and the singular values are $(10, 9)$.

(e) Large fraction of missing entries: rank k_0 is 6, n is 20, and the singular values are $(10, 9, 8, 7, 6, 5)$. The fraction of observed entries is 0.6.

For all settings, E is a matrix of i.i.d. Gaussian noise with $E_{ij} \sim N(0, \sigma^2)$, and $\sigma = 1$. U and V are random orthonormal matrices independent of E. The dimension p is 20 and the number of folds K is 10.

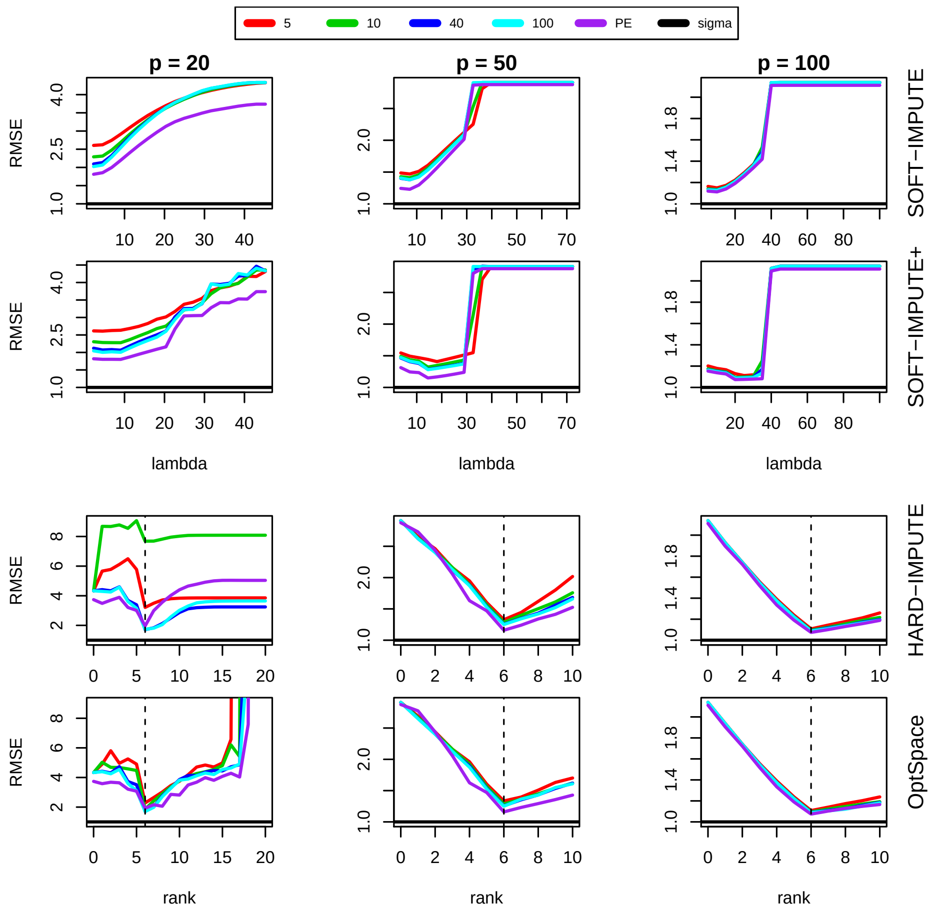


Figure 3.4: Estimated root mean prediction errors with SOFT-IMPUTE, SOFT-IMPUTE+, HARD-IMPUTE and OptSpace cross-validations. The true prediction error is in purple, the noise level is in black, and the CV curves with fold $K = 5, 10, 40$ and 100 are in red, green, blue and light blue respectively. The dotted vertical line indicates the true rank k_0. Cross-validations with all four matrix completion algorithms estimate their corresponding PE(λ) well, except that CV with HARD-IMPUTE is not stable. All four matrix completion algorithms estimate the predictor error better when the matrix dimension p is larger and when the number of folds K is larger, although the marginal gain decreases as K increases.

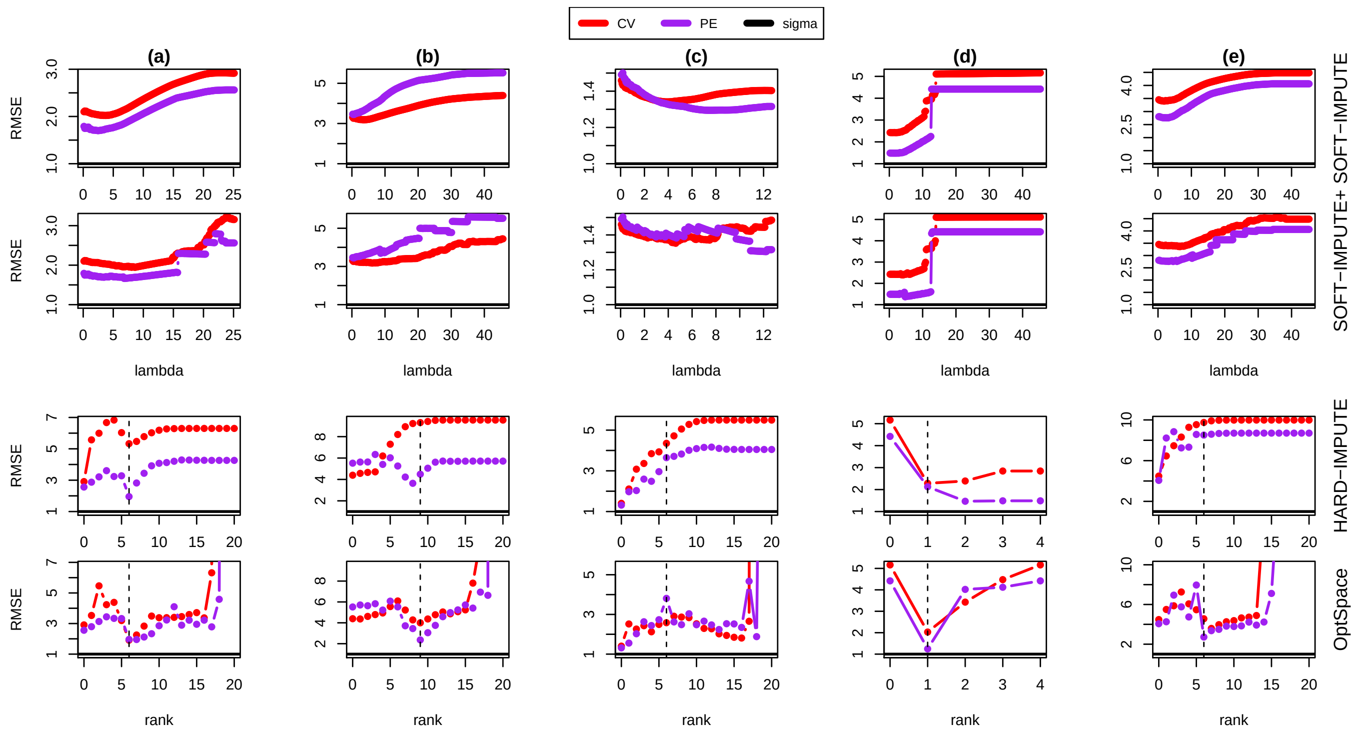


Figure 3.5: Estimated root mean prediction errors with SOFT-IMPUTE, SOFT-IMPUTE+, HARD-IMPUTE and OptSpace cross-validation. The true prediction error is in purple, the noise level is in black, and the CV curves are in red. The dotted vertical line indicates the true rank k_0. The four settings are (a) same non-zero singular values, (b) large rank k_0, (c) low signal-to-noise ratio, (d) small aspect ratio β, and (e) large fraction of missing entries. SOFT-IMPUTE and SOFT-IMPUTE+ cross-validations are more robust than HARD-IMPUTE and OptSpace cross-validations.

The simulation results are shown in Figure 3.5. The results suggest that SOFT-IMPUTE and SOFT-IMPUTE+ cross-validations are more robust than HARD-IMPUTE and OptSpace cross-validations. This may be due to the fact that HARD-IMPUTE and OptSpace can only converge to a local minimum. Even though OptSpace can go wildly when the specified rank is near full-rank, the minimum prediction error is as good as those of SOFT-IMPUTE and SOFT-IMPUTE+. Moreover, the CV error curve estimates the prediction error well for OptSpace.

3.6 Cross-Validation for Clustering

In this section we apply our CV approach to another important unsupervised task – clustering. The clustering problem is ubiquitous in exploratory data analysis. Given a collection of objects, we wish to group them so that members within the same group are similar and members in different groups are dissimilar. One of the key challenges in clustering is choosing the number of clusters. In this section, we rewrite the clustering problem as a matrix estimation problem, and use our CV approach to estimate the out-sample error and to choose the number of clusters. Experiments show that the CV approach is favorable compared to other commonly used methods.

Here we focus on one of the most basic and commonly used clustering methods – k-means clustering. Given the number of clusters k, k-means seeks the partition $\mathcal{C} = (C_1, \ldots, C_k)$ and centroids $B = (b_1, \ldots, b_k)$ that minimize the total within cluster sum of squares

$$\min_{\mathcal{C},B} \sum_{i=1}^{k} \sum_{j \in C_i} \|x_j - b_i\|_2^2. \tag{3.35}$$

To fit k-means into the setting of matrix estimation, we write the above optimization problem as

$$\min_{A \in H, B} \|X - AB^T\|_F^2, \tag{3.36}$$

where A encodes the cluster memberships and $H = \left\{A \in \{0,1\}^{N \times k} : A\mathbf{1} = \mathbf{1}\right\}$. Therefore, to find the optimal number of clusters, we can apply the cross-validation strategy in Section 3.3 to a grid of values of k and choose the one that minimizes the cross-validation error.

For each left-out fold, we need to solve

$$\min_{A \in H, B} \|P_\Omega(X - AB^T)\|_F^2. \tag{3.37}$$

Chi et al. [63] propose a method of k-means clustering on partially observed data, called k-POD, which doesn't require assumptions on the missingness pattern. The k-POD method employs a majorization-minimization algorithm to identify a clustering that is in accord with the observed data. This algorithm is very similar to HARD-IMPUTE, and the difference is the additional constraint of $A \in H$ in the matrix factorization step. The details of the k-POD algorithm are in Algorithm 3.5.

Algorithm 3.5 k-POD

1: Initialize $(A^{(0)}, B^{(0)})$
2: **while** not converged **do**
3: Fill in missing entries

$$X^{(i)} \leftarrow P_\Omega(X) + P_{\Omega^\perp}(A^{(i)}B^{(i)T})$$

4: Update the clustering

$$(A^{(i+1)}, B^{(i+1)}) \leftarrow k\text{-means}(X^{(i)})$$

5: **end while**
6: **Return:** $(A^{(i)}, B^{(i)})$

3.6.1 Experiments

We present results from a simulation study to compare the performance of the cross-validation approach with six standard approaches.

1. CH: Calinski and Harabasz [64] propose choosing the maximizer of

$$\mathrm{CH}(k) = \frac{B(k)/(k-1)}{W(k)/(N-k)}, \tag{3.38}$$

   where $B(k)$ and $W(k)$ are the between- and within- cluster sum of squares with k clusters.

2. KL: Krzanowski and Lai [65] suggest choosing the maximizer of

$$\mathrm{KL}(k) = \left| \frac{\mathrm{DIFF}(k)}{\mathrm{DIFF}(k+1)} \right|, \tag{3.39}$$

   where $\mathrm{DIFF}(k) = (k-1)^{2/p} W(k-1) - k^{2/p} W(k)$.

3. Hartigan: Hartigan [66] chooses the smallest value of k such that $H(k) \leq 10$, where

$$H(k) = (N-k-1) \left(\frac{W(k)}{W(k+1)} - 1 \right). \tag{3.40}$$

4. Silhouette: Kaufman and Rousseeuw [67] propose choosing the number of clusters that maximizes the average value of $s(i)$, where

$$s(i) = \frac{b(i) - a(i)}{\max[a(i), b(i)]} \tag{3.41}$$

   is a measure of how well the ith point is clustered. Here $a(i)$ is the average distance between the ith point and all other observations in its cluster, and $b(i)$ is the average distance to points in the "nearest" cluster.

5. Jump: Sugar and James [68] choose the maximizer of

$$J(k) = \hat{d}(k)^{-Y} - \hat{d}(k-1)^{-Y}, \tag{3.42}$$

where

$$\hat{d}(k) = \frac{1}{p}\frac{1}{N}\min_{\mathcal{C},B}\sum_{i=1}^{k}\sum_{j\in C_i}\|x_j - b_i\|_2^2. \tag{3.43}$$

is the minimum distortion obtained by applying the k-means clustering algorithm to the data. The transformation power parameter Y is positive, and a typical value is $Y = p/2$.

6. Gap: Tibshirani et al. [69] use the gap statistics

$$\mathrm{Gap}(k) = \frac{1}{B}\sum_{b}\log(W_b^*(k)) - \log(W(k)), \tag{3.44}$$

where B independent uniform datasets, each with the same range as the original data, are produced, and $W_b^*(k)$ is the within-cluster sum of squares for the bth uniform dataset. They suggest choosing the smallest value of k such that

$$\mathrm{Gap}(k) \geq \mathrm{Gap}(k+1) - s_{k+1}, \tag{3.45}$$

where s_k is the standard error of $\log(W_b^*(k))$.

We examine the performance of our cross-validation approach in four different simulations. For all simulations, the number of clusters is 5. The first simulation, which was designed to test the effectiveness of the methods on low-dimensional data, used a basic two-dimensional mixture of five Gaussian clusters, each with identity covariance, and with cluster means (0, 0), (2.5, 2.5), (5, 5), (-2.5, -2.5), and (-5,-5). The second simulation used a Gaussian mixture with identity covariance, but in dimension 10, and the cluster centers are evenly spaced on a line with separations of 3.5 in each dimension. The rest of the four simulations consider multivariate data in dimension 10, and the cluster centers are generated randomly from $N(0, 4I_{10})$. The third simulation examined performance when there was dependence among the dimensions and tested the effect of differing covariances. It used a distribution with five

Gaussian clusters each with 10-by-10 covariance matrix with correlations of -0.1, 0, 0.1, 0.3, and 0.5. In simulation four we produced five non-Gaussian clusters using t-distributions with variance 1 independently in each dimension. Each simulation is repeated 20 times. All of the simulated datasets contained 100 observations equally divided among the clusters. For each of the four scenarios, we ran cross-validation with 10 folds on each.

The results are shown in Table 3.2. The cross-validation approach appears to be very robust. It performs well in all of the scenarios, whereas each of the other approaches does poorly in at least two.

Table 3.2: Simulation results for k-means clustering. Simulation 1 had cluster means of (0, 0), (2.5, 2.5), (5, 5), (-2.5, -2.5), and (-5,-5). Simulations 2 had clusters evenly spaced on a line with separations of 3.5 in each dimension. Simulations 3-4 had cluster means generated randomly from $N(0, 4I_p)$. All simulations had within-cluster standard deviations of 1 in each dimension. The values are the average difference between the estimated number of clusters and the true number of clusters for 20 replicates of k-means clustering. The cross-validation approach appears to be very robust. It performs well in all of the scenarios, whereas each of the other approaches does poorly in at least two.

Simulation	**CV**	**CH**	**KL**	**Hartigan**	**Silhouette**	**Jump**	**Gap**
1. Two dimensions, identity covariance							
	-1.00	-1.80	2.90	2.40	-3.00	.80	-2.35
	(.65)	(2.04)	(2.07)	(1.67)	(.00)	(2.07)	(1.31)
2. Ten dimensions, identity covariance							
	.05	-1.60	3.35	.00	.00	.00	-.85
	(.51)	(2.01)	(1.31)	(.00)	(.00)	(.00)	(.67)
3. Ten dimensions, differing covariance							
	.20	-2.00	3.75	.20	-.90	1.50	-.65
	(.62)	(2.05)	(.91)	(.41)	(1.41)	(2.35)	(.49)
4. Ten dimensions, t distribution clusters							
	.15	-2.00	3.30	.50	-.20	2.00	-.70
	(.67)	(2.05)	(1.03)	(.83)	(.41)	(2.13)	(.73)

3.7 Conclusion

We propose a general and well-specified procedure for model selection in unsupervised learning. Our procedure is motivated by Cross-validation (CV), which is probably the simplest and most widely used method for model evaluation in supervised setting, but uses a novel method to partition data and estimate generalization error. Our procedure is nonparametric and can work in a variety of contexts, including principal component analysis, clustering and matrix completion. Moreover, we give a computationally efficient approximation to leave-one-out cross-validation.

Specifically, our procedure randomly partitions the elements of the data matrix into K folds. For each of the K folds, we leave it out as the hold-out set, fit to the remaining $K-1$ folds using some matrix completion algorithm, and evaluate its performance on the hold-out set. In general, the matrix completion algorithm used for estimating the hold-out set can be an expectation-maximization (EM) adaption of the unsupervised learning algorithm that we are considering. Then the average of model errors over all hold-out sets is our estimate of the generalization error.

We conduct an extensive simulation analysis comparing our procedure with other state-of-the-art model selection procedures for unsupervised learning. The results show that our procedure is the most generally applicable one and has the best performance in most settings.

Bibliography

[1] Ya Le and Trevor Hastie. Sparse quadratic discriminant analysis and community bayes. *arXiv preprint arXiv:1407.4543*, 2014.

[2] John C Duchi, Michael I Jordan, Martin J Wainwright, and Yuchen Zhang. Optimality guarantees for distributed statistical estimation. *arXiv preprint arXiv:1405.0782*, 2014.

[3] John C Duchi, Alekh Agarwal, and Martin J Wainwright. Dual averaging for distributed optimization: Convergence analysis and network scaling. *IEEE Transactions on Automatic control*, 57(3):592–606, 2012.

[4] Jason D Lee, Yuekai Sun, Qiang Liu, and Jonathan E Taylor. Communication-efficient sparse regression: a one-shot approach. *arXiv preprint arXiv:1503.04337*, 2015.

[5] Yuchen Zhang, Martin J Wainwright, and John C Duchi. Communication-efficient algorithms for statistical optimization. In *Advances in Neural Information Processing Systems*, pages 1502–1510, 2012.

[6] Ryan Mcdonald, Mehryar Mohri, Nathan Silberman, Dan Walker, and Gideon S Mann. Efficient large-scale distributed training of conditional maximum entropy models. In *Advances in Neural Information Processing Systems*, pages 1231–1239, 2009.

[7] Ryan McDonald, Keith Hall, and Gideon Mann. Distributed training strategies for the structured perceptron. In *Human Language Technologies: The 2010 Annual Conference*

of the North American Chapter of the Association for Computational Linguistics, pages 456–464. Association for Computational Linguistics, 2010.

[8] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603, 2010.

[9] Jonathan D Rosenblatt and Boaz Nadler. On the optimality of averaging in distributed statistical learning. *Information and Inference*, 5(4):379–404, 2016.

[10] Michael I Jordan, Jason D Lee, and Yun Yang. Communication-efficient distributed statistical inference. *arXiv preprint arXiv:1605.07689*, 2016.

[11] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011.

[12] Ohad Shamir, Nati Srebro, and Tong Zhang. Communication-efficient distributed optimization using an approximate newton-type method. In *International conference on machine learning*, pages 1000–1008, 2014.

[13] Finbarr O'Sullivan. A statistical perspective on ill-posed inverse problems. *Statistical science*, pages 502–518, 1986.

[14] DM Titterington. Common structure of smoothing techniques in statistics. *International Statistical Review/Revue Internationale de Statistique*, pages 141–170, 1985.

[15] Peter J Bickel and Elizaveta Levina. Some theory for fisher's linear discriminant function,'naive bayes', and some alternatives when there are many more variables than observations. *Bernoulli*, pages 989–1010, 2004.

[16] Line Clemmensen, Trevor Hastie, Daniela Witten, and Bjarne Ersbøll. Sparse discriminant analysis. *Technometrics*, 53(4), 2011.

[17] Sandrine Dudoit, Jane Fridlyand, and Terence P Speed. Comparison of discrimination methods for the classification of tumors using gene expression data. *Journal of the American statistical association*, 97(457):77–87, 2002.

[18] Yaqian Guo, Trevor Hastie, and Robert Tibshirani. Regularized linear discriminant analysis and its application in microarrays. *Biostatistics*, 8(1):86–100, 2007.

[19] Trevor Hastie, Andreas Buja, and Robert Tibshirani. Penalized discriminant analysis. *The Annals of Statistics*, pages 73–102, 1995.

[20] WJ Krzanowski, Philip Jonathan, WV McCarthy, and MR Thomas. Discriminant analysis with singular covariance matrices: methods and applications to spectroscopic data. *Applied statistics*, pages 101–115, 1995.

[21] Robert Tibshirani, Trevor Hastie, Balasubramanian Narasimhan, and Gilbert Chu. Diagnosis of multiple cancer types by shrunken centroids of gene expression. *Proceedings of the National Academy of Sciences*, 99(10):6567–6572, 2002.

[22] Ping Xu, Guy N Brock, and Rudolph S Parrish. Modified linear discriminant analysis approaches for classification of high-dimensional microarray data. *Computational Statistics & Data Analysis*, 53(5):1674–1687, 2009.

[23] Jerome H Friedman. Regularized discriminant analysis. *Journal of the American statistical association*, 84(405):165–175, 1989.

[24] Bradley S Price, Charles J Geyer, and Adam J Rothman. Ridge fusion in statistical learning. *arXiv preprint arXiv:1310.3892*, 2013.

[25] Quefeng Li and Jun Shao. Sparse quadratic discriminant analysis for high dimensional data. *Statist. Sinica*, 2013.

[26] Jiehuan Sun and Hongyu Zhao. The application of sparse estimation of covariance matrix to quadratic discriminant analysis. *BMC bioinformatics*, 16(1):48, 2015.

[27] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[28] Patrick Danaher, Pei Wang, and Daniela M Witten. The joint graphical lasso for inverse covariance estimation across multiple classes. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2013.

[29] Peter Bühlmann, Philipp Rütimann, Sara van de Geer, and Cun-Hui Zhang. Correlated variables in regression: clustering and sparse estimation. *Journal of Statistical Planning and Inference*, 143(11):1835–1858, 2013.

[30] Nadine Hussami and Robert Tibshirani. A component lasso. *arXiv preprint arXiv:1311.4472*, 2013.

[31] Mee Young Park, Trevor Hastie, and Robert Tibshirani. Averaged gene expressions for regression. *Biostatistics*, 8(2):212–227, 2007.

[32] B Boser Le Cun, John S Denker, D Henderson, Richard E Howard, W Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*. Citeseer, 1990.

[33] Kean Ming Tan, Daniela Witten, and Ali Shojaie. The cluster graphical lasso for improved estimation of gaussian graphical models. *arXiv preprint arXiv:1307.5339*, 2013.

[34] Han Liu, John Lafferty, and Larry Wasserman. The nonparanormal: Semiparametric estimation of high dimensional undirected graphs. *The Journal of Machine Learning Research*, 10:2295–2328, 2009.

[35] Han Liu, Fang Han, Ming Yuan, John Lafferty, and Larry Wasserman. The nonparanormal skeptic. *arXiv preprint arXiv:1206.6488*, 2012.

[36] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

[37] Anthony John Robinson. *Dynamic error propagation networks*. PhD thesis, University of Cambridge, 1989.

[38] Svante Wold. Cross-validatory estimation of the number of components in factor and principal components models. *Technometrics*, 20(4):397–405, 1978.

[39] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 79–86. ACM, 2010.

[40] David M Blei, Perry R Cook, and Matthew Hoffman. Bayesian nonparametric matrix factorization for recorded music. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 439–446, 2010.

[41] Nathan Srebro, Jason Rennie, and Tommi S Jaakkola. Maximum-margin matrix factorization. In *Advances in neural information processing systems*, pages 1329–1336, 2005.

[42] Jacob Abernethy, Francis Bach, Theodoros Evgeniou, and Jean-Philippe Vert. Low-rank matrix factorization with attributes. *arXiv preprint cs/0611124*, 2006.

[43] Daniela M Witten, Robert Tibshirani, and Trevor Hastie. A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. *Biostatistics*, 10(3):515–534, 2009.

[44] K Ruben Gabriel. The biplot-tool for the exploration of multidimensional data. *Journal of the French Society of Statistics*, 143(3-4).

[45] Art B Owen and Patrick O Perry. Bi-cross-validation of the svd and the nonnegative matrix factorization. *The annals of applied statistics*, pages 564–594, 2009.

[46] Rahul Mazumder, Trevor Hastie, and Robert Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *Journal of machine learning research*, 11(Aug):2287–2322, 2010.

[47] Raghunandan H Keshavan, Andrea Montanari, and Sewoong Oh. Matrix completion from a few entries. *IEEE Transactions on Information Theory*, 56(6):2980–2998, 2010.

[48] Trevor Hastie, Rahul Mazumder, Jason D Lee, and Reza Zadeh. Matrix completion and low-rank svd via fast alternating least squares. *Journal of Machine Learning Research*, 16:3367–3402, 2015.

[49] Emmanuel J Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717, 2009.

[50] Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

[51] Emmanuel J Candes and Yaniv Plan. Matrix completion with noise. *Proceedings of the IEEE*, 98(6):925–936, 2010.

[52] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

[53] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[54] Ian Jolliffe. Principal component analysis. In *International encyclopedia of statistical science*, pages 1094–1096. Springer, 2011.

[55] Patrick O Perry. Cross-validation for unsupervised learning. *arXiv preprint arXiv:0909.3052*, 2009.

[56] Raj Rao Nadakuditi and Alan Edelman. Sample eigenvalue based detection of high-dimensional signals in white noise using relatively few samples. *IEEE Transactions on Signal Processing*, 56(7):2625–2638, 2008.

[57] Jushan Bai and Serena Ng. Determining the number of factors in approximate factor models. *Econometrica*, 70(1):191–221, 2002.

[58] Edmund R Malinowski. Statistical f-tests for abstract factor analysis and target testing. *Journal of Chemometrics*, 3(1):49–60, 1989.

[59] Klaas Faber and Brace R Kowalski. Critical evaluation of two f-tests for selecting the number of factors in abstract factor analysis. *Analytica chimica acta*, 337(1):57–71, 1997.

[60] Mati Wax and Thomas Kailath. Detection of signals by information theoretic criteria. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33(2):387–392, 1985.

[61] Shira Kritchman and Boaz Nadler. Determining the number of components in a factor model from limited noisy data. *Chemometrics and Intelligent Laboratory Systems*, 94(1):19–32, 2008.

[62] Samarendra Nath Roy. On a heuristic method of test construction and its use in multivariate analysis. *The Annals of Mathematical Statistics*, pages 220–238, 1953.

[63] Jocelyn T Chi, Eric C Chi, and Richard G Baraniuk. K-pod: A method for k-means clustering of missing data. *The American Statistician*, 70(1):91–99, 2016.

[64] Tadeusz Caliński and Jerzy Harabasz. A dendrite method for cluster analysis. *Communications in Statistics-theory and Methods*, 3(1):1–27, 1974.

[65] Wojtek J Krzanowski and YT Lai. A criterion for determining the number of groups in a data set using sum-of-squares clustering. *Biometrics*, pages 23–34, 1988.

[66] John A Hartigan and JA Hartigan. *Clustering algorithms*, volume 209. Wiley New York, 1975.

[67] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

[68] Catherine A Sugar and Gareth M James. Finding the number of clusters in a dataset: An information-theoretic approach. *Journal of the American Statistical Association*, 98(463):750–763, 2003.

[69] Robert Tibshirani, Guenther Walther, and Trevor Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001.