MATRIX FACTORIZATION UNDER CONTAMINATION

Peter Ballen

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania
in
Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy
2020

Supervisor of Dissertation

_____

Aaron Roth, Class of 1940 Bicentennial Term Associate Professor, Computer and Information Science

Graduate Group Chairperson

_____

Rajeev Alur, Professor, Computer and Information Science

Dissertation Committee:
Anyndia De, Assistant Professor, Computer and Information Science
Brett Falk, Research Assistant Professor, Computer and Information Science
Edgar Dobriban, Assistant Professor, Statistics
Sudipto Guha, Research Scientist, Amazon

Acknowledgment

This dissertation would not have been possible without the support of countless people. I would especially like to thank the following individuals:

Sudipto Guha, for years of guidance. Under his mentorship, I grew from a young student uncertain of my interests and direction to a confident scholar aware of my place in the world and my ability to contribute to it. Aaron Roth, for advising and support while I was writing this dissertation. My dissertation committee - Anindya De, Brett Falk, and Edgar Dobriban - for being generous with their time and feedback. Cheryl Hickey and Sandy Herman, for nonacademic support for the duration of my time at the University of Pennsylvania.

My friends, who were always there for me. Stephen Philips, for being a great roommate for the past five years.

My family, particularly my parents Robert and Debra Ballen, who inspired in me a lifelong love of mathematics and learning, and who have always given me their love and support.

ABSTRACT

MATRIX FACTORIZATION UNDER CONTAMINATION

Peter Ballen

Aaron Roth

In the nonnegative matrix factorization problem, the user inputs a nonnegative matrix $\mathbf{V}$ and wants to factor $\mathbf{V} \approx \mathbf{WH}$, with both $\mathbf{W}$ and $\mathbf{H}$ nonnegative. Standard factorization techniques make unrealistic assumptions about the noise present in the data: that the noise is generated from independent and identically distributed Gaussian process. However, real world datasets are unlikely to satisfy this simplistic assumption. In particular, real world datasets suffer from contamination, anomalies, and outliers that cannot be modeled by simple Gaussian distributions. In this dissertation, we discuss novel techniques for matrix factorization under contamination and non-standard noise models. These techniques can be used both as a replacement for a standard factorization algorithm, or as an independent contamination detection procedure. We also prove a number of complexity bounds on the hardness of the problem.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Contaminated NMF

In the modern era, large high-dimensional datasets have become commonplace. However, these large datasets are difficult to work with - both because they have a large number of dimensions that make interpretation and analysis difficult, and because of the existence of contamination and anomalies in the data. Traditional matrix factorization solves the first problem by reducing the high-dimensional data into a low-dimensional factorization, but struggles in the presence of nonstandard noise. Traditional data cleaning solves the second problem by looking at statistical properties of the data, but struggles on high-dimensional data. In this dissertation, we combine the two approaches by considering the contaminated nonnegative matrix factorization problem (Contaminated NMF). Contaminated NMF takes as input a $n \times m$ matrix $\mathbf{V}$. The primary goal is to both find the underlying factorization of the

non-contaminated elements of $\mathbf{V}$ and to identify the contaminated entries.

There are two main applications of this work, depending on which direction you approach the problem. The algorithms can be viewed as factorization algorithms which are designed to find a better factorization than existing techniques on noisy data. Under this interpretation, the algorithms are dimensionality reduction algorithms that can be used in place of existing factorization techniques. Alternatively, the algorithms can be viewed as contamination detection algorithms which are designed to identify contaminated elements in high-dimensional data. Under this interpretation, the algorithms are classification algorithms which classify every entry as either clean or contaminated.

The biggest challenge encountered when solving the Contaminated NMF problem is coming up with a mathematically rigorous definition of contamination. Informally, contamination represents 'anomalous entries' - data which does not fit into the underlying factorization model. However, contamination need not satisfy simplistic mathematical modeling assumptions; for example, it does not have to be mean-zero, independent, or identically distributed.

This dissertation considers two approaches to formally define contamination. The first approach is the loss function approach commonly used in data science: we define a loss function L that will model the contamination. The problem of finding contamination is then translated into a problem of minimizing L, and we create algorithms that minimize this function. The second approach is a theoretical approach com-

monly seen in mathematical work: we define contamination as a property inherent to a matrix via a theory of critical sets. A critical set is a set of elements that can be altered to reduce the nonnegative rank of a matrix. The problem of finding contamination is then translated into a problem of deciding whether a critical set with certain properties exists.

1.2 Roadmap

Nonnegative Matrix Factorization lies at the intersection of data science, theory, machine learning, & statistics. In the one extreme, a data scientist may just want a simple algorithm that finds better quality nonnegative factorizations than the traditional Lee and Sheung algorithm and not particularly care about the underlying mathematics. In the other extreme, a theoretician can view NMF as a fascinating optimization problem and consider the fact that there are real-world datasets attached to the problem to be largely inconsequential. Neither approach is inherently right or wrong, and this dissertation draws upon ideas from all fields. In general, the most algorithmic results are in Chapter 3, and the dissertation becomes more theoretical in the later chapters, with Chapter 6 being almost entirely theory.

Chapter 2 gives introduces some notation that will be used for the remainder of the dissertation. It also gives some background on both linear algebra programming and nonnegative matrix factorization without contamination.

Chapter 3 begins the discussion on the loss function approach to contamination.

In this chapter, we assume the ϵ-contamination model commonly used in robust statistical inference. Under this model, we define two loss functions that model contamination: one built on the theory of M-estimators and one built on W-estimators. It also describes algorithms to optimize these loss functions, along with some useful optimizations that improve performance. The algorithms are then testing on a range of experiments. Chapter 4 discusses the convergence of these algorithms. Chapter 5 moves beyond the ϵ-contamination framework and describes how the loss functions and algorithms can be altered to handle alternative contamination models. We then test the modified algorithms on more datasets. These ideas and algorithms appeared in [5, 4].

Chapter 6 leaves the loss function approach and considers contamination as a property inherent in a matrix. This chapter relies heavily on theoretical computer science and graph theory. Many of the ideas and theorems appeared in [3]. Chapter 7 pulls the two strands together to discuss the NMF problem as a whole.

Chapter 2

Nonrobust Factorization

In this chapter, we discuss the multiplicative update rule framework for nonnegative matrix factorization. The algorithms in the subsequent chapters will involve multiplicative updates and will build upon this framework. The chapter is broken into three sections. Section 2.1 defines useful notation we will use for the entirety of the dissertation. Section 2.2 discusses basic linear algebra preliminaries and implementations. Section 2.3 discusses nonrobust NMF algorithms.

2.1 Notation

In this dissertation, we use capital bold letters ($\mathbf{V}, \mathbf{W}, \mathbf{H}$) to denote matrices. Unless otherwise specified, we will use n as the number of rows in the input matrix, m as the number of columns, and k as the inner factorization dimension. In other words, $\mathbf{V}$ is a $n \times m$ matrix with n rows in m columns, $\mathbf{W}$ is a $n \times k$ matrix with n rows

and k columns, and $\mathbf{H}$ is a $k \times m$ matrix with k rows and m columns.

We use subscripts i and j to denote a specific index of a matrix, for example $\mathbf{V}_{ij}$ refers to the element in the i-th row and j-th column of the $n \times m$ matrix $\mathbf{V}$. We also use this notation to denote a specific index of a matrix product. $\mathbf{WH}_{ij}$ is the element in the i-th row and j-th column of the $n \times m$ matrix $\mathbf{WH}$. It does *not* refer to a $n \times k$ matrix where each element of $\mathbf{W}$ is multiplied by the scalar value $\mathbf{H}_{ij}$. We use the standard linear algebra convention that matrix indices start at 1, not the computer science convention that array indices start at 0.

We use $\mathbf{X} \odot \mathbf{Y}$ to denote elementwise multiplication: $[\mathbf{X} \odot \mathbf{Y}]_{ij} = \mathbf{X}_{ij} * \mathbf{Y}_{ij}$. Similarly, matrix division is always done elementwise.

When meaning is obvious, we use $\sum_i(\cdot)$ as a shorthand for $\sum_{i=1}^{n}(\cdot)$, $\sum_j(\cdot)$ as shorthand for $\sum_{j=1}^{m}(\cdot)$, and $\sum_{ij}(\cdot)$ as shorthand for $\sum_{i=1}^{n}\sum_{j=1}^{m}(\cdot)$.

When discussing graphs in Chapter 6, we will use italics $(\mathcal{V}, \mathcal{E})$ to discuss graph theory concepts. We use $|\mathcal{V}|$ and $|\mathcal{E}|$ to represent the number of vertices and edges in graph $G = (\mathcal{V}, \mathcal{E})$. We will never use n or m to refer to the number of vertices or edges in a graph.

2.2 Linear Algebra Programming

There are a few standard linear algebra operations which are so fundamental they are referred to as basic matrix operations. These are also called atomic matrix operations or standard matrix operations in some books.

Definition 2.1. The following operations are basic matrix operations

- Matrix Multiplication - Given a $n_1 \times k$ matrix $\mathbf{X}$ and $k \times n_2$ matrix $\mathbf{Y}$, compute a $n_1 \times n_2$ matrix $\mathbf{Z} = \mathbf{XY}$ with $\mathbf{Z}_{ij} = \sum_{a=1}^{k} \mathbf{X}_{ia}\mathbf{Y}_{aj}$

- Matrix Transpose - Given a $n \times k$ matrix $\mathbf{X}$, compute a $k \times n$ matrix $\mathbf{Z} = \mathbf{X}^T$ with $\mathbf{Z}_{ij} = \mathbf{X}_{ji}$

- Element-wise Operations (also called ufuncs or map-only functions): Given a $n \times k$ matrix $\mathbf{X}$ and a function $f : \mathbb{R} \to \mathbb{R}$, compute a $n \times k$ matrix $\mathbf{Z} = f(\mathbf{X})$ with $\mathbf{Z}_{ij} = f(\mathbf{X}_{ij})$. Examples of element-wise operations include element-wise absolute value and element-wise maximum.

- Element-wise Arithmetic (also called broadcasting ufunc or map-reduce functions): Given a $n \times k$ matrix $\mathbf{X}$ and a $n \times k$ matrix $\mathbf{Y}$ and a function $g : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$, compute a $n \times k$ matrix $\mathbf{Z} = g(\mathbf{X}, \mathbf{Y})$ where $\mathbf{Z}_{ij} = g(\mathbf{X}_{ij}, \mathbf{Y}_{ij})$, Examples of element-wise arithmetic include elementwise addition or element-wise multiplication.

- Aggregation: Given a $n \times k$ matrix $\mathbf{X}$, return $\sum \mathbf{X}_{ij}$.

- Composition: Combine multiple of the above operations.

An example of a basic matrix operation formed by composition is computing $\sum_{ij}[\mathbf{V}_{ij} - \mathbf{WH}_{ij}]^2$, which can be done by composing matrix multiplication, element-wise subtraction $g(x, y) = x - y$, elementwise-operation $f(x) = x^2$, and aggregation.

2.2.1 Numpy

For a casual reader, it suffices to know that basic matrix operations defined above, and the algorithms described in this dissertation built on top of those operations, can be implemented in Python or in a distributed framework like Dask with good performance. The remainder of this section gives a more advanced look at linear algebra programming. Interested readers are further directed to the Numpy and Dask documentation [35, 12].

Basic matrix operations described in the prior section are used in countless applications and domains. Because they are so ubiquitous, a lot of work has been put into implementing these operations efficiently. A popular framework for writing linear algebra code is to write the code using a high-level language (Python) and a matrix library (Numpy). Numpy is an open-source library that widely used for working on high dimensional numeric data. Many other open source libraries - including the popular Scipy library and the distributed Dask library - are written on top of Numpy.

Numpy implements a special class called `ndarray` - a multidimensional array. A matrix is considered a two-dimensional ndarray. Older versions of Numpy implemented a separate `matrix` class, but this class has been depreciated in favor of the `ndarray` class. Data in a standard Python list may be scattered across system memory. However, data in an `ndarray` is stored in a single contiguous block. This storage philosophy allows Numpy to benefit from locality of reference: when Numpy needs to load data from memory, it can load the data in one large chunk and store the excess

data in cache.

The second optimization that Numpy employs is that because data is stored in one contiguous block, it can be accessed by other non-Python libraries. The `umath` subpackage is a set of functions that implements several common operations (including ufuncs). When the user calls the appropriate method, the `ndarray` passes the memory address of its data to `umath`, and then `umath` does the actual computational work in C. Because C is substantially faster than Python, this offers substantial speed improvement.

The third optimization Numpy employs is to combine `umath` with even more low-level libraries. A Basic Linear Algebra Subsystem (`blas`) is a library that implements common linear algebra operations. A Linear Algebra Package (`lapack`) is a library that implements more linear algebra operations. Unix comes with a default `blas` and `lapack`, but these default libraries are considered slow by modern day standards. Instead, multiple heavily optimized `blas` and `lapack` libraries have been developed and can be called by `umath`. ATLAS, OpenBLAS, and the Intel Math Kernel Library are three popular libraries. These libraries take advantage of multithreading and the machine architecture to offer substantial speed improvements. Furthermore, they use state of the art matrix algorithms. For example, the Strassen algorithm multiplies matrices in time $O(n^{2.807735})$, as compared to the naive algorithm which takes time $O(n^3)$.

The combination of these optimizations makes basic linear algebra operations

Table 2.1: Matrix Operation Speed Comparisons

Operation	Naive	Numpy
Matrix Sum	2.16s	**0.01s**
Elementwise Add	4.55s	**0.06s**
Matrix Multiplication	55.2s	**0.03s**

very fast in Numpy. As a demonstration, we compare Numpy's performance using the OpenBLAS with a naive Python implementation. These comparisons are run on $2^{12} \times 2^{12}$ matrices using an Intel i7 12-core Unix machine (the machine in my office, and the machine used for most of the experiments in this dissertation). We report the results in Table 2.2.1. As the table demonstrates, basic linear algebra operations are incredibly fast when done in Numpy. Additional tricks to improve the performance of Numpy are discussed in Chapter 3 (particularly Section 3.3.2).

2.3 Nonrobust Matrix Factorization

2.3.1 Traditional NMF

Given a $n \times m$ matrix $\mathbf{V}$, the goal of matrix factorization is to approximate $\mathbf{V} \approx \mathbf{WH}$ where $\mathbf{W}$ is $n \times k$ and $\mathbf{H}$ is $k \times m$, with k much smaller than both n and m. When $\mathbf{V}$, $\mathbf{W}$, and $\mathbf{H}$ are all required to be nonnegative, the problem is referred to as nonnegative matrix factorization (NMF). NMF offers several benefits over traditional factorization that allows negative values. A few such benefits include:

- NMF returns nonnegative factor matrices, which can be interpreted as scaled-up probabilities.

- NMF does not allow cancellation: the phenomenon where positive and negative entries in $\mathbf{W}$ and $\mathbf{H}$ cancel each other out. If $\mathbf{WH}_{ij} = 0$ (or is small), then it must be the case that $\mathbf{W}_{ia}$ or $\mathbf{H}_{aj}$ are 0 (or are small) for all values of a. In traditional factorization, $\mathbf{WH}_{ij} = 0$ implies nothing about $\mathbf{W}$ and $\mathbf{H}$

- NMF maps directly onto domain-specific applications, including the latent factor model and certain biomedicine applications [23, 34]

- NMF is less prone to overfitting than standard matrix factorization.

- NMF can be easily applied to a wide range of penalty functions and easily incorporates missing data and similar restrictions. We discuss a few in Section 2.3.2.

Early work on nonnegative matrix factorization was done in the 1990s [37]. The goal of this work was to minimize the Frobenius loss between $\mathbf{V}$ and $\mathbf{WH}$. This loss function, define in Equation 2.1, is also called the squared loss, L_2 loss, or Euclidean loss in certain papers.

$$L_{fro}(\mathbf{W}, \mathbf{H}) = \frac{1}{2} \sum_{ij} [\mathbf{V}_{ij} - (\mathbf{WH})_{ij}]^2 \tag{2.1}$$

The earliest algorithms were alternating regression algorithms which essentially performed gradient descent and had slow convergence. In 2001, Lee and Seung [27] gave a novel algorithm to optimize L_{fro} that took advantage of multiplicative updates. At iteration $t = 0$, $\mathbf{W}$ and $\mathbf{H}$ are randomly initialized. In each subsequent iteration, there algorithm has two steps: (1) Fix $\mathbf{H}$, update $\mathbf{W}$ (2) Fix $\mathbf{W}$, update $\mathbf{H}$.

Algorithm 2.1 Traditional NMF

1: Randomly Initialize $\mathbf{W}$ and $\mathbf{H}$
2: **for** $t = 1, 2, 3 \ldots$ **do**
3: $\quad \mathbf{W} \leftarrow \mathbf{W} \odot \frac{\mathbf{V}\mathbf{H}^T}{\mathbf{W}\mathbf{H}\mathbf{H}^T}$
4: $\quad \mathbf{H} \leftarrow \mathbf{H} \odot \frac{\mathbf{W}^T\mathbf{V}}{\mathbf{W}^T\mathbf{W}\mathbf{H}}$
5: **end for**

Lee and Seung prove that their algorithm is monotonic in L_{fro}, i.e. every step of the algorithm decreases L_{fro}. They notably do not prove any convergence result: we discuss convergence issues in Chapter 4.

2.3.2 Beyond Traditional NMF

After Lee and Seung published their Traditional NMF algorithm, interest in the NMF problem exploded. In the 2000s, large nonnegative datasets were abundant, and scientists were excited about the possiblities of NMF.

An important subsequent work is the case where $\mathbf{V}$ contains missing data. Let $\mathbf{Z} \geq 0$ be a $n \times m$ matrix of nonnegative weights. The weighted Frobenius loss is defined as

$$L_{weighted}(\mathbf{W}, \mathbf{H}) = \frac{1}{2} \sum_{ij} \mathbf{Z}_{ij} \odot [\mathbf{V}_{ij} - (\mathbf{W}\mathbf{H})_{ij}]^2 \tag{2.2}$$

The most common application of the weighted Frobenius loss is to set $\mathbf{Z}_{ij} = 1$ for a data point that is present in $\mathbf{V}$ and $\mathbf{Z}_{ij} = 0$ for a missing data. Once $\mathbf{W}$ and $\mathbf{H}$ are estimated using the present data, the missing data can be predicted using the

corresponding entries in the product $\mathbf{WH}$. Mao and Saul [30] modify Algorithm 2.1 to account for the $\mathbf{Z}$-weights. This algorithm is monotonic in $L_{weighted}(\mathbf{W}, \mathbf{H})$.

Algorithm 2.2 Weighted NMF

1: Randomly Initialize $\mathbf{W}$ and $\mathbf{H}$
2: **for** $t = 1, 2, 3 \ldots$ **do**
3: $\quad \mathbf{W} \leftarrow \mathbf{W} \odot \dfrac{(\mathbf{Z} \odot \mathbf{V})\mathbf{H}^T}{(\mathbf{WH} \odot \mathbf{Z})\mathbf{H}^T}$
4: $\quad \mathbf{H} \leftarrow \mathbf{H} \odot \dfrac{\mathbf{W}^T(\mathbf{Z} \odot \mathbf{V})}{\mathbf{W}^T(\mathbf{Z} \odot \mathbf{WH})}$
5: **end for**

Recognizing the limitations of the Frobenius norm on non-Gaussian data, a second line of work has modified the Lee and Seung algorithm to replace the Frobenius norm with a wide array of loss functions. We discuss a few common extensions. First, it is possible to add regularization terms to both $\mathbf{W}$ and $\mathbf{H}$ to enforce sparsity on the factor matrices. These regularization terms can include the L_1 norm, the L_2 norm, or a linear combination of the two norm. Let $\alpha_w, \alpha_h, \psi_w, \psi_h$ all be nonnegative constants (possibly zero). Then the regularized Frobenius loss is defined as

$$L_{regularized} = L_{fro}(\mathbf{W}, \mathbf{H}) + \sum_{i=1}^{n}\sum_{a=1}^{k} \alpha_w * (\mathbf{W}_{ia}) + \frac{\psi_w}{2} * (\mathbf{W}_{ia})^2 + \sum_{a=1}^{k}\sum_{j=1}^{m} \alpha_h * (\mathbf{H}_{aj}) + \frac{\psi_h}{2} * (\mathbf{H}_{aj})^2 \tag{2.3}$$

Algorithm 2.1 is easily modified to incorporate these additional restraints. Note Algorithm 2.1 is a special case of Algorithm 2.3 when $\alpha_w, \alpha_h, \psi_w, \psi_h$ are all zero.

Fevotte and Idier [14] consider replacing the Frobenius loss with the beta loss.

Algorithm 2.3 Regularized NMF

1: Randomly Initialize $\mathbf{W}$ and $\mathbf{H}$
2: **for** $t = 1, 2, 3 \ldots$ **do**
3: $\quad \mathbf{W} \leftarrow \mathbf{W} \odot \dfrac{\mathbf{V}\mathbf{H}^T}{\mathbf{W}\mathbf{H}\mathbf{H}^T + \alpha_w + \psi_w \odot \mathbf{W}}$
4: $\quad \mathbf{H} \leftarrow \mathbf{H} \odot \dfrac{\mathbf{W}^T\mathbf{V}}{\mathbf{W}^T\mathbf{W}\mathbf{H} + \alpha_h + \psi_h \odot \mathbf{H}}$
5: **end for**

The beta loss was defined in Basu et al [6] and is a generalization of the Frobenius loss. Fix a constant β. When $\beta \neq 0, 1$, the beta loss is defined as

$$L_{beta} = \sum_{ij} bfunc\left(\mathbf{V}_{ij}, \mathbf{W}\mathbf{H}_{ij}\right) \tag{2.4}$$

$$bfunc(x, y) = \frac{1}{\beta(\beta - 1)}\left(x^\beta + (\beta - 1)y^\beta - \beta x y^{\beta - 1}\right)$$

When $\beta = 0$, L_{beta} is defined as the Itakura-Saito divergence with $bfunc(x, y) = \frac{x}{y} - \log\frac{x}{y} - 1$. When $\beta = 1$, L_{beta} is defined as the Kullback-Leibler divergence with $bfunc(x, y) = x \log\frac{x}{y} - x + y$. When $\beta = 2$, L_{beta} is the Frobenius loss with $bfunc(x, y) = \frac{1}{2}(x^2 + y^2 - 2xy) = \frac{1}{2}(x - y)^2$.

Algorithm 2.4 Beta Loss NMF

1: Randomly Initialize $\mathbf{W}$ and $\mathbf{H}$
2: **for** $t = 1, 2, 3 \ldots$ **do**
3: $\quad \mathbf{W} \leftarrow \mathbf{W} \odot \dfrac{\mathbf{V}(\mathbf{W}\mathbf{H})^{\beta-2}\mathbf{H}^T}{(\mathbf{W}\mathbf{H})^{\beta-1}\mathbf{H}^T}$ ▷ exponent taken elementwise
4: $\quad \mathbf{H} \leftarrow \mathbf{H} \odot \dfrac{\mathbf{W}^T(\mathbf{W}\mathbf{H})^{\beta-2}\mathbf{V}}{\mathbf{W}^T(\mathbf{W}\mathbf{H})^{\beta-1}}$
5: **end for**

Note Algorithm 2.1 is a special case of Algorithm 2.4 when $\beta = 2$. The regularized

and weighted beta loss is identical to the regularized Frobenius loss in Equation 2.3, but with L_{beta} taking the roll of L_{fro}. The regularized beta loss is particularly important because it is the loss function implemented by scikit-learn, which is one of the most widely used Python libraries and a fundamental part of the PyData stack.

Separately, Kong et al [24] consider replacing the Frobenius norm with a different matrix norm. The absolute loss, also called the L_1 loss, is defined as

$$L_{abs} = \sum_{ij} |\mathbf{V}_{ij} - \mathbf{WH}_{ij}| \approx \sum_{ij} \frac{(\mathbf{V}_{ij} - \mathbf{WH}_{ij})^2}{\sqrt{(\mathbf{V}_{ij} - \mathbf{WH}_{ij})^2 + \epsilon}} \tag{2.5}$$

The critical insight of Kong et al's algorithm is that the absolute loss is approximately equal to the Frobenius loss with a weight $\mathbf{Z}_{ij} = ((\mathbf{V}_{ij} - \mathbf{WH}_{ij})^2 + \epsilon)^{-1/2} \approx |\mathbf{V}_{ij} - \mathbf{WH}_{ij}|^{-1}$. Based on this insight, they derive the following algorithm (note the similarity between Algorithm 2.5 and Algorithm 2.2).

Algorithm 2.5 Absolute Loss NMF

1: Randomly Initialize $\mathbf{W}$ and $\mathbf{H}$
2: **for** $t = 1, 2, 3 \ldots$ **do**
3: $\quad \mathbf{Z} \leftarrow ((\mathbf{V} - \mathbf{WH})^2 + \epsilon)^{-1/2}$ ▷ exponents taken elementwise
4: $\quad \mathbf{W} \leftarrow \mathbf{W} \odot \frac{(\mathbf{Z} \odot \mathbf{V})\mathbf{H}^T}{(\mathbf{Z} \odot \mathbf{WH})\mathbf{H}^T}$
5: $\quad \mathbf{Z} \leftarrow ((\mathbf{V} - \mathbf{WH})^2 + \epsilon)^{-1/2}$
6: $\quad \mathbf{H} \leftarrow \mathbf{H} \odot \frac{\mathbf{W}^T(\mathbf{Z} \odot \mathbf{V})}{\mathbf{W}^T(\mathbf{Z} \odot \mathbf{WH})}$
7: **end for**

Using a similar weighting scheme, Kong et al [24] also give an algorithm to update the L_{21} loss, defined as

$$L_{21}(\mathbf{W}, \mathbf{H}) = \sum_j \sqrt{\sum_i (\mathbf{V}_{ij} - \mathbf{W}\mathbf{H}_{ij})^2}$$

Chapter 3

Algorithms for Robust NMF

While NMF can be viewed as a pure optimization problem, it is useful to consider it as a statistical estimation problem. The ϵ-contamination model is a simple statistical model for modeling contamination; we will discuss more complex contamination models in Chapter 5.

In the ϵ-contamination model, we assume that a true low-rank matrix $\mathbf{V}_{true}$ exists. However, we only get access to $\mathbf{V}$, which is a noisy sampling of $\mathbf{V}_{true}$. Each entry (i, j) independently flips a bias ϵ-coin. With probability $1 - \epsilon$, an entry $\mathbf{V}_{ij}$ is drawn from $\mathbf{V}_{true}$ plus independent and identically distributed Gaussian noise. With probability ϵ, an entry $\mathbf{V}_{ij}$ is drawn from some alternative contamination distribution. Depending on the setting, the contamination distribution may be related to $\mathbf{V}_{true}$ (for example, adding high variance noise to $\mathbf{V}_{true}$) or unrelated (for example, overwriting the true value with a new value). Also depending on the setting, the user may or may not know the parameters of the Gaussian distribution or the value of ϵ. The user is given

access to $\mathbf{V}$, but does not know the results of the coinflips.

The user has two primary goals: (1) to determine which elements are contaminated, and (2) find a low rank factorization of $\mathbf{V}_{true}$. Note that if the user knew $\mathbf{V}_{true}$, they could just use Traditional NMF (Algorithm 2.1) on $\mathbf{V}_{true}$. If the user knew the result of the ϵ-coinflips, the user could use Weighted NMF (Algorithm 2.2) by assigning each entry $\mathbf{Z}_{ij} = 0$ or $\mathbf{Z}_{ij} = 1$ depending on the result of the coinflip. The problem is challenging specifically because the user does not know the results of the ϵ-coins.

In this chapter, we discuss two approaches to solving this problem: one built on M-estimators and one built on W-estimators. We test both approaches on experiments and show that they perform better at both goals than alternative algorithms.

3.1 Robust Statistics

Robust statistics has been a topic of interest in the mathematical community for decades. Statistical inference is based on both the given data and underlying assumptions of the underlying distribution. When those assumptions are invalid, the technique may return faulty solutions. A good robust estimator should: (1) be almost as statistically efficient as the standard estimator on uncontaminated data (2) be more resilient to invalid assumptions on contaminated data.

When statisticians refer to efficiency, they generally do not refer to computational efficiency or big-O runtime analysis. Statistical relative efficiency is a mathematical

term that refers to the performance of the estimator on a fixed amount of data. A more efficient estimator will return a more accurate prediction than a less efficient estimator.

For example, consider a set of n data points $y_1 \dots y_n$ drawn from the standard Gaussian distribution $Normal(0, 1)$. The mean is equal to $\sum y_i/n$ and can be computed in time $O(n)$. The median is equal to the midpoint of the data and requires time $O(n \log n)$ to compute due to the time required to sort the data points. Finally, consider the coinflip-mean which throws out 25% of the data at random and computes the mean on the remaining datapoints - this also takes time $O(n)$.

When given an infinite amount of data, the mean, the coinflip-mean, and the median all converge the true center of the underlying distribution (0 for the standard Gaussian). When only given n data points, we expect there to be some variation between the true center of the data and these statistical estimators. For any two estimators θ and θ', the asymptotic relative efficiency is the ratio of the variance of θ and the variance of θ'. In this dissertation, we will always define the mean and Frobenius loss (defined later in Equation 2.1) as the baseline for the relative efficiency.

Definition 3.1. The relative statistical efficiency of an estimator θ on a distribution $\mathcal{D}$ is the expected asymptotic variance of that estimator of θ on that distribution divided by the expected asymptotic variance of the mean or Frobenius loss on the same distribution.

Example 3.2. Suppose n data points are drawn from the standard normal

$Normal(0, 1)$. The mean is distributed as $Normal(0, \frac{1}{n})$ and has efficiency 1.00. The coinflip-mean is distributed as $Normal(0, \frac{3}{4n})$ and has efficiency $\frac{3}{4n}/\frac{1}{n} = \frac{3}{4}$. The median is distributed as $Normal(0, \frac{2}{\pi n}))$ and has efficiency ≈ 0.63.

Put another way, the coinflip mean actually does a better job at estimating the center of the underlying distribution than the median, at least on data that hasn't undergone contamination.

Example 3.3. Suppose n data points are drawn from the following procedure: with probability $1 - \epsilon$, the point is drawn from the standard normal $Normal(0, 1)$. With probability ϵ, the mean is drawn from $Normal(0, \sigma^2)$, a Gaussian distribution with a much larger variance. This corresponds to the ϵ-contamination model described earlier where the contamination distribution is $Normal(0, \sigma^2)$.

The mean is distributed as $Normal(0, \frac{1}{n}(1 - \epsilon + \sigma^2))$ and has relative efficiency 1.00. The coinflip-mean is distributed as $Normal(0, \frac{3}{4n}(1 - \epsilon + \sigma^2))$ and has relative efficiency 0.75. The median is distributed as $Normal(0, \frac{2}{\pi n}\sigma(\sigma + \epsilon - \epsilon\sigma)^{-2})$. The relative efficiency of the median is equal to $\frac{2}{\pi}(1 - \epsilon + \epsilon/\sigma)^2(1 - \epsilon + \epsilon\sigma^2)$. When $\sigma = 3$, the relative efficiency of the median is plotted in Figure 3.1. As the figure demonstrates, the median is more efficient than the mean (i.e. the relative efficiency is greater than one) when $0.10 < \epsilon < 0.81$.

Observe that the big-O runtime and runtime efficiency required to calculate of the mean, coinflip-mean, and median do not change between the two examples - statistical efficiency is unrelated to computational efficiency. A common tradeoff is

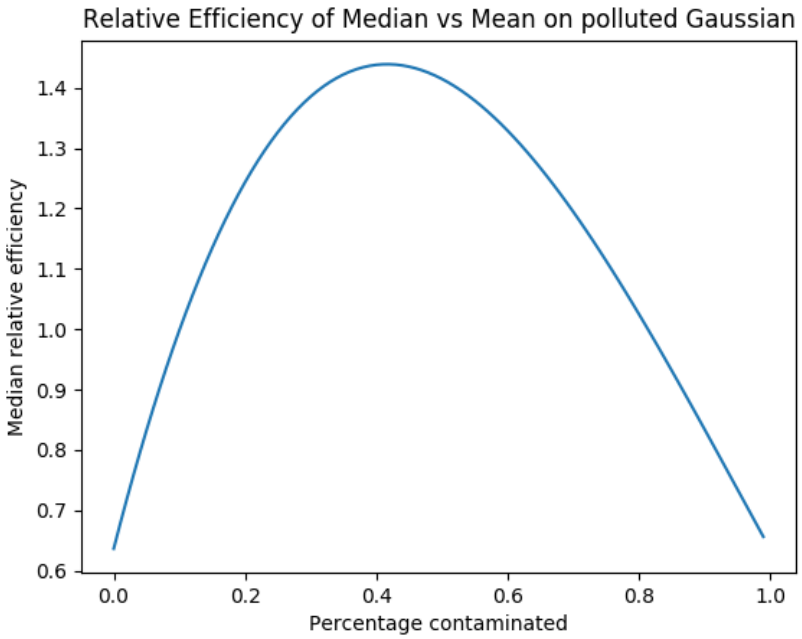


Figure 3.1: Relative efficiency of median vs mean on the distribution described in Example 3.3. X-axis repreents ϵ, y-axis represents efficiency of median on ϵ-contamination model.

making an estimator less efficient on Gaussian noise while making the estimator on ϵ-contamination. Information theory proves that the mean is the optimal estimator on Gaussian noise, i.e. no estimator can have relative efficiency greater than 1.00 on Gaussian noise. Therefore, the goal is to be close to 1 on Gaussian noise while being robust to outliers.

3.2 M-Robust Loss Function

Peter Huber popularized the field of robust statistics in 1984 with his theory of M-estimators. M-estimators are modified loss functions that are more resistant to contamination than the mean and/or squared loss. Given a set of points $\{y_i\}$, the mean can be defined as the value of θ that minimizes $\sum_i (y_i - \theta)^2$. The idea of M-estimators is to replace $(y_i - \theta)^2$ with some other function $\rho(x)$. A good M-estimator should behave similarly to the squared loss for small errors and should be more resilient for large errors.

Definition 3.4. Let $\rho(x)$ be a nonnegative function such that $\rho(x) \approx x^2$ when $|x| \leq \lambda$ and $\rho(x) \leq x^2$ when $|x| > \lambda$. The M-estimator of a dataset $\{y_i\}$ is the value of θ that minimizes $\sum_i \rho(y_i - \theta)$. The value λ is called the cutoff point.

The L_{pq} matrix norm (including the absolute loss and L_{21} loss) are M-estimator. However, these norms are known to have low statistical efficiency (the absolute loss is the multidimensional analog to the median), and as we will demonstrate in experiments, do not provide satisfactory results. Peter Huber suggests the Huber Loss as a

superior M-estimator [18]. For $\lambda > 0$, the Huber function and corresponding Huber Loss when applied to the matrix factorization problem are defined as

$$L_{huber}(\mathbf{W}, \mathbf{H}) = \sum_{ij} Huber(\mathbf{V}_{ij} - (\mathbf{WH})_{ij}, \lambda)$$

$$Huber(x, \lambda) = \begin{cases} \frac{1}{2}x^2 & |x| \leq \lambda \\ \lambda(|x| - \frac{1}{2}\lambda) & \text{o/w} \end{cases}$$

The Huber loss is quadratic for uncontaminated elements with error less than λ and is linear for contaminated elements with error greater than λ, thus satisfying Definition 3.4. As λ increases, the Huber loss becomes more conservative and is less likely to mark an element as contaminated. In statistical terms, increasing λ increases the efficiency of the algorithm but decreases the robustness. If the underlying data is drawn from Gaussian noise with standard deviation σ, setting $\lambda = 1.345\sigma$ gives the Huber loss an efficiency of 0.95 on the uncontaminated model while still providing good performance on ϵ-contamination. Decreasing λ will make the algorithm more robust on ϵ-contamination in exchange for making the loss less efficient on uncontaminated data.

We next consider the Robust Loss. For $\lambda > 0$ and for a $n \times m$ matrix $\mathbf{S}$, the Robust Loss is defined as

$$L_{robust}(\mathbf{W}, \mathbf{H}, \mathbf{S}) = \sum_{ij} \frac{1}{2}[\mathbf{V}_{ij} - (\mathbf{WH})_{ij} - \mathbf{S}_{ij}]^2 + \lambda|\mathbf{S}_{ij}| \tag{3.1}$$

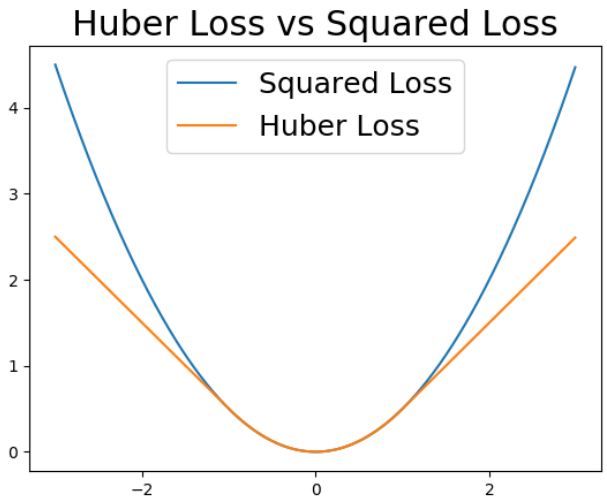


Figure 3.2: Huber Loss (orange) vs Squared Loss (blue) with $\lambda = 1$

$\mathbf{S}$ is a $n \times m$ 'correction' matrix where nonzero elements of $\mathbf{S}$ correspond to contaminated elements of $\mathbf{V}$. The matrix $\mathbf{V} - \mathbf{S}$ will be our estimate for the values of the uncontaminated matrix, observing that when $\mathbf{S} = 0$, $\mathbf{V} - \mathbf{S} = \mathbf{V}$ and the given matrix is our estimate for the uncontaminated matrix.

The Robust Loss has been applied to standard matrix factorization by Candès et al [8], but their algorithm don't extend to the NMF setting. It has also been applied to nonnegative matrix factorization by Shen et al [41], but their algorithm is not a multiplicative update algorithm and lacks theoretical guarantees. It's possible to replace the $|\mathbf{S}|$ term with other terms such as the L_{12} matrix norm, as done by Kannan et al [21]. However, replacing $|\mathbf{S}|$ with other penalty functions breaks the connection to the Huber loss.

The following theorem proves the correspondence between the Huber Loss and the Robust Loss.

Theorem 3.5. *For any* $\mathbf{W}, \mathbf{H} \geq 0$, $L_{huber}(\mathbf{W}, \mathbf{H}) = \min_{\mathbf{S}} L_{robust}(\mathbf{W}, \mathbf{H}, \mathbf{S})$

Proof. Fix $\mathbf{W}$ and $\mathbf{H}$. Then

$$\frac{\partial}{\partial \mathbf{S}_{ij}} L_{robust}(\mathbf{W}, \mathbf{H}, \mathbf{S}) = -(\mathbf{V}_{ij} - (\mathbf{WH})_{ij} - \mathbf{S}_{ij}) + \lambda * \frac{\partial}{\partial \mathbf{S}_{ij}} |\mathbf{S}_{ij}|$$

Setting the derivative to 0 and solving for $\mathbf{S}_{ij}$ gives that

$$\underset{\mathbf{S}_{ij}}{argmin}\, L_{robust} = \begin{cases} \mathbf{V}_{ij} - (\mathbf{WH})_{ij} + \lambda & \mathbf{V}_{ij} - \mathbf{WH}_{ij} < -\lambda \\ 0 & |\mathbf{V}_{ij} - (\mathbf{WH})_{ij}| \leq \lambda \\ \mathbf{V}_{ij} - (\mathbf{WH})_{ij} - \lambda & \mathbf{V}_{ij} - \mathbf{WH}_{ij} > \lambda \end{cases}$$

When $\mathbf{V}_{ij} - [\mathbf{WH}]_{ij} < -\lambda$, then $\mathbf{S}_{ij} = \mathbf{V}_{ij} - [\mathbf{WH}]_{ij} + \lambda$, and

$$\begin{aligned} \tfrac{1}{2}(\mathbf{V}_{ij} - (\mathbf{WH})_{ij} - \mathbf{S}_{ij})^2 + \lambda |\mathbf{S}_{ij}| &= \tfrac{1}{2}(-\lambda)^2 + \lambda |\mathbf{V}_{ij} - (\mathbf{WH})_{ij} + \lambda| \\ &= \tfrac{1}{2}\lambda^2 + \lambda * |\mathbf{V}_{ij} - (\mathbf{WH})_{ij}| - \lambda^2 \\ &= \lambda \left(|\mathbf{V}_{ij} - (\mathbf{WH})_{ij}| - \tfrac{1}{2}\lambda \right) \\ &= Huber(\mathbf{V}_{ij} - (\mathbf{WH})_{ij}, \lambda) \end{aligned}$$

When $\mathbf{V}_{ij} - [\mathbf{WH}]_{ij} > \lambda$, the same argument holds. Otherwise, when $-\lambda \leq \mathbf{V}_{ij} - [\mathbf{WH}]_{ij} \leq \lambda$, $\mathbf{S}_{ij} = 0$. In all three cases, $\frac{1}{2}(\mathbf{V}_{ij} - [\mathbf{WH}]_{ij} - \mathbf{S}_{ij})^2 + \lambda |\mathbf{S}_{ij}| = Huber(\mathbf{V}_{ij} - [\mathbf{WH}]_{ij})$. It follows that the Huber loss is equal to the Robust loss. □

Boundary Restrictions Our estimate for the uncontaminated matrix is $\mathbf{V} - \mathbf{S}$, which we assume is nonnegative. However, if $\mathbf{S} > \mathbf{V}$, then $\mathbf{V} - \mathbf{S} < 0$, which violates this assumption. Thus, we impose the restriction that $-\infty \leq \mathbf{S}_{ij} \leq \mathbf{V}_{ij}$. We call this restraint the Bounded $\mathbf{S}$-Restriction.

It is possible, and occasionally desirable, to impose a stronger restriction on $\mathbf{S}$. A common restriction is the Nonnegative $\mathbf{S}$-Restriction that requires that $0 < \mathbf{S}_{ij} < \mathbf{V}_{ij}$. It is appropriate when we have external knowledge that the contamination is purely additive. Additionally, under the Nonnegative $\mathbf{S}$-Restriction, if $\mathbf{V}_{ij} = 0$, then $\mathbf{S}_{ij} = 0$. In other words, this restriction imposes the constraint that 0-entries in $\mathbf{V}$ are never considered contaminated. In settings where $\mathbf{V}$ is sparse and the 0 entries are known with 100% certainty to be uncontaminated, this restriction is desirable.

3.3 M-Robust Algorithm

We now discuss an algorithm to minimize $L_{robust}(\mathbf{W}, \mathbf{H}, \mathbf{S})$, which will be equivalent to minimizing $L_{huber}(\mathbf{W}, \mathbf{H})$. The algorithm has three steps: (1) Update $\mathbf{W}$, keeping $\mathbf{H}$ and $\mathbf{S}$ fixed (2) Update $\mathbf{H}$, keeping $\mathbf{W}$ and $\mathbf{S}$ fixed (3) Update $\mathbf{S}$ to its optimal value in the constrained domain, keeping $\mathbf{W}$ and $\mathbf{H}$ fixed. These three steps are repeated until convergence.

Steps (1) and (2) derives immediately from the standard NMF multiplicative update rules, replacing $\mathbf{V}$ with $\mathbf{V} - \mathbf{S}$. Step (3) derives from $\operatorname{argmin} L_{robust}(\mathbf{W}, \mathbf{H}, \mathbf{S})$, which we computed as part of Theorem 3.5.

$$\mathbf{W} \leftarrow \mathbf{W} \odot \frac{(\mathbf{V}-\mathbf{S})\mathbf{H}^T}{\mathbf{W}\mathbf{H}\mathbf{H}^T} \qquad \mathbf{H} \leftarrow \mathbf{H} \odot \frac{\mathbf{W}^T(\mathbf{V}-\mathbf{S})}{\mathbf{W}^T\mathbf{W}\mathbf{H}} \tag{3.2}$$

$$\mathbf{S}_{ij} \leftarrow \begin{cases} (\mathbf{V}-\mathbf{W}\mathbf{H})_{ij} + \lambda & \text{if } (\mathbf{V}-\mathbf{W}\mathbf{H})_{ij} < -\lambda \\ 0 & \text{if } -\lambda \leq (\mathbf{V}-\mathbf{W}\mathbf{H})_{ij} < \lambda \\ (\mathbf{V}-\mathbf{W}\mathbf{H})_{ij} - \lambda & \text{if } \lambda \leq (\mathbf{V}-\mathbf{W}\mathbf{H})_{ij} < \mathbf{V}_{ij} + \lambda \\ \mathbf{V} & \text{if } (\mathbf{V}-\mathbf{W}\mathbf{H})_{ij} > \mathbf{V}_{ij} + \lambda \end{cases} \tag{3.3}$$

Observe that equation 3.3 takes $(\mathbf{V}-\mathbf{W}\mathbf{H})^2$ and 'pulls' the entries towards 0. Entries that are very large are instead pulled to $\mathbf{V}$ instead.

Putting the upper bound on $\mathbf{S}$ ensures that $\mathbf{V} \geq \mathbf{S}$ as we discussed above. In order to enforce the nonnegative $\mathbf{S}$-restriction, any time $\mathbf{S}$ would be set to a negative number, it is set to zero instead. Equation 3.3 sets $\mathbf{S}$ to its optimal value on either $[-\infty, \mathbf{V}]$ or $[0, \mathbf{V}]$ as appropriate.

Algorithm 3.6 M-Robust NMF

1: **for** $t = 1, 2, 3 \ldots$ **do**
2: Update each entry of $\mathbf{W}$ using Equation 3.2
3: Update each entry of $\mathbf{H}$ using Equation 3.2
4: Update each entry of $\mathbf{S}$ using Equation 3.3
5: Set $\mathbf{S} = \max(\mathbf{S}, 0)$ if imposing Nonnegative $\mathbf{S}$-contamination
6: **end for**

Steps (1) and (2) inherit the monotonic properties of Standard NMF. Step (3) is monotonic in L_{robust} because $\mathbf{S}$ is always set to its optimal value in the constrained domain. Convergence is discussed in the next chapter. There are two potential

optimizations that can be further applied to Algorithm 3.6 to minimize the amount of time required to update $\mathbf{S}$. Neither optimization increases the asymptotic runtime of the algorithm or break monotonicity, but can reduce the time required to update $\mathbf{S}$ by up 80%. When both optimizations are applied, updating $\mathbf{S}$ on a $2^{12} \times 2^{12}$ matrix only requires about 75ms.

3.3.1 Optimization: Update Sets

Algorithm 3.6 updates every element of $\mathbf{S}$ (nm entires total) at every iteration. This is wasteful: if we expect $\mathbf{S}$ to be sparse, then updating an element from 0 to 0 hasn't improved L_{robust}. We modify Algorithm 3.6 to incorporate the idea of an update set U_t. We will only modify elements of $\mathbf{S}$ that are in the update set.

Algorithm 3.7 M-Robust NMF with update sets

1: **for** $t = 1, 2, 3 \ldots$ **do**
2: Update each entry of $\mathbf{W}$ using Equation 3.2
3: Update each entry of $\mathbf{H}$ using Equation 3.2
4: Generate update set U_t
5: Update entries of $\mathbf{S}$ in U_t using Equation 3.3
6: **end for**

Setting U_t to be large means doing more computational work for every $\mathbf{S}$-update set, which is slow when n and m are large or in a distributed setting . Setting U_t to be small means doing less computational work, but may slow down the convergence of the algorithm. Thus, our goal is to find update sets that are small so that updates are fast, but large enough to contain the contaminated entries. We consider four update set rules.

The Total Update Set rule sets $U_t = ALL$; note that this reduces Algorithm 3.7 to Algorithm 3.6. The Deterministic Update Sets update $1/a$ fraction of the indices in each iteration in a cyclical order, where $a \geq 1$ is a fixed constant. For example, if $a = 5$, we update the first row of $\mathbf{S}$ in iterations $1, 6, 11, 16, \ldots$. The Random Update Sets update a random $1/a$ fraction of the indices.

The Greedy Update set is a different style of heuristic. At iteration 0, we put every element into U_t. If a index (i, j) sets $\mathbf{S}_{ij}$ in iteration t, it gets kicked out of the update set. Thus, the update set will shrink over time. This corresponds to a kind of greedy update rule - we greedily keep the elements with high error in the update set (recall $\mathbf{S}_{ij} \neq 0$ if $|\mathbf{V}_{ij} - [\mathbf{WH}]_{ij}| > \lambda$). However, to maintain the theoretical convergence guarantee, we must add a small random set of indices to U_t at every iteration. At heart, this is an exploration-exploitation tradeoff. We want to explore by putting random entries into U_t. At the same time we want to exploit our knowledge by putting entries with high error into U_t and taking entries with low error out of U_t. We summarize the four update set generation rules below.

- Total Update Sets - $U_t = ALL$, where ALL is the set of all nm indices of $\mathbf{S}$
- Deterministic Update Sets - $U_t = \{(i, j) : i \mod a = t \mod a\}$ for fixed constant $a \geq 1$
- Random Update Sets $U_t = RAND(1/a)$, i.e. $(i, j) \in U_t$ with probability $1/a$ independent of all other entries.

- Greedy Update Sets - $U_0 = ALL$, $U_t = \{(i,j) \in U_{t-1} : \mathbf{S}_{ij} \neq 0\} + RAND(1/a)$.

Note that it may be tempting to do an expensive preprocessing step to find an optimal U_t. However, we want the generation of U_t to be very fast. For example, a seemingly-plausible way to generate U_t would be to compute $\mathbf{V} - \mathbf{WH}$ and putting entries with high error into U_t. But this is unacceptably expensive - if you've computed $\mathbf{V} - \mathbf{WH}$ at every entry, you might as well set $U_t = ALL$.

The next definition comes from stochastic convergence theory. The above four update set rules are all recurrent and thus guarantee convergence (which we discuss in the next chapter). The above four rules also happen to be Markovian, but this is not a requirement for the convergence result to hold.

Definition 3.6. Element (i,j) is recurrent if for any fixed t_0,

$$\Pr\left[(i,j) \notin \bigcup_{t=t_0}^{\tau} U_t\right] \to 0 \text{ as } \tau \to \infty$$

An update set rule $\{U_t\}$ is recurrent if every element (i,j) is recurrent.

3.3.2 Optimization: Numba

Any implementation of Standard NMF can easily be modified to implement Equation 3.2 simply by replacing $\mathbf{V}$ with $\mathbf{V} - \mathbf{S}$. Updating $\mathbf{S}$ using Equation 3.3 can be done using built-in Numpy functionality, as demonstrated in Algorithm 3.8. However, by adding the just-in-time compiler Numba, it is easy to compile the entire

Algorithm 3.8 Numpy Implementation of Update-S

```
#Raw Numpy, less efficient than Numba version
from numpy import dot
Dif = V - dot(W,H)
S = updateS_basic(V, Err)

# Update S using build-in numpy primitives
def updateS_basic(V, Err):
        S = Dif + lamb
        S[Dif > -1 * lamb] = 0
        S[Dif > lamb] = (Dif - lamb)[Dif > lamb]
        S[Dif > V + lamb] = V[Dif > V+lamb]
        return S
```

S-update into a single `ufunc`, as demonstrated in Algorithm 3.9 (see Section 2.2.1 for background on ufuncs). The Numba algorithm is substantially faster than algorithm that relies on Numpy built-in functionality. On benchmark testing, `updateS_vec` was about 5x faster than `updateS_basic`.

3.4 W-Robust Loss Function

An alternative M-estimator to the Huber loss is the Winsor loss, originally proposed by Charles Winsor as an alternative way to estimate the mean on data that has undergone contamination. For $\lambda > 0$, the Winsor Loss is defined as follows.

Algorithm 3.9 Numba Implementation of Update-S

```
from numpy import dot
from numba import float64, vectorize
Dif = V - np.dot(W,H)
S = updateS_vec(V, Dif)

# Numba will compile operations into a single ufunc
@vectorize([(float64(float64, float64))])
def updateS_vec(v, d):
        if d < -1 * lamb:
                return e + lamb
        elif -1 * lamb <= d and d < lamb:
                return 0
        elif lamb <= d and d < v + lamb:
                return d - lamb
        else:
                return v
```

$$L_{winsor}(\mathbf{W}, \mathbf{H}) = \sum_{ij} Winsor(\mathbf{V}_{ij} - (\mathbf{WH})_{ij}, \lambda)$$

$$Winsor(x, \lambda) = \begin{cases} \frac{1}{2}x^2 & |x| \leq \lambda \\ \\ \frac{1}{2}\lambda^2 & \text{o/w} \end{cases}$$

The Winsor loss is quadratic for uncontaminated elements with error less than λ, just like the Huber loss. However, the Winsor loss constant for contaminated entries with error greater than λ, while the Huber loss is linear. This makes the Winsor loss more robust against outliers that are very far from the rest of the data. Like the Huber loss, the Winsor loss becomes more conservative as λ increases, i.e. increasing λ increases the relative statistical efficiency of the algorithm, but decreases

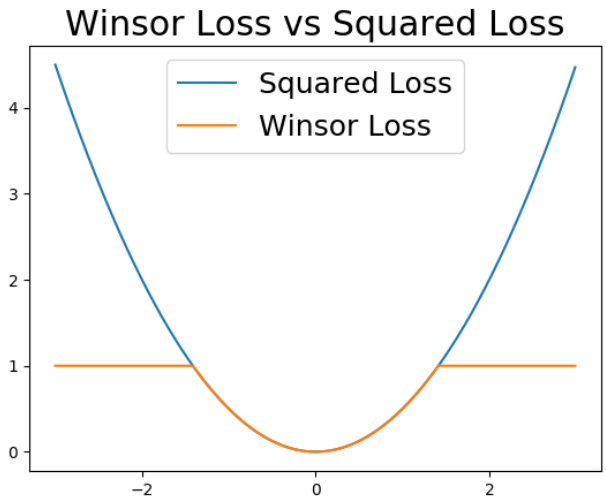


Figure 3.3: Winsor Loss (orange) vs Squared Loss (blue) with $\lambda = 1$

the robustness.

The Winsor loss is difficult to optimize directly because it 'flattens out' for values greater than λ and has derivative equal to zero. To optimize the Winsor less, we will treat the Winsor loss as an W-estimator instead of an M-estimator. W-estimators are statistical estimators that assign a weight to each entry and minimize a weighted sum of the individual errors. The weight should be close to 1 for small errors and should decrease for larger errors. The formal definition is given below

Definition 3.7. Let $w(x)$ be a nonnegative function such that $w(x) \approx 1$ when $|x| \leq \lambda$ and $w(x) \leq 1$ when $|x| > \lambda$. The W-estimator of a dataset $\{y_i\}$ is the value of θ that minimizes $\sum_i w(y_i - \theta) * (y_i - \theta)^2$. The value λ is called the cutoff.

Most M-estimators can be written as W-estimators using an appropriate choice of weighting function. For example, the absolute loss (Equation 2.5) can be written

as a W-estimator with $w(x) = sgn(x)/x$.

We next consider the Weighted Loss with a constant penalty term. For $\lambda > 0$ and for a $n \times m$ matrix $\mathbf{Z}$, where every entry of $\mathbf{Z}$ is between 0 and 1, the weighted loss is defined as

$$L_{weighted}(\mathbf{W}, \mathbf{H}, \mathbf{Z}) = \sum_{ij} \mathbf{Z}_{ij} \odot \frac{1}{2}[\mathbf{V}_{ij} - (\mathbf{WH})_{ij}]^2 + (1 - \mathbf{Z}_{ij}) \odot \frac{1}{2}\lambda^2 \tag{3.4}$$

$\mathbf{Z}$ is a $n \times m$ 'weighting' function where small elements of $\mathbf{Z}$ correspond to contaminated elements of $\mathbf{V}$, and large elements of $\mathbf{Z}$ correspond to uncontaminated elements. The second $\lambda(1-\mathbf{Z})$ term ensures that the loss function 'pays' a penalty of λ for marking an element as contaminated. Much the same way the Huber loss and Robust loss were equivalent, the Winsor loss is equivalent to the weighted loss with a constant λ penalty.

Theorem 3.8. *For any* $\mathbf{W}, \mathbf{H}$, $L_{winsor}(\mathbf{W}, \mathbf{H}) = \min_{\mathbf{Z}} L_{weighted}(\mathbf{W}, \mathbf{H}, \mathbf{Z})$

Proof. Fix $\mathbf{W}$ and $\mathbf{H}$. Observe that Equation 3.4 is minimized when $\mathbf{Z}_{ij} = 0$ or $\mathbf{Z}_{ij} = 1$ for every element of $\mathbf{Z}$.

Consider the case where $|\mathbf{V}_{ij} - \mathbf{WH}_{ij}| \leq \lambda$. Then setting $\mathbf{Z}_{ij} = 1$ gives the weighted loss a penalty of $\frac{1}{2}(\mathbf{V}_{ij} - \mathbf{WH}_{ij})^2 \leq \frac{1}{2}\lambda^2$, while setting $\mathbf{Z}_{ij} = 0$ gives a penalty of $\frac{1}{2}\lambda^2$. It is optimal to set $\mathbf{Z}_{ij} = 1$ and pay the first term, which is equal to $L_{winsor}(\mathbf{W}, \mathbf{H})$.

Alternatively, suppose $|\mathbf{V}_{ij} - \mathbf{WH}_{ij}| > \lambda$. Then setting $\mathbf{Z}_{ij} = 1$ gives the weighted

loss a penalty of $\frac{1}{2}(\mathbf{V}_{ij} - \mathbf{WH}_{ij})^2 > \frac{1}{2}\lambda^2$, while setting $\mathbf{Z}_{ij} = 0$ gives a penalty of $\frac{1}{2}\lambda^2$. It is optimal to set $\mathbf{Z}_{ij} = 0$ and pay the second term, which is again equal to $L_{winsor}(\mathbf{W}, \mathbf{H})$. □

3.5 W-Robust Algorithm

We now discuss an algorithm to minimize $L_{weighted}(\mathbf{W}, \mathbf{H}, \mathbf{Z})$, again noting that this is equivalent to minimizing $L_{winsor}(\mathbf{W}, \mathbf{H})$. This algorithm is similar to the iterative reweighting algorithms used for robust regression. Each entry in $\mathbf{V}$ is assigned a weight given by the corresponding entry in $\mathbf{Z}$. We will alternate between learning the model parameters $\mathbf{W}$ and $\mathbf{H}$, and updating the weights in $\mathbf{Z}$.

If the weights are fixed at 1 (which will happen if λ is sufficiently large), the algorithm transforms into traditional NMF. If the weights are fixed but not all equal to 1, then algorithm transforms into weighted NMF. A somewhat similar approach is used by Kong's algorithm for Absolute NMF (Algorithm 2.5), but instead of using the weights to transform the Frobenius Loss into the Absolute Loss, we are using the weights to deal with the contaminated entries. In this equation, $\odot$ represents elementwise multiplication.

$$\mathbf{W} \leftarrow \mathbf{W} \odot \frac{(\mathbf{V} \odot \mathbf{Z})\mathbf{H}^T}{(\mathbf{WH} \odot \mathbf{Z})\mathbf{H}^T} \qquad \mathbf{H} \leftarrow \mathbf{H} \odot \frac{\mathbf{W}^T(\mathbf{V} \odot \mathbf{Z})}{\mathbf{W}^T(\mathbf{WH} \odot \mathbf{Z})} \tag{3.5}$$

$$\mathbf{Z}_{ij} \leftarrow \begin{cases} 0 & \text{if } (\mathbf{V} - \mathbf{WH})_{ij} < -\lambda \\ 1 & \text{if } -\lambda \leq (\mathbf{V} - \mathbf{WH})_{ij} < \lambda \\ 0 & \text{if } (\mathbf{V} - \mathbf{WH})_{ij} > \lambda \end{cases} \tag{3.6}$$

Algorithm 3.10 W-Robust NMF

1: Initialize $\mathbf{Z} = 1$
2: **for** $t = 1, 2, 3 \ldots$ **do**
3: Update each entry of $\mathbf{W}$ using Equation 3.5
4: Update each entry of $\mathbf{H}$ using Equation 3.5
5: Update each entry of $\mathbf{Z}$ using Equation 3.6
6: **end for**

3.5.1 Optimization: Non-Integral Weights

Iterative reweighted regression when the weights are updated as in Equation 3.6 is also called trimmed regression, as the contaminated points are 'trimmed' out of the dataset by setting $\mathbf{Z} = 0$. However, Algorithm 3.10 can handle float-valued weights without issue. Allowing non-integral $\mathbf{Z}$ weights improves the stability of the algorithm: instead of a sharp jump from $\mathbf{Z} = 0$ to $\mathbf{Z} = 1$, the algorithm is allowed to use float values in between. This allows the algorithm to represent 'uncertainty' by setting $\mathbf{Z} \approx 0.5$.

The easiest way to implement nonintegral weights is to limit the amount $\mathbf{Z}$ can

shift in a given iteration. This reduces the impact of bad initial conditions, at the cost of requiring more iterations until convergence is achieved. Note that the algorithm may eventually set $\mathbf{Z} = 0$ and mark an element as fully contaminated, but it will spend some amount of time considering the possibilities before doing so. Equation 3.6 is replaced with the following, where 0.02 can be any small number.

$$\mathbf{Z}_{ij} \leftarrow \begin{cases} \max(\mathbf{Z}_{ij} - 0.02, 0) & \text{if } (\mathbf{V} - \mathbf{WH})_{ij} < -\lambda \\ \min(\mathbf{Z}_{ij} + 0.02, 1) & \text{if } -\lambda \leq (\mathbf{V} - \mathbf{WH})_{ij} < \lambda \\ \max(\mathbf{Z}_{ij} - 0.02, 0) & \text{if } (\mathbf{V} - \mathbf{WH})_{ij} > \lambda \end{cases} \tag{3.7}$$

3.6 Experiments

We now run a series of experiments to evaluate the performance of both the M-robust and W-robust algorithms (using all optimizations described above).

3.6.1 Evaluation Metrics

There are several popular error metrics used when evaluated the quality of a factorization on noisy data. We discuss three such metrics below. These error metrics are not equivalent - an algorithm could hypothetically perform well on one metric and poorly on another. $\mathbf{V}_{true}$ denotes the true (uncontaminated) data and $\mathbf{V}_{contam}$ denotes the contaminated data. In a non-experimental setting, a user would only have access to

$\mathbf{V}_{contam}$.

Factorization Error NMF takes $\mathbf{V}_{contam}$ as input and tries to optimize $\phi(\mathbf{V}_{contam} - \mathbf{WH})$. Thus, a common evaluation metric is to test how well it performed the factorization on the noncontaminated entries. Define $\mathbf{B}_{ij} = 1$ if $\mathbf{V}_{ij}$ is non-contaminated; note $\sum \mathbf{B}_{ij}$ is the number of noncontaminted entries in the matrix. We then define

$$
\begin{aligned}
ERR_{fs} &= \frac{\sum_{ij} \mathbf{B}_{ij} * (\mathbf{V}_{contam} - \mathbf{WH})_{ij}^2}{\sum_{ij} \mathbf{B}_{ij}} \\
ERR_{11} &= \frac{\sum_{ij} \mathbf{B}_{ij} * |\mathbf{V}_{contam} - \mathbf{WH}|_{ij}}{\sum_{ij} \mathbf{B}_{ij}} \\
ERR_{12} &= \frac{\sqrt{\sum_j \left(\sum_i \mathbf{B}_{ij} * |\mathbf{V}_{contam} - \mathbf{WH}|_{ij}\right)^2}}{\sum_{ij} \mathbf{B}_{ij}} \\
ERR_{21} &= \frac{\sum_j \sqrt{\sum_i \mathbf{B}_{ij} * (\mathbf{V}_{contam} - \mathbf{WH})_{ij}^2}}{\sum_{ij} \mathbf{B}_{ij}} \\
ERR_{\infty} &= \max(\mathbf{B}_{ij} * |\mathbf{V}_{contam} - \mathbf{WH}|_{ij})
\end{aligned}
$$

ERR_{fs} is interpreted as the average Frobenius-squared norm on the noncontaminated entries. The other error measures have the same interpretation for the L_{11} norm, the L_{12} norm, the L_{21} norm, and the L_{∞} norm respectively.

Reconstruction Error An alternative error metric is the reconstruction error, which specifically measures how well the algorithm can estimate the contaminated

entries (i.e. reconstructing the correct values). Note this metric can only be evaluated on datasets where we know the ground truth $\mathbf{V}_{true}$. We again define $\mathbf{B}_{ij} = 1$ if $\mathbf{V}_{ij}$ is non-contaminated.

$$REC_{fs} = \frac{\sum_{ij}(1 - \mathbf{B}_{ij}) * (\mathbf{V} - \mathbf{WH})^2_{ij}}{\sum_{ij}(1 - \mathbf{B}_{ij})}$$

REC_{11}, REC_{21}, REC_{12}, and REC_{∞} are all defined similarly by taking the respective ERR function, replacing $\mathbf{B}_{ij}$ with $1 - \mathbf{B}_{ij}$ and $\mathbf{V}_{contam}$ with $\mathbf{V}_{true}$.

Precision / Recall A third metric we consider is identifying which elements are contaminated instead of trying to minimize the error. We compare the elements the algorithm marks as contaminated against the true location of the contamination, and compute precision, recall, and F_1 score. Note that Traditional NMF and other nonrobust algorithms do not mark any entries as contaminated, so instead we compute $(\mathbf{V} - \mathbf{WH})^2$ and choose the entries with the highest error as contaminated.

Comparisons: We compare against Traditional NMF, Absolute NMF, and the L_{21} loss algorithms (all discussed in Chapter 2). We also compare against Lin's [29] algorithm to minimize the Frobenius loss (discussed in Chapter 4). Shen's algorithm to minimize L_{robust} (this is not a multiplicative update algorithm, but is minimizing the same function, so we include it as a comparison point), and Kannan's algorithm to minimize $\sum(\mathbf{V}_{ij} - \mathbf{WH}_{ij} - \mathbf{S}^2_{ij} + \lambda|\mathbf{S}|_{12}$. All algorithms are given the same random initializations of $\mathbf{W}$ and $\mathbf{H}$ and given the same computational resources. To ensure

Table 3.1: Factorization Error on Synthetic Data

Alg	ERR_{fs}	ERR_{11}	ERR_{12}	ERR_{21}	ERR_{∞}
Traditional [28]	0.285	0.407	0.012	0.017	6.134
Lin [29]	0.285	0.407	0.012	0.017	6.330
Absolute [24]	0.283	0.406	0.012	0.017	6.148
L21 Loss [24]	0.284	0.407	0.012	0.017	6.176
Kannan et al [21]	0.224	0.394	0.012	0.015	2.797
Shen et al [41]	0.278	0.402	0.012	0.017	6.197
M-Robust	0.021	0.117	0.003	0.004	6.197
W-Robust	0.017	0.104	0.003	0.004	0.916

Table 3.2: Reconstruction Error on Synthetic Data

Alg	REC_{fs}	REC_{11}	REC_{12}	REC_{21}	REC_{∞}
Traditional [28]	1.880	1.195	0.039	0.159	9.015
Lin [29]	1.888	1.195	0.039	0.159	9.158
Absolute [24]	1.910	1.194	0.039	0.014	9.588
L21 Loss [24]	1.907	1.194	0.039	0.159	9.588
Kannan et al [21]	0.800	0.766	0. 248	0.105	4.348
Shen et al [41]	1.867	1.184	0.309	0.159	9.076
M-Robust	0.146	0.292	0.009	0.042	5.139
W-Robust	0.037	0.157	0.005	0.022	5.881

Table 3.3: Precision and Recall on Synthetic Data

Alg	Precision	Recall	$F1$
All algorithms achieved > 0.99 precision and recall			

a fair comparison, all algorithms are reimplemented in Python from their respective papers and are given sufficient time to converge.

3.6.2 Experimental Results

Synthetic: We generate a 1000×80 matrix $\mathbf{W}_{true}$ and a 80×1000 matrix $\mathbf{H}_{true}$. 25% of the entries of $\mathbf{W}_{true}$ and $\mathbf{H}_{true}$ are set to 1 - the remaining entries are set to 0. We set $\mathbf{V}true = \mathbf{W}_{true}\mathbf{H}_{true}$. We then create $\mathbf{V}_{contam}$ by randomly selecting 7% of the elements of $\mathbf{V}_{true}$, and adding 5 to those indices.

Faces: The ORL Database of Faces [40] is a database of image files that has been used in several NMF studies [26, 17]. An image file is represented by a 112×92 matrix which we rescale from $[0, 255]$ grayscale values to $[0, 1]$ decimal values. To generate $\mathbf{V}_{contam}$, we randomly choose 8% of the entries. Chosen entries with values less than 0.5 are set to 1, chosen entries with values greater than 0.5 are set to 0.

Text: 20Newsgroups is a collection of news documents that is commonly used for text experiments. Using sci-kit learn, we load 500 documents about computers (documents that have been assigned to topic 'comp.*'). We strip off headers, footers, and quotations. We then select the 500 most common words, excluding words that appear in more than 95% of the documents and common English stopwords (the, and, or, etc.). For each document and each word, we count the number of times that word appears in that document. The result is a 500×500 matrix of nonnegative integer values. To generate $\mathbf{V}_{contam}$, we choose 8% of the entries with $\mathbf{V} = 0$ and add 1 to those entries. This represent adding random words to documents.

Summary: On almost all experiments and metrics, the M-Robust and W-Robust algorithms outperformed alternative approaches.

Table 3.4: Factorization Error on Face Data

Alg	ERR_{fs}	ERR_{11}	ERR_{12}	ERR_{21}	ERR_{∞}
Traditional [28]	0.012	0.071	0.007	0.013	0.923
Lin [29]	0.012	0.071	0.007	0.013	0.851
Absolute [24]	0.009	0.055	0.007	0.011	0.739
L21 Loss [24]	0.010	0.062	0.007	0.009	0.838
Kannan et al [21]	0.020	0.082	0.008	0.013	1.935
Shen et al [41]	0.010	0.062	0.006	0.009	0.889
M-Robust	0.007	0.050	0.004	0.008	1.140
W-Robust	0.005	0.045	0.02	0.002	0.042

Table 3.5: Reconstruction Error on Face Data

Alg	REC_{fs}	REC_{11}	REC_{12}	REC_{21}	REC_{∞}
Traditional [28]	0.121	0.264	0.036	0.095	1.351
Lin [29]	0.121	0.254	0.034	0.092	1.215
Absolute [24]	0.135	0.225	0.036	0.080	0.999
L21 Loss [24]	0.124	0.224	0.034	0.081	1.070
Kannan et al [21]	0.106	0.259	0.033	0.105	1.764
Shen et al [41]	0.127	0.253	0.035	0.096	1.652
M-Robust	0.119	0.228	0.013	0.079	1.698
W-Robust	0.117	0.200	0.004	0.069	1.345

Table 3.6: Precision and Recall on Face Data

Alg	Precision	Recall	$F1$
Traditional [28]	0.759	0.940	0.843
Lin [29]	0.758	0.947	0.843
Absolute [24]	0.731	0.913	0.812
L21 Loss [24]	0.755	0.944	0.839
Kannan et al [21]	0.763	0.953	0.848
Shen et al [41]	0.772	0.964	0.857
M-Robust	0.849	0.997	0.917
W-Robust	0.800	0.998	0.889

Table 3.7: Factorization Error on Text Data

Alg	ERR_{fs}	ERR_{11}	ERR_{12}	ERR_{21}	ERR_{∞}
Traditional [28]	0.133	0.269	0.012	0.016	5.816
Lin [29]	0.133	0.267	0.012	0.016	6.000
Absolute [24]	0.132	0.267	0.012	0.016	5.834
L21 Loss [24]	0.132	0.267	0.012	0.016	5.815
Kannan et al [21]	0.139	0.273	0.013	0.017	5.931
Shen et al [41]	0.132	0.267	0.012	0.016	5.815
M-Robust	0.007	0.050	0.004	0.008	5.140
W-Robust	0.105	0.168	0.008	0.013	5.086

Table 3.8: Reconstruction Error on Face Data

Alg	REC_{fs}	REC_{11}	REC_{12}	REC_{21}	REC_{∞}
Traditional [28]	0.164	0.303	0.014	0.063	5.766
Lin [29]	0.160	0.300	0.014	0.062	5.541
Absolute [24]	0.112	0.225	0.036	0.080	5.999
L21 Loss [24]	0.106	0.224	0.034	0.081	5.070
Kannan et al [21]	0.165	0.305	0.014	0.064	5.600
Shen et al [41]	0.160	0.297	0.014	0.062	5.764
M-Robust	0.105	0.175	0.010	0.042	5.215
W-Robust	0.063	0.056	0.003	0.023	6.503

Table 3.9: Precision and Recall on Face Data

Alg	Precision	Recall	$F1$
Traditional [28]	0.815	0.846	0.820
Lin [29]	0.813	0.844	0.828
Absolute	0.815	0.845	0.830
L21 Loss [24]	0.815	0.846	0.830
Kannan et al [21]	0.790	0.820	0.805
Shen et al [41]	0.816	0.850	0.832
M-Robust	0.836	0.871	0.854
W-Robust	0.841	0.873	0.857

Chapter 4

Convergence

In this chapter, we prove that multiplicative update algorithms, including Traditional NMF, M-robust NMF, and W-robust NMF, converge to a KKT optimality point of a perturbed error function as long as 0/0 is treated as 0. Section 4.1 defines the Clarke derivative, Section 4.2 discusses existing convergence results and why a standard derivative proof of convergence does not hold. Section 4.3 proves convergence of Algorithm 2.1 using the Clarke derivative and Section 4.4 proves the convergence of Algorithm 3.6 and Algorithm 3.10

4.1 Matrix Derivatives

Let L be a function that maps one or more matrices to a scalar. The Frobenius loss, robust loss, weighted loss, huber loss, and truncated loss are all examples of such scalar functions. The definitions of derivative and partial derivative of L extend

naturally from their standard calculus definitions.

Definition 4.1 (Matrix Derivative)**.** Let $L(\mathbf{W}, \mathbf{H})$ be a differentiable scalar function. The matrix derivative of L is denoted ∇L and is a $(\mathbf{M}_w, \mathbf{M}_h)$ pair, where the (i, a)-th element of $\mathbf{M}_w$ is the partial derivative of L with respect to $\mathbf{W}_{ia}$ and the (a, j)-th element of $\mathbf{M}_h$ is the partial derivative of L with respect to $\mathbf{H}_{aj}$

Definition 4.2 (Partial Matrix Derivative)**.** Let $L(\mathbf{W}, \mathbf{H})$ be a differentiable scalar function. The matrix derivative of L with respect to $\mathbf{W}_{ia}$ is denoted $\nabla_w L_{ia}$ and is a scalar value equal to the partial derivative of L with respect to $\mathbf{W}_{ia}$. The partial derivative of L with respect to $\mathbf{H}_{aj}$ is denoted $\nabla_h L_{aj}$ and defined similarly.

However, if L is not differentiable, these quantities might not be well defined. The Clarke derivative is a generalization of the regular derivative for nonconvex and nonsmooth functions where the regular derivative does not exist. See [38] for a more comprehensive overview of the Clarke derivative.

Definition 4.3. A function L is locally Lipschitz at a point θ if there exists $\epsilon > 0$ and a $\Gamma > 0$ such that $|L(\theta) - L(\theta')| \leq \Gamma|\theta - \theta'|$ for all x' where $|\theta - \theta'| < \epsilon$.

Intuitively, L is locally Lipschiz at θ means that in the neighborhood around θ, L remains in a bounded neighborhood around $L(x)$. All of the loss functions described in the previous chapters are locally Lipschitz.

Definition 4.4. The Clarke directional derivative of a locally Lipschitz function L at a point θ in the direction of $\vec{d}$ is $L^\circ(x; \vec{d}) = \limsup_{\theta' \to \theta, t \downarrow 0} \left[L(\theta' + t\vec{d}) - L(\theta') \right] / t$

Definition 4.5. The Clarke generalized subderivative of L at a point θ is the set

$$\widetilde{\nabla} L(\theta) = \{\vec{\zeta} : L^{\circ}(\theta; \vec{d}) \geq \vec{\zeta} \bullet \vec{d} \ \forall \vec{d}\}$$

where $\vec{\zeta} \bullet \vec{d}$ is inner product. Elements of $\widetilde{\nabla} L$ are called Clarke subgradients.

The standard notion of the derivative ∇f assumes a derivative that is continuous, has a single value everywhere, and takes 0 at the minimum point. The Clarke derivative allows for nonsmooth functions and is set-valued. When L is a smooth function and the regular derivative exists, $\widetilde{\nabla} L$ is a singleton set that contains the derivative. If L is a convex function and the regular subderivative exists, $\widetilde{\nabla} L$ is set that contains the subderivative. L is said to be Clarke regular at these locations.

Corollary 4.6 ([38])**.** *If L is locally Lipschitz at θ, $\widetilde{\nabla} L(x)$ is nonempty. If L is smooth and differentiable at θ, then $\widetilde{\nabla} L(\theta) = \{L'(\theta)\}$. If θ is a local minima, $0 \in \widetilde{\nabla} L(\theta)$. Analogues to the chain rule, product rule, and sum rule all apply to subdifferential calculus using the Clarke derivative.*

We now introduce a bit of notation that will help prove the convergence results.

Definition 4.7 (Clarke matrix derivative)**.** Let $L(\mathbf{W}, \mathbf{H})$ be a locally Lipschitz scalar function. The Clarke matrix derivative of L with respect to $\mathbf{W}_{ia}$ is

$$\widetilde{\nabla}_w L_{ia} \quad = \quad \{w : (\mathbf{M}_w, \mathbf{M}_h) \in \widetilde{\nabla} L(\mathbf{W}, \mathbf{H}) \text{ entry } (i, a) \text{ of } \mathbf{M}_w \text{ equals } w\}$$

and define $\widetilde{\nabla}_h F_{aj}$ similarly.

When meaning is obvious and the Clarke matrix derivative is a singleton set, we will sometimes use equality as a shorthand: $u = \widetilde{\nabla}_w L_{ia}$ is shorthand for setting u to be equal to the single element in $\widetilde{\nabla}_w L_{ia}$.

Note that while Definition 4.2 and 4.7 may appear similar, they are not identical - there is no such thing as a Clarke partial derivative.

4.1.1 KKT Conditions

The Karush Kuhn Tucker (KKT) are a set of conditions are necessary conditions for a point θ to be an optimal point of a function L under certain constraints (for example, the nonnegativity constraint of NMF).

Definition 4.8 (KKT Conditions)**.** Given a function $f(\theta)$ to minimize and a set of inequality constraints $C_r(\theta) \leq 0$, θ satisfies the KKT optimality conditions if for each inequality constraint C_r, there exists a corresponding $u_r \in \mathbb{R}$ such that the following three conditions are satisfied

- $0 \in \widetilde{\nabla}_\varphi \overline{L}(\theta) + \sum_r u_r \widetilde{\nabla}_\varphi C_i(\theta)$ for each variable φ in θ
- $u_r C_r(\theta) = 0$, $u_r \geq 0$
- $C_r(\theta) \leq 0$

A point is called a KKT optimality point if it satisfies the KKT optimality conditions described above. When inequality constraint C_r is slack, i.e. $C_r(\theta) < 0$, then the corresponding $u_r = 0$ and we expect $\widetilde{\nabla}_r f(\theta) = 0$. When constraint C_r is tight, i.e.

$C_r(x) = 0$, then the corresponding $u_r > 0$. Intuitively, for a point θ to be optimal, it should either be in the interior of the constrained space with a zero derivative, or it should be on the boundary of the constrained space and the derivative is pointing 'outwards'.

4.2 Convergence and Underflow

The traditional Lee and Seung multiplicative update steps in Algorithm 2.1 are not well defined when $\mathbf{WHH}^T = 0$ or $\mathbf{W}^T\mathbf{WH} = 0$ in some entry. There are two major implications of this problem. The first issue is that the Lee and Seung algorithm is not actually guaranteed to converge to a KKT point. If the limit point is on the boundary where $\mathbf{WHH}^T = 0$, the update rule is not actually defined and thus the limit point cannot be declared a KKT optimality point. Lin [29] offers a modified algorithm that is theoretically guaranteed to converge to a KKT optimality point. Essentially, Lin adds a $+\epsilon$ term into the denominator. However, he needs to replace $\mathbf{W}$ and $\mathbf{H}$ with $\mathbf{W}^\circ$ and $\mathbf{H}^\circ$ to maintain the monotonic guarantee and ensure convergence.

Algorithm 4.11 Lin NMF

1: Randomly Initialize $\mathbf{W}$ and $\mathbf{H}$
2: **for** $t = 1, 2, 3 \ldots$ **do**
3: $\quad \mathbf{W}^\circ_{ia} = \mathbf{W}_{ia}$ if $\nabla_w(L_{fro})_{ia} > 0$ else $\mathbf{W}^\circ_{ia} = \max(\mathbf{W}_{ia}, \epsilon)$
4: $\quad \mathbf{W} \leftarrow \mathbf{W} - \dfrac{\mathbf{W}^\circ}{\mathbf{W}^\circ\mathbf{H}\mathbf{H}^T + \epsilon} \odot \nabla_w(L_{fro})_{ia}$
5: $\quad \mathbf{H}^\circ_{aj} = \mathbf{H}_{aj}$ if $\nabla_h(L_{fro})_{aj} > 0$ else $\mathbf{H}^\circ_{aj} = \max(\mathbf{H}_{aj}, \epsilon)$
6: $\quad \mathbf{H} \leftarrow \mathbf{H} - \dfrac{\mathbf{H}^\circ}{\mathbf{W}^T\mathbf{W}\mathbf{H}^\circ + \epsilon} \nabla_h(L_{fro})_{aj}$
7: **end for**

Table 4.1: Approaches to Underflow

Approach	Formal Update Step	Implementation
Treat $0/0 = 0$	$\mathbf{W} \odot \frac{\mathbf{VH}^T}{\max(\epsilon, \mathbf{WHH}^T)}$	Scikit-Learn
Treat $0/0 = 1$	$\mathbf{W} \odot \frac{\max(\epsilon, \mathbf{VH}^T)}{\max(\epsilon, \mathbf{WHH}^T)}$	Nimfa
Force $\mathbf{W}, \mathbf{H} > \epsilon$	$\max\left(\epsilon, \mathbf{W} \odot \frac{\mathbf{VH}^T}{\mathbf{WHH}^T}\right)$	MatLab
Throw an error	Divide-by-zero Error	CloudNMF
Lin's Rule	See Algorithm 4.11	None

Putting aside the convergence issue, the second problem with Standard NMF is the divide-by-zero problem. If $\mathbf{W}$ and $\mathbf{H}$ are initialized to be nonzero positive numbers, and in a machine with infinite decimal precision, the algorithm will never encounter a divide by zero issue. On a real-world machine where decimal underflow is a practical concern, some approach needs to be taken to handle the case where a small number underflows into 0/0. There is no uniform approach to handling decimal underflow; Lee and Seung's original paper does not discuss the issue, and as a result different implementations have made different decisions.

One potential solution solution is to replace any denominator less than ϵ as ϵ. This is the approached used by Scikit-Learn, which corresponds to the idea of treating 0/0 as 0. An alternate solution is to replace any numerator or denominator less than ϵ with ϵ. This is the approach used by the Nimfa library. A third solution to is replace any value in $\mathbf{W}$ or $\mathbf{H}$ that is less than ϵ with ϵ. This is the approach taken by MatLab. A fourth solution is to throw an error when decimal underflow occurs, which is the approach the bio-medicine-focused library CloudNMF takes. No major

matrix library has chosen to implement Lin's approach. For the reasons discussed in Section 2.2.1, the standard multiplicative rules are fast to compute, while replacing $\mathbf{W}$ with $\mathbf{W}^{\circ}$ is not. We summarize the various approaches in Table 4.1. None of these libraries offers a particular justification for the decision they make about how to handle decimal underflow.

4.3 Convergence of Traditional NMF

In this section, we describe a new result for the convergence of Traditional NMF, along with a justification for the correct way to handle underflow. Under Definition 4.2 for the matrix partial derivative, the Lee and Seung update rules can be written as

$$\mathbf{W}_{ia} \leftarrow \mathbf{W}_{ia} - \frac{\mathbf{W}_{ia}}{[\mathbf{W}\mathbf{H}\mathbf{H}^T]_{ia}} \odot \nabla_w [L_{fro}]_{ia} \tag{4.1}$$

$$\mathbf{H}_{aj} \leftarrow \mathbf{H}_{aj} - \frac{\mathbf{H}_{aj}}{[\mathbf{W}^T\mathbf{W}\mathbf{H}]_{aj}} \odot \nabla_h [L_{fro}]_{aj} \tag{4.2}$$

Thus, the Lee and Seung can be viewed as a variation of gradient descent using a nonstandard step size, specifically targeted at L_{fro}. We can view the Lee and Seung update steps as a special case of the following updates.

$$\mathbf{W}_{ia} \leftarrow \mathbf{W}_{ia} - \mathbf{X}_w \odot \nabla_w L_{ia} \tag{4.3}$$

$$\mathbf{H}_{aj} \leftarrow \mathbf{H}_{aj} - \mathbf{X}_h \odot \nabla_h L_{aj}$$

where $\mathbf{X}_w, \mathbf{X}_h > 0$ as long as $\mathbf{W}, \mathbf{H} > 0$

but $\mathbf{X}_w, \mathbf{X}_h$ may not be well-defined when $\mathbf{W}, \mathbf{H} = 0$

We now take inspiration from Lagrangian optimization techniques and modify the loss function to encode the nonnegativity restriction.

Definition 4.9. For any loss function $L(\mathbf{W}, \mathbf{H})$, the perturbed loss function $\overline{L}(\mathbf{W}, \mathbf{H})$ is defined as $\overline{L}(W, \mathbf{H}) = L\left(\max(\mathbf{W}, 0), \max(\mathbf{H}, 0)\right)$, where max() is taken elementwise.

Note that the original loss function and the perturbed loss function agree on all values where $\mathbf{W}, \mathbf{H} \geq 0$. However, the perturbed loss function 'cuts off' at 0. Alternatively, the perturbed loss applies a penalty for assigning negative values to $\mathbf{W}$ and $\mathbf{H}$, and this penalty is exactly equal to the benefit gained by using a negative value.

Theorem 4.10. *Let L is a Locally Lipschitz function (not necessarily differentiable) and that the the step sizes in the generic multiplicative update rules (Equation 4.3) are chosen so that L is non-increasing. Then those rules are guaranteed to a KKT optimality point of $\overline{L}$ as long $\mathbf{X}_w, \mathbf{X}_h > 0$ when $\mathbf{W}, \mathbf{H} > 0$ and $\mathbf{X}_w, \mathbf{X}_h = 0$ when $\mathbf{W}, \mathbf{H} = 0$.*

Corollary 4.11. *The Lee and Seung multiplicative update rules converge to a KKT optimality point of* $\overline{L}_{fro}$ *as long as* $0/0$ *is replaced with* 0.

Proof. Let $\theta = (\mathbf{W}, \mathbf{H})$ be the limit point of the generic NMF multiplicative algorithm. First, we need to convert the nonnegativity constraint into a inequality constraint. This is easily done: for each $\mathbf{W}_{ij}$, define $C_{ijw}(\theta) = -\mathbf{W}_{ij}$ and for each $\mathbf{H}_{ij}$, define $C_{ijh}(\theta) = -\mathbf{H}_{ij}$. For each constraint, set $U_{ijw} = U_{ihj} = 0$. Then θ lies within the constrained domain as long as all of the C-constraints are nonpositive and conditions two and three from Definition 4.8 are satisfied.

Suppose $\mathbf{W}_{ia} > 0$. Recall from Equation 4.3 that the $\mathbf{W}$ update step can be written as $\mathbf{W} \leftarrow \mathbf{W} - \mathbf{X}_w \odot \nabla L(\theta)$. If $\mathbf{W}_{ia} > 0$, then $[\mathbf{X}_w]_{ia} > 0$. If convergence has been achieved, then it must be the case that $\nabla L_{ia} = 0$. Since $\nabla_w L_{ia} \subseteq \widetilde{\nabla}_w L_{ia}$, we immediately have that $0 \in \widetilde{\nabla}_w L_{ia}$.

Suppose $\mathbf{W}_{ia} = 0$. We can no longer rely on the above argument. Instead, we note that by Corollary 4.6 (specifically the Chain Rule) that $\widetilde{\nabla}_w \overline{L}_{ia}(\theta) = \nabla_w L_{ia}(\theta) * \widetilde{\nabla}_w max(\mathbf{W}_{ia}, 0)$ and that

$$\widetilde{\nabla}_w \max(\mathbf{W}_{ia}, 0) = \begin{cases} 1 & \mathbf{W}_{ia} > 0 \\ [0,1] & \mathbf{W}_{ia} = 0 \\ 0 & \mathbf{W}_{ia} < 0 \end{cases}$$

It follows that $0 \in \widetilde{\nabla}_w \overline{L}_{ia}(\theta)$ and the first condition on Definition 4.8 are satisfied.

$0 \in \widetilde{\nabla}_h \overline{L}_{aj}(\theta)$ follows by similar logic. $\square$

4.4 Convergence of Robust NMF

Theorem 4.12. *Algorithm 3.6 converges to a KKT optimality point of* $\overline{L}_{robust}$ *under the Bounded* $\mathbf{S}$*-Contamination constraint* $(\mathbf{S} \leq \mathbf{V})$ *, where*

$\overline{L}_{robust}(\mathbf{W}, \mathbf{H}, \mathbf{S}) = L_{robust}(\max(\mathbf{W}, 0), \max(\mathbf{H}, 0), \mathbf{S})$

Proof. The algorithm is monotonic in $\overline{L}_{robust}$ for the $\mathbf{W}$-update steps and $\mathbf{H}$-update steps by the same argument as the standard multiplicative update algorithm. For the $\mathbf{S}$ update step, we note that Equation 3.3 sets $\mathbf{S}_{ij}$ to $\text{argmin}_{\mathbf{S}}\, L_{robust}(\mathbf{W}, \mathbf{H}, \mathbf{S})$, which is guaranteed to decrease L_{robust}. This proves monotonicity.

To prove convergence, let $\theta = (\mathbf{W}, \mathbf{H}, \mathbf{S})$ be the limit point of the algorithm, and for notational convenience define $\overline{L} = \overline{L}_{robust}$. As before, we need to convert the restrictions on $\mathbf{W}, \mathbf{H}, \mathbf{S}$ into inequality constraints. Define $C_{ijw}(\theta) = -\mathbf{W}_{ij}$ and $C_{ijh}(\theta) = -\mathbf{H}_{ij}$ as before. Define $C_{ijs}(\theta) = \mathbf{S}_{ij} - \mathbf{V}_{ij} - \lambda$. The algorithm enforces the nonnegative restriction on $\mathbf{W}, \mathbf{H} \geq 0$ and the Bounded $\mathbf{S}$-contamination restriction on $\mathbf{S}$. $0 \in \widetilde{\nabla}_w \overline{L}_{ia}$ and $0 \in \widetilde{\nabla}_h \overline{L}_{aj}$ for the same logic as in Theorem 4.10. Thus, we need to prove that there exists U_{ijs} such

$$0 \in \widetilde{\nabla}_s \overline{L}_{ij}(\theta) + U_{ijs} \widetilde{\nabla}_s C_{ijs}(\theta) \text{ and } U_{ijs} C_{ijs}(\theta) = 0 \text{ and } U_{ijs} \geq 0 \tag{4.4}$$

If $\mathbf{S}_{ij} < \mathbf{V}_{ij}$, then $\mathbf{S}_{ij}$ is is the argmin of $\overline{L}_{robust}$, so $0 \in \widetilde{\nabla}_s \overline{L}_{ij}$ and Equation 4.4

is satisfied for $U_{ijs} = 0$. If $\mathbf{S}_{ij} = \mathbf{V}_{ij}$, then setting $U_{ijz} = -\widetilde{\nabla}_s \overline{L}_{ij}(\theta)$ ensures the first requirement is satisfied. The fact that $C_{ijs} = 0$ when $\mathbf{S}_{ij} = \mathbf{V}_{ij}$ ensures the second requirement is satisfied. For the third requirement, note that

$$-(\mathbf{V}_{ij} - \mathbf{W}\mathbf{H}_{ij} - \mathbf{S}_{ij}) + \lambda \in \widetilde{\nabla}_s \overline{L}_{ij}(\theta) \tag{4.5}$$

By Equation 3.3, we have that $\mathbf{S}_{ij} = \mathbf{V}_{ij}$ iff $\mathbf{V}_{ij} - \mathbf{W}\mathbf{H}_{ij} \geq \mathbf{V}_{ij} + \lambda$. Therefore, we have that

$$-(\mathbf{V}_{ij} - \mathbf{W}\mathbf{H}_{ij} - \mathbf{S}_{ij}) + \lambda \leq -(\mathbf{V}_{ij} + \lambda - \mathbf{V}_{ij}) + \lambda \leq 0$$

and so $U_{ijs} = -\widetilde{\nabla}_s \overline{L}_{ij}(\theta) \geq 0$. Thus, for all values of $\mathbf{S}$ Equation 4.4 is satisfied and θ is a KKT optimality point.

□

Corollary 4.13. *Algorithm 3.6 converges to a KKT optimality point of* $\overline{L}_{robust}(\mathbf{W}, \mathbf{H}, \mathbf{S})$ *under the Nonnegative* $\mathbf{S}$*-Contamination constraint*

Proof. We add the additional inequality constraint that $C_{ijs2}(\theta) = -\mathbf{S}_{ij}$. We need to prove Equation 4.4 hold for C_{ijs2} and U_{ijs2}. If $\mathbf{S}_{ij} > 0$, set $U_{ijs2} = 0$ and the same argument as the previous theorem holds. If $\mathbf{S}_{ij} = 0$, then set $U_{ijs2} = \widetilde{\nabla}_s \overline{L}_{ij}(\theta)$ ensures the first requirement is satisfied. The fact that $C_{ijs2} = 0$ when $\mathbf{S}_{ij} = 0$ ensures the second requirement is satisfied. Finally, if $\mathbf{S}_{ij} = 0$, then by Equation 3.3,

$\mathbf{V}_{ij} - \mathbf{WH}_{ij} \leq \lambda$. Therefore, again applying Equation 4.5, we have that

$$-(\mathbf{V}_{ij} - \mathbf{WH}_{ij} - \mathbf{S}_{ij}) + \lambda \geq -(\lambda + 0) + \lambda = 0$$

and so $U_{ijs2} = \widetilde{\nabla}_s \overline{L}_{ij}(\theta) \geq 0$. Thus, for all values of $\mathbf{S}$ Equation 4.4 is satisfied and θ is a KKT optimality point.

□

Corollary 4.14. *If the update steps are recurrent, then Algorithm 3.7 almost surely converges to a KKT point of* $\overline{L}_{robust}(\mathbf{W}, \mathbf{H}, \mathbf{S})$ *under either Bounded* $\mathbf{S}$*-contamination or Nonnegative* $\mathbf{S}$*-contamination.*

Proof. Let $\theta_t = (\mathbf{W}, \mathbf{H}, \mathbf{S})$ be the (random) values of $\mathbf{W}, \mathbf{H}, \mathbf{S}$ at iteration t. Observe that while θ_t are random variables, Algorithm 3.7 is monotone in $\overline{L}_{robust}$ with probability 1. This means that the θ_t converge almost surely to a limit point θ. The proof that this limit point satisfies the KKT optimality follows from identical logic as the above theorems. □

Theorem 4.15. *Algorithm 3.10 converges to a KKT optimality point of* $\overline{L}_{reweighted}$ *, where* $\overline{L}_{reweighted}(\mathbf{W}, \mathbf{H}, \mathbf{Z}) = L_{reweighted}(\max(\mathbf{W}, 0), \max(\mathbf{H}, 0), \mathbf{Z})$

Proof. The algorithm is monotonic in $\overline{L}_{reweighted}$ for the $\mathbf{W}$-update and $\mathbf{H}$-update steps by the same argument as the weighted multiplicative update algorithm. For the $\mathbf{Z}$ update step, we note that the algorithm always increases or decreases $\mathbf{Z}$ towards the direction that decreases $L_{reweighted}$. This proves monotonicity.

To prove convergence, Let $\theta = (\mathbf{W}, \mathbf{H}, \mathbf{S})$ be the limit point of the algorithm, and for notational convenience define $\overline{L} = \overline{L}_{reweighted}$. Note that when convergence is achieved, either $\mathbf{Z}_{ij} = 0$ or $\mathbf{Z}_{ij} = 1$ for all $\mathbf{Z}$ values. Define $C_{ijw} = -\mathbf{W}_{ij}$, $C_{ijh} = -\mathbf{H}_{ij}$ as before. Define $\mathbf{Z}_{ijz} = \mathbf{Z}_{ij}(\mathbf{Z}_{ij} - 1)$. Observe $C_{ijz} \leq 0$ iff $0 \leq \mathbf{Z}_{ij} \leq 1$.

$0 \in \widetilde{\nabla}_w \overline{L}_{ia}$ and $0 \in \widetilde{\nabla}_h \overline{L}_{aj}$ for the same logic as in Theorem 4.10. Thus, we need to prove that there exists U_{ijz} such

$$0 \in \widetilde{\nabla}_z \overline{L}_{ij}(\theta) + U_{ijs} \widetilde{\nabla}_z C_{ijz}(\theta) \text{ and } U_{ijz} C_{ijz}(\theta) = 0 \text{ and } U_{ijz} \geq 0 \tag{4.6}$$

Taking the Clarke derivative, we have that

$$\tfrac{1}{2}(\mathbf{V}_{ij} - \mathbf{WH}_{ij})^2 - \tfrac{1}{2}\lambda^2 \in \widetilde{\nabla}_z \overline{L}_{ij} \text{ and } 2\mathbf{Z}_{ij} - 1 \in \widetilde{\nabla}_z C_{ijz}$$

If $\mathbf{Z}_{ij} = 0$, set $U_{ijz} = \widetilde{\nabla}_z \overline{L}_{ij}$. Note $\widetilde{\nabla}_z C_{ijz} = -1$ and $U_{ijz} \widetilde{\nabla}_z C_{ijz} = -\widetilde{\nabla}_z \overline{L}_{ijz}$, so the first requirement is satisfied. The fact that $C_{ijz} = 0$ when $\mathbf{Z}_{ij} = 0$ proves the second requirement. For the third requirement, Equation 3.6 proves $\mathbf{Z}_{ij} = 0$ iff $(\mathbf{V}_{ij} - \mathbf{WH}_{ij})^2 > \lambda^2$, so $U_{ijz} = \widetilde{\nabla}_z C_{ijz} > 0$ and the third requirement is satisfied.

If $\mathbf{Z}_{ij} = 1$, then set $U_{ijz} = -\widetilde{\nabla}_z \overline{L}_{ij}$. Note $\widetilde{\nabla}_z C_{ijz} = 1$ and $U_{ijz} \widetilde{\nabla}_z C_{ijz} = -\widetilde{\nabla}_z \overline{L}_{ijz}$, so the first requirement is satisfied. The fact that $C_{ijz} = 0$ when $\mathbf{Z}_{ij} = 1$ proves the second requirement. For the third requirement, Equation 3.6 proves $\mathbf{Z}_{ij} = 1$ iff $(\mathbf{V}_{ij} - \mathbf{WH}_{ij})^2 < \lambda^2$, so $U_{ijz} = -\widetilde{\nabla}_z C_{ijz} > 0$ and the third requirement is satisfied. $\square$

Chapter 5

Complex Contamination Models

Chapter 3 focused on the ϵ-contamination model. We assume the existence of a low-rank matrix $\mathbf{V}_{true}$. However, we only have access to $\mathbf{V}$, a noise sampling of $\mathbf{V}_{true}$. In the ϵ-contamination model, we assumed that mean-zero Gaussian noise affected all entries and that each entry was contaminated with independent probability ϵ. In Section 5.1, we describe this model probabilistically, and give a probabilistic interpretation of the λ value used in Robust and Weighted Loss.

The remainder of the chapter focuses on more complex contamination models. In Section 5.2, we consider the case where $\mathbf{V}$ contains missing elements, i.e. instead of a full sampling, we only receive a partial sampling. In Section 5.3, we consider the case where the noise is non-Gaussian, but the contamination is still independent. In Section 5.4, we consider the case where the noise is Gaussian, but the contamination is not independent.

5.1 Choosing Lambda

In Standard NMF, Lee and Seung assume each entry of $\mathbf{V}$ is generated from $\mathbf{V}_{true}$ plus independent and identically distributed Gaussian noise with standard deviation σ. Let $A_{ij}(\mathbf{W}, \mathbf{H})$ be the log likelihood of generating the observed $\mathbf{V}_{ij}$ from the estimations of $\mathbf{W}, \mathbf{H}$, under the assumption of Gaussian noise. In other words,

$$\begin{aligned} A_{ij}(\mathbf{W}, \mathbf{H}) &= \log \Pr[(\mathbf{V}_{true})_{ij} = \mathbf{V}_{ij} \mid \mathbf{W}, \mathbf{H}] \\ &= \log \Pr[Normal(\mathbf{V}_{ij}, \sigma) = \mathbf{WH}_{ij}] \\ &= a(\mathbf{V}_{ij} - \mathbf{WH}_{ij})^2 + b \\ &\text{where } a = -\frac{1}{2\sigma^2} \text{ and } b = -\frac{1}{2}\log(2\pi\sigma^2) \end{aligned} \tag{5.1}$$

Define $A(\mathbf{W}, \mathbf{H}) = \sum A_{ij}(\mathbf{W}, \mathbf{H})$, which is the log-likelihood of generating all of $\mathbf{V}$ under the assumption of Gaussian contamination. Minimizing the squared loss $L_{fro}(\mathbf{W}, \mathbf{H})$ is equivalent to maximizing $A(\mathbf{W}, \mathbf{H})$.

We now consider the ϵ-contamination model. Every entry independently flips an ϵ-coin with $0 < \epsilon < 1$. With probability $1 - \epsilon$, that entry is generated from $\mathbf{V}_{true}$ plus Gaussian noise - this is the good distribution. With probability ϵ, that entry is generated by a contaminated distribution unrelated to $\mathbf{V}_{true}$.

Set $\lambda = \left(-\frac{\log \epsilon + b}{a}\right)^{1/2}$. If $|\mathbf{V}_{ij} - \mathbf{WH}_{ij}| < \lambda$, than $A_{ij}(\mathbf{W}, \mathbf{H}) > \log \epsilon$ and it is more likely that the point was generated from the good distribution than the contamination distribution. If $|\mathbf{V}_{ij} - \mathbf{WH}_{ij}| > \lambda$, then $A_{ij}(\mathbf{W}, \mathbf{H}) < \log \epsilon$ and it

is more likely that the entry was generated from the contamination distribution. In other words, points are marked as clean or contaminated based on whether they are imperialistically more likely to be clean or contaminated. In this interpretation, λ can be interpreted as an assumption about the variance of the good distribution, plus the likelihood that a point is contaminated.

5.2 Missing Data

In the previous sections, we assumed that the user is given a full sampling of $\mathbf{V}$. This holds true even if $\mathbf{V}$ is sparse: a 0 element in $\mathbf{V}$ represents a 0 element in $\mathbf{V}_{true}$, Broadly speaking, it represents evidence of absence. For example, in the text experiment, a 0 in $\mathbf{V}$ represents the situation where a word does not appear in a given document. We have the full text of the document - the value 0 does not correspond to uncertainty about whether the word appears or not.

An alternative model is when we are only given a partial sampling of $\mathbf{V}$. In this model, $\mathbf{V}$ is allowed to have 'missing' elements, which we will denote using $\emptyset$. $\mathbf{V}$ may still be sparse, but the missing elements to represent absence of information. For example, in a recommendation system, a null value $\emptyset$ in $\mathbf{V}$ represents that a user has not assigned a movie a score. We do not know what score the user would have given - the $\emptyset$ value corresponds to uncertainty about the correct score. The weighted loss is easily modified to ignore missing values.

$$L_{weighted}(\mathbf{W},\mathbf{H},\mathbf{Z}) = \sum_{\substack{ij \\ \mathbf{V}_{ij}\neq\emptyset}} \mathbf{Z}_{ij} \odot \tfrac{1}{2}(\mathbf{V}_{ij} - \mathbf{WH}_{ij})^2 + (1-\mathbf{Z}_{ij}) \odot \tfrac{1}{2}\lambda^2$$

Let $\mathbf{M}$ be a $n \times m$ zero-one matrix where $\mathbf{M}_{ij} = 0$ if $\mathbf{V}_{ij} = \emptyset$ and $\mathbf{M}_{ij} = 1$ otherwise. Then the update rules becomes as follows.

$$\mathbf{W} \leftarrow \mathbf{W} \odot \frac{(\mathbf{V} \odot \mathbf{Z} \odot \mathbf{M})\mathbf{H}^T}{(\mathbf{WH} \odot \mathbf{Z} \odot \mathbf{M})\mathbf{H}^T} \qquad \mathbf{H} \leftarrow \mathbf{H} \odot \frac{\mathbf{W}^T(\mathbf{V} \odot \mathbf{Z} \odot \mathbf{M})}{\mathbf{W}^T(\mathbf{WH} \odot \mathbf{Z} \odot \mathbf{M})} \tag{5.2}$$

$$\mathbf{Z}_{ij} \leftarrow \begin{cases} 0 & \text{if } (\mathbf{V} - \mathbf{WH})_{ij} < -\lambda \\ 1 & \text{if } -\lambda \leq (\mathbf{V} - \mathbf{WH})_{ij} < \lambda \\ 0 & \text{if } (\mathbf{V} - \mathbf{WH})_{ij} > \lambda \end{cases} \tag{5.3}$$

Algorithm 5.12 W-Robust NMF with missing data

1: Initialize $\mathbf{Z} = 1$
2: **for** $t = 1, 2, 3 \ldots$ **do**
3: Update each entry of $\mathbf{W}$ using Equation 5.2
4: Update each entry of $\mathbf{H}$ using Equation 5.2
5: Update each entry of $\mathbf{Z}$ using Equation 5.3
6: **end for**

Note that the $\mathbf{Z}$-update rule is unchanged from the case with no missing data. However, since $\mathbf{Z} \odot \mathbf{M} = 0$ whenever $\mathbf{M} = 0$, it is not necessary to compute $\mathbf{Z}$ for the missing entries, which speeds up computation considerably. Additionally, the alternative $\mathbf{Z}$-update rules discussed in Section 3.5.1 can be applied here as well.

5.3 Non-Gaussian Distributions

The Gaussian distribution is nice because it is a reasonable model for several real world system and leads to clean update steps. However, the probabilistic ϵ-model can be extended beyond the Gaussian setting. Consider the setting where the good distribution is an arbitrary distribution from the exponential family instead of the Gaussian distribution, Every likelihood function of a distribution in the exponential family can be written in the following canonical form:

$$\begin{aligned} &\Pr[(\mathbf{V}_{true})_{ij} = \mathbf{V}_{ij} \mid \mathbf{W}, \mathbf{H}] \\ &= \nu(\mathbf{V}_{ij})\gamma(\mathbf{WH}_{ij}, \sigma_a) \exp\left(\sum_{c=1}^{C} \alpha_c(\mathbf{V}_{ij})\beta_c(\mathbf{WH}_{ij}, \sigma_a)\right) \\ &\text{where } \nu, \gamma, \alpha_c, \beta_c \text{ are fixed functions} \end{aligned} \tag{5.4}$$

Under this definition, the log likelihood $A_{ij}(\mathbf{W}, \mathbf{H})$ from Equation 5.1 can be replaced with the following function

$$A_{ij}(\mathbf{W}, \mathbf{H}) = \log \nu(\mathbf{V}_{ij}) + \log \gamma(\mathbf{WH}_{ij}, \sigma_a) + \sum_{c=1}^{C} \alpha_c(\mathbf{V}_{ij})\beta_c(\mathbf{WH}_{ij}, \sigma_a) \tag{5.5}$$

The matrix derivatives of A_{ij} have a nice closed form.

$$\nabla_w A_{ia} = (\Phi \mathbf{H}^T - \Psi \mathbf{H}^T)_{ia} \qquad \nabla_h A_{aj} = (\mathbf{W}^T \Phi - \mathbf{W}^T \Psi)_{aj} \tag{5.6}$$

$$\text{where } \Phi_{ij} = \sum_c \alpha_c(\mathbf{V}_{ij}) \beta_c'(\mathbf{W}\mathbf{H}_{ij}, \sigma_a) \text{ and } \Psi_{ij} = -\gamma'(\mathbf{W}\mathbf{H}_{ij})/\gamma(\mathbf{W}\mathbf{H}_{ij})$$

Cheung and Tresch [9] and Sra and Dillon [42] give a set of NMF multiplicative update rules to maximize A. If A is the Gaussian log likelihood function, $\Phi = \mathbf{V}$, $\Psi = \mathbf{WH}$, and these rules revert to the Lee and Seung's Traditional NMF rules.

We now extend $L_{weighted}(\mathbf{W}, \mathbf{H}, \mathbf{Z})$ to replace $\frac{1}{2}(\mathbf{V}_{ij} - \mathbf{W}\mathbf{H}_{ij})^2$ with A_{ij}. It is standard to minimize loss functions and maximize likelihood functions. To avoid confusion, we will define $L_{expweighted}$ as follows and maintain our establishing convention of minimizing functions (note Lemma 5.1 could also written as a maximization problem)

$$L_{expweighted}(\mathbf{W}, \mathbf{H}, \mathbf{Z}) = \sum_{ij} \mathbf{Z}_{ij} \odot -A_{ij}(\mathbf{W}, \mathbf{H}) + (1 - \mathbf{Z}_{ij}) \odot \frac{1}{2}\lambda^2 \tag{5.7}$$

Lemma 5.1. *Suppose $A_{ij}(\mathbf{W}, \mathbf{H}, \sigma_a)$ is a distribution from the exponential family, written as in Equation 5.4. Consider the update rules*

$$\mathbf{W} \leftarrow \mathbf{W} \odot \frac{(\mathbf{Z} \odot \Phi)\mathbf{H}^T}{(\mathbf{Z} \odot \Psi)\mathbf{H}^T} \qquad \mathbf{H} \leftarrow \mathbf{H} \odot \frac{\mathbf{W}^T(\mathbf{Z} \odot \Phi)}{\mathbf{W}^T(\mathbf{Z} \odot \Psi)} \tag{5.8}$$

where Φ and Ψ are defined as in Equation 5.6. Then $L_{expweighted}(\mathbf{W}, \mathbf{H}, \mathbf{Z})$ is nonincreasing under these update rules.

Proof. It suffices to prove that $\sum_{ij} \mathbf{Z}_{ij} \odot -A_{ij}(\mathbf{W}, \mathbf{H}, \mathbf{Z})$ is nonincreasing.

For a fixed $\mathbf{W}_{ij}$, define $\gamma(\mathbf{WH}_{ij}) = \gamma(\mathbf{WH}_{ij})^{\mathbf{Z}_{ij}}$ and $\tilde{\beta}_c(\mathbf{WH}_{ij}) = \mathbf{Z}_{ij}\beta_c(\mathbf{WH}_{ij})$. Let $\tilde{A}_{ij}$ be the distribution in the exponential family defined by $\nu, \tilde{\gamma}, \alpha_c, \tilde{\beta}_c$. Observe that $\mathbf{Z}_{ij} \odot A_{ij} = \tilde{A}_{ij}$. Thus, it suffices to argue $\sum_{ij} \tilde{A}_{ij}$ is nondecreasing. However, $\tilde{A}$ is also a distribution from the exponential family, and we have (applying the chain rule for $\tilde{\Psi}$)

$$\nabla_w \tilde{A}_{ia} = (\tilde{\Phi}\mathbf{H}^T - \tilde{\Psi}\mathbf{H}^T)_{ia} \qquad \nabla_h A_{aj} = (\mathbf{W}^T\tilde{\Phi} - \mathbf{W}^T\tilde{\Psi})_{aj}$$

$$\text{where } \tilde{\Phi}_{ij} = \sum_c \alpha_c(\mathbf{V}_{ij})\tilde{\beta}'_c(\mathbf{WH}_{ij}) = \mathbf{Z}_{ij} \odot \Phi_{ij} \text{ and}$$

$$\tilde{\Psi}_{ij} = -\frac{\tilde{\gamma}'(\mathbf{WH}_{ij})}{\gamma(\mathbf{WH}_{ij})} = -\frac{\mathbf{Z}_{ij}\gamma(\mathbf{WH}_{ij})^{\mathbf{Z}_{ij}-1}\gamma'(\mathbf{WH}_{ij})}{\gamma(\mathbf{WH})_{ij}^{\mathbf{Z}_{ij}}} = -\frac{\mathbf{Z}_{ij}\gamma'(\mathbf{WH}_{ij})}{\gamma(\mathbf{WH}_{ij})} = \mathbf{Z}_{ij} \odot \Psi_{ij}$$

Substituting $\tilde{\Phi}$ and $\tilde{\Psi}$ into the Cheung and Tresch multiplicative rules gives the desired result. □

With this lemma, we get the following monotonic algorithm.

Algorithm 5.13 W-Robust NMF with non-Gaussian noise

1: Initialize $\mathbf{Z} = 1$
2: **for** $t = 1, 2, 3 \ldots$ **do**
3: Update each entry of $\mathbf{W}$ using Equation 5.8
4: Update each entry of $\mathbf{H}$ using Equation 5.8
5: Update each entry of $\mathbf{Z}$ using Equation 5.3
6: **end for**

The $\mathbf{Z}$-update rule is unchanged from the case with Gaussian noise. Additionally,

the alternative $\mathbf{Z}$-update rules discussed in the previous chapter can be applied here as well.

5.4 Non-Independent Contamination

The ϵ-contamination model assumes that contamination occurs uniformly at random, i.e. that knowing $\mathbf{V}_{ij}$ is contaminated gives no additional information about whether other elements are contaminated. This assumption is natural for one dimensional data or robust linear regression. However, in the matrix setting, it is natural to consider the case where the contamination is not distributed uniformly among the matrix, but is centered in a few rows or columns. One particularly important situation is when contamination is centered around a small number of rows. This can occur when each row represents a user, vector, or item in the input dataset, and some rows represent anomalous items.

The W-robust algorithm and $L_{weighted}(\mathbf{W}, \mathbf{H}, \mathbf{Z})$ can be modified to capture non-independent contamination by adding a restriction on the elements of $\mathbf{Z}$. In the rowwise contamination model, either an entire row is marked as contaminated, or the entire row is marked as clean. This is easily captured by requiring $\mathbf{Z}_{ia} = \mathbf{Z}_{ib}$ for all $a, b \in [1, m]$.

$$\mathbf{W} \leftarrow \mathbf{W} \odot \frac{(\mathbf{V} \odot \mathbf{Z})\mathbf{H}^T}{(\mathbf{WH} \odot \mathbf{Z})\mathbf{H}^T} \qquad \mathbf{H} \leftarrow \mathbf{H} \odot \frac{\mathbf{W}^T(\mathbf{V} \odot \mathbf{Z})}{\mathbf{W}^T(\mathbf{WH} \odot \mathbf{Z})} \tag{5.9}$$

$$\mathbf{Z}_{ij} \leftarrow \begin{cases} 0 & \text{if } \sum_i (\mathbf{V}_{ij} - \mathbf{WH}_{ij})^2 < -n * \lambda^2 \\ 1 & \text{if } -n * \lambda \leq \sum_i (\mathbf{V}_{ij} - \mathbf{WH}_{ij})^2 < n * \lambda^2 \\ 0 & \text{if } \sum_i (\mathbf{V}_{ij} - \mathbf{WH}_{ij})^2 > n * \lambda^2 \end{cases} \tag{5.10}$$

Algorithm 5.14 W-robust NMF with rowwise contamination

1: Initialize $\mathbf{Z} = 1$
2: **for** $t = 1, 2, 3 \ldots$ **do**
3: Update each entry of $\mathbf{W}$ using Equation 5.9
4: Update each entry of $\mathbf{H}$ using Equation 5.9
5: Update each entry of $\mathbf{Z}$ using Equation 5.10
6: **end for**

5.5 Experiments

We consider both experiments with missing data in the recommendation setting (a popular application for NMF algorithms) and experiments with non-independent contamination.

5.5.1 Missing Data Experiments

We consider the Movielens 100M movie score dataset [15], a large dataset with 100M user-created movie scores and many missing entries. Each entry is a integer between 1 and 5, representing how a user rated a specific movie. This dataset has been used in

previous NMF experiments [25, 31]. We randomly partition the dataset into training (90%) and testing (10%). We also choose a random 'target movie' that the informed attacks will try to contaminate. We consider three contamination patterns. The later two attacks are based on injection attacks on recommendation systems [36, 33].

Untargeted Attack: We choose 10% of the training data and swap scores of 1-3 to 5, and swap scores of 4-5 to 1. The test set never undergoes contamination.

Low-Knowledge Attack: We choose a 'target movie' randomly. We then augment the training set with synthetic users who rate 19 movies randomly and give the target movie a score of 1. We add enough synthetic users to ensure that 10% of the scores assigned to the target movie are synthetic. The test set never undergoes contamination.

Informed Attack: We choose a 'target movie' randomly. We then convert real users in the training set into adversarial users who give the target movie a score of 1, but leave their other scores unchanged. We convert enough adversarial users to ensure that 10% of the scores assigned to the target movie are adversarial. The test set never undergoes contamination.

The contaminated training set is converted into a matrix $\mathbf{V}$ with many missing entries. We run both Weighted NMF (Algorithm 2.2), W-Robust NMF with missing data (Algorithm 5.12), and Zhang's expectation-maximization factorization algorithm [45] which attempts to 'densify' a matrix by replacing the missing entries with their current predictions. In all three experiments, we randomly permute the generated

Table 5.1: Test Error and Adversary Effect on Movie Data

Model	Alg	Test Err	Adv. Effect
Untargeted	Fixed Weight [30]	0.168	n/a
	Zhang [45]	0.163	n/a
	W-robust	0.164	n/a
Low Knowledge	Fixed Weight [30]	0.589	0.102
	Zhang [45]	0.587	0.072
	W-robust	0.588	0.054
Informed	Fixed Weight [30]	0.596	0.083
	Zhang [45]	0.598	0.050
	W-robust	0.595	0.019

matrix. All algorithms are given the same random initializations of $\mathbf{W}$ and $\mathbf{H}$ and the same computational resources.

Evaluation Metrics: We consider two error metrics. First, we consider the predictive error, defined as the total squared error between $\mathbf{WH}$ found by the algorithm and the scores in the testset (note the algorithms do not have access to these scores, and these scores are never contaminated). Following Zhang [45], we normalize the value by 1/5.

We also compute the adversary effect. We begin by running traditional NMF on the uncontaminated training set and call the resulting factors $\mathbf{WH}_{pure}$. We compute the total mean absolute error between $\mathbf{WH}$ found on the contaminated dataset with $\mathbf{WH}_{pure}$ only on the column associated with the targeted matrix. This evaluation metric was used in [36, 33] and measures how effective the adversary was at harming the targeted movie's rating. In all cases, we run the experiment 25 times and report the average results in Table 5.1.

Summary: Both Zhang's expectation minimization algorithm and the W-robust algorithm find better factorizations then Fixed Weight NMF. However, when looking at the adversary effect, we see that the W-robust algorithm has a substantially lower adversary effect than Zhang's algorithm. This is because we are explicitly modeling the adversarial entries as contamination and down-weighting those entries, while Zhang treats them as legitimate and allows them to pull down the score of the targeted movie.

5.5.2 Rowwise Contamination

We consider three datasets that have undergone rowwise (non-IID) contamination. We begin by creating $\mathbf{V}_{true}$, then choose a random row and add Gaussian $Normal(0, \sigma^2)$ noise to every element in that row. This makes the 'contamination row' substantially more noisy than the other rows. As the magnitude of the Gaussian noise increases, the contaminated row drifts further and further away from the underlying factorization, thus making it easier to identify. We consider three datasets.

Synthetic: To generate the matrices, we set $\mathbf{W} = \text{RAND}(100, 5)$ and $\mathbf{H} = \text{RAND}(5, 1000)$, where $\text{RAND}(n, m)$ is a $n \times m$ matrix with each entry drawn from the uniform $[0, 1]$ distribution. Values in $\mathbf{V}$ range from 0 to 5, with a mean of 1.25 and a stdev of 0.25. $\mathbf{V}_{true} = \mathbf{WH}$.

Hyperspectral We load a hyperspectral satellite image [11], which has been used before in NMF experiments. The original data is a 145×145 image with 224 values

associated with each pixel (a three dimensional matrix). We collapse the matrix into a 224×21025 matrix, where each row corresponds to a specific spectral band and each column corresponds to a pixel in the image. Values in $\mathbf{V}_{true}$ range from 0 to 10000, with a mean of 2652 and a stdev of 1592.3.The goal of finding the contamination row is equivalent to finding the spectral band that is returning incorrect values.

Medical We load Medulloblastoma microarray data [7] (Medulloblastoma is a type of brain tumor). This dataset has been used in previous NMF experiments and was one of the early successes of NMF. $\mathbf{V}_{true}$ is a 34×5893 matrix, where each row corresponds to an individual and each column corresponds to a gene. Values in $\mathbf{V}_{true}$ range from 20 to 1600 with a mean of 372 and a standard deviation of 972. The goal of finding the critical row is equivalent to finding the individual whose gene scan is abnormal. This application is particularly relevant as abnormal patients are known to be challenging when applying statistical methods to medical data.

In all three experiments, we randomly permute the generated matrix. We run both Traditional NMF (Algorithm 2.1) and W-robust NMF with rowwise contamination (Algorithm 5.14). Both algorithms are given the same amount of time to run and the same random initializations of $\mathbf{W}$ and $\mathbf{H}$.

Evaluation Metrics: For each value of σ, we run the experiment 50 times. For each trial, we compute the total squared error in each row. We sort this list in descending order and compute the index of the critical row in this list. For example, an index of 3 means that the contaminated row had the third highest error among all

rows, and that the contaminated row would have been the algorithm's third choice. The optimal index is 1, and lower numbers indicate better performance.

We also compute the improvement in Frobenius error from the original factorization to the new factorization with the contaminated row removed. To compute the original error, we run Traditional NMF on the input matrix. To compute the new error, we compute the Frobenius error between $\mathbf{V}'$ and $\mathbf{W}', \mathbf{H}'$ returned by the algorithm, excluding the contaminated row. Deleting a random row has an expected improvement ratio of $(n-1)/n$ if the error was uniform. Deleting the contaminated row should ideally improve the factorization by more than deleting a random row.

Summary: In all cases, both Traditional NMF and W-robust NMF with contaminated rows succeed at finding which row is contaminated (as long as the noise added to the contaminated row is sufficiently large). Additionally, both algorithms find a better quality factorization then removing a 'random' row (as seen in the green baseline in Figure 5.1). On the Synthetic and Hyperspectral data, the W-robust algorithm substantially outperforms Traditional NMF. On the Medulloblastoma data, there is no statistically significant difference in performance. Medical data is often messier than other datasets, making it challenging to achieve dramatic performance improvements.

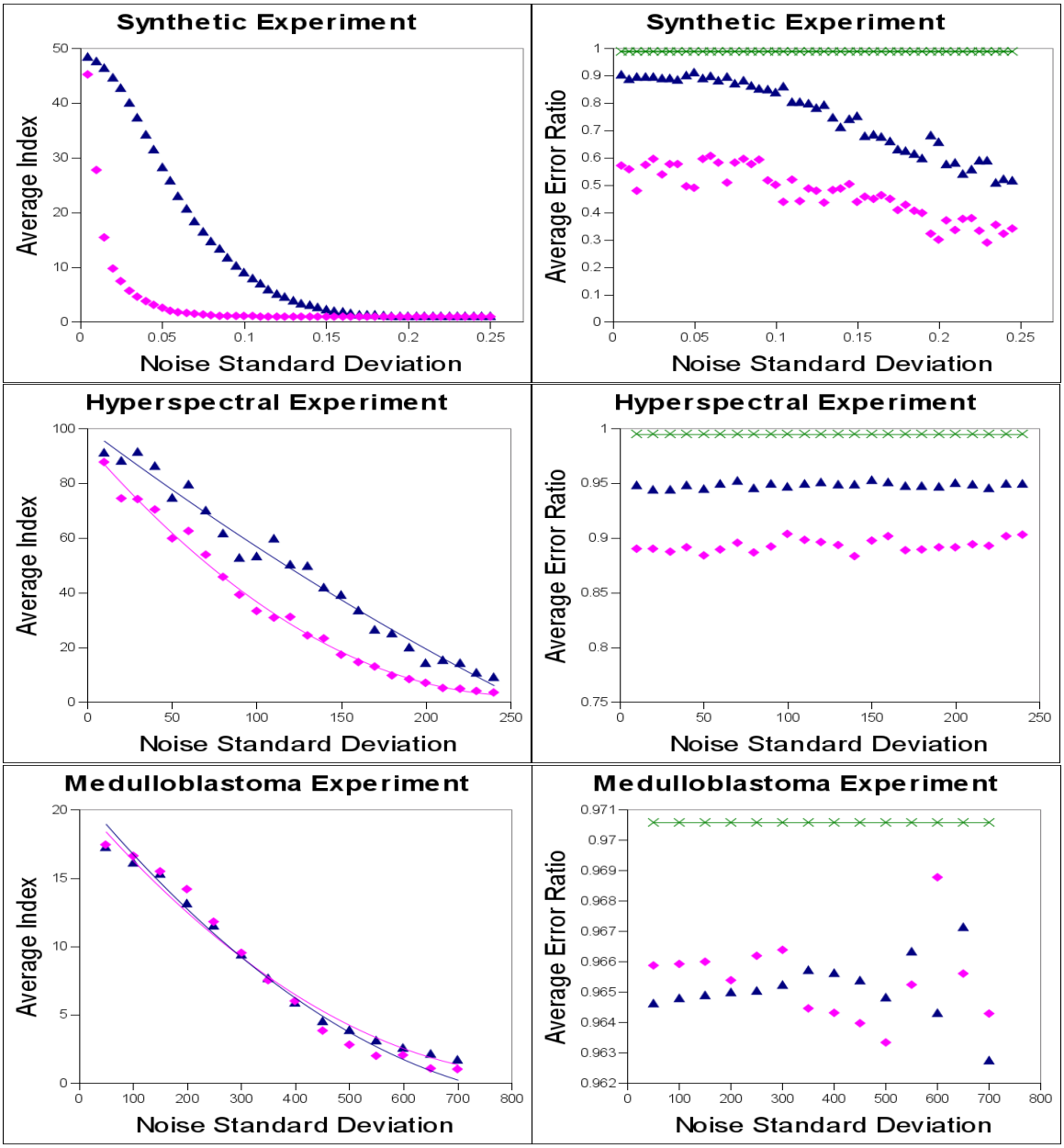


Figure 5.1: Experimental results. Traditional NMF is blue triangles, Algorithm 5.14 is pink dots, Baseline $(n-1)/n$ is green crosses. x-axis denotes noise stdev - larger values indicate a noiser contamination row; y-axis denotes index of the contamination row (left) and improvement in error ratio (right)

Table 5.2: Summary of Experimental Datasets

Dataset	n	m	k	mean($\mathbf{V}$)	stdev($\mathbf{V}$)	$(n-1)/n$
Synthetic	100	1000	5	1.25	0.50	0.990
Hyperspectral	224	21025	20	2652	1592	0.995
Medulloblastoma	34	5893	2	372	972	0.970

Chapter 6

Complexity Results

6.1 Introduction

In the previous chapters, we defined loss functions $L_{robust}(\mathbf{W}, \mathbf{H}, \mathbf{S})$ and $L_{reweighted}(\mathbf{W}, \mathbf{H}, \mathbf{Z})$, and said that $\mathbf{V}_{ij}$ was contaminated if $\mathbf{S}_{ij}$ was large or $\mathbf{Z}_{ij}$ was small. This approach connects to the theory of M-estimators and W-estimators, and let us convert the problem of finding contaminated elements into a problem of optimizing a function. However, these definitions are in some sense circular: a contaminated element is defined as an element that the algorithm marks as contaminated. For example, the determinate or rank is an intrinsic mathematical property of a matrix.

In this chapter, we leave the loss function approach behind mathematical definitions for what counts as a contaminated element. We begin with the following definition

Definition 6.1 (Nonnegative Rank)**.** The nonnegative rank of a nonnegative $n \times m$ matrix $\mathbf{V}$ is the smallest integer k such that there exist a $n \times k$ matrix $\mathbf{W}$ and a $k \times m$ matrix $\mathbf{H}$ such that $\mathbf{V} = \mathbf{WH}$.

Definition 6.2 (Critical Sets)**.** Let $\mathbf{V}$ be a nonnegative $n \times m$ matrix. A set of indices Ψ is an critical set if there exists a nonnegative $n \times m$ matrix $\mathbf{V}'$ such that $\mathbf{V}$ and $\mathbf{V}'$ agree on all indices not in Ψ and the nonnegative rank of $\mathbf{V}'$ is less than the nonnegative rank of $\mathbf{V}$.

Intuitively, the critical set is the set of 'contaminated elements', and fixing those elements reduces the nonnegative rank of the matrix. Note that every nonzero matrix has a critical set with nm elements (trivially, you can replace the entire matrix with a new matrix with lower rank). Thus, instead of asking whether a matrix has a critical set, it makes more sense to ask whether a matrix has a critical set that satisfies certain properties.

Summary of Results

In Section 6.2, we discuss existing complexity results on nonnegative matrix factorization and graph problems. Section 6.3 describes the first property: a critical set contained entirely within a single row. It also contains a novel proof that NMF is NP-hard. Section 6.4 discusses the second property: a critical set with at most c elements, or a minimal critical set. Section 6.5 discusses the third property: a critical set that obeys certain types of side channel information. For each property, deter-

mining whether a critical set that satisfies the desired property exists is NP-hard. In Section 6.6, we consider a very important type of side information: the nonnegative rank of the uncontaminated matrix (which we often know from domain knowledge). We prove the problem remains NP-hard even with this extra piece of information. Finally, we discuss a few additional results not entirely related to critical sets in Section 6.7.

6.2 Background

In this section, we discuss some important complexity theory background.

Definition 6.3 (NP-hard)**.** A problem is called NP-hard if a polynomial time algorithm to solve that problem implies that there exist a polynomial time algorithm to solve any other NP-hard problem, including the boolean satisfaction problem.

It remains an open question whether any NP-hard problem can be solved in polynomial time. A full discussion of the topic is beyond the scope of this dissertation, interested readers are directed to [16]. The common way to prove a problem is NP-hard is to prove there exists a polynomial time reduction between that problem and another NP-hard problem. Vavasis [44] constructs such a reduction between nonnegative rank and a geometric problem called intermediate simplex, then proves intermediate simplex is NP-hard. Aurora et al [1] later proved that a subexponential time algorithm to solve intermediate simplex would imply that the boolean satisfaction problem could be solved in subexponential time, which would disprove the

subexponential time hypothesis [16].

Definition 6.4 (Intermediate Simplex)**.** Given a polyhedron in $\mathbb{R}^{k-1}$ defined by n inequalities and a set of m points in $\mathbb{R}^{k-1}$, does there exist a $(k-1)$ dimensional simplex such that all the points are contained in the simplex and the simplex is contained in the polyhedron?

Theorem 6.5 ([44])**.** *Given an integer k, it is NP-hard to determine whether $\mathbf{V}$ has nonnegative rank k.*

Theorem 6.6 ([1])**.** *Given an integer k, a subexponential time algorithm to determine whether $\mathbf{V}$ has nonnegative rank k would imply the existence of a subexponential time algorithm to solve the boolean satisfaction problem.*

There are two main limitations to this reduction. The first limitation is that the proof is nonconstructive: it cannot be used to actually construct matrices which are difficult to factor. The second and more fatal limitation (at least for our purposes) is that there is no way to incorporate contamination into the reduction: there is no geometric analog to 'contaminated entries'. As such, the theoretical results in this chapter will derive not from geometry, but from graph theory.

A graph $G = (\mathcal{V}, \mathcal{E})$ is a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$ that connect those vertices. In this dissertation, we assume that all graphs are simple, i.e. every element of $\mathcal{E}$ is unique and there are no self loops with a vertex connecting to itself.

The minimum vertex cover problem and the maximal coverage problem are classical graph theory problems. We define both problems, along with the useful notation

of a coverage set, below.

Definition 6.7. Let $G = (\mathcal{V}, \mathcal{E})$. For any set $\mathcal{C} \subseteq \mathcal{V}$, the coverage set $\mathit{coverage}(\mathcal{C}) \subseteq \mathcal{E}$ is the set of edges (v_i, v_j) such that either $v_i \in \mathcal{C}$ or $v_j \in \mathcal{C}$ or both.

Definition 6.8 (Minimum Vertex Cover)**.** Let $G = (\mathcal{V}, \mathcal{E})$. A vertex cover is a set $\mathcal{C} \subseteq \mathcal{V}$ with $\mathit{coverage}(\mathcal{C}) = \mathcal{E}$. A vertex cover is minimal if no vertex cover with strictly fewer elements exists.

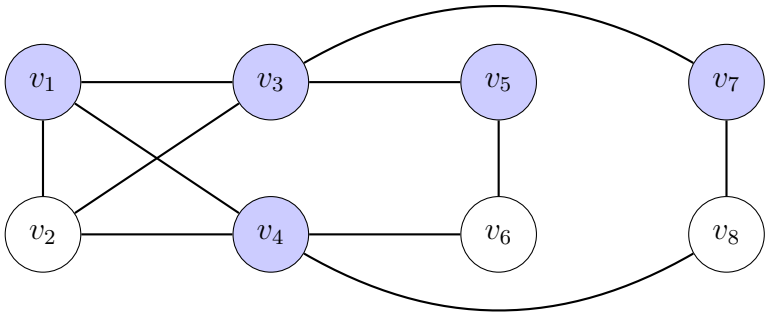
Definition 6.9 (Maximal Coverage)**.** Let $G = (\mathcal{V}, \mathcal{E})$ be a graph and let k be an integer. A set $\mathcal{C} \subseteq \mathcal{V}$ with k vertices is a maximal k-cover if $\mathit{coverage}(\mathcal{C})$ contains as many edges as the coverage set of any other set of k vertices.

Note that if $\mathcal{C}$ is a vertex cover with k elements, then it is by definition a maximal k-cover. Deciding whether a graph has a minimum vertex cover with k elements and/or finding such a vertex cover is NP-hard, as is finding the maximal k-cover of a graph. The critical edge problem is more recent, it was first proposed by Erdos and Gallai [13] while exploring graph colorings. The concept was later extended to vertex covers by Jakoby et al [19]. Finding the critical edge of a graph is also NP-hard.

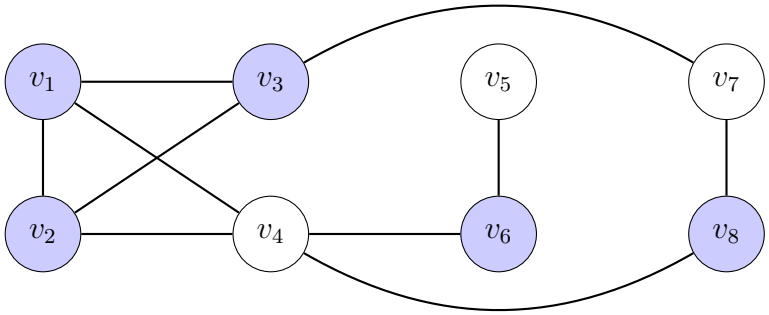
Definition 6.10 (Critical Edge)**.** Let $G = (\mathcal{V}, \mathcal{E})$ be a graph. An edge $e \in \mathcal{E}$ is MVC-critical if the subgraph $G' = (\mathcal{V}, \mathcal{E} - e)$ has a minimum vertex cover with fewer elements than the minimum vertex cover of G.

Effectively, deleting e from G reduces the size of the minimum vertex cover by one. Figure 6.1 has an example of a graph with a critical edge - deleting edge (v_1, v_2)

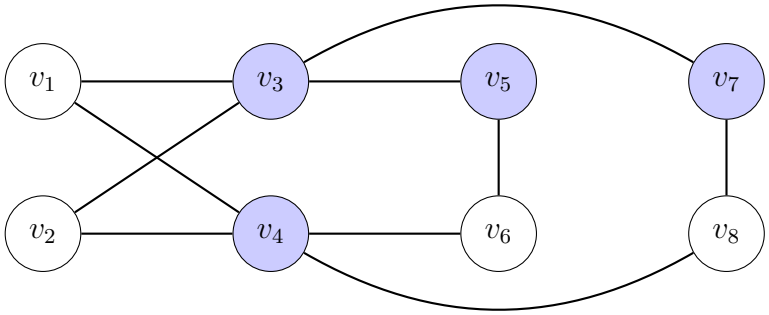
reduces the size of the minimum vertex cover from 5 to 4. Note that an edge may not be critical even if both of its endpoints are in a minimum vertex cover. Figure 6.1 visually depicts such a case: edge (v_3, v_5) is not a critical edge even though both v_3 and v_5 are in the minimum vertex cover of the original graph. It is possible that a graph has no critical edges, for example the star graph. It is also possible that every edge in a graph is a critical edge: such graphs are called critical graphs and are a topic of study. Cycle graphs are examples of such a graph, and Jakoby [19] gives an algorithm to generate random graphs with many critical edges.



vertex cover $= \{v_1, v_3, v_4, v_5, v_7\}$



vertex cover $= \{v_1, v_2, v_3, v_6, v_8\}$



vertex cover $= \{v_3, v_4, v_5, v_7\}$

Figure 6.1: The edge (v_1, v_2) is a critical edge because deleting this edge from G reduces the size of the MVC from five to four. The edge (v_3, v_5) is not a critical edge, even though both v_3 and v_5 are in the MVC of the original graph

6.3 Critical Rows

The first restriction we consider is whether a matrix has a critical set contained entirely in one row. This corresponds to the rowwise contamination model discussed in Section 5.4.

Definition 6.11 (Critical Row)**.** A critical row is a row whose indices form a critical set.

Theorem 6.12. *For any connected graph $G = (\mathcal{V}, \mathcal{E})$, there exists a nonnegative matrix $\mathbf{V}$ such that G has a vertex cover of size s iff $\mathbf{V}$ has a nonnegative rank of $k = s + |\mathcal{V}| + 4|\mathcal{E}|$. Additionally, if $e \in \mathcal{E}$ is a critical edge in G, then the corresponding rows in $\mathbf{V}$ are critical rows.*

Corollary 6.13. *It is NP-hard to determine whether a matrix has a critical row.*

We break the proof into three parts. First, we define the reduction from graphs to matrices (Figure 6.2). Lemma 6.15 proves one direction of the theorem. and Lemma 6.16 proves the other direction. This reduction takes inspiration from the work of L.B. Thomas [43] on the gap between rank and nonnegative rank, who uses the 5×4 matrix that appears in the bottom-right corner of Figure 6.2. It also takes inspiration from the work of Miettinen et al [32] and Jiang et al [20], who use similar ideas on on boolean rank and finite state automata respectively. However, because of the existence of the critical row, and because $\mathbf{H}$ is real-valued and not a subset of a finite collection of sets, these proofs do not apply and a new analysis is needed.

Reduction 6.14. Let $G = (\mathcal{V}, \mathcal{E})$ be a connected graph. We construct a nonnegative matrix $\mathbf{V}$ with $|\mathcal{V}| + 5|\mathcal{E}|$ rows and $2|\mathcal{V}| + 4|\mathcal{E}|$ columns. We begin by defining the columns. For each vertex $v_i \in \mathcal{G}$, we create two corresponding columns labeled x_i and y_i. For each edge $(v_i, v_j) \in \mathcal{E}$, we create four corresponding columns labeled a_{ij}, b_{ij}, c_{ij}, and d_{ij}.

We next define the rows and entries on $\mathbf{V}$. For each vertex $v_i \in \mathcal{V}$, we define a single row r_i. In each r_i row, set the corresponding x_i and y_i to be 1 and set all other entries in this row to 0. This creates the first $|\mathcal{V}|$ rows. Then, for each edge $(v_i, v_j) \in \mathcal{E}$, we define five rows as follows. All values not explicitly set to 1 are set to 0.

- The first row of $\mathbf{V}$ for (v_i, v_j) sets x_i, a_{ij}, and b_{ij} to 1.
- The second row of $\mathbf{V}$ for (v_i, v_j) sets y_i, c_{ij}, and d_{ij} to 1.
- The third row of $\mathbf{V}$ for (v_i, v_j) sets x_j, b_{ij}, and c_{ij} to 1.
- The fourth row of $\mathbf{V}$ for (v_i, v_j) sets y_j, a_{ij}, and d_{ij} to 1.
- The fifth row of $\mathbf{V}$ for (v_i, v_j) sets $a_{ij}, b_{ij}, c_{ij}, d_{ij}$ to 1.

This construction is visually depicted in Figure 6.2, which depicts $v_i, v_j \in \mathcal{V}$ with edge $(v_i, v_j) \in \mathcal{E}$.

Lemma 6.15. *If $G = (\mathcal{V}, \mathcal{E})$ has a minimum vertex cover $\mathcal{C}$ of size s, then $\mathbf{V}$ as defined in Reduction 6.14 has a nonnegative rank of $k = s + |\mathcal{V}| + 4|\mathcal{E}|$. Additionally,*

$$
\mathbf{V} = \begin{array}{c}
\begin{array}{cccccccccc} x_i & y_i & x_j & y_j & \dots & a_{ij} & b_{ij} & c_{ij} & d_{ij} & \end{array} \\
\left(\begin{array}{cccccccccc}
 & \vdots & & & & & & \vdots & & \\
1 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & \\
0 & 0 & 1 & 1 & \dots & 0 & 0 & 0 & 0 & \\
 & \vdots & & & & & \vdots & & & \\
1 & 0 & 0 & 0 & \dots & 1 & 1 & 0 & 0 & \\
0 & 1 & 0 & 0 & \dots & 0 & 0 & 1 & 1 & \\
0 & 0 & 1 & 0 & \dots & 0 & 1 & 1 & 0 & \\
0 & 0 & 0 & 1 & \dots & 1 & 0 & 0 & 1 & \\
0 & 0 & 0 & 0 & \dots & 1 & 1 & 1 & 1 & \\
 & \vdots & & & & & \vdots & & &
\end{array}\right)
\begin{array}{l}
\\
\text{row for } v_i \\
\text{row for } v_j \\
\\
\text{row1 for } (v_i, v_j) \\
\text{row2 for } (v_i, v_j) \\
\text{row3 for } (v_i, v_j) \\
\text{row4 for } (v_i, v_j) \\
\text{row5 for } (v_i, v_j) \\
\\
\end{array}
\end{array}
$$

Figure 6.2: Reduction of a graph to a matrix. Figure demonstrates two vertices v_i and v_j, along with an edge (v_i, v_j)

if $e \in \mathcal{E}$ is a critical edge, then all five rows of $\mathbf{V}$ corresponding to that edge are critical rows.

Proof. If $v_i \in \mathcal{C}$, add one row to $\mathbf{H}$ with $x_i = 1$ and all other entries set to zero, then add a second row to $\mathbf{H}$ with $y_i = 1$ and all other entries set to zero. If $v_i \notin \mathcal{C}$, add one row to $\mathbf{H}$ with $x_i = y_i = 1$ and all other entries set to zero. This process defines the first $s + |\mathcal{V}|$ rows of $\mathbf{H}$.

Next, for each edge $(v_i, v_j) \in \mathcal{G}$, either v_i or v_j must be in the vertex cover. Suppose that v_i is in the cover. Then we will add four rows to $\mathbf{H}$, defined as follows. All values not explicitly set to 1 are set to 0.

- The first row of $\mathbf{H}$ for (v_i, v_j) sets a_{ij}, and b_{ij} to 1.
- The second row of $\mathbf{H}$ for (v_i, v_j) sets c_{ij}, and d_{ij} to 1.
- The third row of $\mathbf{H}$ for (v_i, v_j) sets x_j, b_{ij}, and c_{ij} to 1.

$$
\mathbf{H} = \begin{array}{c}
\begin{array}{cccccccccc} x_i & x_j & y_i & y_j & \dots & a_{ij} & b_{ij} & c_{ij} & d_{ij} & \end{array} \\
\begin{pmatrix}
 & \vdots & & & & & & \vdots & & \\
1 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & \dots & 0 & 0 & 0 & 0 \\
 & \vdots & & & & & \vdots & & & \\
0 & 0 & 0 & 0 & \dots & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & \dots & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & \dots & 1 & 0 & 0 & 1 \\
 & \vdots & & & & & \vdots & & &
\end{pmatrix}
\end{array}
\begin{array}{l}
\\
\text{row1 for } v_i \in \mathcal{C} \\
\text{row2 for } v_i \in \mathcal{C} \\
\text{row1 for } v_j \notin \mathcal{C} \\
\\
\text{row1 for } (v_i, v_j) \\
\text{row2 for } (v_i, v_j) \\
\text{row3 for } (v_i, v_j) \\
\text{row4 for } (v_i, v_j) \\
\end{array}
$$

Figure 6.3: Reduction of a vertex cover to a matrix. Figure demonstrates case where v_i is in the cover, v_j is not in the cover, and (v_i, v_j) is an edge in the graph.

- The fourth row of $\mathbf{H}$ for (v_i, v_j) sets y_j, a_{ij}, and d_{ij} to 1.

This defines the remaining $4|\mathcal{E}|$ rows of $\mathbf{H}$. If v_j was in the cover, we would instead modify these four rows so that $x_i = 1$ in the first row, $y_i = 1$ in the second row, $x_j = 0$ in the third row, and $y_j = 0$ in the fourth row. This construction is visually depicted in Figure 6.3.

We now observe that every row of $\mathbf{V}$ can be written as a nonnegative linear combination of the rows in $\mathbf{H}$. Every row in $\mathbf{V}$ corresponding to vertex v_i can be written as the sum of the one or two rows in $\mathbf{H}$ corresponding to vertex v_i. Additionally, for each edge (v_i, v_j) with v_i in the vertex cover

1. Row1 (resp. row2) for (v_i, v_j) in $\mathbf{V}$ is the sum of the row1 (resp. row2) for v_i in $\mathbf{H}$ and row1 (resp. row2) for (v_i, v_j) in $\mathbf{H}$.

2. Row3 (resp. row4) for (v_i, v_j) in $\mathbf{V}$ is equal to row3 (resp. row4) for (v_i, v_j) in

H.

3. Row5 for (v_i, v_j) in $\mathbf{V}$ is the sum of the row1 and row2 for (v_i, v_j) in $\mathbf{H}$.

Therefore, every row of $\mathbf{V}$ can be written as a nonnegative linear combination of the rows of $\mathbf{H}$, and $\mathbf{H}$ has $k = s + |\mathcal{V}| + 4|\mathcal{E}|$ rows.

For the second part of the lemma, assume $e \in \mathcal{E}$ is a MVC-critical edge of G. Then there exists a cover $\mathcal{C}'$ with $s - 1$ vertices that covers every edge except e. To construct $\mathbf{H}'$, construct $(s-1) + |\mathcal{V}| + 4|\mathcal{E}| - 4$ rows associated with each vertex in the cover and each edge other than e. Every row of $\mathbf{V}$ - other than the five rows of $\mathbf{V}$ corresponding to e - can be written as a nonnegative combination of the rows in $\mathbf{H}$. Choose one of these rows as the critical row, then insert the remaining four rows into $\mathbf{H}'$. Now $\mathbf{H}'$ has $(s-1) + |\mathcal{V}| + 4|\mathcal{E}|$ rows, and every row of $\mathbf{V}$ except the critical row can be written as a nonnegative combination of the rows in $\mathbf{H}'$.

□

Lemma 6.16. *If* $\mathbf{V}$ *as defined in Reduction 6.14 has a nonnegative rank of* $k = s + |\mathcal{V}| + 4|\mathcal{E}|$*, then the corresponding* $G = (\mathcal{V}, \mathcal{E})$ *has a minimum vertex cover* $\mathcal{C}$ *of size* s*. Additionally, if a row in* $\mathbf{V}$ *that corresponds to* $e \in \mathcal{E}$ *is a NNR-critical row, then* e *is a MVC-critical edge in* G*.*

Proof. Let $\mathbf{H}$ be a set of k rows such that every row of $\mathbf{V}$ can be written as a nonnegative linear combination of the elements in $\mathbf{V}$. We make the following two observations.

Observation 1: For the row in $\mathbf{V}$ corresponding to a v_i, either

- The row can be written as a scalar multiple of a row in $\mathbf{H}$ with $x_i = y_i \neq 0$ and all other entries are 0. –or–

- The row can be written as a nonnegative linear combination of two rows in $\mathbf{H}$: one row with $x_i > 0$ and all other entries equal zero and one row with $y_i > 0$ and all other entries equal to zero. In this case, v_i is called a saturated vertex.

Observation 2: For the five rows in $\mathbf{V}$ corresponding to edge (v_i, v_j), these five rows cannot be written as the nonnegative linear combination of three or fewer rows, and that any row with that has a nonzero value in a column other than $x_i, y_i, x_j, y_j, a_{ij}, b_{ij}, c_{ij}, d_{ij}$ is useless to represent these rows. Thus, either:

- v_i (resp. v_j) is saturated, and these five rows in $\mathbf{V}$ can be written as the linear combination of the rows in $\mathbf{H}$ associated with v_i (resp. v_j), plus four additional rows.

- These five rows can be written as the linear combination of five or more rows in $\mathbf{H}$. In this case, edge (v_i, v_j) is called a saturated edge.

In both cases, the rows in $\mathbf{H}$ uses to represent the rows that are not associated with a vertex cannot be used to represent other rows. Thus, k is equal to $|\mathcal{V}|+4|\mathcal{E}|$ plus the number of saturated vertices, plus the number of rows in excess of four required to represent the saturated edges. This immediately implies that the number of saturated vertices plus the number of saturated edges is less than or equal to $k - |\mathcal{V}| - 4|\mathcal{E}|$. Furthermore, for every edge (v_i, v_j), either one of the vertices is saturated or the edge itself is saturated.

Define $\mathcal{C}$ to be the set of saturated vertices. Then, for each saturated edge, add either endpoint of the edge to $\mathcal{C}$. $\mathcal{C}$ is now a vertex cover with at most $k-|\mathcal{V}|-4|\mathcal{E}| = s$ elements, proving the first part of the lemma.

For the second part of the lemma, assume one of the rows in $\mathbf{V}$ corresponding to edge (v_i, v_j) is a critical row. Let $k' = k - 1$. Observe that the four remaining rows in $\mathbf{V}$ corresponding to the edge (v_i, v_j) still cannot be written as the nonnegative linear combination of three or fewer rows. Then k' is equal to $|\mathcal{V}| + 4|\mathcal{E}|$, plus the number of saturated vertices, plus the number of rows in excess of four required to represent the saturated edges. However it is possible that neither v_i nor v_j nor edge (v_i, v_j) is saturated. Construct $\mathcal{C}'$ identically as before. $\mathcal{C}'$ is a set with at most $k' - |\mathcal{V}| - 4|\mathcal{E}| = s - 1$ elements that covers every edge except (v_i, v_j), proving the second part of the lemma. □

6.4 Minimal Critical Set

The second restriction we consider is whether a matrix has a critical set that contains a fixed number of elements. An algorithm that could answer this query could be used to find the size of the minimal critical set. Alas, no polynomial time algorithm can solve this problem. Additionally, no polynomial time algorithm could achieve a 1.49-approximation of the minimal critical set.

Theorem 6.17. *For any connected graph $G = (\mathcal{V}, \mathcal{E})$, there exists a nonnegative matrix $\mathbf{V}$ such that G has a critical row iff $\mathbf{V}$ has a critical set with 2 elements in it.*

The theorem also holds for a critical set with 3 *elements.*

Corollary 6.18. *For any $c \geq 2$, it is NP-hard to determine whether a matrix has a critical set with at most c elements.*

Corollary 6.19. *Given a matrix* $\mathbf{V}$*, it is NP-hard to find a critical set that has at most* $(1.5 - \epsilon) * OPT$ *elements, where* OPT *is the size of the smallest critical set of* $\mathbf{V}$ *and* $\epsilon > 0$*.*

To prove Theorem 6.17, we need to modify the reduction slightly.

Reduction 6.20. Let $G = (\mathcal{V}, \mathcal{E})$ be a connected graph. We construct a nonnegative matrix $\mathbf{V}$ with $4|\mathcal{V}| + 5|\mathcal{E}|$ rows and $2|\mathcal{V}| + 4|\mathcal{E}|$ columns. The columns are defined identically to Reduction 6.14. We next define the rows on $\mathbf{V}$. For each vertex $v_i \in \mathcal{V}$, we define **four** row r_i^1, r_i^2, r_i^3, and r_i^4. In each r_i row, set the corresponding x_i and y_i to be 1 and set all other entries in this row to 0. This creates the first $4|\mathcal{V}|$ rows. The remaining $5|\mathcal{E}|$ rows are constructed as in Reduction 6.14

Proof of Theorem 6.17. We need to prove that $\mathbf{V}$ as defined in Reduction 6.20 has a critical set with two (or three) elements iff the corresponding $\mathcal{G}$ has a critical row.

For the first direction, suppose (v_i, v_j) is a critical edge in $\mathcal{G}$, then consider the elements in row5 for (v_i, v_j) in $\mathbf{V}$ in the columns corresponding to x_i and y_i. Both of these elements are 0. If these elements were flipped to 1, Row5 can be written as the sum of Row1 and Row2 for edge (v_i, v_j), which means that the five rows of $\mathbf{V}$ for (v_i, v_j) can be written as the linear combination of four rows in $\mathbf{H}$. The remainder of

the argument remains unchanged from Lemma 6.15, proving that these two elements form a critical set. Adding an arbitrary element to this set also generates a (non-minimal) critical set with three elements.

In the other direction, assume that $\mathbf{V}$ has a critical set with either two or three elements. Let $\mathbf{H}$ be a set of k rows such that every row of $\mathbf{V}$ can be written as a nonnegative linear combination of $\mathbf{V}$, with the exception of indices in the critical set which we will call the critical elements.

Consider the case where a critical element appears in one of the $3|\mathcal{V}|$ vertex rows. Observe every vertex row appears four times, and the critical set contains at most three elements. One copy of the vertex row must contains no critical elements, and can be written as the nonnegative linear combination of rows in $\mathbf{H}$. Therefore, all four copies can be written as linear combination of rows in $\mathbf{H}$. This means any critical element that appears in a vertex row can be removed from the critical set without increasing the nonnegative rank.

We now observe that Observation 2 from Lemma 6.16 still holds if the five rows in $\mathbf{V}$ corresponding to edge (v_i, v_j) have at most one element from the critical set. This means the five (unaltered) rows in $\mathbf{V}$ corresponding to edge (v_i, v_j) can be written as four/five rows in $\mathbf{H}$ (depending on whether v_i or v_j is saturated). This implies that replacing the element in the critical set with its original value does not reduce the number of rows needed. Therefore, if a set of five rows corresponding to an edge has one critical element, that element can be removed from the critical set without

increasing the nonnegative rank.

Let Φ be a critical set with three elements. By the above arguments, we can construct a new critical set $\Phi' \subseteq \Phi$ where all the elements in Φ' appear in the five edges associated with some edge $(v_i, v_j) \in \mathcal{E}$. Call that edge the critical edge and apply the argument in the last part of the proof for Lemma 6.16. □

The Proof of Corollary 6.18 follows from Theorem 6.17, 'stacking' multiply copies of a matrix along the diagonal, and applying the pigeonhole principal

Proof of Corollary 6.18. Theorem 6.17 already handles the case with $c = 2, 3$. For $c \geq 4$, we will prove for any matrix $\mathbf{V}$, there exists a matrix $\mathbf{V}'$ such that $\mathbf{V}$ has a critical set with at most 2 elements iff $\mathbf{V}'$ as a critical set with at most c elements. Theorem 6.17 will then prove the corollary.

To construct such a $\mathbf{V}'$ we create a block diagonal matrix by stacking $\lfloor c/2 \rfloor$ copies of $\mathbf{V}$, as demonstrates in Equation 6.1

$$\mathbf{V}'' = \left[\begin{array}{c|c|c|c} \mathbf{V} & 0 & \dots & 0 \\ \hline 0 & \mathbf{V} & \dots & 0 \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \dots & \mathbf{V} \end{array}\right] \tag{6.1}$$

If $\mathbf{V}$ has a critical set with 2 indices, then $\mathbf{V}'$ has a critical set with $2 * \lfloor c/2 \rfloor \leq c$ elements by taking those two indices and stacking them in the appropriate place on each copy of $\mathbf{V}$. In the other direction, suppose if $\mathbf{V}'$ has a critical set with at most c elements. Then by the pidgeonhold principal, there must exist one copy of $\mathbf{V}$ that contains at most 2 indices from the critical set, which proves the other direction. □

The proof of Corollary 6.19 follows from Theorem 6.5. G has a critical edge iff the minimum critical set of the corresponding matrix has two elements in it. Thus, a 1.49-approximation algorithm could distinguish between a matrix with a minimal critical set of size 2 (G has a critical edge) and a minimal critical set of size 3 (G has no critical edge).

6.5 Side Channel Critical Sets

It is plausible that certain types of side channel information might help the algorithm find contamination elements, specifically the number of contaminated elements in each row. We show that this type of information does not help.

Theorem 6.21. *For any connected graph $G = (\mathcal{V}, \mathcal{E})$, there exists a nonnegative matrix $\mathbf{V}$ and nonnegative integers $c_1 \dots c_n$ with $\sum c_i \geq 1$ such that G has a vertex cover of size s iff $\mathbf{V}$ has a critical set where each column has at most c_i elements in the critical set.*

Corollary 6.22. *It is NP-hard to determine whether a matrix has a critical set where each column has at most c_i elements from the critical set.*

To prove the thorem, we need to modify the reduction again. Unlike the previous reduction where we added extra rows, this reduction requires adding an extra column.

Reduction 6.23. Let $G = (\mathcal{V}, \mathcal{E})$ be a connected graph. We construct a nonnegative matrix $\mathbf{V}$ with $|\mathcal{V}| + 5|\mathcal{E}|$ rows and $2|\mathcal{V}| + 4|\mathcal{E}| + 1$ columns. We begin by defining the

columns. One column will be defined as the α column. The remaining columns are defined as in Reduction 6.14.

We next define the rows and entries on $\mathbf{V}$. For each vertex $v_i \in \mathcal{V}$, we define a single row r_i. In each r_i row, set the corresponding x_i and y_i to be 1, set α to be 1, and set all other entries to be 0. This creates the first $|\mathcal{V}|$ rows. For each of edge $(v_i, v_j) \in \mathcal{E}$, construct the five rows as in Reduction 6.14 and set α in all those rows to be 0.

Lemma 6.24. *$G = (\mathcal{V}, \mathcal{E})$ has a minimum vertex cover of size s iff $\mathbf{V}$ as defined in Reduction 6.23 has a critical set where each row contains at most c_i elements, where the c_i associated with the checksum column is equal to s and all other c_i are set to 0.*

Proof. Let $\mathcal{C}$ be a vertex cover of $\mathcal{G}$. If $v_i \in \mathcal{C}$, add one row to $\mathbf{H}$ with $x_i = 1$ and $\alpha = 0$ and all other entries set to zero, then add a second row to $\mathbf{H}$ with $y_i = 1$ and $\alpha = 0$ and all other entries set to zero. If $v_i \notin \mathcal{C}$, add one row to $\mathbf{H}$ with $x_i = y_i = \alpha = 1$ and all other entries set to zero. This process defines the first $s + |\mathcal{V}|$ rows of $\mathbf{H}$.

Construct the edge rows in $\mathbf{H}$ using the same procedure as Lemma 6.15 with $\alpha = 0$ in all of those rows. Define the critical set to be the α elements in the vertex rows of $\mathbf{V}$ with $v_i \in \mathcal{C}$. Observe that if each of these element in the critical set was flipped from 1 to 0, then every row of $\mathbf{V}$ could be written as a nonnegative linear combination of the rows in $\mathbf{H}$ defined above. This proves the first direction of the lemma.

For the second direction, we note that Observation 1, Observation 2, and the argument of Lemma 6.16 is completely unaffected by the existence of the α column,

so the same lemma also applies to the matrix defined in Reduction 6.23 □

6.6 Blocking Sets

Consider a matrix $\mathbf{V}$ whose true nonnegative rank is k, but contains five contaminated elements. According to the definition of critical set (Definition 6.2), the critical set of $\mathbf{V}$ would contain 1 element, and fixing that element would reduce the rank from $k+5$ to $k+4$. However, a more intuitive definition of contamination might state that $\mathbf{V}$ should have five contaminated elements, not one. Additionally, it is reasonable in many applications to assume that knowledge of the true rank of the uncontaminated matrix can be known or guessed.

We modify the definition of critical sets, and call the resulting definition a blocking set. Under this definition, the contaminated elements is the smallest set of elements that can be altered to reduce the nonnegative rank down to k (as opposed to the elements that can be altered to reduce the rank by 1). Note that the critical set of a nonzero matrix is never empty, while the k-blocking set of a nonzero matrix may indeed be empty.

Intuitively, critical sets are the matrix equivalent to critical edges of vertex cover, while blocking sets are the matrix equivalent to maximal k-covers (Definition 6.9).

Definition 6.25 (k-Blocking Set)**.** Let $\mathbf{V}$ be a nonnegative $n \times m$ matrix and let k be an integer. A set of indices Ψ is a k-blocking set if there exists a nonnegative $n \times m$ matrix $\mathbf{V}'$ such that $\mathbf{V}$ and $\mathbf{V}'$ agree on all elements not in Ψ and $\mathbf{V}'$ has rank

k.

An algorithm to return a blocking set that is approximately minimal would suggest that it might be possible for an algorithm to return a set that contains all contaminated elements in a matrix along with a small number of false positives. Unfortunately, finding a blocking set that is approximately minimal is NP-hard.

Theorem 6.26. *For any nonnegative matrix* $\mathbf{V}$*, there exists a nonnegative matrix* $\mathbf{V}'$ *such that* $\mathbf{V}$ *has rank at most* k *iff the* k*-blocking set of* $\mathbf{V}'$ *has at most* 1 *element in it*

Corollary 6.27. *For any nonnegative integer* c*, it is NP-hard to determine whether a matrix has a* k*-blocking set with at most* c *elements.*

Corollary 6.28. *Given a matrix* $\mathbf{V}$*, it is NP-hard to find a* k*-blocking set that has at most* $(1.5 - \epsilon) * OPT$ *elements, where* OPT *is the size of the smallest* k*-blocking set of* $\mathbf{V}$*.*

Proof of Theorem 6.26. For a $n \times m$ matrix $\mathbf{V}$, define the $(n+1) \times (m+1)$ matrix $\mathbf{V}'$ as

$$\mathbf{V}' = \left[\begin{array}{c|c} \mathbf{V} & 0 \\ \hline 0 & 1 \end{array}\right] \tag{6.2}$$

We will argue that $\mathbf{V}$ has rank k if and only if the minimal blocking set of $\mathbf{V}'$ has at most one element. In the first direction, if $\mathbf{V}$ has rank k, then the singleton set

containing the index of the bottom-left corner of $\mathbf{V}'$ is a blocking set of $\mathbf{V}'$ (switch the 1 to a 0).

In the other direction, if there exists a k-blocking set with no elements, than $\mathbf{V}'$ has rank k, and $\mathbf{V}$ must also have rank at most k. Thus, we need only consider the case where there exists blocking set has a single element, which we will call the blocking element.

By definition, there must exist $\mathbf{W}', \mathbf{H}'$ where $\mathbf{V}' = \mathbf{W}'\mathbf{H}'$ on all but one element, where $\mathbf{W}'$ is $(n+1) \times k$ and $\mathbf{H}'$ is $k \times (m+1)$. If the blocking element is not in the top $n \times m$ block of $\mathbf{V}'$, then $\mathbf{V}$ has a k-rank factorization formed by the top n rows of $\mathbf{W}'$ and the right m rows of $\mathbf{H}'$ and the proof is complete. Thus, we need only consider the case where the blocking element is somewhere in the top $n \times m$ block of $\mathbf{V}'$. This implies that the bottom-row of $\mathbf{W}'\mathbf{H}'$ has no contaminated elements and is equal to $[\ 0 \quad 0 \quad 0 \quad \dots \quad 1\ ]$.

Let s be the number of zeros in the bottom row of $\mathbf{W}'$. If the bottom row of $\mathbf{W}'$ contained no zeros, then the leftmost m columns of $\mathbf{H}'$ must all be zero. However, this would imply that $\mathbf{W}'\mathbf{H}'$ is 0 in the top $n \times m$ block, which implies $\mathbf{V} = 0$ and has rank k. Alternatively, if every element of the bottom row of $\mathbf{W}'$ was zeros, then there would be no way for the bottom row of $\mathbf{W}'\mathbf{H}'$ to be $[0 \dots 1]$. We may therefore assume $1 \leq s \leq k-1$

Without loss of generality, we assume that the leftmost s entries of the bottom row of $\mathbf{W}'$ are zero by permuting the columns of $\mathbf{W}'$ and $\mathbf{H}'$. However, we know that

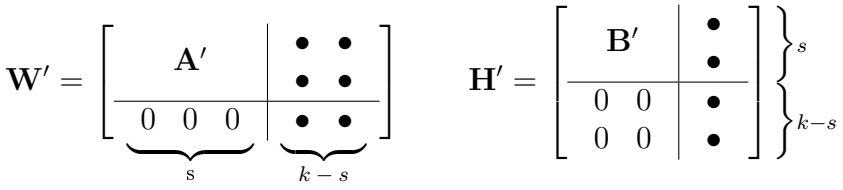


Figure 6.4: Visual representation of the sparsity pattern for $\mathbf{W}'$ and $\mathbf{H}'$. 0 represents a 0 element and $\bullet$ represents an element that may be zero or nonzero. Note $\mathbf{A}'$ is $n \times s$ and $\mathbf{B}'$ is $s \times m$, and that $s \leq k-1$

the bottom row of $\mathbf{W}'\mathbf{H}'$ is equal $[0 \ldots 1]$. It must be the case that the bottom-left $(k-s) \times m$ block of $\mathbf{H}'$ must consist of 0s. If this block contains a nonzero value, then $\mathbf{W}'\mathbf{H}'$ would have a nonzero value in a place where it must have a zero. Figure 6.4 visually represents the zero pattern of $\mathbf{W}'$ and $\mathbf{H}'$ described above.

Define $\mathbf{A}'$ to be the $n \times s$ top-left block of $\mathbf{W}'$ and define $\mathbf{B}'$ to be the top-left $s \times m$ block of $\mathbf{H}'$. $\mathbf{A}'\mathbf{B}'$ is a $n \times m$ matrix, and furthermore that the top-left $n \times m$ block of $\mathbf{W}'\mathbf{H}'$ is equal to $\mathbf{A}'\mathbf{B}'$ on every entry but one. This implies $\mathbf{V} = \mathbf{A}'\mathbf{B}'$ on every entry but one (the blocking element). Intuitively, we complete the proof by adding one additional row to $\mathbf{A}'$ and column to $\mathbf{B}'$ to handle the blocking element. Since $s \leq k-1$, $s+1 \leq k$, $\mathbf{V}$ has a k-rank factorization, and $\mathbf{V}$ has rank at most k.

Formally, let (i, j) denote the index of the blocking entry. Let $\mathbf{e}_i$ denote the k-dimensional vector whose i-th coordinate is 1 and with every other coordinate is 0, define $\mathbf{e}_j$ similarly. Finally, let $x = \mathbf{V}_{ij} - [\mathbf{A}'\mathbf{B}']_{ij}$. Define $\mathbf{A} = \begin{bmatrix} \mathbf{A}' & (\mathbf{e}_i)^T \end{bmatrix}$ and $\mathbf{B} = \begin{bmatrix} \mathbf{B}' \\ x * \mathbf{e}_j \end{bmatrix}$. Then $\mathbf{V} = \mathbf{A}\mathbf{B}$ on every entry of $\mathbf{V}$. Additionally, $\mathbf{A}$ is $n \times (s+1)$ and $\mathbf{B}$ is $(s+1) \times m$. Again, since $1 \leq s \leq k-1$, then $s+1 \leq k$. Thus, we have

constructed two nonnegative matrices with the desired dimensions whose product is $\mathbf{V}$. $\square$

Much like in Corollary 6.19, to prove Corollary 6.27, we need to stack multiple copies of the matrix.

Proof of Corollary 6.27. If $c = 0$, then Corollary 6.18 is equvialent to asking whether a matrix has rank at most k, which is NP-hard. If $c = 1$, Theorem 6.12 applies. Therefore, we may assume without loss of generality that $c \geq 2$.

Let $\mathbf{V}'$ be the matrix defined in Equation 6.1. We will construct a matrix $\mathbf{V}''$ such that $\mathbf{V}'$ has a blocking set with at most 1 element iff $\mathbf{V}''$ has a blocking set with at most c elements. Theorem 6.26 will then prove the corollary. To construct such a $\mathbf{V}''$, we construct a block diagonal matrix by stacking $c+1$ copies of $\mathbf{V}$, as demonstrated in Equation 6.2.

$$\mathbf{V}' = \left[\begin{array}{c|c|c|c|c} \mathbf{V} & 0 & \dots & 0 & 0 \\ \hline 0 & \mathbf{V} & \dots & 0 & 0 \\ \hline \vdots & \vdots & \ddots & \vdots & \vdots \\ \hline 0 & 0 & \dots & \mathbf{V} & 0 \\ \hline 0 & 0 & \dots & 0 & 1 \end{array}\right] \tag{6.3}$$

If $\mathbf{V}'$ has a blocking set of size at most 1, then applying the same logic as Theorem 6.26, the original $\mathbf{V}$ has rank at most k, and $\mathbf{V}''$ has a k-blocking set that is the singleton set containing the bottom-left element of $\mathbf{V}''$.

In the other direction, if $\mathbf{V}''$ has a blocking set with at most c elements, then one copy of $\mathbf{V}$ must have zero elements in the blocking set and therefore the original $\mathbf{V}$

has rank at most k. Therefore, $\mathbf{V}'$ has a blocking set with at most one element. $\square$

6.7 Additional Results

We briefly state a few additional results that are not related to critical sets and contamination, but easily follow from the reductions and ideas states in this chapter. Each of these corollaries follow from observing that one of the reductions has the desired property.

The first corollaries prove that finding the nonnegative rank remains hard even for sparse matrices and/or sparse factors. This result does not immediately follow from Vavasis's geometric reduction, but since many real world applications are focused on sparse matrices, it is a useful result.

Corollary 6.29. *Given a $n \times m$ sparse nonnegative matrix $\mathbf{V}$ with sparsity $4/n$ and an integer k, it is NP-hard to determine whether a matrix $\mathbf{V}$ has nonnegative rank k.*

Corollary 6.30. *Given a $n \times m$ nonnegative matrix $\mathbf{V}$ and an integer k, it is NP-hard to determine if there exist a nonnegative $n \times k$ matrix $\mathbf{W}$ with sparsity $4/n$ and a nonnegative $m \times k$ matrix $\mathbf{H}$ with sparsity $4/m$ such that $\mathbf{V} = \mathbf{WH}$.*

The next corollary focuses on the pure pixel assumption, also called the separability assumption [39]. Note that an alternative and equivalent definition sometimes used is that the columns of $\mathbf{W}$ appear in $\mathbf{V}$, since $\mathbf{V} = \mathbf{WH}$ iff $\mathbf{V}^T = \mathbf{H}^T\mathbf{W}^T$.

Definition 6.31 (Pure Pixel Assumption)**.** Given a $n \times m$ nonnegative matrix $\mathbf{V}$

and integer k, $\mathbf{V}$ satisfies the sparse pixel assumption if there exists a nonnegative $n \times k$ matrix $\mathbf{W}$ and a nonnegative $k \times m$ matrix $\mathbf{H}$ such that $\mathbf{V} = \mathbf{WH}$ and every row of $\mathbf{H}$ also appears in $\mathbf{V}$.

If $\mathbf{V}$ is square and satisfies the pure pixel assumption, then $\mathbf{V}$ can be factored in polynomial time by solving the linear program, where ρ is any vector with positive distinct entries.

$$\min_{\mathbf{X}>0} \rho^T diag(\mathbf{X})$$

$$\text{s.t.} |\mathbf{V} - \mathbf{VX}|_1 \leq \epsilon . trace(\mathbf{X}) = k$$

$$\mathbf{X}_{ii} \leq 1, \mathbf{X}_{ij} \leq \mathbf{X}_{ii} \text{ for all } i, j$$

With the optimal $\mathbf{X}$, set $\mathbf{H}$ to be equal to the k rows in $\mathbf{V}$ that correspond to the k largest diagonal entries of $\mathbf{X}$, at which point $\mathbf{W}$ can be found by solving a system of polynomials. However, as the next corollary will demonstrate, if the pure pixel assumption is only mostly satisfied, there's no guarantee that $\mathbf{X}$ will tell you the correct $\mathbf{H}$ rows. It may seem plausible to modify the linear program by adding some other constraint and salvage this approach. The next corollary makes clear this cannot work.

Corollary 6.32. *Given a $n \times m$ nonnegative matrix $\mathbf{V}$ and an integer k, it is NP-hard to determine whether there exists a nonnegative $n \times k$ matrix $\mathbf{W}$ and a nonnonegative $k \times m$ matrix $\mathbf{H}$ such that $\mathbf{V} = \mathbf{WH}$ and every row of $\mathbf{H}$ is within L_1 distance 1 of*

a row in $\mathbf{V}$. *This corollary also holds if every row of* $\mathbf{H}$ *is required to be within* L_0 *distance 1 (i.e. edit distance 1) of a row in* $\mathbf{V}$.

The third corollary relates to boolean matrix factorization (BMF) and effectively states that all of the theorems in the previous sections apply when critical sets (Definition 6.2) are defined in terms of boolean rank instead of nonnegative rank.

Definition 6.33 (Boolean Rank)**.** In boolean matrix multiplication, $\mathbf{V}, \mathbf{W}, \mathbf{H}$ must contain boolean true/false values. Boolean addition is the OR operation and boolean multiplication is the AND operation. The boolean rank of a $n \times m$ matrix $\mathbf{V}$ is the smallest value of k such that there exist boolean $n \times k$ matrix $\mathbf{W}$ and a $k \times m$ matrix $\mathbf{H}$ such that $\mathbf{V} = \mathbf{WH}$.

Miettinen et al [32] prove finding the boolean rank is NP-hard, but our proofs and reduction about critical sets naturally extend to the boolean case.

Corollary 6.34. *Theorems 6.12, 6.17, 6.21, and 6.26, along with all corollaries of those theorems, hold when the definition of critical set uses boolean rank instead of nonnegative rank.*

Chapter 7

Comparison between Approaches

The previous chapters created four definitions of contaminated elements:

1. M-Robust: Find the optimal point of $L_{robust}(\mathbf{W}, \mathbf{H}, \mathbf{S})$, then return elements with $\mathbf{S}_{ij} \neq 0$ as contaminated.

2. W-Robust: Find the optimal point of $L_{reweighted}(\mathbf{W}, \mathbf{H}, \mathbf{Z})$, then return elements with $\mathbf{Z}_{ij} = 0$ as contaminated.

3. Critical Set: Find the minimal critical set and return the elements in that critical set as contaminated.

4. Blocking Set: Find the minimal blocking set and return the elements in that blocking set as contaminated.

In this chapter, we discuss the differences between the four approaches, ending with a comparison between the algorithmic concepts (M-Robust and W-Robust) and

the theoretic concepts (the Critical and Blocking Set).

7.1 M-robust vs W-robust

In this section, we compare the M-robust and W-robust definitions by constructing a example where the two definitions return different sets. This highlights the challenge of using a loss function to define contamination: different loss functions return different sets.

Define $\mathbf{A}$ as a $(n-1) \times 2$ random matrix for sufficiently large n, with each entry of $\mathbf{A}$ drawn from Uniform(0,4). Define $\mathbf{W}_0$ as a $n \times 2$ matrix where the first row is $[0, 0]$ and the remaining $(n-1)$ rows are equal to $\mathbf{A}$. Define $\mathbf{H}_0$ and $\mathbf{V}$ as follows. Here, $\mathbf{V}$ is a $n \times 4$ matrix.

$$\mathbf{W}_0 = \left[\begin{array}{cc} 0 & 0 \\ \hline \multicolumn{2}{c}{\mathbf{A}} \end{array}\right] \qquad \mathbf{H}_0 = \begin{bmatrix} 6 & 3 & 3 & 2 \\ 1 & 1 & 1 & 2 \end{bmatrix} \qquad \mathbf{V} = \left[\begin{array}{cccc} 12 & 4 & 4 & 4 \\ \hline \multicolumn{4}{c}{\mathbf{AH}_0} \end{array}\right] \tag{7.1}$$

$$\mathbf{V} = \begin{bmatrix} 12 & 9 & 9 & 14 \\ 12 & 4 & 4 & 4 \\ 8 & 5 & 5 & 6 \\ 8 & 6.5 & 6.5 & 11 \\ 7 & 6 & 6 & 2.33 \\ 6 & 3 & 3 & 2 \\ 5 & 3.5 & 3.5 & 5 \end{bmatrix} \tag{7.2}$$

Suppose $\mathbf{V}$ is treated as a contaminated matrix. The following theorem describes

how the M-robust definition and W-robust definition return different solutions.

Theorem 7.1. *The following holds true with probability $e^{-O(n)}$, where n is the number of rows in $\mathbf{V}$.*

For any choice $0.01 < \lambda < 2\sqrt{2}$, the M-robust definition returns the set $\{(1,2),(1,3)\}$ as the set of contaminated elements. For any choice of $\lambda > 2\sqrt{2}$, the M-robust definition returns no contaminated elements.

For any choice $0.01 < \lambda < 2$, the W-robust definition returns the set $\{(1,1)\}$ as the set of contaminated elements. For any choice of $\lambda > 2$, the W-robust definition almost surely returns no contaminated elements.

Proof. Let $\mathbf{W}_1, \mathbf{H}_1, \mathbf{Z}_1$ be the values that minimize $L_{robust}(\mathbf{W}, \mathbf{H}, \mathbf{S})$. In general, finding the optimal factorization of a nonnegative matrix is NP-hard, but this example has been constructed so that an optimal factorization can be computed in polynomial time.

In order to compute the optimal factorization, we first argue $\mathbf{H}_1$ is either approximately equal to $\mathbf{H}_0$ or a rotation/dilation of $\mathbf{H}_0$ ($\mathbf{RH}_0$ for some invertible matrix $\mathbf{R}$).

Assume this was not the case. Consider the errors associated with rows 2-n in the factorization. If $\mathbf{H}_1 \neq \mathbf{RH}_0$, then with high probability then there are at least $n/2$ rows that have error $\sum_i |\mathbf{V}_{ij} - \mathbf{WH}_{ij}| > \epsilon$. If n is sufficiently large, then $n\epsilon/2 > 3$. We will momentarily construct a factorization with error approximately 2.758. Thus, H_1 would not be part of the optimal factorizations, which is a contradiction.

Once we know that $\mathbf{H}_1$ is a rotation/dilation of $\mathbf{H}_0$, we can use local search to compute the optimal $\mathbf{W}_1$ and $\mathbf{Z}_1$. When $\lambda = 1$, the following matrices are one of the optimal solution (again, note that $\mathbf{W}_1\mathbf{R}^{-1}$ and $\mathbf{R}\mathbf{H}_1$ are also optimal solutions).

$$\mathbf{W}_1 = \left[\begin{array}{cc} 1.925 & 0 \\ \hline \multicolumn{2}{c}{\mathbf{A}} \end{array}\right] \qquad \mathbf{H}_1 = \mathbf{H}_0 \qquad \mathbf{S}_1 = \left[\begin{array}{cccc} 0 & -0.275 & -0.275 & 0 \\ \hline \multicolumn{4}{c}{\mathbf{0}} \end{array}\right] \tag{7.3}$$

As λ increases, the nonzero entry in $\mathbf{W}_1$ and $\mathbf{S}_1$ approach 0. At $\lambda = \sqrt{2} + \epsilon$, the optimal solution sets $\mathbf{W}_1$ to $[1.7931 \quad 0]$ and sets all entries in $\mathbf{S}_1$ to 0.

Let $\mathbf{W}_2, \mathbf{H}_2, \mathbf{Z}_2$ be the values that minimize $L_{reweighted}(\mathbf{W}, \mathbf{H}, \mathbf{Z})$. In general, finding the optimal factorization is NP-hard, but this example has been constructed so that the optimal solution can be computed in polynomial time. The same argument that applies to $\mathbf{H}_1$ applies to $\mathbf{H}_2$. Again using local search, we get that the optimal solution for $\lambda = 1$ is

$$\mathbf{W}_2 = \left[\begin{array}{cc} 1 & 1 \\ \hline \multicolumn{2}{c}{\mathbf{A}} \end{array}\right] \qquad \mathbf{H}_2 = \mathbf{H}_0 \qquad \mathbf{Z}_2 = \left[\begin{array}{cccc} 0 & 1 & 1 & 1 \\ \hline \multicolumn{4}{c}{\mathbf{1}} \end{array}\right] \tag{7.4}$$

When $\lambda > 2.7586$, instead the optimal solution sets the first row of $\mathbf{W}_2$ to $[1.7931 \quad 0]$ and all entries in $\mathbf{Z}_2$ to be 1. □

On the experiments run in Section 3.6.2, the W-robust algorithm slightly outperformed the M-robust algorithm. However, one relevant point is that the M-robust

algorithm was substantially faster when implemented in a distributed setting, making it more appropriate for very large datasets that cannot fit on a single machine.

7.2 Critical Set vs Blocking Set

Suppose $\mathbf{V}$ has rank $k+2$. The critical set is the smallest set of elements that need to be changed to reduce $\mathbf{V}$ to rank $k+1$, while the blocking set is the smallest set of element that need to be changed to reduce $\mathbf{V}$ to rank k. Clearly these sets are different. However, an obvious modification of the definition is to compose multiple critical sets. Consider Algorithm 7.15.

Algorithm 7.15 Critical Set Composition

1: Define $nnr(\mathbf{V})$ as an oracle function that returns the nonnegative rank of $\mathbf{V}$
2: Define $mincrit(\mathbf{V})$ as an oracle function that returns both the minimal critical set Ψ of $\mathbf{V}$ and a matrix $\mathbf{V}'$ with $nnr(\mathbf{V}') = nnr(\mathbf{V}) - 1$ and $\mathbf{V} = \mathbf{V}'$ on all elements not in Ψ.
3:
4: Set $\mathbf{V}_{current} = \mathbf{V}$, $\Psi_{final} = \emptyset$
5: **while** $nnr(\mathbf{V}_{current}) > k$ **do**
6: $\quad \mathbf{V}', \Psi = mincrit(\mathbf{V}_{current})$
7: $\quad \mathbf{V}_{current} = \mathbf{V}$, $\Psi_{final} = \Psi_{final} \cup \Psi$
8: **end while**
9: Return Ψ_{final} as the 'composition critical set'

Both the minimum blocking set and the composition critical set reduce $\mathbf{V}$ to rank k, so it is at least plausible that the two could be equivalent. The remainder of this section is dedicated to constructing a counterexample where the two are distinct. Define the 4×17 matrix $\mathbf{V}$ as follows, where $\mathbf{1}_9$ is the 1×9 matrix filled by 1s. $\mathbf{V}$ has nonnegative rank 4. Define $k = 2$.

$$\mathbf{V} = \left[\begin{array}{ccc|ccccc|c} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}_9 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & \mathbf{1}_9 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}_9 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}_9 \end{array}\right] \tag{7.5}$$

Theorem 7.2. *When $k = 2$, the Minimal Blocking Set of $\mathbf{V}$ is $\{(3,1),(3,2),(3,3),(4,2),(4,3)\}$ and the Composed Critical Set of $\mathbf{V}$ is $\{((4,1),(2,4),(2,5),(2,6),(2,7),(2,8)\}$.*

Proof. As in the previous section, while finding the optimal factorization of a nonnegative matrix is NP-hard, this example has been constructed so that an optimal factorization can be computed in polynomial time. Define Ψ_1 as the minimal blocking set, and define $\mathbf{V}_1$ as a matrix with nonnegative rank 2 such that $\mathbf{V}_1$ and $\mathbf{V}$ agree on all elements not in Ψ. Further define $\mathbf{W}_1, \mathbf{H}_1$ as a 4×2 and 2×14 matrices with $\mathbf{V}_1 = \mathbf{W}_1\mathbf{H}_1$. Again, note that factorization is not unique, and $\mathbf{V}_1 = \mathbf{W}_1\mathbf{R}^{-1}\mathbf{R}\mathbf{H}_1$ for any invertible matrix $\mathbf{R}$. We highlight the elements in Ψ in the equation below.

$$\mathbf{W}_1 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{H}_1 = \left[\begin{array}{c|c|c} \mathbf{0}_3 & \mathbf{1}_5 & \mathbf{0}_9 \\ \mathbf{0}_3 & \mathbf{0}_5 & \mathbf{1}_9 \end{array}\right]$$

$$\mathbf{V}_1 = \left[\begin{array}{ccc|ccccc|c} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}_9 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & \mathbf{1}_9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}_9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}_9 \end{array}\right]$$

Now consider the critical set composition set. Let $\mathbf{V}_2$ be the intermediate with

rank 3 matrix temporarily stored by Algorithm 7.15 with $\mathbf{V}_2 = \mathbf{W}_2\mathbf{H}_2$. Observe that $\mathbf{W}_2$ and $\mathbf{H}_2$ are not reducible. We highlight the intermediate Ψ from line 6.

$$\mathbf{W}_2 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \qquad \mathbf{H}_2 = \left[\begin{array}{c|c|c} \mathbf{1}_3 & \mathbf{0}_5 & \mathbf{0}_9 \\ \mathbf{0}_3 & \mathbf{1}_5 & \mathbf{0}_9 \\ \mathbf{0}_3 & \mathbf{0}_5 & \mathbf{1}_9 \end{array}\right]$$

$$\mathbf{V}_2 = \left[\begin{array}{ccc|ccccc|c} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}_9 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & \mathbf{1}_9 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}_9 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}_9 \end{array}\right]$$

Let $\mathbf{V}_3$ be the final with rank 2 matrix returned by Algorithm 7.15 with $\mathbf{V}_3 = \mathbf{W}_3\mathbf{H}_3$. Observe that $\mathbf{W}_3$ and $\mathbf{H}_3$ are not reducible. We highlight the final Ψ returned in red.

$$\mathbf{W}_3 = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad \mathbf{H}_3 = \left[\begin{array}{c|c|c} \mathbf{1}_3 & \mathbf{0}_5 & \mathbf{0}_9 \\ \mathbf{0}_3 & \mathbf{0}_5 & \mathbf{1}_9 \end{array}\right]$$

$$\mathbf{V}_2 = \left[\begin{array}{ccc|ccccc|c} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}_9 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}_9 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}_9 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & \mathbf{1}_9 \end{array}\right]$$

□

7.3 Theory vs Practice

In Chapter 6, we constructed matrices (Reduction 6.14, 6.20, 6.23) that encoded instances of vertex cover. An obvious question is to ask what happens when Traditional NMF or one of the Robust NMF algorithms is run on these hard instances. The answer is underwhelming: all algorithms completely fail to find reasonable factorizations.

As an experiment, we convert Figure 7.1 into a 58×56 matrix using Reduction 6.14. We then run the W-robust algorithm for Rowwise Contamination described in Section 5.4 on this matrix. We compare to a 58×56 random matrix and run the same set of experiments as described in Section 5.5.2 and report the results in Figure 7.1. As the figure demonstrates, both traditional NMF and Robust NMF fail to find a high good factorization of the matrix which encodes vertex cover. Benchmark testing on other vertex cover graphs shows similar results.

Summary , the Robust algorithms work well on real-world datasets and on random matrices, but perform badly on this artificially hard instance that encode vertex cover. Looking at Theorem 6.12 gives a possible explanation for why this is the case. There are several greedy vertex cover algorithms [10, 2] that effectively work by (1) defining a heuristic function (2) add a vertex to the cover to minimize that heuristic (3) repeat until the entire graph is covered. These algorithms perform incredibly well on real-world datasets, achieving 1.01-approximations [22]. However, they perform badly on artificially hard graphs, achieving terrible worst case performance. Robust NMF follows a similar paradigm by defining a heuristic function (one of the Robust loss

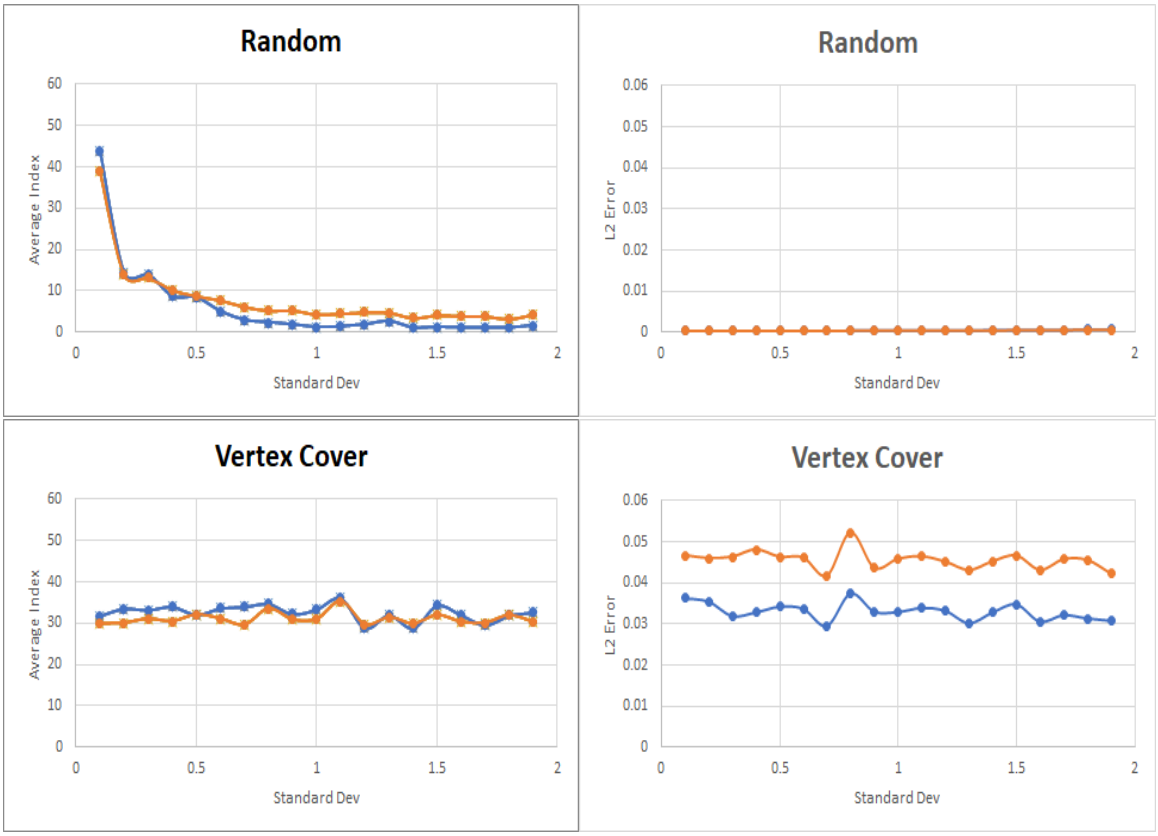


Figure 7.1: NMF run on a graph that encodes vertex cover vs a random matrix. Traditional NMF is orange line, Algorithm 5.14 is blue line. x-axis denotes noise stdev - larger values indicate a noiser contamination row; y-axis denotes index of the contamination row (left) and improvement in error ratio (right)

functions) and mark elements as contaminated to minimize the hueristic. Robust-NMF perform quite well on real-world datasets, but can be thwarted by carefully constructed hard instances. For both Vertex Cover and Robust-NFM, real world datasets do not look like these artificially hard instances. As a result, the Robust NMF algorithms are useful in many real world settings even if they are not theoretically guaranteed to provide good solutions on every possible instance.

Bibliography

[1] Sanjeev Arora, Rong Ge, Ravindran Kannan, and Ankur Moitra. Computing a nonnegative matrix factorization–provably. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 145–162. ACM, 2012.

[2] S Balaji, V Swaminathan, and K Kannan. Optimization of unweighted minimum vertex cover. *World Academy of Science, Engineering and Technology*, 43:716–729.

[3] Peter Ballen. Critical rows of almost-factorable matrices. In *2019 Combinatorial Optimization and Applications*. Springer, 2019.

[4] Peter Ballen. Nonnegative matrix factorization under adversarial noise. In *2019 Conference on Data Mining and Applications*, volume 9. AIRCC, 2019.

[5] Peter Ballen and Sudipto Guha. Elastic nonnegative matrix factorization. In *2018 International Conference on Data Mining Workshops*, pages 1271–1278. IEEE, 2018.

[6] Ayanendranath Basu, Ian R Harris, Nils L Hjort, and MC Jones. Robust and efficient estimation by minimising a density power divergence. *Biometrika*, 85(3):549–559, 1998.

[7] Jean-Philippe Brunet, Pablo Tamayo, Todd R Golub, and Jill P Mesirov. Metagenes and molecular pattern discovery using matrix factorization. 101(12):4164–4169, 2004.

[8] Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):11, 2011.

[9] Vincent CK Cheung and Matthew C Tresch. Non-negative matrix factorization algorithms modeling noise distributions within the exponential family. In *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the*, pages 4990–4993. IEEE, 2006.

[10] Kenneth L Clarkson. A modification of the greedy algorithm for vertex cover. *Information Processing Letters*, 16(1):23–25, 1983.

[11] University of Basque County Computational Intelligence Group. `http://www.ehu.eus/ccwintco/index.php/hyperspectral_remote_sensing_Scenes`.

[12] Dask Development Team. *Dask: Library for dynamic task scheduling*, 2016.

[13] Paul Erdos and Tibor Gallai. On the minimal number of vertices of a graph representing the edges of a graph, 1961.

[14] Cédric Févotte and Jérôme Idier. Algorithms for nonnegative matrix factorization with the β-divergence. *Neural computation*, 23(9):2421–2456, 2011.

[15] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):19, 2016.

[16] A Condon D Harel J Hartmanis, T Henzinger, J Hromkovic N Jones T Leighton, and M Nivat. Texts in theoretical computer science an eatcs series. 2006.

[17] Patrik Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research*, 5(Nov):1457–1469, 2004.

[18] Peter J Huber. Robust estimation of a location parameter. In *Breakthroughs in statistics*, pages 492–518. Springer, 1992.

[19] Andreas Jakoby, Naveen Kumar Goswami, Eik List, and Stefan Lucks. Critical graphs for the minimum-vertex-cover problem. In *Proceedings of the International Conference on Foundations of Computer Science (FCS)*, pages 3–9. The Steering Committee of The World Congress in Computer Science, Computer ..., 2017.

[20] Tao Jiang and Bala Ravikumar. Minimal nfa problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993.

[21] Ramakrishnan Kannan, Hyenkyun Woo, Charu C Aggarwal, and Haesun Park. Outlier detection for text data. pages 489–497, 2017.

[22] Imran Khan and Sangeen Khan. Experimental comparison of five approximation algorithms for minimum vertex cover. *International Journal of u-and e-Service*, 7(6):69–84, 2014.

[23] Wooyoung Kim, Bernard Chen, Jingu Kim, Yi Pan, and Haesun Park. Sparse nonnegative matrix factorization for protein sequence motif discovery. *Expert Systems with Applications*, 38(10):13198–13207, 2011.

[24] Deguang Kong, Chris Ding, and Heng Huang. Robust nonnegative matrix factorization using l21-norm. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 673–682. ACM, 2011.

[25] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, (8):30–37, 2009.

[26] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788, 1999.

[27] Daniel D Lee and H Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in neural information processing systems*, pages 556–562, 2001.

[28] Chih-Jen Lin. On the convergence of multiplicative update algorithms for nonnegative matrix factorization. *IEEE Transactions on Neural Networks*, 18(6):1589–1596, 2007.

[29] Chih-Jen Lin. Projected gradient methods for nonnegative matrix factorization. *Neural computation*, 19(10):2756–2779, 2007.

[30] Yun Mao and Lawrence K Saul. Modeling distances in large-scale networks by matrix factorization. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 278–287. ACM, 2004.

[31] Prem Melville and Vikas Sindhwani. Recommender systems. *Encyclopedia of Machine Learning and Data Mining*, pages 1056–1066, 2017.

[32] Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 335–346. Springer, 2006.

[33] Bamshad Mobasher, Robin Burke, and Jeff J Sandvig. Model-based collaborative filtering as a defense against profile injection attacks. 2006.

[34] Oleg Okun and Helen Priisalu. Fast nonnegative matrix factorization and its application for protein fold recognition. *EURASIP Journal on Advances in Signal Processing*, 2006(1):071817, 2006.

[35] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[36] Michael O'Mahony, Neil Hurley, Nicholas Kushmerick, and Guénolé Silvestre. Collaborative recommendation: A robustness analysis. *ACM Transactions on Internet Technology (TOIT)*, 4(4):344–377, 2004.

[37] Pentti Paatero and Unto Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994.

[38] Neittaanmaki Pekka et al. Nonsmooth optimization: analysis and algorithms with applications to optimal control. 1992.

[39] Ben Recht, Christopher Re, Joel Tropp, and Victor Bittorf. Factoring nonnegative matrices with linear programs. In *Advances in Neural Information Processing Systems*, pages 1214–1222, 2012.

[40] Ferdinando S Samaria and Andy C Harter. Parameterisation of a stochastic model for human face identification. In *Applications of Computer Vision, 1994., Proceedings of the Second IEEE Workshop on*, pages 138–142. IEEE, 1994.

[41] B. Shen, B. D. Liu, Q. Wang, and R. Ji. Robust nonnegative matrix factorization via l1 norm regularization by multiplicative updating rules. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 5282–5286, Oct 2014.

[42] Suvrit Sra and Inderjit S Dhillon. Generalized nonnegative matrix approximations with bregman divergences. In *Advances in neural information processing systems*, pages 283–290, 2006.

[43] L.B. Thomas. Rank factorization of nonnegative matrices. *SIAM Review*, 16(3):393–394, 1974.

[44] Stephen A Vavasis. On the complexity of nonnegative matrix factorization. *SIAM Journal on Optimization*, 20(3):1364–1377, 2009.

[45] Sheng Zhang, Weihong Wang, James Ford, and Fillia Makedon. Learning from incomplete ratings using non-negative matrix factorization. In *Proceedings of the 2006 SIAM international conference on data mining*, pages 549–553. SIAM, 2006.