2023-08-21

Characterization of Stability of Non-Negative Matrix Factorization Models: An Application to Single-Cell Data

Liu, Alexander EJ

UNIVERSITY OF CALGARY

Characterization of Stability of Non-Negative Matrix Factorization Models: An Application to Single-Cell Data

by

Alexander EJ Liu

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

GRADUATE PROGRAM IN MATHEMATICS AND STATISTICS

CALGARY, ALBERTA
AUGUST, 2023

Abstract

The non-negative matrix factorization (NMF) is a powerful machine learning technique used in mathematics, computer science, and data science. This technique has applications in a wide range of fields including recommender systems, image processing, signal processing, machine learning and genetics. Recently, NMF has gained popularity in the analysis of single-cell gene expression data to identify cell types and gene expression patterns. In this thesis, we have studied the NMF, its rank estimation, classification, and stability using both simulated data and real single-cell gene expression data. We have designed two simulated data sets with desired features and tested two seeding methods, eight NMF algorithms and five rank estimation criteria. Additionally, a real single-cell gene expression data has been used to further characterize the NMF algorithms. We have also investigated the stability of NMF, first over the sample size consideration and then on initialization. The detailed conditions that have been revealed by this thesis may generate practical impact in directing the appropriate use of NMF in analyzing single-cell gene expression data.

Acknowledgements

I would like to extend my deepest gratitude and appreciation to several individuals who have played a significant role in the completion of this thesis.

First and foremost, I would like to express my heartfelt thanks to my supervisor, Dr. Qingrun Zhang, for her unwavering guidance, expertise, and constant support throughout this research journey. Her dedication, insightful feedback, and encouragement have been instrumental in shaping the direction and quality of this thesis. I am also grateful to Dr. Quan Long, for his valuable insights and support during this process.

I am thankful to Dr. Jingjing Wu and Dr. Yuan Xu for serving as my thesis defense committee members. I also want to express gratitude to my graduate course instructors Dr. Thierry Chekouo, Dr. Gemai Chen, Dr. Xuewen Lu, Dr. Anatoliy Swishchuk, and Dr. Qingrun Zhang.

I am incredibly fortunate to have the love and support of my parents, Jiny and Shawn Liu, who have been my pillars of strength, especially during the challenging times of the COVID-19 pandemic. Their unwavering belief in my abilities, encouragement, and sacrifices have been invaluable, and I am truly grateful for their presence in my life.

I would like to express my sincere appreciation to all my lab mates who have contributed to this thesis in various ways. Special thanks go to Qing (Leah) Li, Jiayi (Anne) Bian, Xiang He, Hamid Hamidi, Dinghao Wang, and Sandesh Acharya for their guidance, insightful discussions, and camaraderie. Their collective expertise and support have greatly enriched my research experience.

I would also like to acknowledge the friendships I have formed during my time at the University of Calgary. The support, laughter, and shared experiences with my friends have made my academic journey more enjoyable and memorable. I am thankful for their presence in my life.

Lastly, I would like to express my gratitude to the University of Calgary for providing me with the necessary resources, facilities, and opportunities to conduct this research. The academic environment and the assistance received from the university have been instrumental in the successful completion of this thesis.

Table of Contents

List of Figures

List of Tables

List of Symbols, Abbreviations, and Nomenclature

Symbol	Definition
NMF	Non-Negative Matrix Factorization
FA	Factor Analysis
PCA	Principal Component Analysis
ICA	Independent Component Analysis
ED	Euclidean Distance
S_i	ith Cell Sparseness
NS_i	Normalized ith Cell Sparseness
E_i	ith Cell Entropy
IG_i	ith Cell Information Gain
TIG	Total Information Gain
$NTIG$	Normalized Total Information Gain
r	Rank (Hidden factors)
DC	Degree of Confusion
DC_i	ith row Degree of Confusion
RT_i	ith Row Total of Confusion Matrix
GT	Grand Total of Confusion Matrix
$RS1$	Random Seeding 1
$RS2$	Random Seeding 2
$BIRS$	Built in Random Seeding
α	Cutoff value

Epigraph

To achieve great things, two things are needed; a plan, and not quite enough time.

- Leonard Bernstein,

Chapter 1

Introduction

Non-negative matrix factorization (NMF) is an important technique in many applications, especially the analysis of modern single-cell RNA-seq data. In this work, we studied the properties of multiple variants of NMF, focusing on their performance and stability. We have tackled some of the issues and challenges about NMF which include rank estimation, classification, initialization, and stability. We have done a large amount of simulation study as well as real data investigation for comparing the performance of different factorization algorithms. During the course of the analysis, some techniques and ideas were developed. The thorough analyses provide useful information of advantages and disadvantages of different algorithms and some guidance on how to use them in practice.

1.1 Matrix Factorization

Matrix factorization is a powerful technique used in mathematics, computer science, and data science to represent a matrix as the product of two matrices. This technique has applications in a wide range of fields including recommender systems, image processing, signal processing, machine learning and recently in biological data including single-cell data analysis.

By breaking down a large matrix into smaller matrices, matrix factorization can help extract useful information including hidden dimension and patterns from the data. It is especially useful in the situations where the data matrix is sparse or has missing values. Matrix factorization has become increasingly popular in recent years, in part due to the rise of big data and the need for efficient methods to analyze and process it. In this context, matrix factorization is a powerful tool for reducing the dimensionality of data, revealing hidden structures, and extract insights.

There are many different matrix factorization algorithms have been developed over the years. They are mainly fall into two categories: One is traditional statistics based and the other is modern machine learning based.

The statistics based methods mainly include Factor Analysis (FA), Principal Component Analysis (PCA) and some of the variations.

Factor analysis is a statistical method to identify the underlying factors or latent variables that explain the correlations among a set of observed variables. These factors are hypothetical constructs that cannot be directly observed but can be inferred from the observed variables. Factor analysis seeks to reduce the complexity of a dataset by grouping variables that share common patterns of variance. It does this by identifying the minimum number of factors needed to explain the maximum amount of variability in the dataset.

Similarly, Principal Component Analysis (PCA) is a statistical method to reduce the dimensionality of a dataset while retaining as much of the original variation as possible. PCA achieves this by identifying a smaller set of linear combinations of variables (covariates), called principal components, which capture the most significant patterns and relationships in the data. These principal components are orthogonal and are sorted by the amount of variance they explain. By analysing these components, we can gain the insights into underlying structure of the data and identify the most important components that influence the variability in the dataset.

Mathematically, both FA and PCA are working with the variance-covariance matrix of the original dataset. Given $m \times n$ matrix $X_{m\times n}$ where m is the number of covariates and n is the number of samples. We find the variance-covariance matrix $V_{m\times m}$ and its eigenvalues. It is known that the variance-covariance matrix $V_{m\times m}$ is positive semi-definite. Furthermore as $V_{m\times m}$ is a real symmetric matrix, hence all its eigenvalues are real and non-negative, $\lambda_1 \geq \lambda_1 \geq ... \geq \lambda_m$.

We choose the first k ($k \leq m$) eigenvalues and their corresponding eigenvectors to form a loading matrix $L_{m\times k}$ such that $V_{m\times m} \approx L_{m\times k} \times L^T_{m\times k}$ (matrix factorization) [16]. We call these methods statistics based as the factorization is performed on the variance-covariance matrix instead of the original data matrix. Sometimes, statistical assumptions may be required for the analysis.

On the other hand, the machine learning based methods work directly with the original dataset and no statistical assumptions are needed. The machine learning based methods include Non-negative Matrix Factorization (NMF), Dictionary Learning (DL), and Latent Dirichlet Allocation (LDA) etc. In this category, NMF is a very important method as in many applications the datasets involved are non-negative in nature.

1.2 Non-negative Matrix Factorization

NMF is a machine learning technique that is used to factorize a non-negative matrix into two low-rank matrices. Unlike some other matrix factorization methods, NMF restricts the factor matrices to be non-negative, which makes the resulting factors easier to interpret in terms of the original data. The main idea behind the NMF is to represent a high-dimensional dataset as a linear combination of a small number of base vectors, with non-negative coefficients. It can be used to identify pattens and relationships in data that are not directly observed. NMF is a type of matrix factorization that seeks to factorize a non-negative matrix into two non-negative matrices, where the product of the two matrices is the approximation of the original matrix. The resulting matrices represent the underlying structure of the data and can be used to identify latent variables that explain the variation in the data. Here is a brief mathematical setting of NMF: Given a non-negative matrix $X_{m\times n}$ and a natural number r (the rank), find two non-negative matrices, $W_{m\times r}$ and $H_{r\times n}$ such that $X_{m\times n} \approx W_{m\times r} \times H_{r\times n}$. Or more formally, for a given loss function $l(\cdot)$ such that $l(X_{m\times n} - W_{m\times r} \times H_{r\times n})$ is minimized. [18] [19].

Let us look at the following motivation example: Assume that we have m words and n articles. Let $X_{m\times n}$ be a data matrix and the entry X_{ij} is the frequency of the i-th word in the j-th article. Hence, $X_{m\times n}$ is a non-negative matrix. Among all the n articles there might be r different hidden topics (say science, music, sports, etc.). We will have to estimate the r value and use NMF to find $W_{m\times r}$ and $H_{r\times n}$ such that $X_{m\times n} \approx W_{m\times r} \times H_{r\times n}$. Here both $W_{m\times r}$ and $H_{r\times n}$ are non-negative matrices. The $W_{m\times r}$ represents the relationship between words and topics and $H_{r\times n}$ represents the relationship between the topics and articles. This factorization has two utilities: (1) It reveals the number of topics (the r value), the topic of each document, and the membership of words belonging to the topics. (2) The r value is typically much smaller than m and n values. Hence it will lead to substantial dimension reduction.

One important application of NMF is the recommender system. A recommender system is a type of information filtering system that provides personalized recommendations to users based on their past behavior, preferences, and interests. It is commonly used in online platforms such as e-commerce websites, social me-

dia, music, and video streaming services. Recommender systems use different algorithms and techniques to analyze user data, such as browsing and purchase history, search queries, and ratings, to create user profiles and generate recommendations. Let $R_{m\times n}$ be a non-negative user verse item rating matrix where R_{ij} is the rating of ith user to jth item. An important step is to find a suitable value of r and decompose the rating matrix $R_{m\times n}$into two non-negative matrices $P_{m\times r}$ and $Q_{n\times r}$ such that $R_{m\times n} \approx P_{m\times r}Q^T_{n\times r}$. Here $P_{m\times r}$ is the users representation matrix (users verse latent factors) and $Q_{n\times r}$ is the items representation matrix (items verse latent factors). NMF is a commonly used method for this purpose [6].

Recently, NMF has gained popularity in the analysis of single-cell gene expression data. In the context of single-cell gene expression data, NMF is used to identify cell types and gene expression pattens. The gene expression data is represented as a matrix, where each row corresponds to a gene and each column corresponds to a single cell (or its transpose, where each row corresponds to a single cell and each column corresponds to a gene). NMF can be applied to this matrix to identify a smaller number (the rank) of gene sets, called metagenes, that capture the variation in the data. These metagenes represent group of genes that are co-expressed and may have functional relationship. Similar, based on the transposed matrix (cell verse gene), NMF can be used to identity metacells. One advantage of using NMF in single-cell gene expression analysis is that it can handle the sparsity and noise in the data. NMF can identify pattens in the data even when some genes are not expressed in certain cells or when there is variability in the expression levels due to experimental noise (robust).

Although NMF has shown promise in the analysis of single-cell gene expression data, there are challenges associated with its application to this type of data. One challenge is to choose appropriate rank value r (the number of metagenes or the number of metacells), as smaller r value may not capture most variation in the data, while too large r value may result in overfitting and reduce interpretability. In addition, the choice of the initialization values and optimization algorithm can affect the quality of the factorization and the resulting metagenes or metacells (non-robust).

Another challenge is the interpretation of the resulting metagenes. While NMF can identify patterns in the data and group co-expressed genes, it may be difficult to determine the biological significance of these gene sets without additional experimental validation. Additionally, the interpretation of the resulting cell types may be complex, as there can be substantial heterogeneity within a given cell population.

1.3 The Stability of Non-negative Matrix Factorization

Other than the challenges we mentioned in Section 1.2, the stability of non-negative matrix factorization has drowned large attention from the researchers in this area. One of the concerns is the impact of the sample size. This is one of the issues we have discussed in this thesis.

A more common concern of NMF stability is the impact of initialization. Although the NMF algorithms guarantee the convergence of the iterations, however, they may convergence to some local minimums rather than the global minimum. Depend on the choice of the initial W and H values, the NMF procedure may convergence to different minimums (non-robust). This pitfall is due to the non-convexity of the NMF loss functions [8] [12].

1.4 Organization

This thesis is organized in the following way:
In Chapter 2, we provide some basics about NMF, some preliminaries needed for this thesis. We have proposed two new random seeding methods and also some new ideas about rank estimation for NMF.

In Chapter 3, we have designed two artificial data sets that imitate the real single-cell gene expression data and with some known (desired) features. An extensive amount of simulation study is done for the comparison over the initializations and over the different algorithms for rank estimation, classification, and accuracy of the matrix approximation through its Euclidean distance.

In Chapter 4, we have carried out all the simulation study we have designed in Chapter 3 to a real single-cell gene expression data. We have also briefly looked at the confusion matrices of single-cell gene expression data.

In Chapter 5, we first have studied the stability of NMF over the sample size on two of our simulated data sets and then on the single-cell gene expression data. We have also compared the performance of our proposed random seedings with the package build in random seeding. Then we have had a discussion on the initialization and stability of NMF.

Chapter 6 is a brief summary of the whole thesis, and some further studies could be done in the area that related to this thesis.

Chapter 2

Preliminaries and New Ideas

In this chapter, we provide some basic knowledge about NMF, some preliminaries needed for this thesis, and also some new ideas about rank estimation and initialization for NMF.

2.1 NMF Algorithm and Initialization

In this section, we give technical details of NMF, a brief description of eight NMF algorithms under comparison, and two default seeding strategies (as built-in functions in the NMF package). We also introduce two of our own random seeding approaches.

2.1.1 NMF Algorithms

In Chapter 1, we have mentioned a few examples that involve non-negative matrix factorization. That is, for a given non-negative matrix $X_{m\times n}$ and a natural number r (the rank), we have to find two non-negative matrices, $W_{m\times r}$ and $H_{r\times n}$ such that $l(X_{m\times n} - W_{m\times r} \times H_{r\times n})$ is minimized. The pioneer work in this area was done by Lee and Seung [18],[19]. In these two papers, they considered the following two loss functions. One loss function is based on the squared Euclidian distance and the other is based on the divergence between two matrices. Here are the definitions:

Definition 1: Given two non-negative matrices $A_{m\times n}$ and $B_{m\times n}$. Then

$$ED(A,B) = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(A_{ij} - B_{ij})^2}$$

is the Euclidean distance between $A_{m\times n}$ and $B_{m\times n}$.

Definition 2: Given two non-negative matrices $A_{m\times n}$ and $B_{m\times n}$. Then

$$D(A||B) = \sum(A_{ij}log\frac{A_{ij}}{B_{ij}} - A_{ij} + B_{ij})$$

is the Kullback-Leibler divergence of A from B.

Lee and Seung [18],[19] considered two formulations of NMF as the following two optimization problems:

Problem 1: For given matrix $X_{m\times n}$ and a natural number r,

minimize $l_1(X_{m\times n} - W_{m\times r} \times H_{r\times n}) = ED(X_{m\times n} - W_{m\times r} \times H_{r\times n})^2$
with respect to $W_{m\times r}$ and $H_{r\times n}$, subject to the constraints $W_{m\times r}$, $H_{r\times n} \geq 0$.

Problem 2: For given matrix $X_{m\times n}$ and a natural number r,

minimize $l_2(X_{m\times n} - W_{m\times r} \times H_{r\times n}) = D(X_{m\times n}||W_{m\times r} \times H_{r\times n})$
with respect to $W_{m\times r}$ and $H_{r\times n}$, subject to the constraints $W_{m\times r}$, $H_{r\times n} \geq 0$.

To solve these two optimization problems, they proposed two multiplicative update rules and proved the following two convergence theorems.

Theorem 1:
The squared Euclidean distance $ED(X_{m\times n} - W_{m\times r} \times H_{r\times n})^2$ is nonincreasing under the update rules

$$H_{kj} := H_{kj}\frac{(W^TX)_{kj}}{(W^TWH)_{kj}}, \quad W_{ik} := W_{ik}\frac{(XH^T)_{ik}}{(WHH^T)_{ik}} \quad (1)$$

where $i = 1,\ldots,m; j = 1,\ldots,n; k = 1,\ldots,r$.
The squared Euclidean distance is invariant under these updates if and only if $W_{m\times r}$ and $H_{r\times n}$ are at a

stationary point of the distance.

We include this result here as we are using this update rules for some of our simulation studies.
For coding, we will use

$$H_{kj} := H_{kj} \frac{(W^T X)_{kj}}{(W^T W H)_{kj} + \epsilon}, \quad W_{ik} := W_{ik} \frac{(X H^T)_{ik}}{(W H H^T)_{ik} + \epsilon} \qquad (2)$$

where $\epsilon = 10^{-9}$ instead of (1) to avoid division by zero.

Theorem 2:
The divergence $D(X_{m\times n}||W_{m\times r} \times H_{r\times n})$ is nonincreasing under the update rules

$$H_{kj} := H_{kj} \frac{\sum_i W_{ik} X_{ij}/(WH)_{ij}}{\sum_s W_{sk}}, \quad W_{ik} := W_{ik} \frac{\sum_i H_{kj} X_{ij}/(WH)_{ij}}{\sum_t H_{kt}} \qquad (3)$$

where $i = 1, \ldots, m; j = 1, \ldots, n; k = 1, \ldots, r; s = 1, \ldots, m; t = 1, \ldots, n$.

The divergence is invariant under these updates if and only if $W_{m\times r}$ and $H_{r\times n}$ are at a stationary point of the divergence.

After Lee and Seung [19], there are many other non-negative matrix factorization algorithms have been developed. Some of methods improved the convergence rate, some considered difference optimization criteria while the others tackled stability issues. Some of the software packages have been developed recently as well.

For our simulation study and application, we are using a NMF package by Gaujoux [10]. We are using eight build-in NMF algorithms named Lee, Brunet, Frobenius, KL, nsNMF, Offset, snmf/r, and snmf/l. The following are the brief description of these eight algorithms.

Lee: Standard NMF. Based on Euclidean distance, it uses simple multiplicative updates [19].

Brunet: Standard NMF. Based on Kullback-Leibler divergence, it uses simple multiplicative updates from [19] , enhanced to avoid numerical underflow [4].

Frobenius: NMF based on the Frobenius norm. It penalizes large errors in individual matrix elements, which aligns with the goal of finding a good overall approximation.

KL: The KL divergence is used as a cost or objective function to quantify the dissimilarity between the original matrix and its NMF approximation.

nsNMF: Non-smooth NMF. Uses a modified version of Lee and Seung's multiplicative updates for Kullback-Leibler divergence to fit a extension of the standard NMF model. It is meant to give sparser results [23].

Offset: Uses a modified version of Lee and Seung's multiplicative updates for Euclidean distance, to fit a NMF model that includes an intercept [1].

snmf/r, snmf/l : Alternating Least Square (ALS) approach. It is meant to be very fast compared to other approaches [17].

2.1.2 Initialization of NMF

The NMF algorithms need to be initialized with a seed (i.e. a value for W_0 and/or H_0), from which to start the iteration process. Because there is no global minimization algorithm due to the non-convexity of the loss function, and due to the problem's high dimensionality, the choice of the initialization is in fact very important to ensure meaningful results.

There are many different ways to choose the initial W_0 and H_0 matrices. One of the popular categories is deterministic seedings. There is a NMF package build-in seeding, called ICA seeding which uses the result of an Independent Component Analysis (ICA) [22].

Another popular category is random seedings. Here are two common approaches of the random seeding:

1. For a given matrix $X_{m\times n}$ and r value, randomly choose r columns from $X_{m\times n}$ to form W_0 and randomly choose r rows from $X_{m\times n}$ to form H_0.

2. For a given matrix $X_{m\times n}$ and r value, randomly choose $m \times r$ numbers from the range of all entries of

matrix $X_{m\times n}$ to form W_0 and randomly choose $r \times n$ numbers from the range of all entries of matrix $X_{m\times n}$ to form H_0.

The advantage of random seedings is cost nearly nothing (running time) to construct the W_0 and H_0 matrices. One of the pitfalls is that there is little relevance to the matrix $X_{m\times n}$ when we form the initial W and H matrices. Hence, we are going to introduce two different random seedings with more merits and more relevance to the matrix $X_{m\times n}$. The following are the necessary preparations needed to introduce our two new random seedings.

Claim 1: Let $X_{m\times n}$ be a non-negative matrix. For $r < min\{m, n\}$, let $W_{m\times r}$ and $H_{r\times n}$ be two constant matrices such that $W_{ik} = H_{kj} = c, c > 0; i = 1, ..., m; j = 1, ..., n; k = 1, ..., r$. Then the $ED(X, W \times H)$ is a function of c which is minimized when

$$c = \sqrt{\frac{\sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij}}{mnr}}.$$

Proof. It is clear that $(W \times H)_{m\times n}$ is also a constant matrix with $(W \times H)_{ij} = rc^2$. Hence

$$ED(X, W \times H) = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(X_{ij} - rc^2)^2}$$

Let $SED(X, W \times H)$ be the squared Euclidean distance. Then

$$SED(X, W \times H) = \sum_{i=1}^{m}\sum_{j=1}^{n}(X_{ij} - rc^2)^2$$

Take derivative with respect to c, we have

$$\frac{dSED(X, W \times H)}{dc} = \sum_{i=1}^{m}\sum_{j=1}^{n} 2(X_{ij} - rc^2)(-2rc) := 0$$

This implies that

$$\sum_{i=1}^{m}\sum_{j=1}^{n}(X_{ij} - rc^2) = 0,$$

and

$$\sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij} - mnrc^2 = 0.$$

Hence, we have

$$c = \sqrt{\frac{\sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij}}{mnr}}.$$

The same c will minimize the $ED(X, W \times H)$ as well. □

Definition 3. Let $M_{m\times n}$ be a non-negative matrix. Then

$$||M||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} M_{ij}^2}$$

is the Frobenius norm of matrix M.

Claim 2: Let $X_{m\times n}$ be a non-negative matrix. For $r < min\{m, n\}$, let $W_{m\times r}$ and $H_{r\times n}$ be two constant matrices such that $W_{ik} = H_{kj} = c, c > 0; i = 1, ..., m; j = 1, ..., n; k = 1, ..., r$. Then $||X||_F = ||W \times H||_F$ when

$$c = \sqrt{\frac{||X||_F}{r\sqrt{mn}}} = \sqrt{\frac{\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij}^2}}{r\sqrt{mn}}}.$$

Proof. It is clear that $(W \times H)_{m\times n}$ is also a constant matrix with $(W \times H)_{ij} = rc^2$. Hence

$$||W \times H||_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(rc^2)^2} = \sqrt{mnr^2c^4} = r\sqrt{mn}c^2.$$

Let $||X||_F = ||W \times H||_F$. we have

$$||X||_F = r\sqrt{mn}c^2.$$

Solve for c, we find

$$c = \sqrt{\frac{||X||_F}{r\sqrt{mn}}} = \sqrt{\frac{\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij}^2}}{r\sqrt{mn}}}.$$

□

One of the idea of initializations is to choose W_0 and H_0 matrices so that X and $W_0 \times H_0$ are relatively close to each other. By Claim 1, we know that $ED(X, W \times H)$ is minimized if we choose the c value in Claim 1 to form constant matrices W and H among all the other constant matrices W and H. However, if the initial W and H matrices are constant matrices, some of the matrix factorization algorithms may not run. Hence, we are going to introduce a random seeding based on the c value in Claim 1.

Random Seeding 1 (RS1)

For a given matrix $X_{m\times n}$ and r value, let

$$c_1 = \sqrt{\frac{\sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij}}{mnr}}$$

and $\delta_1 = min\{c_1, d_1\}$ where d_1 is some chosen positive number. We then construct $W_{m\times r}$ by $W_{ik} \sim Uniform(c_1 - \delta_1, c_1 + \delta_1)$ (to ensure non-negativity) $i = 1, ...m, k = 1, ...r$ and construct $H_{r\times n}$ by $H_{kj} \sim Uniform(c_1 - \delta_1, c_1 + \delta_1)$ $k = 1, ...r, j = 1, ..., n$.

We could also have a random seeding based on the c value in Claim 2.

Random Seeding 2 (RS2)

For a given matrix $X_{m\times n}$ and r value, let

$$c_2 = \sqrt{\frac{||X||_F}{r\sqrt{mn}}} = \sqrt{\frac{\sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij}^2}}{r\sqrt{mn}}}$$

and $\delta_2 = min\{c_2, d_2\}$ where d_2 is some chosen positive number. We then construct $W_{m\times r}$ by $W_{ik} \sim Uniform(c_2 - \delta_2, c_2 + \delta_2)$ $i = 1, ...m, k = 1, ...r$ and construct $H_{r\times n}$ by $H_{kj} \sim Uniform(c_2 - \delta_2, c_2 + \delta_2)$ $k = 1, ...r, j = 1, ..., n$.

2.2 Classification and Rank Estimation

In this section, we will first discuss the classification and confusion matrix and then the criteria of rank estimation.

2.2.1 Classification and Confusion Matrix

The gene expression data is represented as a matrix, where each row corresponds to a gene and each column corresponds to a single cell (gene-centered) or its transpose, where each row corresponds to a single cell and each column corresponds to a gene (cell-centered). In our application, the single-cell data is cell-centered, that is the matrix X_{mn} is a cell verse gene matrix.

Assuming that W and H are available, each column $\mathbf{x}_i$ of X can be approximated as $\mathbf{x}_i \approx W \times \mathbf{h}_i$, thus W

is referred to as the basis matrix and H as the coefficient matrix.

For each given r, we apply NMF to the matrix X. The basis matrix W returned by NMF has size $m \times r$, while m denotes the number of cells and r is the rank. Then we normalize the basis matrix to make each row sum to unity and obtain $\bar{W}$. For classifying of the single cells, we use the largest metacell contribution in $\bar{W}$. In this case, all the cells will be classified.

The confusion matrix is constructed by intersecting the metacell classification obtained via NMF against the known biological classification (e.g., the cell types). Hence, a confusion matrix M of dimension $k \times r$ means that there are k biological subgroups (cell types) and r metacells (hidden factors). The confusion matrix contains useful information for further analysis and biological interpretation.

2.2.2 Criteria for Rank Estimation

One of the important and challenge issues in NMF is the rank estimation. There are different ways of estimating r value. For instance, approaches based on cophenetic correlation or residual sum of squares have been proposed to estimate the proper rank value in NMF decomposition [4] [15]. Shao and Hofer [24] found that these approaches successfully uncovered the correct rank for some data sets but failed to find the appropriate rank value for noisy, high-dimensional RNAseq data. Hence, they suggested using cell sparseness and information gain to estimate the rank r value.

In this thesis, we will consider five different criteria and compare their performances in rank estimation. Other than cell sparseness and information gain, we will consider three different criteria for rank estimation. We are now describing five criteria in details.

1. Criterion Based on Cell Sparseness

For each given r, we calculate the i-th cell sparseness for each cell, i, as

$$S_i = \sqrt{\sum_{j=1}^{r} (\bar{W}_{ij})^2}$$

and normalized i-th cell sparseness as

$$NS_i = (S_i - a)/(1 - a), \ where \ a = \frac{1}{\sqrt{r}}.$$

Hence we have,

$$0 \leq NS_i \leq 1.$$

A higher cell sparseness indicates a lower ambiguity in classification results. Thus, a significant drop of cell sparseness would suggest at a certain rank the cell population should not be further divided.

2. Criterion Based on Information Gain

First, we define the entropy of each cell i as

$$E_i = -\sum_{j=1}^{r} \bar{W}_{ij} \log_2 \bar{W}_{ij}$$

In a completely non-informative classification, cell i would have the same contribution $1/r$ in row i of $\bar{W}$. Thus the entropy is

$$-\sum_{j=1}^{r} \frac{1}{r} log_2 \frac{1}{r} = -\sum_{j=1}^{r} \frac{1}{r} log_2 r^{-1} = \sum_{j=1}^{r} \frac{1}{r} log_2 r = log_2 r.$$

Hence, we define the information gain of cell i, as

$$IG_i = log_2 r - \sum_{j=1}^{r} \bar{W}_{ij} \log_2 \bar{W}_{ij}.$$

and the total information gain as

$$TIG = \sum_{i=1}^{m} (log_2 r - \sum_{j=1}^{r} \bar{W}_{ij} log_2 \bar{W}_{ij}) = m log_2 r - \sum_{i=1}^{m} \sum_{j=1}^{r} \bar{W}_{ij} \log_2 \bar{W}_{ij}$$

and the normalized total information gain as

$$NTIG = \frac{TIG}{mlog_2 r} = 1 - \frac{1}{mlog_2 r} \sum_{i=1}^{m} \sum_{j=1}^{r} \bar{W}_{ij} log_2 \bar{W}_{ij}, \quad 0 \leq NTIG \leq 1.$$

The normalized total information gain indicates how well NMF performs in unambiguously classifying cells. We are interested in a clear drop of information gain and use that as an indication to estimate the rank value.

3. Criterion Based on Euclidean Distance

The Euclidean distance between $X_{m\times n}$ and $W_{m\times r} \times H_{r\times n}$ measures the accuracy of the approximation of the matrix factorization. Mathematically we know that the Euclidean distance of the approximation is decreasing when r is increasing. See Claim 3 below. Hence, when there is no clear decline in Euclidean distance provides an indication of proper rank r value. That is, further increasing r value will not improve too much the accuracy of the approximation.

Claim 3: Given $X_{m\times n} \geq 0$ *and* $1 \leq r \leq n$–1. *Let* $W_{m\times r} \geq 0$ *and* $H_{r\times n} \geq 0$; $W_{m\times(r+1)} \geq 0$ *and* $H_{(r+1)\times n} \geq 0$ be any non-negative matrices with specified dimensions. Then, we have

$$min_{W_{m\times(r+1)}, H_{(r+1)\times n}} \{ED(X_{m\times n} - W_{m\times(r+1)} \times H_{(r+1)\times n})\} \leq min_{W_{m\times r}, H_{r\times n}} \{ED(X_{m\times n} - W_{m\times r} \times H_{r\times n})\}$$

Proof. For any $W_{m\times r} \geq 0$, let $W^*_{m\times(r+1)} \geq 0$ be the matrix formed by $W_{m\times r}$ add an additional $(r+1)th$ column with all the components of zero. Similarly, for any $H_{r\times n} \geq 0$, let $H^*_{(r+1)\times n} \geq 0$ be the matrix formed by $H_{r\times n}$ add an additional $(r+1)th$ row with all the components of zero. Then, we have,

$$\{W_{m\times(r+1)} \times H_{(r+1)\times n} | W_{m\times(r+1)} \geq 0 \; and \; H_{(r+1)\times n} \geq 0\}$$

$$\supseteq \{W^*_{m\times(r+1)} \times H^*_{(r+1)\times n} | W^*_{m\times(r+1)} \geq 0 \; and \; H^*_{(r+1)\times n} \geq 0\}$$

$$= \{W_{m\times r} \times H_{r\times n} | W_{m\times r} \geq 0 \; and \; H_{r\times n} \geq 0\}$$

Therefore, we have

$$min_{W_{m\times(r+1)}, H_{(r+1)\times n}} \{ED(X_{m\times n} - (W_{m\times(r+1)} \times H_{(r+1)\times n}))\}$$

$$\leq min_{W^*_{m\times(r+1)}, H^*_{(r+1)\times n}} \{ED(X_{m\times n} - (W^*_{m\times(r+1)} \times H^*_{(r+1)\times n}))\}$$

$$= min_{W_{m\times r}, H_{r\times n}}\{ED(X_{m\times n} - (W_{m\times r} \times H_{r\times n}))\}.$$

□

4. Criterion Based on the Counts of Classified Cells

Sometimes, we only want to classify the cells when the largest metacell contribution in $\bar{W}$ is above some prespecified (cut-off) value α, say, $\alpha = 0.5$ $(0.6, 0.7, 0.8,$ *or* $0.9)$. Now for a given α, we find the counts of classified cells over a range of different r values. We suggest using the r value with the highest percentage (or counts) of classified cells as the proper (estimated) rank.

5. Criterion Based on the Degree of Confusion of Confusion Matrix

We are now going to introduce a numerical quantity to measure the degree of confusion for any confusion matrix. Consider a confusion matrix M with dimension $k \times r$. We normalize the confusion matrix to make each row sum to unity and obtain $\bar{M}$.

For each row i of $\bar{M}$, we define the row i sparseness as

$$\sqrt{\sum_{j=1}^{r}(\bar{M}_{ij})^2}$$

This quantity is bounded between $\frac{1}{\sqrt{r}}$ (when all the entries of row i are equal to $\frac{1}{r}$) and 1 (when there is only one non-zero entry with value 1 in row i). That is

$$\frac{1}{\sqrt{r}} \leq \sqrt{\sum_{j=1}^{r}(\bar{M}_{ij})^2} \leq 1$$

Hence we have,

$$0 \leq \sqrt{\sum_{j=1}^{r}(\bar{M}_{ij})^2} - \frac{1}{\sqrt{r}} \leq 1 - \frac{1}{\sqrt{r}}$$

and

$$0 \leq \frac{\sqrt{\sum_{j=1}^{r}(\bar{M}_{ij})^2} - \frac{1}{\sqrt{r}}}{1 - \frac{1}{\sqrt{r}}} \leq 1.$$

We define the following quantity as the degree of confusion for row i,

$$DC_i = 1 - \frac{\sqrt{\sum_{j=1}^{r}(\bar{M}_{ij})^2} - \frac{1}{\sqrt{r}}}{1 - \frac{1}{\sqrt{r}}}.$$

Note that $0 \leq DC_i \leq 1$ as well.

Let

RT_i - be the number of total counts in row i;

GT — be the grand total counts of matrix M (the summation of all the entries of M).

We define the degree of confusion of matrix M, DC as the weighted average of DC_i,

$$DC = \sum_{i=1}^{k} \frac{RT_i}{GT} DC_i = \frac{1}{GT} \sum_{i=1}^{k} RT_i \times DC_i.$$

It is clear that $0 \leq DC \leq 1$.

With this measure (DC), for the confusion matrix M with one non-zero entry in every row will have $DC = 0$. With all the entries in a row having the same value for all the rows will have $DC = 1$. We now can use DC to estimate the r value by finding the r with the smallest DC value (or for the r value before a big increasing in DC value take place) as the proper rank value.

We are going to run NMF by using eight algorithms, two seeding methods, and five rank estimation criteria to our two simulated data sets and a real single-cell gene expression data set. The results, comparisons, and conclusions will be in Chapter 3 and Chapter 4.

Chapter 3

Assessment on Simulated Data

As we mentioned before, one of the challenges for NMF is the rank estimation. The estimated rank r value may depend on the seedings and also depend on the factorization algorithms. With the configuration of single-cell data in mind, we have designed and generated two simulated data here for our investigation of NMF and its stability. We use these two data sets to compare the performances among different seedings and factorization algorithms. The performance criteria include rank estimation, classification, and accuracy of the matrix approximation. We refer Simulation I as the simulation study based on our First Simulated Data (FSD) and Simulation II as the simulation study based on our Second Simulated Data (SSD).

We applied the rank estimation and classification methods in Chapter 2 to our two simulated data. The results and conclusions are summarized in Section 3.1.2 and Section 3.2.2.

3.1 Simulation I

3.1.1 The Design of the First Simulated Data (FSD)

Our first simulated data, stored in a matrix $X_{m\times n}$ has size $m = 400$ and $n = 5000$. We consider each row as "individual cell" and each column as "gene expression". We further consider the first 50 rows, row 1 – row 50, as group 1 (cell type 1); row 51 – row 100 as group 2 (cell type 2); row 101 – row 150 as group 3 (cell type 3); row 151 – row 200 as group 4 (cell type 4); row 201 – row 250 as group 5 (cell type 5), row 251 – row 300 as group 6 (cell type 6); row 301 – row 350 as group 7 (cell type 7), row 351 – row 400 as group 8 (cell type 8). The matrix $X_{m\times n}$ is designed with the following features:

1. All the rows in the same group are highly correlated.
2. The rows from different groups are not correlated.

With the above features, we should expect the estimated rank value r = 8 when we run the matrix factorization procedures. Hence, our simulation study could compare and identify which algorithms, seeding methods, and estimation criteria are better than the others.

The way of design a $X_{m\times n}$ is not unique. The following is the way of our design:

The Design of Group 1:

Step 1

Row 1 has 45% (2250) components with values 0 and 55% (2750) components with values randomly selected between 1 and 10 with different weights (see the table below for the weights). Shuffle the components in random order.

Counts	1	2	3	4	5	6	7	8	9	10
Weights %	55	20	10	5	4	2	1.6	1.2	0.8	0.4

In the above table, the weights of different counts are the imitation of a real single-cell gene expression data. Also dimension of our simulated data are compatible with real single-cell gene expression data. Along with the amount of zero entries in the matrix, our simulated data is sparse, non-negative and with high dimension which are the common natures of real single-cell gene expression data.

Step 2

Let Row 1 = Row 2 = ... = Row 50

(Duplicate row 2, row 3, ..., row 50. At this moment, row 1, row 2, ..., row 50 are all the same).

Step 3

Change all the rows (row 1, row 2, ..., row 50):

For each row i, ($i = 1, ..., 50$), randomly choose 10% (500) numbers between 0 and 9 (uniformly) and use these values to replace 10% (500) randomly chosen components in row i. (Now, all the 50 rows in group 1 are similar but different from each other).

The design of the other Seven Groups:

Do the same thing for Group 2, Group 3, ..., Group 8 as we did for Group 1 except the following changes:

Group 2

Each row in Group 2, has 50% (2500) components with values 0 and 50% (2500) components with values randomly selected between 1 and 10 with different weights (use the same table for the weights).

Group 3

Each row in Group 3, has 55% (2750) components with values 0 and 45%(2250) components with values randomly selected between 1 and 10 with different weights (use the same table for the weights).

Group 4

Each row in Group 4, has 60% (3000) components with values 0 and 40% (2000) components with values randomly selected between 1 and 10 with different weights (use the same table for the weights).

Group 5

Each row in Group 5, has 65% (3250) components with values 0 and 35% (1750) components with values randomly selected between 1 and 10 with different weights (use the same table for the weights).

Group 6

Each row in Group 6, has 70% (3500) components with values 0 and 30% (1500) components with values randomly selected between 1 and 10 with different weights (use the same table for the weights).

Group 7

Each row in Group 7, has 75% (3750) components with values 0 and 25% (1250) components with values randomly selected between 1 and 10 with different weights (use the same table for the weights).

Group 8

Each row in Group 8, has 80% (4000) components with values 0 and 20% (1000) components with values randomly selected between 1 and 10 with different weights (use the same table for the weights).

Note that the change of the percentage of zero counts in each group is to make some difference for each group. Which is arbitrarily and not unique.

The matrix $X_{m\times n}$ is in Supplementary 1 and its covariance matrix is in Supplementary 2.
When we look at the covariance matrix of $X_{m\times n}$, we find that our designed data really achieves our purpose. That is: (1) all the rows in the same group are highly correlated; (2) the rows from different groups are not correlated.

3.1.2 The Results and Conclusions of Simulation I

We are using a NMF package, Algorithms and Framework for Non-negative Matrix Factorization, to carry out our simulation study. We choose two seeding methods (a fixed random seeding and ICA seeding) and eight different factorization algorithms to run the simulations. We use a fixed random seeding for the sake of reproducibility and comparison purpose. The eight factorization algorithm are: Lee, Brunet, nsNMF, KL, Frobenius, Offset, Snmf/r, and Snmf/l. For detailed description of these algorithms, see Section 2.1.1 and [10].

As we know the way how the first simulated data is designed, we should expect the rank value r is 8. To save space, we only present the tables for ICA seeding and leave the tables for random seeding in Supplementary 5.

In Shao, C. and Hofer, T. [24], they used the cell-sparseness and information gain to estimate the rank r values for their data sets and found quite successful. We have also used cell-sparseness and information gain to our first simulated data for rank r value between 4 and 10. The results are presented in the following Table 3.1 and Table 3.2.

Table 3.1: Mean Normalized Spareness and (SD) for Different Algorithms and Ranks for FSD

Method \ Rank	4	5	6	7	8	9	10
Lee	0.54 (0.46)	0.65 (0.44)	0.75 (0.39)	0.85 (0.29)	0.94 (0.02)	0.91 (0.07)	0.77 (0.14)
Brunet	0.90 (0.08)	0.89 (0.10)	0.92 (0.09)	0.95 (0.05)	0.96 (0.04)	0.95 (0.06)	0.87 (0.10)
nsNMF	0.94 (0.08)	0.93 (0.10)	0.97 (0.06)	0.96 (0.09)	0.99 (0.03)	0.90 (0.16)	0.84 (0.16)
KL	0.88 (0.10)	0.87 (0.13)	0.93 (0.05)	0.94 (0.06)	0.96 (0.04)	0.89 (0.12)	0.85 (0.13)
Frobenius	0.54 (0.46)	0.65 (0.44)	0.75 (0.39)	0.85 (0.29)	0.94 (0.02)	0.85 (0.13)	0.83 (0.11)
Offset	0.79 (0.27)	0.82 (0.25)	0.83 (0.26)	0.89 (0.16)	0.94 (0.03)	0.89 (0.07)	0.82 (0.15)
Snmf/r	0.55 (0.45)	0.65 (0.43)	0.74 (0.38)	0.82 (0.28)	0.90 (0.02)	0.86 (0.05)	0.83 (0.07)
Snmf/l	0.55 (0.45)	0.65 (0.43)	0.75 (0.39)	0.85 (0.29)	0.94 (0.02)	0.92 (0.04)	0.91 (0.04)

Table 3.2: Information Gain for Different Algorithms and Ranks for FSD

Method \ Rank	4	5	6	7	8	9	10
Lee	0.891	0.903	0.920	0.941	0.963	0.952	0.911
Brunet	0.964	0.958	0.965	0.973	0.979	0.977	0.951
nsNMF	0.980	0.976	0.989	0.986	0.995	0.971	0.952
KL	0.958	0.952	0.970	0.971	0.979	0.958	0.947
Frobenius	0.891	0.903	0.918	0.940	0.964	0.937	0.927
Offset	0.947	0.948	0.943	0.957	0.965	0.948	0.925
Snmf/r	0.892	0.900	0.913	0.927	0.944	0.926	0.911
Snmf/l	0.893	0.903	0.920	0.941	0.965	0.957	0.949

Table 3.1 provides the average cell-sparseness and the standard deviation (in parentheses). We can see that for all eight matrix factorization algorithms the cell-sparseness has the highest values and smallest standard deviation values when $r = 8$. Similarly, Table 3.2 shows that for all eight matrix factorization algorithms, the information gain has the highest values when $r = 8$ as well. When we consider the higher cell-sparseness and information gain as the criteria, both criteria indicate that $r = 8$ is the proper rank value.

We now look at the rank estimation approach based on percentage (or counts) of classified cells (see Section 2.2.2). For each matrix factorization algorithms, we calculate the counts of classified cells for the cut-off value $\alpha = 0.5, 0.6, 0.7, 0.8,$ *and* 0.9. We only present Table 3.3 for Lee algorithm and leave all the other seven tables in Supplementary 5.

Table 3.3: Number of Classified Cells for Given Rank and Cutoff, Lee Algorithm, for FSD

Cutoff \ Rank	4	5	6	7	8	9	10
0.5	211	250	300	350	400	400	398
0.6	200	250	300	350	400	396	378
0.7	200	250	300	350	400	393	325
0.8	200	250	300	350	400	388	242
0.9	200	250	300	350	400	356	145

The sample size of first simulated data is $n = 400$. There is a very clear indication that $r = 8$ is the proper rank value. Other seven matrix factorization algorithms also give the same rank estimation $r = 8$. However, in turns of clarity, Lee and Frobenius give the most clear indication followed by Offset, Snmf/r, and Snmf/l and then Brunet, nsNMF, and KL.

Another idea of rank estimation is to use the degree of confusion of the confusion matrix. For all eight matrix factorization algorithms, we have done the classifications and found the confusion matrices and then the degree of confusion of the confusion matrix. The results are in Table 3.4.

Table 3.4: Degree of Confusions for Different Algorithms and Ranks for FSD

Rank	4	5	6	7	8	9	10
Lee	0.096	0.094	0.016	0.000	0.000	0.038	0.040
Brunet	0.000	0.000	0.000	0.000	0.000	0.052	0.000
nsNMF	0.000	0.000	0.000	0.000	0.000	0.000	0.018
KL	0.000	0.000	0.000	0.000	0.000	0.000	0.004
Frobenius	0.096	0.092	0.020	0.000	0.000	0.000	0.055
Offset	0.108	0.047	0.075	0.012	0.000	0.004	0.000
Snmf/r	0.000	0.038	0.012	0.000	0.000	0.004	0.007
Snmf/l	0.000	0.076	0.012	0.004	0.000	0.004	0.028

From Table 3.4, we can see that the degree of confusion is zero when rank $r = 8$ for all eight matrix factorization algorithms. For some algorithms, the degree of confusion also equal to zero when the rank r is less than 8. The reason for that is because of the fact that all the rows in each group from first simulated data are highly correlated. Hence, they will stay together in classification process. Collectively, when we look at the degree of confusion for all eight methods, it will suggest that the rank value $r = 8$.

As we mentioned in Section 2.2.2, we can also use the Euclidean distance to estimate the rank value r. For eight matrix factorization algorithms, we have calculated the Euclidean distance between $X_{m\times n}$ and

$W_{m\times r} \times H_{r\times n}$ for rank r value between 4 and 10. It is very clear that the Euclidean distances decreasing significantly when r increasing from 4 to 8 and from $r = 8$ and above, the Euclidean distances remain more or less the same for all eight matrix factorization algorithms. These facts indicate that further increasing r value beyond 8 do not improve the accuracy of the approximation of matrix factorization. Hence $r = 8$ is the proper rank value. See Table 3.5 below.

Table 3.5: Euclidean Distance for Different Algorithms and Ranks for FSD

Method \ Rank	4	5	6	7	8	9	10
Lee	1976.14	1849.25	1724.67	1612.46	1512.66	1510.31	1507.91
Brunet	2008.75	1883.10	1767.66	1619.39	1514.15	1512.13	1514.35
nsNMF	2140.37	2036.33	1933.50	1824.66	1730.49	1740.30	1744.80
KL	2027.93	1899.19	1791.87	1626.47	1514.13	1514.44	1514.55
Frobenius	1976.14	1849.25	1724.66	1612.45	1512.67	1512.01	1507.87
Offset	2083.92	1967.88	1764.40	1634.21	1512.81	1510.44	1510.92
Snmf/r	1976.14	1849.25	1724.69	1612.48	1512.72	1510.30	1507.99
Snmf/l	1976.14	1849.24	1724.66	1612.44	1512.66	1510.36	1507.96

We have also carried out the same simulation study for a fixed random seeding. The numerical results are kept in Supplementary 5 and the summary of them is in Figure 3.1. The findings are summarized as follows.

1. Based on the cell-sparseness

We found that for all eight matrix factorization algorithms the cell-sparseness has the highest values when $r = 8$ and also has the smallest standard deviation values when $r = 8$.

2. Based on the information gain

Similarly, we found that for all eight matrix factorization algorithms, the information gain has the highest value when $r = 8$ as well.

3. Based on the counts of classified cells

There is a very clear indication that $r = 8$ is the proper rank value by using all eight matrix factorization algorithms. In turns of clarity, Lee and Frobenius give the most clear indication followed by Offset, Snmf/r, and Snmf/l and then Brunet, nsNMF, and KL.

4. Based on the degree of confusion of confusion matrix

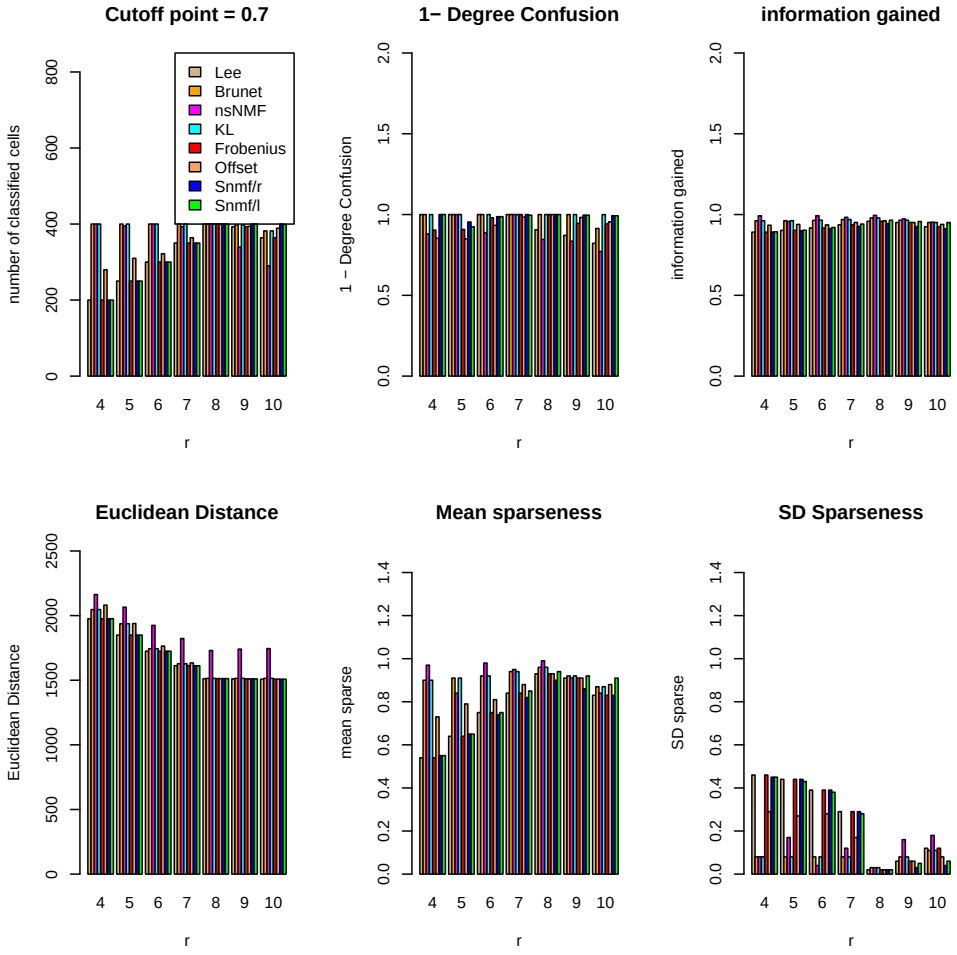
Similar to the ICA seeding case, the estimated rank value is $r = 8$.

5. Based on the Euclidean distance

For eight matrix factorization algorithms, we have calculated the Euclidean distance between $X_{m\times n}$ and $W_{m\times r} \times H_{r\times n}$ for rank r value between 4 and 10. It is very clear that the Euclidean distances decreasing significantly when r increasing from 4 to 8 and from $r = 8$ and above, the Euclidean distances remain more or less the same for all eight matrix factorization algorithms.

All the above conclusions are the same as when we were using ICA seeding. Hence, for our first simulated data, we find that two different seedings, eight different matrix factorization algorithms, and five different rank estimation criteria have reached the same conclusion. That is, the proper rank for the first simulated data is $r = 8$ although there are some minor differences in classification. The reason for this perfect result is due to the perfectly designed data set. When the data set is not that ideal, the conclusions would not be that great anymore. We will see this in the next section.

Figure 3.1: Rank Estimation for FSD using Random Seeding



3.2 Simulation II

3.2.1 The Design of the Second Simulated Data (SSD)

We have designed our second simulated data, stored in matrix $Y_{m\times n}$, in a similar manner as the first simulated data. However, the second simulated data is designed to have the following features instead:

1. All the rows within group 1, group 2, and group 3 are highly correlated and they are not correlated with the rows in any other groups.
2. All the rows within group 4, group 5, and group 6 are correlated. All the rows among group 4, group 5, and group 6 are also correlated but they are not correlated to the rows in any other groups.
3. All the rows within group 7 and group 8 are correlated. All the rows between group 7 and group 8 are a bit less correlated but they are not correlated to the rows in any other groups.

With the above features, a good NMF algorithm should classify the cells in group 1, 2, or 3 into three different metacells and classify all the cells in group 4, 5, and 6 into a single metacell. For the cells in group 7 and 8, the NMF algorithm may classify all of them into a single metacell or into two different metacells if it is more sensitive. Hence, we should expect the estimated rank value r = 5 or 6 when we run the matrix factorization procedures. With this expected outcome, our simulation study could compare the performances among all the eight algorithms, two different seeding methods and five different rank estimation criteria.

Again, the way of design a $Y_{m\times n}$ is not unique. The following is the way of our design:

The Design of Group 1:

Step 1

Row 1 has 50% (2500) components with values 0 and 50% (2500) components with values randomly selected between 1 and 10 with different weights (see the table below for the weights). Shuffle the components in random order.

Counts	1	2	3	4	5	6	7	8	9	10
Weights %	55	20	10	5	4	2	1.6	1.2	0.8	0.4

Step 2

Let Row 1 = Row 2 = ... = Row 50

(Duplicate row 2, row 3, ..., row 50. At this moment, row 1, row 2, ..., row 50 are all the same).

Step 3

Change all the rows (row 1, row 2, ..., row 50):
For each row i, ($i = 1, ..., 50$), randomly choose 10% (500) numbers between 0 and 9 (uniformly) and use these values to replace 10% (500) randomly chosen components in row i. (Now, all the 50 rows in group 1 are similar but different from each other).

The Design of Group 2 and Group 3:

Do the same thing for Group 2 and Group 3 as we did for Group 1 except the following changes:

Group 2
Each row in Group 2, has 55% (2750) components with values 0 and 45%(2250) components with values randomly selected between 1 and 10 with different weights (use the same table for the weights).

Group 3
Each row in Group 3, has 60% (3000) components with values 0 and 40%(2000) components with values randomly selected between 1 and 10 with different weights (use the same table for the weights).

The Design of Group 4, Group 5, and Group 6:

Step 1
Row 151 has 65% (3250) components with values 0 and 35%(1750) components with values randomly selected between 1 and 10 with different weights (use the same table for the weights). Shuffle the components in random order.

Step 2
Let Row 151 = Row 152 = ...= Row 300
Duplicate row 152, row 153, ..., row 300. At this moment, all the rows in group 4, Group 5, and Group 6 (from row151 to row 300) are all the same.

Step 3

For each row i in Group 4, ($i = 151, ..., 200$), randomly choose 5% (250) numbers between 0 and 9 (uniformly) and use these values to replace 5% (250) randomly chosen components in row i. (Now, all the 50 rows in group 4 are similar but different from each other).

Step 4

For each row i in Group 5, ($i = 201, ..., 250$), randomly choose 10% (500) numbers between 0 and 9 (uniformly) and use these values to replace 10%(500) randomly chosen components in row i. (Now, all the 50 rows in group 5 are similar but different from each other).

Step 5

For each row i in Group 6, ($i = 251, ..., 300$), randomly choose 15% (750) numbers between 0 and 9 (uniformly) and use these values to replace 15%(750) randomly chosen components in row i. (Now, all the 50 rows in group 6 are similar but different from each other).

The Design of Group 7 and Group 8:

Step 1

Row 301 has 70% (3500) components with values 0 and 30% (1500) components with values randomly selected between 1 and 10 with different weights (use the same table for the weights). Shuffle the components in random order.

Step 2

Let Row 301 = Row 302 = ...= Row 400 Duplicate row 302, row 303, ..., row 400. At this moment, all the rows in group 7 and Group 8, (from row 301 to row 400) are all the same.

Step 3

For each row i in Group 7, ($i = 301, ..., 350$), randomly choose 500 numbers between 0 and 9 (uniformly) and use these values to replace 500 randomly chosen components within the FIRST 2500 components (do not change the Last 2500 components) in row i. (Now, all the 50 rows in group 7 are similar but different from each other).

Step 4

Do the same to Group 8 as what we did to Group 7: For each row i in Group 8, ($i = 351, ..., 400$), randomly

choose 500 numbers between 0 and 9 (uniformly) and use these values to replace 500 randomly chosen components within the LAST 2500 components (do not change the FIRST 2500 components) in row i. (Now, all the 50 rows in group 8 are similar but different from each other).

The matrix $Y_{m\times n}$ is in Supplementary 3 and the covariance matrix of $Y_{m\times n}$ is in Supplementary 4. From the covariance matrix of $Y_{m\times n}$, we clearly see that $Y_{m\times n}$ has all our desired features.

3.2.2 The Results and Conclusions of Simulation II

The simulation study and discussion in this section are parallel to the previous Section 3.1.2 except using the second simulated data instead of the first simulated data. We have conducted the study for both a fixed random seeding and ICA seeding over eight different matrix factorization algorithms. This time, we are presenting the tables for the random seeding and keep all the tables for ICA seeding in Supplementary 6.

The second simulated data is designed in the way that the proper rank r should be 6 or 5 (second best choice). The following tables are the simulation results for a fixed random seeding.

Table 3.6: Mean Normalized Spareness and (SD) for Different Algorithms and Ranks for SSD

Method \ Rank	4	5	6	7	8	9	10
Lee	0.81 (0.24)	0.85 (0.14)	0.88 (0.15)	0.85 (0.16)	0.77 (0.19)	0.75 (0.19)	0.71 (0.20)
Brunet	0.84 (0.18)	0.84 (0.20)	0.91 (0.13)	0.87 (0.17)	0.82 (0.18)	0.76 (0.19)	0.86 (0.18)
nsNMF	0.88 (0.16)	0.84 (0.21)	0.78 (0.24)	0.98 (0.07)	0.91 (0.17)	0.92 (0.14)	0.92 (0.13)
KL	0.84 (0.18)	0.85 (0.20)	0.91 (0.13)	0.87 (0.17)	0.82 (0.18)	0.76 (0.19)	0.86 (0.18)
Frobenius	0.81 (0.24)	0.85 (0.14)	0.88 (0.15)	0.85 (0.16)	0.77 (0.19)	0.75 (0.19)	0.71 (0.20)
Offset	0.89 (0.14)	0.92 (0.05)	0.93 (0.04)	0.88 (0.12)	0.85 (0.15)	0.83 (0.17)	0.81 (0.19)
Snmf/r	0.81 (0.23)	0.86 (0.10)	0.89 (0.09)	0.73 (0.18)	0.69 (0.19)	0.68 (0.17)	0.67 (0.16)
Snmf/l	0.84 (0.24)	0.91 (0.08)	0.93 (0.08)	0.78 (0.19)	0.88 (0.13)	0.76 (0.19)	0.84 (0.15)

We can see that seven matrix factorization algorithms have the highest average cell-sparseness occurred at r = 6 except nsNMF at r = 7. However, there is no clear big drop of average cell-sparseness take place from r = 5 to r = 6 or from r = 6 to r = 7 for some of the algorithms. Hence, the rank estimation based on the cell-sparseness among different algorithms are not very consistent.

Table 3.7: Information Gain for Different Algorithms and Ranks for SSD

Method \ Rank	4	5	6	7	8	9	10
Lee	0.943	0.941	0.948	0.938	0.914	0.905	0.891
Brunet	0.951	0.945	0.963	0.954	0.941	0.923	0.952
nsNMF	0.968	0.948	0.938	0.991	0.974	0.973	0.971
KL	0.951	0.945	0.964	0.954	0.941	0.923	0.952
Frobenius	0.943	0.941	0.948	0.938	0.914	0.905	0.891
Offset	0.963	0.962	0.966	0.949	0.937	0.928	0.918
Snmf/r	0.942	0.940	0.950	0.904	0.887	0.876	0.865
Snmf/l	0.952	0.959	0.965	0.925	0.946	0.910	0.931

In Table 3.7, we find that seven matrix factorization algorithms have the highest information gain occurred at $r = 6$ except nsNMF at $r = 7$. However, the information gain for other rank values are very close to each other cross the range of rank r values for every one of the eight matrix factorization algorithms. Also, there is no significant drop of information gain between any two consecutive rank r values for every matrix factorization algorithm. Therefore, information gain criterion is unable to estimate the proper rank value convincingly.

As we discussed in Section 2.2.2, we can also estimate the rank value based on the counts of classified cells for different cut-off value $\alpha = 0.5, 0.6, 0.7, 0.8,$ *and* 0.9. Again, we only present the Table 3.8 for Lee algorithm and leave all the other seven tables in Supplementary 6.

Table 3.8: Number of Classified Cells for Given Rank and Cutoff, Lee Algorithm, for SSD

Cutoff \ Rank	4	5	6	7	8	9	10
0.5	400	400	400	394	377	375	347
0.6	350	400	400	381	354	331	308
0.7	350	393	361	358	316	287	284
0.8	308	350	350	323	285	240	220
0.9	299	283	330	275	180	176	122

In Table 3.8, we can see clearly Lee algorithm gives the preference to $r = 6$ first and to $r = 5$ second.

When we exam the tables of the counts of classified cells for all eight algorithms, we find the following:

1. Brunet and KL algorithms give the clear preference to $r = 6$ only.

2. Lee, Frobenius, Offset, Snmf/r, and Snmf/l give the first preference to $r = 6$ and the second preference to $r = 5$.

3. Unfortunately, nsNMF method gives the preference to $r = 4$ only. Hence it has failed to estimate the proper rank value.

The following Table 3.9 are the degree of confusion for the second simulated data with a fixed random seeding and $r = 4, 5, ...10$ for all eight matrix factorization algorithms.

Table 3.9: Degree of Confusions for Different Algorithms and Ranks for SSD

Method \ Rank	4	5	6	7	8	9	10
Lee	0.000	0.000	0.000	0.041	0.070	0.128	0.094
Brunet	0.000	0.000	0.000	0.019	0.079	0.163	0.183
nsNMF	0.000	0.000	0.059	0.000	0.108	0.163	0.213
KL	0.000	0.000	0.000	0.019	0.079	0.163	0.183
Frobenius	0.000	0.000	0.000	0.041	0.070	0.128	0.094
Offset	0.069	0.000	0.000	0.041	0.058	0.074	0.094
Snmf/r	0.000	0.000	0.000	0.094	0.129	0.081	0.103
Snmf/l	0.000	0.000	0.000	0.000	0.064	0.065	0.130

From Table 3.9, we find that the degree of confusion is zero when rank $r = 5$ for all eight matrix factorization algorithms. For $r = 4$ and $r = 6$, the degree of confusion is also zero for seven out of eight algorithms. This will suggest that the proper rank value could be 5, or 6, or possibly 4.

We are now using the Euclidean distance to estimate the rank r value. For eight matrix factorization algorithms, we have calculated the Euclidean distance between $Y_{m\times n}$ and $W_{m\times r} \times H_{r\times n}$ for rank r value between 4 and 10. We find that, other than nsNMF algorithm, all seven matrix factorization algorithms have a significant drop of Euclidean distance from $r = 4$ to $r = 5$ and a quite big drop from $r = 5$ to $r = 6$. From $r = 6$ and above, the Euclidean distances are stable. These facts suggest that the proper rank value is $r = 5$ or $r = 6$.

For nsNMF algorithm, there is a significant drop of Euclidean distance from $r = 4$ to $r = 5$. From $r = 5$ and above, the Euclidean distances are more or less the same. This fact suggests that the estimated rank value is $r = 5$. This estimation is not as good as the other seven algorithms. See Table 3.10 below.

Table 3.10: Euclidean Distance for Different Algorithms and Ranks for SSD

Method \ Rank	4	5	6	7	8	9	10
Lee	1922.76	1773.80	1746.92	1743.02	1739.23	1735.38	1731.89
Brunet	1945.80	1786.42	1748.55	1746.06	1744.57	1744.01	1740.14
nsNMF	2070.74	1931.88	1952.43	1930.13	1935.68	1943.78	1948.07
KL	1945.80	1786.42	1748.55	1746.06	1744.57	1744.01	1740.14
Frobenius	1922.76	1773.80	1746.92	1743.02	1739.23	1735.38	1731.89
Offset	1987.09	1777.63	1750.12	1745.74	1741.59	1737.30	1733.16
Snmf/r	1922.78	1773.84	1746.95	1745.44	1741.33	1738.01	1734.23
Snmf/l	1922.77	1773.81	1746.92	1745.41	1739.13	1737.56	1731.49

We have also carried out the same simulation study for ICA seeding. The numerical results are kept in Supplementary 6 and the summary of them is in Figure 3.2. The findings are summarized as follows.

1. Based on the cell-sparseness

Similar to the random seeding case, we find that seven matrix factorization algorithms have the highest average cell-sparseness occurred at $r = 4$ except nsNMF at $r = 5$. Also there is no clear patten for the average cell-sparseness values over the rank r value.

2. Based on the information gain

We find that seven matrix factorization algorithms have the highest information gain occurred at $r = 6$ except nsNMF at $r = 7$. We also find that the information gain for other rank values are very close to each other cross the range of rank r values for every one of the eight matrix factorization algorithms and there is no significant drop of information gain between any two consecutive rank r values for every matrix factorization algorithm.

3. Based on the counts of classified cells

We find that Lee, Frobenius, Offset, Snmf/r, and Snmf/l give a clear indication that the proper rank value of r is either $r = 5$ or $r = 6$. However, Brunet and KL give indication of $r = 4$ or $r = 6$, and nsNMF algorithm gives the preference to $r = 7$.

4. Based on the degree of confusion of confusion matrix

Similar to the random seeding case, the estimated r value could be 5, or 6, or possibly 4.

5. Based on the Euclidean distance

When we use the Euclidean distance as a measure to estimate the rank value, the random seeding and ICA seeding behave the same way for all eight different matrix factorization algorithms.

In summary, for the second simulated data, we have the following observations:

1. For the sake of rank estimation, there is a tiny difference between random seeding and ICA seeding.

2. The criteria based on cell-sparseness and information gain have failed to identify the proper rank value.

3. Based on the number of classified counts criterion, seven matrix factorization methods (other than nsNMF algorithm) have been successfully estimating the rank value.

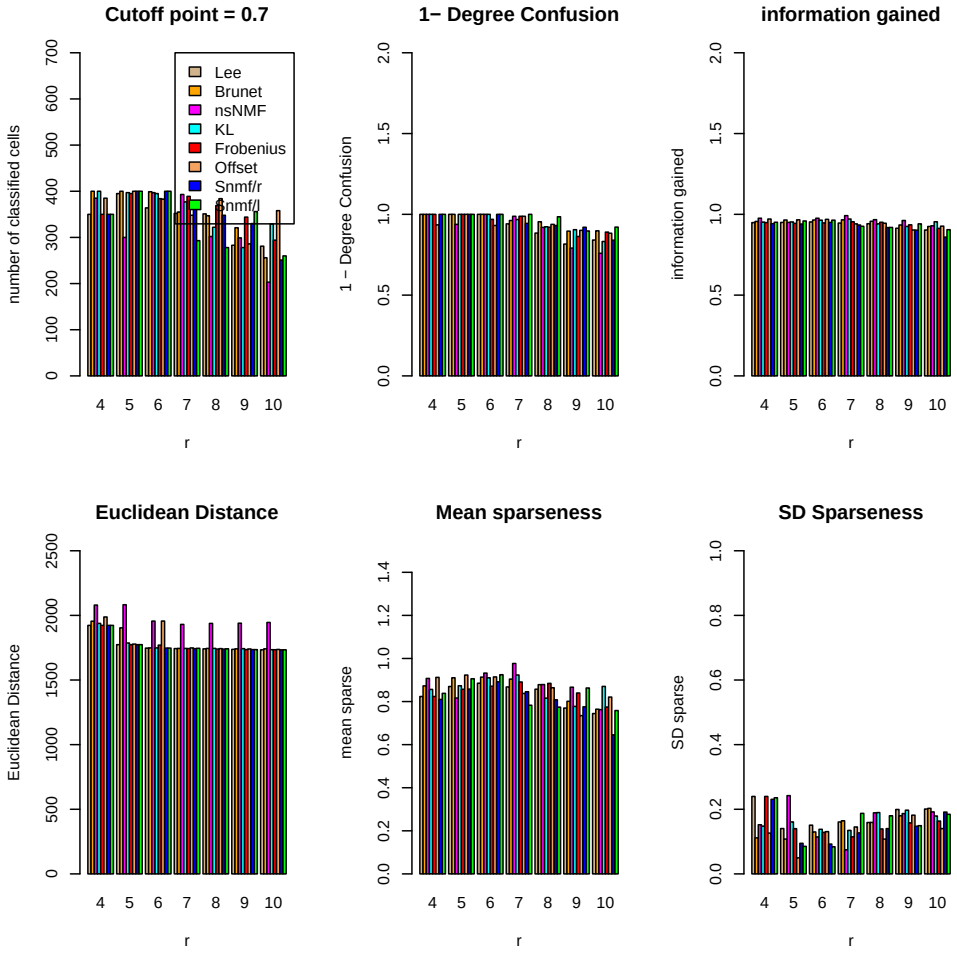
4. All eight matrix factorization algorithms have quite successfully identified the proper rank values when they compound with the criteria of the number of classified counts, the degree of confusion, or the Euclidean distance.

5. In comparison the performance among all eight matrix factorization algorithms, we find that Lee, Frobenius, Offset, Snmf/r, and Snmf/l algorithms are better than Brunet and KL algorithms and then followed by nsNMF algorithm.

As our simulated data here is a very good imitation of the single-cell data, the above summary should provide us some guidelines for applying NMF to the real single-cell gene expression data.

Here are some general recommendations: (1) For the seeding methods, we suggest to using random seeding and to take multiple runs if possible in the hope to achieve better approximation. (2) For the rank estimation criteria, we prefer to using the counts of classified cells, the degree of confusion and the Euclidean distance over the cell sparseness and the information gain. (3) Among eight factorization algorithms (overall speaking), we suggest to using Lee, Frobenius, Offset, Snmf/r, and Snmf/l over Brunet and KL and nsNMF is not recommended.

Figure 3.2: Rank Estimation for SSD using ICA Seeding



Chapter 4

Assessment on a Real Single-Cell Gene Expression Data

In the previous chapter we have investigated NMF using pure simulated data. In this chapter, we move on to test all NMF algorithms by applying them to a real single-cell gene expression data using the same design. The real data do not contain gold-standard truth, however has biologically validated labels of cell-types, which can also be deemed as approximately ground truth in our analysis.

4.1 Descriptions of the Single-Cell Gene Expression Data

The single-cell gene expression data we have used here is from profiling of mouse blood cells using CEL-seq2. The gene expressions were profiled by high throughput sequencing on Mus musculus (house mice) [25].

From computational viewpoint, we are dealing with 383 cells (there is a missing cell) and 19903 genes. Out of the 383 cells there are 16 clusters, labelled A-P, as well as 5 cell types (labelled 1 through 5). We have 48 of cell type 1, 96 of cell type 2, 96 of cell type 3, 95 of cell type 4, and 48 of cell type 5.

4.2 Gene Selection and Data Filtering

The rank estimation is one of the challenges in NMF. The high dimensionality is another challenge when we apply NMF to gene expression data. Our single-cell gene expression data has 383 rows (cells) and 19903 columns (genes). With this high dimension, some of the packages or algorithms may not be able to run or to run within a reasonable amount of time. On the other hand, some of the genes are highly relevant in cell

classification and the others are less relevant in cell classification. We can achieve data reduction by selecting the important genes (delete the less important genes) for classification purpose. There are different ways for gene selection. We may select the important genes based on (1) gene density, (2) gene sparseness, (3) gene's relative expression, and (4) gene's variance.

Here are the detailed descriptions for these gene selection criteria:

1. Gene density

In the original data set, we define the j-th gene density as the ratio of the number of non-zero entries and the length of the column. We may choose the genes with density higher than some predetermined value (say, 0.05, 0.06, 0.07, 0.08, or 0.09) and delete the rest.

2. Gene sparseness

Similar to the cell sparseness, we define the j-th gene sparseness (based on the original data matrix X) as

$$\sqrt{(\sum_{i=1}^{m}(\bar{X}_{ij})^2)}$$

where $\bar{X}$ is the normalized X column wise. We may choose the genes with sparseness higher than some predetermined value.

3. Gene's relative expression

In matrix $\bar{X}$, we view each entry as the relative expression of a gene in a cell. We may choose the genes with the highest value of entries in $\bar{X}$ higher than some predetermined value.

4. Gene's variance

In X matrix, we find the variance for each column (gene). We will select the genes with variance higher than some predetermined value.

In practice we could use any one (or some) of the above criteria to select genes hence reduce the dimension of the original data.

In our study, we are using gene's variance to select genes for our single-cell gene expression data. The

reason of using gene's variance is straight forward as the gene with smaller variance across cells indicates less importance in classification.

4.3 Results on Real Data

Our study in this section is based on reduced data instead of the original single-cell gene expression data. We have used the gene variance criteria (mentioned in previous section) to select 4903 influential genes to get the reduced data. Hence, our reduced data has 383 rows and 4903 columns.

Our two simulated data in Chapter 3 are well designed with known features (hence we know what to expect). When we are dealing with real data sets, things could be vaguer and more complicated. We have conducted a study to our reduced single-cell gene expression data using two seeding methods, eight NMF algorithms, and five rank estimation criteria. First, we discuss the results based on a fixed random seeding. The numerical results for a fixed random seeding are summarized into the following five tables:

Table 4.1: Mean Normalized Spareness and (SD) for Different Algorithms and Ranks

Method \ Rank	2	3	4	5	6	7	8
Lee	0.53 (0.30)	0.53 (0.27)	0.44 (0.22)	0.42 (0.19)	0.40 (0.17)	0.35 (0.17)	0.32 (0.17)
Brunet	0.78 (0.23)	0.77 (0.22)	0.55 (0.22)	0.54 (0.20)	0.57 (0.18)	0.54 (0.18)	0.57 (0.17)
nsNMF	0.90 (0.24)	0.81 (0.27)	0.72 (0.26)	0.72 (0.25)	0.66 (0.24)	0.67 (0.23)	0.62 (0.24)
KL	0.78 (0.23)	0.77 (0.22)	0.55 (0.22)	0.54 (0.20)	0.57 (0.18)	0.54 (0.18)	0.57 (0.17)
Frobenius	0.53 (0.30)	0.53 (0.27)	0.44 (0.22)	0.42 (0.19)	0.40 (0.17)	0.35 (0.17)	0.32 (0.17)
Offset	0.58 (0.29)	0.57 (0.25)	0.43 (0.20)	0.44 (0.17)	0.41 (0.16)	0.39 (0.16)	0.32 (0.16)
Snmf/r	0.58 (0.28)	0.54 (0.24)	0.50 (0.22)	0.41 (0.18)	0.44 (0.17)	0.31 (0.10)	0.26 (0.08)
Snmf/l	0.75 (0.32)	0.75 (0.29)	0.69 (0.27)	0.66 (0.23)	0.71 (0.20)	0.93 (0.15)	0.80 (0.17)

Table 4.2: Information Gain for Different Algorithms and Ranks

Method \ Rank	2	3	4	5	6	7	8
Lee	0.934	0.899	0.859	0.838	0.821	0.790	0.769
Brunet	0.964	0.946	0.891	0.872	0.866	0.845	0.845
nsNMF	0.985	0.963	0.935	0.927	0.905	0.900	0.880
KL	0.964	0.946	0.891	0.872	0.866	0.845	0.845
Frobenius	0.934	0.899	0.859	0.838	0.821	0.790	0.769
Offset	0.939	0.907	0.859	0.846	0.823	0.803	0.770
Snmf/r	0.939	0.903	0.870	0.834	0.832	0.787	0.746
Snmf/l	0.964	0.948	0.926	0.914	0.927	0.979	0.938

Table 4.3: Number of Classified Cells for Given Rank and Cutoff, Lee Algorithms

Cutoff \ Rank	2	3	4	5	6	7	8
0.5	383	352	303	261	217	164	128
0.6	356	307	214	179	145	99	76
0.7	315	253	143	109	75	49	39
0.8	254	181	88	54	24	18	13
0.9	138	84	24	12	5	4	2

Table 4.4: Degree of Confusions for Different Algorithms and Ranks

Method \ Rank	2	3	4	5	6	7	8
Lee	0.232	0.280	0.472	0.481	0.467	0.521	0.566
Brunet	0.092	0.096	0.169	0.277	0.227	0.224	0.197
nsNMF	0.147	0.116	0.173	0.212	0.275	0.311	0.358
KL	0.092	0.096	0.169	0.277	0.227	0.224	0.197
Frobenius	0.232	0.280	0.472	0.481	0.467	0.521	0.566
Offset	0.180	0.243	0.466	0.501	0.462	0.453	0.550
Snmf/r	0.191	0.308	0.312	0.445	0.415	0.301	0.275
Snmf/l	0.184	0.184	0.288	0.376	0.211	0.195	0.348

Table 4.5: Euclidean Distance for Different Algorithms and Ranks

Method \ Rank	2	3	4	5	6	7	8
Lee	5159.78	3586.60	3014.03	2471.43	2305.73	2175.24	2133.20
Brunet	10230.83	9331.46	6152.68	3984.70	3492.36	3090.97	2999.26
nsNMF	13713.12	11958.96	10143.90	9672.59	9356.40	9346.36	10118.21
KL	10230.84	9331.49	6152.97	3984.70	3492.35	3090.97	2999.26
Frobenius	5159.77	3586.58	3014.03	2471.43	2305.73	2175.24	2133.20
Offset	5177.32	3610.19	3039.93	2499.07	2303.73	2226.60	2135.39
Snmf/r	5189.93	3658.62	3149.33	2688.10	2671.68	3110.50	2817.71
Snmf/l	5214.88	4945.25	3329.15	3038.51	8223.60	7988.17	6326.41

Table 4.1 are the average cell-sparseness and the standard deviation values (in parentheses) for eight different matrix factorization algorithms. We can see that all seven matrix factorization algorithms except nsNMF have highest average cell-sparseness value occurred when $r = 2$ and $r = 3$ and there is a big drop occurred from $r = 3$ to $r = 4$. This fact indicates that the proper r value is $r = 3$. For nsNMF method, there is a big drop of average cell-sparseness value from $r = 2$ to $r = 3$ and from $r = 3$ to $r = 4$. This indicates that the proper r value may be $r = 2$ or $r = 3$.

Table 4.2 are the information gain values for eight different matrix factorization algorithms and for rank r values from 2 to 8. We find that some matrix factorization algorithms have a big drop of the information gain values occurred from $r = 2$ to $r = 3$ and some other algorithms have a big drop from $r = 3$ to $r = 4$. This indicates also that the proper r value may be $r = 2$ or $r = 3$.

Table 4.3 presents the number of classified cells for Lee algorithm and for the cut-off value $\alpha = 0.5, 0.6, 0.7, 0.8$ and 0.9. We leave all the other seven tables in Supplementary 7. We can see that the number of classified

cells have highest values when $r = 2$. There is a clear drop off occurred from $r = 2$ to $r = 3$ and from $r = 3$ to $r = 4$. The other seven algorithms exhibit the same thing. Hence, the estimation of r value is $r = 2$ or $r = 3$ by the number of classified cells criterion.

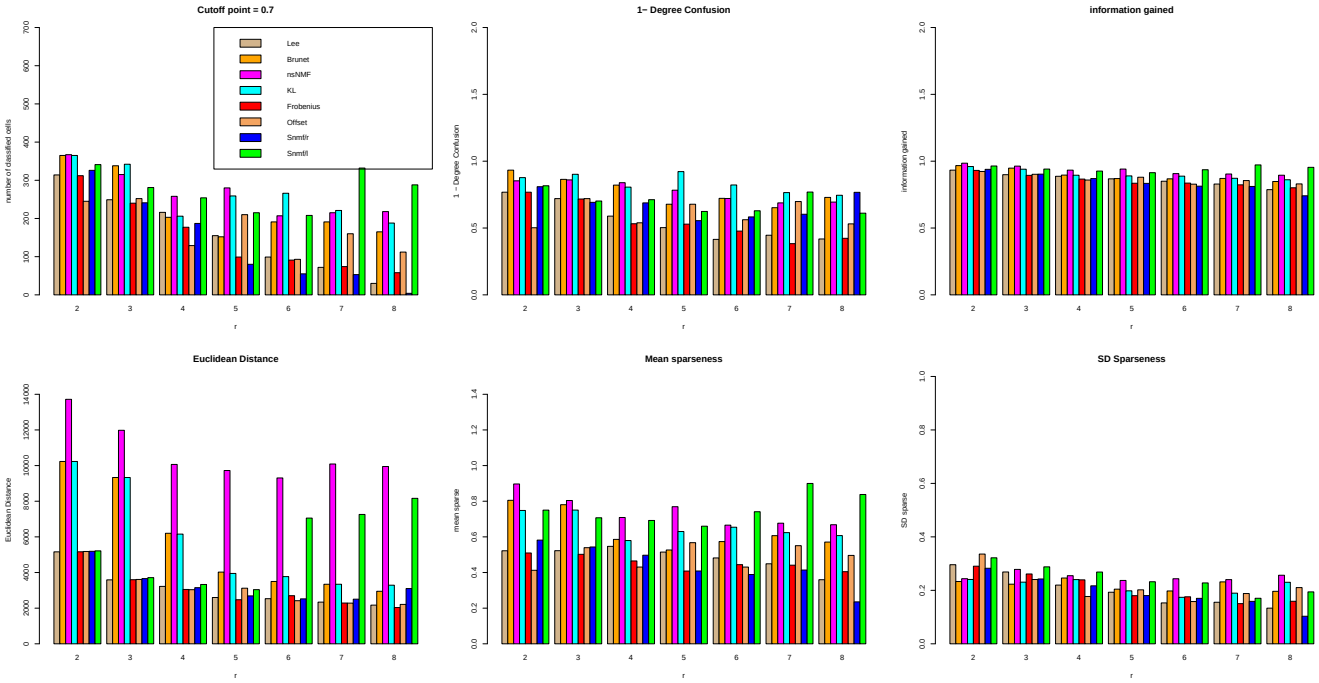
In Table 4.4, we have calculated the degree of confusion for eight matrix factorization algorithms and for rank r values from $r = 2$ to $r = 8$. We have noticed the following:
(1) For Snmf/r method, there is a clear increase in the degree of confusion for r from 2 to 3 and from 4 to 5.
(2) For Brunet, nsNMF, and KL algorithms, there is a clear increase in the degree of confusion for r from 3 to 4 and from 4 to 5.
(3) For the other four algorithms, they all have a clear increase in the degree of confusion for r from 3 to 4.
From the degree of confusion view point, majority of the algorithms suggest that $r = 3$ might be the proper rank value or possibly $r = 4$.

Table 4.5 are the Euclidean distance for eight different algorithms and for r value from 2 to 8. We have observed that the Euclidean distance for nsNMF algorithm has no clear trend. For all the other seven algorithms, there is a clear drop in the Euclidean distance for r value from 4 to 5 and remain stable after $r = 5$. Hence, as far as the Euclidean distance concern, the proper rank r value should be $r = 5$.

When we look into the results based on the ICA seeding, we come to the same conclusion as the fixed random seeding. That is, among all the eight algorithms and five estimation criteria, most of them suggest $r = 3$ and some of them suggest $r = 2$, 4, or 5. In practice, people usually do not consider the case of $r = 2$, as $r = 2$ will have highest cell-sparseness value and information gain values most of the time if not all the time. Hence, for this single-cell gene expression data, we should look at $r = 3$, 4 and 5 for further study.

The numerical results for the ICA seeding are kept in Supplementary 7 and the summary of them is in Figure 4.1.

Figure 4.1: Rank Estimation for Single-Cell Data using ICA Seeding



4.4 Confusion Matrices of Single-Cell Data

Mathematically, we have ways to estimate the proper rank value r or a few possible r values. In practice, the suggested r values will be subject to further biological interpretations. Ideally, the number of metacells is the same as the number of biological subpopulations (cell types in our case). In this case the confusion matrix is a square matrix. Hence, matching the metacells to cell types is clear and the interpretation is straight forward. However, the estimated r value may be less than or greater than the number of cell types. When the r value is less than the number of cell types, it indicates that some of the cells in different cell types will be classified into the same metacell. On the other hand, when the r value is greater than the number of cell types, it indicates that some of the cells in the same cell type will be classified into different metacells. It is beneficial to look at a few possible r values and their confusion matrices and ask for the meaningful interpretations from biologists.

For our single-cell gene expression data, the estimated r value is $r = 3$ (by majority of the methods) and the number of cell types is 5. Here we present the three confusion matrices correspond to $r = 3$, 4, and

5 by Lee algorithm.

$$M_1 = \begin{pmatrix} 8 & 30 & 10 \\ 91 & 5 & 0 \\ 0 & 12 & 84 \\ 2 & 90 & 3 \\ 1 & 38 & 9 \end{pmatrix}$$

$$M_2 = \begin{pmatrix} 5 & 16 & 1 & 26 \\ 81 & 3 & 0 & 12 \\ 0 & 8 & 80 & 8 \\ 2 & 48 & 0 & 45 \\ 1 & 19 & 1 & 27 \end{pmatrix}$$

$$M_3 = \begin{pmatrix} 8 & 5 & 7 & 1 & 27 \\ 2 & 76 & 17 & 0 & 1 \\ 4 & 0 & 4 & 78 & 10 \\ 28 & 0 & 15 & 0 & 52 \\ 11 & 1 & 3 & 1 & 32 \end{pmatrix}$$

We find the degree of confusion for M_1 is 0.2796, for M_2 is 0.4723, and M_3 is 0.4811. From the degree of confusion viewpoint, we may prefer the classification by matrix M_1. However, in practice we will need the knowledge in biology or in genetics to decide which classification will make more sense and will have meaningful interpretation in biological context.

Chapter 5

Stability of NMF

As we mentioned in Section 1.3, one of the challenges in NMF is its stability, especially when the sample size is not substantially larger than the number of terms (which is genes in single-cell data). So, the impact of the sample size to the stability of NMF is an important property to study. Another common concern is the impact of initialization to the stability of NMF. Because there is no global minimization algorithm due to the non-convexity of the loss function and also the structure of loss function is dependent on each target matrix. Researchers found that different initialization could leads to different local minimum of the loss function. So the end result of NMF is sensitive to the choice of the starting point of NMF algorithms. In this chapter, we study the stability of NMF over the sample size first, then a short discussion on initialization and stability.

5.1 The Stability of NMF Over the Sample Size

In this section, we investigate the stability of NMF over the sample size n.

5.1.1 Motivation

The stability of Non-Negative Matrix Factorization (NMF) can vary depending on the sample size. In general, the stability of NMF can be influenced by several factors, including the inherent properties of the data and the specific algorithm or implementation used. However, when it comes to the sample size, the following observations might be possible: (1) When the sample size is small, the stability of NMF may be compromised. Limited data points can lead to instability in the factorization process, making it challenging to obtain consistent and reliable results. (2) As the sample size increases, the stability of NMF tends to improve. With more data points, the algorithm can better estimate the underlying structure of the matrix, leading to more consistent factorization results. However, it's important to note that the stability can still

be influenced by the complexity and variability of the data. (3) With a sufficiently large sample size, NMF tends to become more stable. Adequate data points allow for more accurate estimation of the basis vectors and coefficients, leading to robust factorization results. The extracted factors are more likely to capture the essential patterns and structures present in the data.

In general, as the sample size increases, NMF tends to become more stable and reliable. With a larger number of samples, NMF has access to more information, which can help improve the accuracy and robustness of the factorization. It provides a better estimation of the underlying structure in the data and reduces the potential impact of noise or outliers. However, the exact behavior of NMF with respect to sample size can vary depending on the specific dataset and the algorithm used. In some cases, a small sample size may still yield reliable results if the data exhibits clear patterns or if the algorithm is designed to handle limited data. Conversely, even with a large sample size, NMF may struggle if the data is noisy, highly variable, or lacks a discernible structure.

In summary, while increasing the sample size generally improves the stability of NMF, it is essential to consider the characteristics of the data and the algorithm's limitations. It is always recommended to evaluate the performance of NMF with different sample sizes and assess the robustness of the results across multiple experiments to draw reliable conclusions.

5.1.2 Sample Size Verse Stability

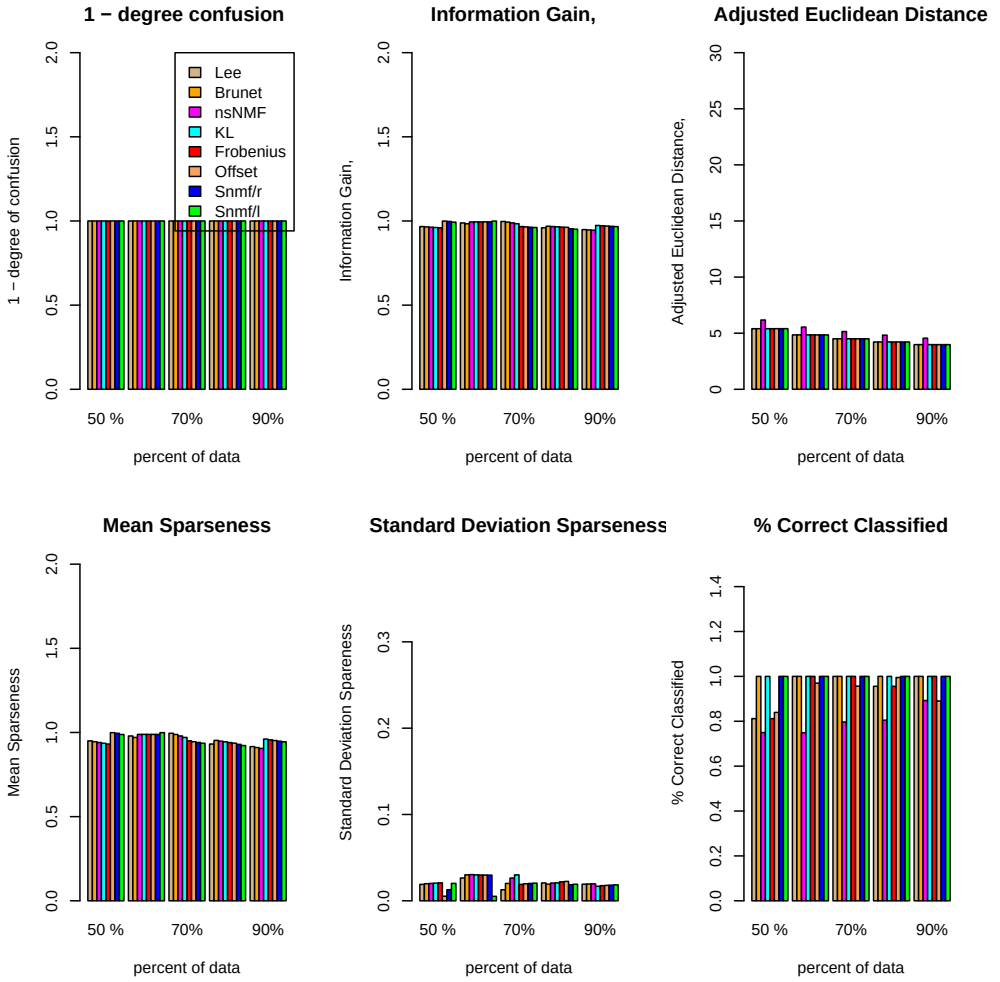
We are now going to investigate the impact of the sample size to the stability of NMF. We start with our two simulated data sets.

For our first simulated data, the sample size is 400 with 50 in each of the eight groups. We proportionally choose 50%, 60%, 70%, 80%, and 90% samples from each group. These percentage correspond to the sample size of 200, 240, 280, 320, 360 respectively. For each percentage, we repeatedly take random sample 50 times and then calculate the following: (1) mean and standard deviation of cell-sparseness, (2) information gain, (3) 1- degree of confusion, (4) adjusted Euclidean distance (the Euclidean distance divided by the number of rows in the target matrix), and (5) percentage of correct classified cells.

The reason to use "1 - degree of confusion" instead of degree of confusion is that it is in line with the other criteria, cell sparseness and information gain.

We run the simulations for all eight matrix factorization algorithms but only for a fixed random seeding and only for rank $r = 8$. The results are presented in the following Figure 5.1.

Figure 5.1: Stability Over Sample Size, First Simulated Data, $r = 8$



From Figure 5.1, we have observed the following:

(1) There is no clear trend for mean sparseness. The standard deviations for all algorithms are small and there is no clear trend.

(2) The Information Gain for all the algorithms are high and stable over the sample size.

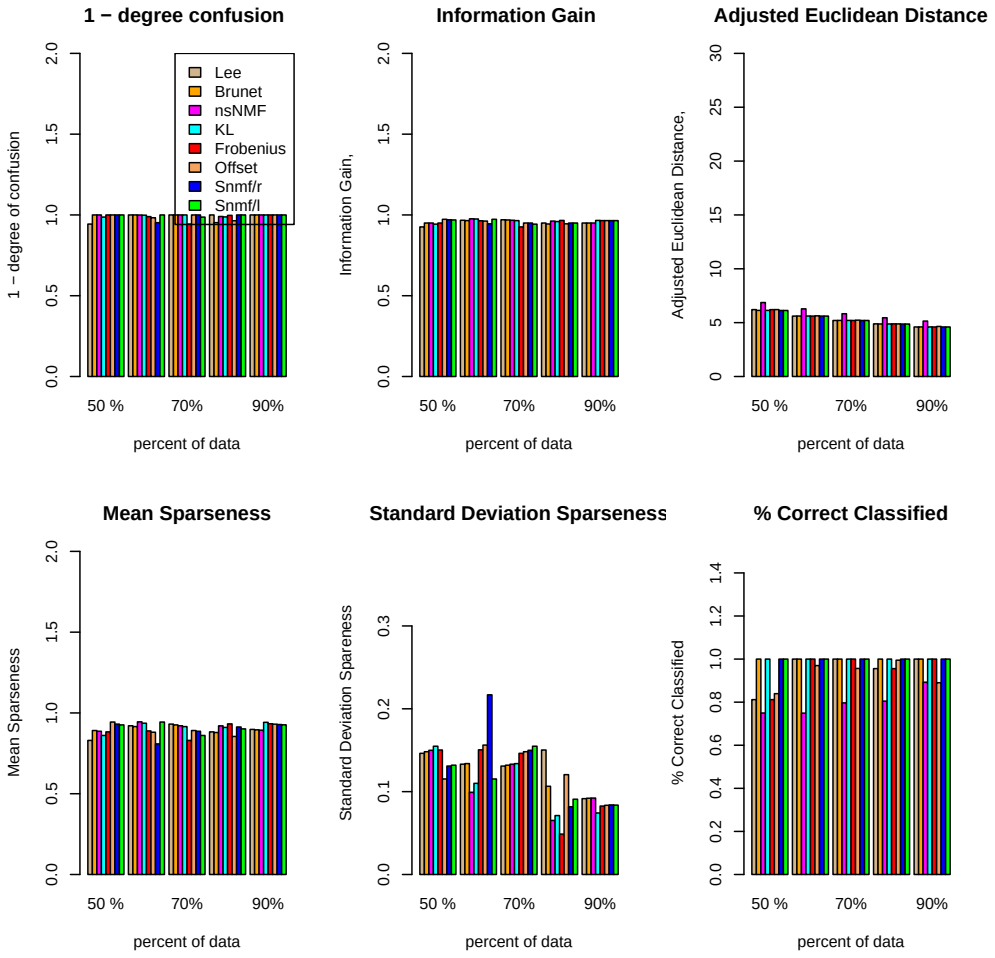
(3) The 1 - Degree of Confusion are stable (equal to one) when the sample sizes are decreased for all

the matrix factorization algorithms. Because the first simulated data was perfectly designed and should cause no confusion for classification.

(4) The Adjusted Euclidean Distance for all the algorithms are increased when sample sizes are decreased with nsNMF having the largest adjusted Euclidean distance and all the others having similar and smaller adjusted Euclidean distances.

We have also done the same simulations for our second simulated data. The results are in Figure 5.2.

Figure 5.2: Stability Over Sample, Second Simulated Data, $r = 6$



For our second simulated data, we have run the simulations for all eight NMF algorithms with a fixed random seeding and rank $r = 6$. We have found the following:

(1) There is no clear trend for mean sparseness. The standard deviations have an increasing trend (with

some fluctuations) when sample sizes are decreasing.

(2) Lee algorithm has a clear decline in information gain when the sample size is $n = 200$ (50% of the original $n = 400$). The information gains are quite stable with small amounts of fluctuations for the other seven NMF algorithms.

(3) For 1 - Degree of Confusion, Lee algorithm has a clear declined when the sample size is 50% of the original sample size. All the other algorithms are stable with some small fluctuations.

(4) The Adjusted Euclidean Distance for all the algorithms are increased when sample sizes are decreased and nsNMF has the largest adjusted Euclidean distance and all the others have similar and smaller adjusted Euclidean distance.

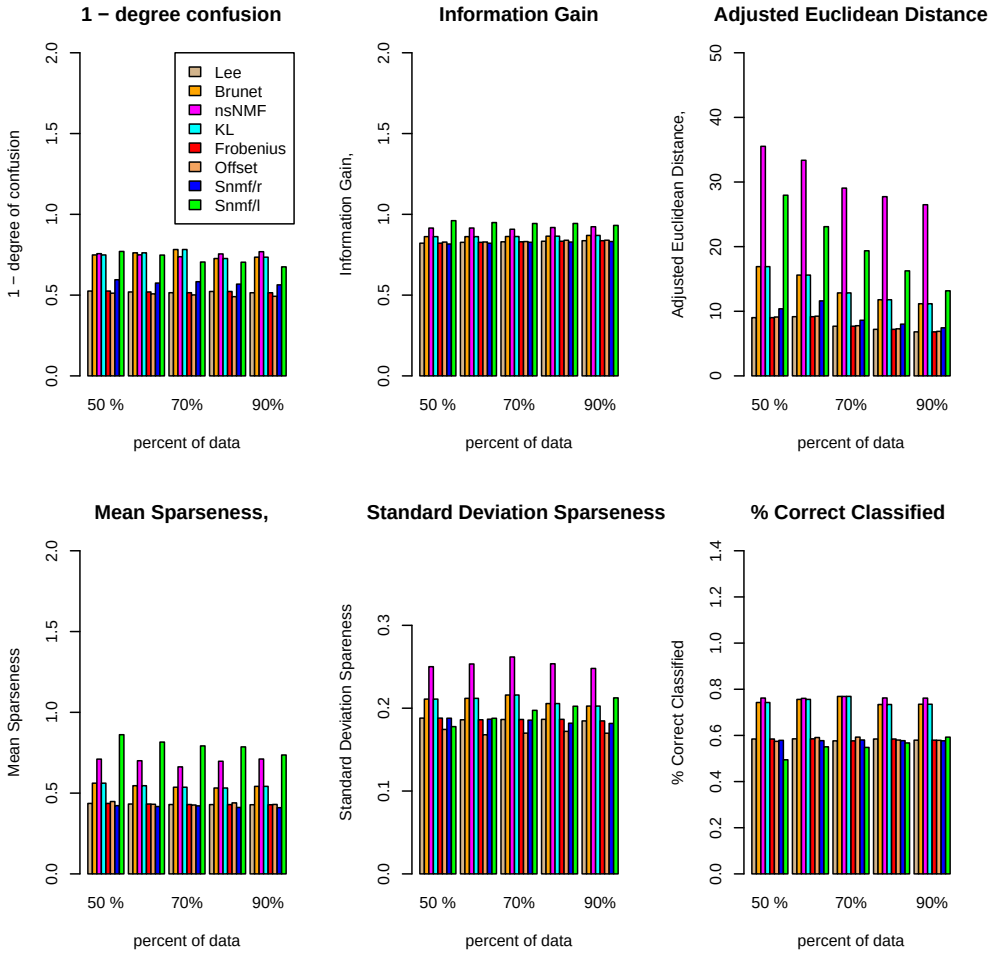
(5) The percentage of correct classified remain the same (stable) for Brunet, KL, Snmf-r, and Snmf-l over the sample size. The percentage of correct classified clearly dropped for Lee and Frobenius when sample size is 50%. The percentage of correct classified for nsNMF is decreased when sample size is decreased. The percentage of correct classified for Offset fluctuates when the sample size is decreased.

One thing in common for both first and second simulated data is that the adjusted Euclidean distance is increased when the sample size is decreased. The adjusted Euclidean distance indicates the accuracy of the matrix factorization. Hence, there is a tradeoff between the sample size and the accuracy of the matrix factorization.

We now use the reduced single-cell gene expression data set to investigate the stability of NMF over the sample size n. The reduced single-cell gene expression data has the sample size of 383 with 5 different known cell types. We proportionally choose 50%, 60% ,70%, 80%, and 90% samples from each cell type. For each percentage, we repeatedly take random sample 50 times and then calculate the following: (1) mean and standard deviation of cell-sparseness, (2) information gain, (3) 1- degree of confusion, (4) adjusted Euclidean distance, and (5) percent of correct classified cells.

We have run the simulations for all eight NMF algorithms but only for a fixed random seeding and only for rank r = 5. The results are presented in the following Figure 5.3.

Figure 5.3: Stability Over Sample Size, Single-Cell Data, $r = 5$



From Figure 5.3, we have found the following:

1. There is no clear trend for mean sparseness. The Snmf/l has the highest mean cell-sparseness values, nsNMF second, Brunet and KL third, and followed by the other four algorithms. Also, the standard deviations of the sparseness have no clear trend. The nsNMF has the largest standard deviation values, Brunet, and KL second, and followed by the other five algorithms.

2. The information gain for all eight algorithms are stable over the sample sizes. The Snmf/l has the highest information gain values, nsNMF second, Brunet and KL third, and followed by the other four algorithms. The order for mean cell sparseness and information gain are the same as these two quantities are very closely related.

3. The 1 - Degree of Confusion also does not affected by the changing of the sample size with Brunet, nsNMF, KL, and Snmf/l have relatively higher values and the other four algorithms have relatively lower values.

4. The adjusted Euclidean distance is the only quantity that is affected by the sample size. The adjusted Euclidean distances are increased when sample sizes are decreased for all the algorithms. The nsNMF has the largest adjusted Euclidean distance, Snmf/l the second, Brunet and KL the third, followed by the other four algorithms. This may suggest that the different matrix factorization algorithms convergent to different local minimum of the loss function used for NMF.

5. The percentage of correct classified remain stable for all the seven algorithms except for Snmf/l. The Snmf/l has a clear decline trend when the sample size is decreased. The Brunet, nsNMF, and KL have relatively higher percentage of correct classified cells than the other five algorithms.

All the above observations are very similar and quite consistent with what we have observed for our two simulated data.

5.2 Initialization and Stability

In Section 2.1.2, we have introduced two new random seedings. In this section, we first compare the performance of our two new random seedings with the package build in random seeding. Then we give a short discussion on the impact of initialization on the stability of NMF.

5.2.1 Compare New Random Seedings with the Build-in Random Seeding

We are now going to compare the performances of our random seedings (RS1 and RS2) with the package build in random seeding (BIRS) in turns of running time. Here, we refer running time as user time. In NMF, user time represents the amount of CPU time used by the NMF algorithm to execute the user's task. It includes the time spent on matrix operations, data manipulation, and other computations specific to NMF. Typically, user time is the more relevant measure when considering the performance and efficiency of the NMF algorithm, as it directly reflects the time spent on the actual computations performed by the algorithm.

We are using the Lee and Brunt algorithms to run NMF with RS1, RS2, and BIRS for two fixed seeds. We have done the simulations using our two simulated data and the single-cell gene expression data (see

Chapter 3 and Chapter 4 for details). The running times are presented in the following six tables. (In these tables, the number 2020 and 2023 are the two fixed random seeds.)

Table 5.1: Running Times for First Simulated Data and Lee Algorithm

	r = 4	r = 5	r = 6	r = 7	r = 8	r = 9	r = 10	Sum
$RS1_{2020}$	115.105	207.536	117.750	211.909	135.483	239.205	251.042	1278.03
$RS1_{2023}$	105.501	184.462	136.353	154.482	212.712	239.825	251.648	1284.983
$RS2_{2020}$	124.159	152.048	114.019	191.188	135.162	252.142	251.785	1220.503
$RS2_{2023}$	108.428	143.636	151.367	192.897	220.957	239.534	250.889	1307.708
$BIRS_{2020}$	181.020	312.078	180.341	340.050	218.228	385.789	391.597	2009.103
$BIRS_{2023}$	166.774	297.096	217.586	239.456	344.560	392.137	395.400	2053.009

Table 5.2: Running Times for Second Simulated Data and Lee Algorithm

	r = 4	r = 5	r = 6	r = 7	r = 8	r = 9	r = 10	Sum
$RS1_{2020}$	235.491	315.722	335.506	352.577	222.216	235.376	246.818	1943.706
$RS1_{2023}$	277.028	317.830	334.633	354.648	225.732	236.155	246.898	1992.924
$RS2_{2020}$	198.068	316.418	332.804	349.922	223.431	235.604	246.669	1902.916
$RS2_{2023}$	257.946	318.695	335.553	244.673	224.684	271.950	246.367	1899.868
$BIRS_{2020}$	220.383	310.564	325.732	345.030	357.744	379.051	399.768	2338.272
$BIRS_{2023}$	280.908	313.354	327.537	342.537	364.385	377.902	398.691	2405.314

Table 5.3: Running Times for Single-Cell Data and Lee Algorithm

	r = 2	r = 3	r = 4	r = 5	r = 6	r = 7	r = 8	Sum
$RS1_{2020}$	187.771	234.242	378.275	388.170	405.209	427.114	393.810	2414.591
$RS1_{2023}$	159.716	284.226	378.388	386.527	406.251	426.648	448.484	2490.24
$RS2_{2020}$	189.706	277.836	350.909	381.554	398.800	429.301	453.253	2481.359
$RS2_{2023}$	187.198	287.402	349.502	352.230	374.401	374.878	395.889	2321.5
$BIRS_{2020}$	196.937	215.580	350.275	352.641	363.613	380.121	389.607	2248.774
$BIRS_{2023}$	161.945	280.173	373.905	382.800	401.274	425.827	444.230	2470.154

Table 5.4: Running Times for First Simulated Data and Brunet Algorithm

	r = 4	r = 5	r = 6	r = 7	r = 8	r = 9	r = 10	Sum
$RS1_{2020}$	222.616	378.222	182.064	224.730	239.802	523.534	586.022	2356.99
$RS1_{2023}$	218.250	216.512	211.871	195.682	407.380	551.275	584.156	2385.12
$RS2_{2020}$	296.282	300.940	250.779	298.182	251.183	524.005	586.226	2507.59
$RS2_{2023}$	338.046	277.543	196.239	206.359	352.635	551.169	582.686	2504.67
$BIRS_{2020}$	215.627	358.435	181.313	219.949	236.596	536.772	588.073	2336.76
$BIRS_{2023}$	205.776	190.029	211.260	193.915	397.759	555.392	589.123	2343.25

Table 5.5: Running Times for Second Simulated Data and Brunet Algorithm

	r = 4	r = 5	r = 6	r = 7	r = 8	r = 9	r = 10	Sum
$RS1_{2020}$	218.888	258.402	276.682	487.782	517.872	549.828	541.090	2850.54
$RS1_{2023}$	209.819	156.184	226.557	490.210	518.344	552.613	316.374	2470.10
$RS2_{2020}$	214.153	220.579	458.876	490.165	515.556	551.696	315.159	2766.18
$RS2_{2023}$	264.703	156.453	338.429	502.774	518.122	552.120	315.080	2647.68
$BIRS_{2020}$	197.695	253.676	284.866	490.183	520.966	556.162	321.226	2624.77
$BIRS_{2023}$	224.692	156.234	227.210	489.914	521.663	556.846	400.700	2577.25

Table 5.6: Running Times for Single-Cell Data and Brunet Algorithm

	r = 2	r = 3	r = 4	r = 5	r = 6	r = 7	r = 8	Sum
$RS1_{2020}$	269.702	109.002	311.936	277.345	281.229	364.884	373.712	1987.81
$RS1_{2023}$	168.264	140.732	303.601	324.469	333.430	360.268	330.036	1960.8
$RS2_{2020}$	244.424	117.437	318.42	327.209	292.067	278.053	375.231	1952.84
$RS2_{2023}$	156.583	196.494	291.539	328.933	279.245	231.899	371.836	1856.53
$BIRS_{2020}$	476.594	249.127	374.854	562.861	592.029	740.744	737.002	3733.21
$BIRS_{2023}$	301.502	290.458	602.875	672.985	661.153	739.168	766.912	4035.05

From the above results, it is clear that running times for RS1 and RS2 are (1) more or less the same, (2) less than, or (3) much less than the running times for BIRS. For instance, the running times for RS1 and RS2 are about 50% of the running times for BIRS with Single-Cell Data and Brunet Algorithm (see Table 5.6).

5.2.2 Discussion on Initialization and Stability

We start with a brief literature review regarding initialization and stability of NMF.

Lee and Seung [18], is the original paper introducing NMF and discussing the multiplicative update rule for solving NMF. It is also briefly mentioning initialization. Hoyer [13], introduces NMF with sparsity constraints and discusses initialization methods that promote sparsity. It also touches on the stability of NMF. In Berry, Browne, Langville., Pauca, and Plemmons [2], it presents various initialization methods for NMF and evaluates their impact on stability and performance. Cichocki, and Zdunek wrote [5] a comprehensive book on non-negative matrix and tensor factorizations that covers various aspects of NMF, including ini-

tialization strategies and stability. Ding, Li, and Jordan [7], discusses the issue of non-convexity in NMF and proposes a convex relaxation framework to enhance stability and uniqueness of the factorization. It also touches on initialization. In Gillis, and Vavasis [11], it proposes a randomized initialization technique called Adaptive Restart for NMF and discusses its stability and efficiency. Hsieh, Dhillon, Ravikumar, and Sustik [14], focuses on stability issues in NMF and discusses the importance of proper initialization and constraint enforcement for obtaining stable factorizations. Wang, and Gao [26], is a review paper provides a comprehensive overview of NMF, including initialization techniques and stability analysis. It discusses various initialization methods and their pros and cons. There are also some recent papers on initialization and stability of NMF by Lin, and Morup [20] and D'Amico, Gabbolini, Bernardis, and Cremonesi, [6].

Although there are many attempts have been made to tackle the issue of initialization and stability of NMF. None of them can solve the issue uniformly and globally which is very likely the mission impossible. People can only solve the problem partially under some special circumstances. Here, we want to make some brief comments between the random seedings and the deterministic seedings.

As we know that one of the common seeding methods is to use a random starting point, where the entries of W_0 and/or H_0 are drawn from a uniform distribution, usually within the same range as the target matrix's entries. This method is very simple to implement. However, a drawback is that in order to achieve stability, one has to perform multiple runs, each with a different starting point. This significantly increases the computation time needed to obtain a better factorization.

To tackle this problem, some methods have been proposed so as to compute a reasonable starting point from the target matrix itself. Their objective is to produce deterministic algorithms that need to run only once, still giving meaningful results[3] [22].

It seems like the deterministic is a better option than the random seedings. However, here is our second thought:

1. For any deterministic seeding and a given data set, it will produce a unique initial W and H matrices as starting point for iteration process. Hence it will be convergent to a minimum value of the loss function of the NMF. We can not tell it is the global minimum or a better local minimum. When we look into the details of any deterministic seedings, we can see that they are just like another matrix factorization process. It is somehow like that to use one matrix factorization process to find a good starting point and

then followed by another matrix factorization process. Hence, it will cost more computational running time. Moreover, the initial W and H matrices produced by deterministic seeding methods are not non-negative. In order to satisfy the non-negative constraints, they either take the absolute values of W and H or make all the negative entries in W and H to be zero. In our opinion, this will compromise the good quality that W and H suppose to have.

2. For any random seeding (with a specific seed) and a given data set, it will produce a unique initial W and H matrices as starting point for iteration process. It will be convergent to a minimum value of the loss function of the NMF. Again, we can not tell it is the global minimum or a better local minimum. However, the random seedings allow us to take multiple runs. Different runs will produce different starting points (initial W and H matrices) and may lead to different local minimums. Hence, we might be able to find a batter local minimum (possibly batter than deterministic seeding can produces).

A final comment:

As we can see from Section 5.1.1, our new random seedings (at least empirically) has less running times than the build in random seeding. Practically, by using our random seedings will allow us to take more runs of NMF. Hence, there is a batter chance to reach a better local minimum or possibly the global minimum of the NMF.

Chapter 6

Summary and Further Study

6.1 A Brief Summary

In this thesis, we have studied the non-negative matrix factorization, its rank estimation, classification, and stability. We have done a large amount of simulations on two simulated data and then extended the study to a real single-cell gene expression data. We have done the simulations by using two different seeding methods, eight different NMF algorithms and five different rank estimation criteria. The comparison of their performances are given. We have also investigated the stability of NMF. We have first studied the stability of NMF over the sample size consideration and then have had a brief discussion on initialization and stability of NMF in general. The comparison of two new random seedings with the built in random seeding are given.

The main contribution of this thesis is to test the performance of NMF and its stability, with respect to various categories of conditions including the choice of algorithms, sample size, and initialization. The detailed conditions that have been revealed by this thesis may generate practical impact in directing the appropriate use of NMF in analyzing single-cell gene expression data.

Additionally, during the course of the investigation, we have created the following techniques to assist the analysis and assessment:
1. We have proposed a numerical quantity to measure the degree of confusion of confusion matrix and then has applied it to rank estimation.
2. We have introduced two new random seedings and the comparison of the performance with the package build in random seeding is given. We find our new random seedings perform better than the build in random

seeding in running time.

3. We have suggested several new criteria for rank estimation.
4. We have suggested some criteria for gene selection and data reduction.
5. We have designed two simulated data sets with desired features for simulation study.

6.2 Potential Further Work

Sometimes, a single-cell gene expression data may contain sub-populations with unknown biological identity and functions or without gold-standard cell types. In this case, it is difficult to set up the r in the NMF analysis.

In the context of biological data analysis, particularly in single-cell RNA sequencing (scRNA-seq) studies, determining the gold-standard cell types and clusters is a crucial step in understanding cellular heterogeneity and function. Determining the gold-standard cell types and clusters in real datasets involves a combination of expert knowledge, data-driven analysis, and validation efforts. While efforts are made to ensure reliability, it's important to acknowledge the inherent complexities and uncertainties in the annotation process and to continually refine annotations as our understanding of biology advances.

Mathematically, one of the approaches for clustering is to use unsupervised clustering algorithms to find a rough estimate of sub-populations among all the cells. A possible method could be the K-means clustering (or any other classification methods) for an initial estimate [21]. Then one can perform NMF as usual.

Matching metacells with cell types involves associating clusters or groups of cells, known as metacells, with known cell types based on their gene expression profiles or other relevant features. This task is commonly performed in single-cell RNA sequencing (scRNA-seq) data analysis to assign biological annotations to clusters of cells.

Mathematically speaking, when the r value is the same as the number of cell types, the confusion matrix is a square matrix. We may want to match the metacells with the cell types based on the confusion matrix. When the degree of confusion is low, the match could be obvious. On the other hand, when the degree of confusion is relatively high, the matching would not be straight forward. A matching method called Gale-Shapley Algorithm (GSA)[9] might be a possible approach.

It's important to note that matching metacells with cell types is an ongoing field of research. The specific method used may depend on the characteristics of the dataset, available references, and the research question at hand. It seems worth some further study in this direction.

Bibliography

[1] Liviu Badea. Extracting gene expression profiles common to colon and pancreatic adenocarcinoma using simultaneous nonnegative matrix factorization. In *Pacific Symposium on Biocomputing*, volume 13, pages 267–278. World Scientific, 2008.

[2] Michael W Berry, Murray Browne, Amy N Langville, V Paul Pauca, and Robert J Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational statistics & data analysis*, 52(1):155–173, 2007.

[3] Christos Boutsidis and Efstratios Gallopoulos. Svd based initialization: A head start for nonnegative matrix factorization. *Pattern Recognition*, 41(4):1350–1362, 2008.

[4] Jean-Philippe Brunet, Pablo Tamayo, Todd R Golub, and Jill P Mesirov. Metagenes and molecular pattern discovery using matrix factorization. *Proceedings of the National Academy of Sciences*, 101(12):4164–4169, 2004.

[5] Andrzej Cichocki and Rafal Zdunek. *Nonnegative matrix and tensor factorizations: Applications to exploratory multi-way data analysis and blind source separation*. John Wiley & Sons, 2009.

[6] Enrico D'Amico, Claudia Gabbolini, Cristian Bernardis, and Paolo Cremonesi. Analyzing and improving stability of matrix factorization for recommender systems. *Journal of Intelligent Information Systems*, 58(2):255–285, 2022.

[7] Chris Ding, Tao Li, and Michael I Jordan. Convex and semi-nonnegative matrix factorizations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(1):45–55, 2010.

[8] Flavio Esposito. A review on initialization methods for nonnegative matrix factorization: Towards omics data experiments. *Mathematics*, 9(9):1006, 2021.

[9] David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–14, 1962.

[10] Renaud Gaujoux and Cathal Seoighe. A flexible r package for nonnegative matrix factorization. *BMC bioinformatics*, 11(1):367, 2010.

[11] Nicolas Gillis and Stephen A Vavasis. Fast and robust recursive algorithms for separable nonnegative matrix factorization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8):1587–1601, 2011.

[12] Saeed F Hafshejani and Zahra Moaberfard. Initialization for non-negative matrix factorization: A comprehensive review. *International Journal of Data Science and Analysis*, 16(1):119–134, 2023.

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

[14] Cho-Jui Hsieh, Inderjit S Dhillon, Pradeep K Ravikumar, and Masa A Sustik. Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. *BMC bioinformatics*, 12(1):1–13, 2011.

[15] Lucie N Hutchins, Shannon E Murphy, Sandeep Singh, and Joel H Graber. Position-dependent motif characterization using non-negative matrix factorization. *Bioinformatics*, 24(23):2684–2690, 2008.

[16] Richard A Johnson and Dean W Wichern. *Applied multivariate statistical analysis*. Pearson Prentice Hall Pearson Education, Inc., 2007.

[17] Hanchuan Kim and Haesun Park. Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. *Bioinformatics*, 23(12):1495–1502, 2007.

[18] Daniel D Lee and H Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

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

[20] Chih-Jen Lin and Morten Morup. Initialization strategies for nonnegative matrix factorization with beta-divergence. *Neurocomputing*, 275:1889–1897, 2018.

[21] James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[22] Jonathan L Marchini, Christopher Heaton, and Brian D Ripley. fastica: Fastica algorithms to perform ica and projection pursuit. *R package version*, 1:2–0, 2013.

[23] Alberto Pascual-Montano, Jose-Maria Carazo, Katsuhisa Kochi, Dominik Lehmann, and Roberto D Pascual-Marqui. Nonsmooth nonnegative matrix factorization (nsnmf). *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(3):403–415, 2006.

[24] Can Shao and Thomas Hofer. Robust classification of single-cell transcriptome data by nonnegative matrix factorization. *Bioinformatics*, 33(2):235–242, 2017.

[25] Lifeng Tian, Sheng Su, Xiaoxi Dong, Daniela Amann-Zalcenstein, and David A Zalcenstein. scpipe: A flexible r/bioconductor preprocessing pipeline for single-cell rna-sequencing data. *PLoS computational biology*, 14(8):e1006361, 2018.

[26] Xiangyu Wang and Xin Gao. Nonnegative matrix factorization: A comprehensive review. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1336–1353, 2013.