Integrative Analyses for Multi-source Data with Multiple Shared Dimensions

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Michael J. O'Connell

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Advised by Eric Lock, PhD and Wei Pan, PhD

July, 2018

Acknowledgements

First and foremost, I would like to thank my advisor, Eric Lock, whose guidance and support made this dissertation possible. I found his research intriguing the first time I saw him present it, and I appreciate his patience with me while I learned the theory and methods surrounding multi-source data analysis. He has consistently provided me with useful guidance and recommendations on my projects, and he has also helped me debug my code many times when I was stuck. I would like also like to thank my committee members Wei Pan, Saonli Basu, and Peter Bitterman for their helpful feedback at various stages of the projects. I would like to thank my advisors from BDAC, Cindy Davies and Kyle Rudser, who mentored me for the first two years at the University of Minnesota, providing me with experience with many different statistical methods as well as with collaborative research.

I would like to thank the University of Minnesota Graduate School for providing the funding for my final year through the Doctoral Dissertation Fellowship. I would also like to thank the support staff at mlb.com, who were prompt and helpful in providing me the data I needed for Chapter 4.

Finally, I would like to thank my family for their moral support. I could not have gotten to this point in my life without the encouragement and guidance of my mother.

Dedication

For my parents, who taught me to always push my boundaries in life.

Abstract

High dimensional data consists of matrices with a large number of features and is common across many fields of study, including genetics, imaging, and toxicology. This type of data is challenging to analyze because of its size, and many traditional methods are difficult to implement or interpret with such data. One way of handling high dimensional data is dimension reduction, which aims to reduce high rank, high-dimensional data sets into low-rank approximations, which maintain important components of the structures of the matrices but are easier to use in models. The most common method for dimension reduction of a single matrix is principal components analysis (PCA). Multi-source data are high dimensional data in which multiple data sources share a dimension. When two or more data sets share a feature set, this is called horizontal integration. When two or more data sets share a sample set, this is called vertical integration. Traditionally, there are two ways to approach such a data set: either analyze each data source separately or treat them as one data set. However, these analyses may miss important features that are unique to each data source or miss important relationships between the data sources. A number of recent methods have been developed for analyzing multi-source data that are either vertically or horizontally integrated. One such method is Joint and Individual Variation Explained (JIVE), which decomposes the variation in multi-source data sets into structure that is shared between data sources (called joint structure) and structure that is unique to each of the data sources (called individual structure) [Lock et al., 2013]. We have created an R package, r.jive, that implements the JIVE algorithm and provides visualization tools for multi-source data, making multi-source methods more accessible. While there are several methods for data sets with horizontal or vertical integration, there have been no previous methods for data sets with simultaneous horizontal and vertical integration (which we call bidimensional integration). We introduce a method called Linked Matrix Factorization that allows for simultaneous decomposition of multi-source data sets with bidimensional integration. We also introduce a method for bidimensionally integrated data that are not normally distributed, called Generalized Linked Matrix Factorization, which is based on generalized linear models rather than ordinary least squares.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Dimension Reduction

Recent technological advances in biomedical research have led to a growing number of platforms for collecting large amounts of health data. Genetic sequencing, gene expression measurements, and MRI scans often result in expansive, unwieldy data sets. These data sets are known as high-dimensional data, and they often contain thousands or more features. Because of their size, these data sets present interesting challenges because they can be difficult to visualize and are often structurally complex. They typically have very high rank, which is a measure of the number of patterns present in the data. One way to reduce their complexity is dimension reduction. Dimension reduction methods decompose large high-rank matrices into smaller, low rank components. These methods extract the features that explain most of the variability while making data much simpler for data visualization and statistical models. There are a number of matrix decomposition methods that can be used for dimension reduction.

1.1.1 Principal Components Analysis

The most commonly used method for dimension reduction is principal component analysis (PCA). PCA is used to find a small number of patterns in a data set that can explain most of its variability. The result of PCA is a set of scores, which represent patterns in the rows, and loadings, which represent patterns in the columns. Principal component analysis is often used for dimension reduction in regression models, as the first few principal components scores can be used as covariates in a regression model. This is especially useful when the number of covariates is greater than the number of observations.

A PCA can be computed using a singular value decomposition (SVD). An SVD of a matrix X has the following form:

$$X = U\Sigma V^T,$$

where Σ is a diagonal matrix containing the singular values of X and U and V are unitary. The columns of U and V are the left and right singular vectors, respectively. U represents the loadings, and V represents the scores. In the context of gene expression data, where the columns of X are the samples and the rows are the genes, then the rows of U represent the patterns that account for the variability across the genes, and the rows of V represent the patterns that account for the variability across the samples. An important component of the SVD is the rank. The rank of the SVD determines number of components in the model. By choosing a rank smaller than the rank of the original data, we can obtain a low-rank approximation of the data. If X is an $m \times n$ matrix, then a rank r SVD will yield $U_{m\times r}$ and $V_{n\times r}$.

When the data set is centered and scaled, PCA is equivalent to computing the SVD. Using the SVD, the scores of a PCA are ΣV^T and the loadings are U.

1.1.2 Alternating Least Squares

Another method for dimension reduction of a matrix is alternating least squares (ALS) [De Leeuw et al., 1976]. Alternating least squares is a method that can be used to estimate regression parameters when both the covariates and the coefficients are unknown. Consider the following regression equation: $Y = \alpha\beta + E$. We can obtain least squares estimates for α and β by alternating between estimation of α holding β fixed and estimation of β holding α fixed. In both cases, the estimates can be obtained using ordinary least squares regression. In a matrix context, we can use this to estimate scores and loadings for a rank r approximation, alternatively estimating U with V fixed and V with U fixed to minimize $\|X - UV^T\|$. Consider a matrix $X_{m\times n}$ with loadings $U_{m\times r}$ and scores $V_{n\times r}$, where $X \approx UV^T$. The algorithm to estimate the ALS decomposition for a matrix is given below:

1. Initialize $\hat{V}$.
2. Treating $\hat{V}$ as fixed, estimate $\hat{U}$ using ordinary least squares ($\hat{U} = (\hat{V}^T\hat{V})^{-1}\hat{V}^T X$)
3. Treating $\hat{U}$ as fixed, estimate $\hat{V}$ using ordinary least squares ($\hat{V} = (\hat{U}^T\hat{U})^{-1}\hat{U}^T X$)
4. Repeat steps 2 and 3 until the solution converges.

If r is chosen smaller than the rank of the original data set, then $\hat{X} = UV^T$ is a low-rank approximation of X.

1.1.3 Exponential PCA

PCA and ALS are both based on an assumption that the residuals from the low-rank approximation are normally distributed. This may not always be a valid assumption, for example binary outcomes may be best modeled with a binomial distribution. Collins et al. [2002] generalized PCA to model any exponential family distribution with the exponential PCA model. A detailed explanation of exponential family distributions is given in Chapter 4.

1.2 Multi-source Data

One particular challenge to analysis of high-dimensional data sets comes when an analysis involves multiple data sets that are related to each other. When multiple data sets share a common sample set (e.g. gene expression and protein expression data for the same set of samples) or a common feature set (e.g. gene expression data from samples of different tumor subtypes), we call this multi-source data. Although they share a sample set or a feature set, each of these data sources may have different features that make them unique. There are three types of multi-source data that we will address in this paper: vertical integration, horizontal integration, and bidimensional integration.

1.2.1 Vertical Integration

Vertically integrated multi-source data sets have a shared sample set. Vertical integration is common in genetics data, where multiple types of data, such as gene expression, DNA methylation, and miRNA expression, are often collected for the same set of samples. Each data source has the same samples along the columns, but the features of each are different along the rows. As an example, the breast cancer data set we consider in Chapter 2 has three types of features (mRNA expression, DNA methylation, and miRNA expression) for one common set of tumor samples [Cancer Genome Atlas Network, 2012].

1.2.2 Horizontal Integration

Horizontally integrated data sets share a feature set. These data involve the same type of data (such as gene expression) collected across multiple different sample sets. For example, Yang et al. [2017] used the `r.jive` package (Chapter 2) to analyze gene expression of glioblastoma samples from both males and females. Both data sources share a common set of features gene expression), but they have different sample sets (male glioblastoma samples and female glioblastoma samples, respectively).

1.2.3 Bidimensional Integration

Bidimensional integration occurs in data sets that involve both vertical and horizontal integration. While there have been many recent methods developed for analyses of vertical and horizontally integrated multi-source data sets, there have been no methods to date for analyzing bidimensional data. In their book *Integrating Omics Data*, Tseng et al. [2015] do not discuss any methods for bidimensionally integrated data sets even though much of the book is about vertical integration and horizontal integration separately. We discuss bidimensional integration in more detail in Chapters 3 and 4.

1.2.4 Motivating Data Sets

In Chapter 2, we consider a breast cancer data set from The Cancer Genome Atlas (TCGA) [Cancer Genome Atlas Network, 2012]. This publicly available multi-source genomic data set contains 3 data sources for 348 breast cancer (BRCA) tumor samples: mRNA expression, DNA methylation, and miRNA data. All three of these data sources share a common set of tumor samples, but each contains a different type of information concerning the genome or its transcription.

We consider bidimensional integration in Chapter 3. The motivating example for that chapter is a cytotoxicity data set, which consists of three data sources: cytotoxicity data (cell lines vs. chemicals), chemical attribute data (chemicals vs. attributes), and genetic data for the cell lines (cell lines vs. genes) [Abdo et al., 2015]. In this case, two of the data sets share a sample space (cell lines) and two share a feature space (chemicals).

We consider another application with bidimensional integration in Chapter 4. Our motivating example in that chapter is a baseball data set with three data sources: batting data, pitching data, and batting averages for specific batters against specific pitchers. The batter versus pitcher data shares its batters with the batting data and its pitchers with the pitching data. In this case, the batter versus pitcher data are proportions, so they are better modeled using a binomial distribution.

1.3 Multi-source Dimension Reduction

Until recently, there was little statistical methodology for integrative analyses of high-dimensional multi-source data. Previously, multi-source data had often been either treated as separate analyses or treated as a single combined data set and analyzed without distinction. However, both of these approaches have limitations. Treating the data as a single data source may miss important features that are unique to each of the data sources. Likewise, analyzing these data sources separately may miss important associations or interactions between them. These limitations have motivated a number of methods for the integrative analysis of multi-source data, allowing analyses of these data sources together while recognizing that each data source may possess unique features.

There are several methods for estimating the structure that is common to multiple data sources (which we refer to as joint structure). One of these models is canonical-correlation analysis (CCA) [Hotelling, 1936]. CCA is a method that focuses on the correlation between data sources. It is used to find a linear combination of two matrices such that their correlation is maximized. For matrices X_1 and X_2, the first pair of CCA components is

$$\text{argmax}_{w_1,w_2}\text{Corr}(w_1X_1, w_2X_2).$$

Additional pairs of CCA components are computed by maximizing this correlation while enforcing orthogonality with previous components. CCA has problems with overfitting when there are more features than samples, so it is generally not useful for high-dimensional data. Partial least squares regression (PLS) [Wold, 2004] is a similar method, but instead of finding a linear combination that maximizes the correlation, PLS finds a linear combination that maximizes the covariance. The first pair of PLS components are

$$\text{argmax}_{w_1,w_2}\text{Cov}(w_1X_1, w_2X_2).$$

Unlike CCA, PLS does not have issues with overfitting and is applicable to high-dimensional data. However, structured noise present in X_1 but not X_2 and visa versa can have a strong influence on the PLS components.

Simultaneous component analysis (SCA) is another set of methods for finding the joint variation [Kiers and ten Berge [1989];Millsap and Meredith [1988];ten Berge et al. [1992];Van Deun et al. [2009]]. For K data sources $\{X_1, ..., X_K\}$ the SCA decomposition can be found using the following expression:

$$\min_{T,P_k}\sum_{k=1}^{K}\|X_k - TP_k^T\|^2$$

In this case, the resulting matrix T represents the joint scores, and each P_k represents the joint loadings for each data source. SCA is essentially a PCA of the concatenated matrix $[X_1^T|...|X_K^T]^T$.

Some recent methods have been developed to use a decomposition of the data that separates the structure that is common to all of the sources (which we refer to as joint structure) and the structure that is unique to each source (which we refer to as individual structure). Some methods for computing joint variation alone are heavily influenced by the individual structure. We alluded to this issue for PLS, and it is also a limitation of SCA [Schouteden et al., 2013]. This issue can be resolved by estimating both the joint and the individual structures of the data and by constraining the components so that they are orthogonal to each other. In addition, this allows us to consider the unique contributions of each data set to the overall structure of the data, capturing some relationships that may have been missed by the joint structure only methods.

As mentioned previously, structured noise present in X_1 but not X_2 and visa versa can have a strong influence on the PLS components. This issue is fixed with O2-PLS [Trygg, 2002].O2-PLS is similar to PLS, but it instead fits the structured noise unique to each of X_1 and X_2. The covariate components and the unique components are estimated such that they are orthogonal to each other. O2-PLS is limited to the case of 2 data sources, but the On-PLS method further extends this algorithm to the case of more than two data sources [Löfstedt and Trygg, 2011b].

DIStinct and COmmon-Simultaneous Component Analysis (DISCO-SCA) is similar to SCA, but it finds both joint (common) and individual (distinct) variation [Schouteden et al., 2013]. To accomplish this, it first finds the SCA components that contribute significantly to the covariance of the data. Then, it rotates the remaining data so that it is orthogonal to the joint components.

Common and Orthogonal Basis Extraction (COBE) is another method for separating the joint and individual structure of a multi-source data set [Zhou et al., 2015]. This algorithm uses a QR decomposition to find the basis vectors for the joint structure, then finds individual structure such that the individual structure is orthogonal to the joint structure.

1.3.1 Joint and Individual Variation Explained

One decomposition method developed to handle these multi-source data sets is Joint and Individual Variation Explained (JIVE) [Lock et al., 2013]. JIVE was developed as a multi-source extension of principal components analysis (PCA) that allows the simultaneous decomposition of multiple matrices which share a common sample set. JIVE quantifies the amount of joint (shared) variation between data sources as well as the amount of individual (unique) variation specific to each data source, reduces dimensionality, and allows for visual exploration of joint and individual structure.

JIVE can also be used as a processing step prior to the application of other methods, such as clustering techniques [Hellton and Thoresen, 2014]. JIVE was designed for the analysis of biomedical data from multiple technologies, but has been used for other diverse applications, such as the analysis of data that were processed using different computational pipelines [Kuligowski et al., 2015] and the analysis of rail commute patterns at different times of day [Jere et al., 2014].

The algorithm alternates between computing singular value decompositions (SVDs) of the joint matrix, which is the vertical concatenation of feature matrices, and of the individual matrices to minimize the overall sum of squared residuals. The decomposition results in three components: (1) a matrix representing the joint structure of the data, (2) a matrix representing the individual structure of each data set, and (3) a residual matrix containing the variability in the data set that cannot be attributed to the joint or individual structures. The factorized form of the JIVE model for two data sources, X_1 and X_2, is shown given by the following equations:

$$X_1 = U_1 S + W_1 S_1 + E_1$$

$$X_2 = U_2 S + W_2 S_2 + E_2$$

U_1 and U_2 are the joint loadings, S is the shared set of joint scores, W_1 and W_2 are the individual loadings, S_1 and S_2 are the individual scores, and E_1 and E_2 are the residuals.

An important factor to consider when estimating matrix decompositions is rank selection. The rank of a matrix decomposition is the number of components in each part. This is a particularly challenging aspect of multi-source data analysis because the overall rank of the full data set can be attributed to a combination of joint structure, individual structure, and random noise. Therefore, with multi-source data, we need to simultaneously estimate the joint and individual ranks of the data. A non-parametric option for rank selection is a permutation test, as originally described in Lock et al. [2013]. Alternatively, BIC model selection offers a parametric alternative for choosing ranks. This method was initially implemented in Jere et al. [2014] and is extended in Chapter 2 [O'Connell and Lock, 2016].

1.3.2 Generalized Association Study

Like SVD, JIVE is based on the assumption that, given the low rank approximation, the residuals are normally distributed. In practice, the data may follow some other distribution. For non-normal multi-source data, Li and Gaynanova [2017] developed the Generalized Association Study (GAS), a method for simultaneous decompositions of multiple matrices that follow exponential family distributions. GAS allows for analysis of heterogeneous data, or data that contain different error distributions. JIVE is a special case of GAS in which the data sets are all normally distributed.

1.4 Contributions

While there have been many recent multi-source data methods developed, they tend to be limited to particular data structures. In particular, most methods, including JIVE [Lock et al., 2013], are limited to the case where only the sample set is shared between data sets. My dissertation research has focused on two primary goals: (1) improving existing methodology for multi-source data (Chapter 2) and (2) developing new methods to accomodate diverse forms of multi-source data that are not addressed with existing methodology (Chapter 3 and 4).

In my first project, we created an R package that implements and extends the JIVE algorithm [O'Connell and Lock, 2016]. Previously, only spartan Matlab code was available to perform JIVE. We created the `R.JIVE` package, for which our intentions were threefold: (1) to improve the accessibility of this method among the bioinformatics community, (2) to implement important extensions and improvements of the JIVE method, and (3) to allow for quick and flexible visualization of JIVE results. This package is covered in more detail in Chapter 2.

My second project was to extend JIVE to situations in which at least one data source shares both its sample set and its feature set with other data sources (in particular, the case where its sample set is shared with one matrix and its feature set with a different matrix, see Figure 1). We have created an algorithm for these data sets which we call Linked Matrix Factorization (LMF), which is discussed in detail in section 3. The motivating example for this project was a toxicology data set, in which there were three data sources: (1) cytotoxicity data, (2) cell line genetics data, and (3) chemical attribute data. In this data set, the cytotoxicity data shares one dimension (cell lines) with the genetic data, but it shares its other dimension (chemicals) with the chemical attribute data. The LMF algorithm allows us to analyze the variability of such data sets, which previous multi-source decomposition methods would not be able to process.

Like JIVE, GAS [Li and Gaynanova, 2017] is limited to data which share either feature sets or sample sets, but not both. My third project was to create a method that was able to handle exponential family data like GAS in the context of matrices with both shared feature sets and sample sets like LMF. This method, Generalized Linked Matrix Factorization is detailed in Chapter 4.

Chapter 2

R.JIVE

In this chapter, we introduce an R [R Core Team, 2018] package called `r.jive` [O'Connell and Lock, 2017], which implements the JIVE algorithm. The primary goal of this project was to improve the accessibilty of the JIVE algorithm [Lock et al., 2013] to other researchers. We also made some improvements and extensions to the algorithm, which give users more flexibility with the package. Additionally, we added several plotting functions to help visualize the results of the JIVE decomposition.

2.1 The JIVE model

JIVE decomposes a multi-source dataset into three components: an approximation of rank r capturing joint variation across sources (J), approximations of rank r_i for structured variation individual to each source (A), and residual noise (E). For a multi-source data set $\{X_1, ..., X_n\}$, the decomposition for the i^th data source is as follows.

$$X_i = J_i + A_i + E_i$$

For the decomposition to be unique, J and A must be orthogonal to each other. This constraint is imposed during the estimation of the model. For dimension reduction and interpretation it is helpful to consider the factorized form of the JIVE decomposition, which is analogous to PCA. In PCA, the factorized decomposition of a matrix X can be written as $X = US$, where U represents the loadings and S represents the scores.

Joint structure corresponds to an r-dimensional sample subspace revealing patterns that explain substantial variability across multiple sources, whereas individual structure corresponds to an r_i-dimensional sample subspace revealing patterns that explain substantial variation in one source but not others. Below is the factorized model for a JIVE decomposition of two data sources, $X_{m_1 \times n}$ and $Y_{m_2 \times n}$.

$$X = U_1 S + W_1 S_1$$

$$Y = U_2 S + W_2 S_2$$

U_1 and U_2 are the source-specific joint loading matrices, which have dimension $m_1 \times r$ and $m_2 \times r$, respectively. S is the $r \times n$ joint score matrix, which is the same for both X and Y. These joint scores can be used to identify any patterns in the joint structure of the data. W_1 and W_2 are the source-specific individual loading matrices, which also have dimension $m_1 \times r$ and $m_2 \times r$, respectively. S_1 and S_2 are the source-specific individual score matrices, which have dimension $r \times n$ and $m_2 \times r$, respectively. These individual score matrices can be used to identify any patterns exclusive to a particular data source. For identifiability of the JIVE decomposition, it is necessary and sufficient that the rows of the joint structure J and the rows of the individual structure A_i are orthogonal for each data source. If we have k data sources, and m is the total rows across all data sources, this means $JA_i^T = 0_{m \times m}$ for $i = 1, ..., k$. Any combination of J and A_i can be transformed such that this condition holds, so this ortogonality constraint does not restrict the solution space.

2.2 Improvements

One improvement we made to the algorithm was adding the ability to handle matrices with missing data. To do this, we used the SVDmiss function from the package SpatioTemporal [Lindstrom et al., 2018]. Normally, an SVD is undefined when missing values are present, but SVDmiss allows us to estimate the SVD by alternating between SVD and least squares estimation to impute missing values. This is an important improvement to the JIVE algorithm because missing data is very common in practice.

Another extension made to the JIVE algorithm was the option to enforce orthogonality between individual structure matrices. The original JIVE algorithm enforces orthogonality between the joint and individual structures but not between each of the individual structures. If ranks are properly specified, then the individual structures will be orthogonal to each other because any non-orthogonal variance will be placed in the joint structure. However, enforcing the orthogonality between individual structure matrices allows the model to be more robust to rank misspecification.

Finally, we included two different rank selection methods in the package. The first one, a permutation-based approach, was presented in the original JIVE paper [Lock et al., 2013]. The second rank selection method is a forward selection algorithm using BIC as the model criterion. The BIC rank selection algorithm included in the package is a modified version of the algorithm proposed in Jere et al. [2014]. This rank selection algorithm is motivated by considering the JIVE results as a parameterized model with normally distributed error:

$$X_i = U_i S + W_i S_i + E_i,$$

where $X_i : d_i \times n$ are data for source i and the entries of E_i (the residual noise) are assumed to be independent and normally distributed with mean 0 and variance σ_i^2. In the factorized decomposition, $U_i : d_i \times r$ gives the loadings for the joint structure, $S : r \times n$ is the joint score matrix, $W_i : d_i \times r_i$ gives the loadings for the individual structure, and $S_i : r_i \times n$ is the individual score matrix. Under this framework, the BIC for the given model depends on the number of entries in each data matrix N_i (because data are compressed via SVD beforehand, $N_i = n^2$ if $d_i > n$ in the original data), the sum of squared error for each data matrix $\text{SSE}_i = ||X_i - U_i S - W_i S_i||_F^2$, and the total number of free parameters p. The number of free parameters p is essentially the total number of entries in the score and loading matrices, but accounts for orthogonality restrictions:

$$p = \sum_{j=0}^{r-1}(n-j) + \sum_{i=1}^{k}\sum_{j=0}^{r-1}(d_i - j) + \sum_{i=1}^{k}\sum_{j=0}^{r_i-1}[(n-r-j)+(d_i-j)].$$

The formula for BIC is then

$$\sum_{i=1}^{k} N_i \log(\mathrm{SSE}_i/N_i)) + p \cdot \log\left(\sum_{i=1}^{k} N_i\right).$$

The algorithm for estimating ranks based on BIC is a forward selection procedure.

Several functions are provided to summarize and generate quick publication-quality visualizations of the results in the form of a barchart, heatmaps, or PCA plots. The first of these are barcharts showing the distribution of variability among the joint structure, individual structure, and residuals. These plots can provide the user with an idea of how much of the variability is attributed to each source. The second type of plots are the heatmaps, which allow the visualization of patterns in the data. The heatmaps can be sorted based on either the joint structure or the indiidual structure, allowing the user to see clustering patterns in both joint and individual structure. The third type of plot is a PCA plot, which plots the components of either the joint of individual structure against each other, which provides another way to search for clusters in the data. Examples of all of these plots are given in section 2.7.

2.3 The JIVE algorithm

The JIVE algorithm as implemented by `r.jive` begins with an SVD using the SVDmiss function from the package SpatioTemporal on each of the data sources. With this initial SVDmiss step, we accomplish two things. First, it allows us to use the JIVE algorithm on data sets with missing data. Second, we can use this step for dimension reduction when the shared dimension is smaller than the non-shared dimension. We can do this because the data is at most rank n, where n is the number of columns in the data set. From the SVD, the reduced data set is ΣV^T for each data source. This data reduction improves the computation time of the algorithm.

The next step is centering and scaling the data set. To center, we subtract off the row means from each data set. To scale, we divide each data source by its Frobenius norm. Without scaling, true joint structure might be misclassified as individual structure because of a difference in variances between data sets. The subsequent steps depend on the chosen rank selection method.

If the ranks are known, the algorithm procedes directly to the JIVE decomposition using the given ranks. For k data sources $\{X_1', ..., X_k'\}'$, the individual structure matrix A = $\{A_1', ..., A_k'\}'$ is initialized to a matrix of 0's. Then it creates a new matrix X$_{Joint}$ = X - A. It then compute the joint structure matrix J = $\{J_1', ..., J_k'\}'$ by a rank r singular value decomposition of X$_{Joint}$. If we are enforcing orthogonality between individual structures, it creates X$_{Individual}$ = (X - J)(I - VV')$\prod_{k \neq i}(I - V_i V_i')$. Otherwise, it sets X$_{Individual}$ = (X - J)(I - VV'). This step ensures the orthogonality of the joint and individual structures by projecting the individual structure matrix onto the space orthogonal to the joint structure. For each data source, it then computes the individual structure matrix A$_i$ by a rank r$_i$ singular value decomposition of X$_{Individual}$. Repeating these steps, it alternates estimating J and A until the solution converges.

If permutation test rank selection option is chosen, we begin with the permutation test, as described in Lock et al. [2013]. Then, the JIVE decomposition is computed as above using the ranks chosen by the permutation test. The permutation test and the JIVE algorithm are then alternated until the permutation test selects the same ranks in consecutive iterations.

For BIC rank selection, the package uses a forward selection algorithm. First, it calculates BIC for the JIVE decomposition with ranks of 0 for joint and individual structure. Then, it calculates the BIC for each model with one added to a single rank (either to joint rank or to one of the individual ranks). These BIC values are then compared. If the lowest BIC is the model with all rank 0, then that model is chosen and the algorithm stops. Otherwise, it continues to add one to each of the ranks and calculate the BIC values for each model. When adding one to any of the ranks increases BIC, then the selection algorithm stops and the model with the lowest BIC is chosen.

2.4 Pseudocode

The `r.jive` package uses four different algorithms to handle the computation of the JIVE decomposition. The pseudocode for the algorithms is provided here.

2.4.1 `jive(...)`

All JIVE decompositions should be called using the `jive(...)` function. The `jive(...)` function then makes calls to other jive-based functions depending on the method. Here is the algorithm:

1. Use SVDmiss (package SpatioTemporal) to predict and replace missing values.

2. Center each row of the data (unless otherwise specified).

3. By default, scale each data set by dividing by the Frobenius norm (unless otherwise specified).

4. If using known ranks or the BIC selection algorithm, reduce each data source using a singular value decomposition. Using the singular values (Σ) and the right singular values (V) of the given source, the reduced data set is $\Sigma V'$. Save the left singular vectors (U). If using the permutation rank selection algorithm, this step is done within that function for each iteration (see `jive.perm`).

5. Call the appropriate function to evaluate the JIVE decomposition. If the ranks are known, go straight to the jive iteration function (see `jive.iter`). If using a permutation test to estimate ranks, go to the permutation test rank selection algorithm (see `jive.perm`). If using BIC to select ranks, go to the BIC rank selection algorithm (see `bic.jive`).

6. For each source, transform the results back to the original dimensions by premultiplying the joint and individual results each by the left-singular vectors for that source (given ranks and BIC-selected ranks only).

7. Return results (lists for joint and individual structure matrices and ranks used in the decomposition).

2.4.2 `jive.iter(...)`

This function evaluates the JIVE decomposition for a list of k data sets for a given set of ranks (joint rank r and individual ranks r_1, ..., r_k). The algorithm is:

1. Initialize individual structure matrix A = $\{A'_1, ..., A'_k\}'$ to a matrix of 0's with dimensions of the original data (X = $\{X'_1, ..., X'_k\}'$).

2. Set X$_{Joint}$ = X - A

3. Set J = $\{J'_1, ..., J'_k\}'$ by a rank r singular value decomposition of X$_{Joint}$. Save the right singular values (V). If r=0, set J and V to matrices of 0's.

4. If enforcing orthogonality between individual structures and this is not the first iteration of this algorithm, set X$_{Individual}$ = (X - J)(I - VV')$\prod_{k\neq i}(I - V_i V'_i)$. Otherwise, set X$_{Individual}$ = (X - J)(I - VV').

5. For each i in 1 to k, set A$_i$ by a rank r$_i$ singular value decomposition of X$_{Individual}$. If enforcing orthogonality between individual structures, save the right singular values (V$_i$).

6. If enforcing orthogonality between individual structures and this is the first iteration, set A$_i$ = A$_i \prod_{k\neq i}(I - V_i V'_i)$.

7. Repeat steps 2-6 until the Frobenius norm of the difference between the current and previous iteration in both J and A is less than some threshold.

8. Return results (J, A, and the ranks used in the decomposition).

2.4.3 `jive.perm(...)`

This function runs the permutation test rank selection algorithm, which is the default method for choosing ranks in **r.jive**. The algorithm is as follows:

1. Choose the number of permutations, nperms, and the significance threshold, α.

2. Estimation of joint rank r:

   (a) Calculate the singular values of the original data (X = $\{X'_1, ..., X'_k\}'$). Let λ_j be the jth singular value.

   (b) Permute the columns within each data set (X$_i$).

   (c) Calculate the singular values of each permutation. Let λ_j^{perm} be the 100(1 - α) percentile of the jth singular values from the permuted matrices.

   (d) Choose r such that $\lambda_j > \lambda_j^{perm}$ for all j $\leq$ r.

3. Estimation of each individual rank r$_i$:

   (a) Calculate the singular values of the original data (X$_i$). Let λ_j be the jth singular value.

   (b) Permute the columns within each row.

   (c) Calculate the singular values of each permutation. Let λ_j^{perm} be the 100(1 - α) percentile of the jth singular values from the permuted matrices.

   (d) Choose r$_i$ such that $\lambda_j > \lambda_j^{perm}$ for all j $\leq$ r$_i$.

4. Reduce each data source with a singular value decomposition. Using the singular values (Σ) and the right singular values (V), the reduced data set is ΣV'. Save the left singular vectors (U).

5. Get the JIVE decomposition (see `jive.iter`) of X using the current ranks r and r$_1$, ..., r$_k$.

6. Transform the results for each data source back to the original dimensions by premultiplying the joint and individual results each by the left singular vectors for that source.

7. Repeat steps 2-6, replacing X with (X - A) in joint rank estimation and X$_i$ with (X$_i$ - J$_i$) in individual rank estimation, until two consecutive iterations give the same ranks.

8. Return results (J, A, and the ranks chosen).

2.4.4 `bic.jive(...)`

This function runs a BIC rank selection algorithm, a modified version of the algorithm proposed in Jere et al. [2014]. This algorithm is motivated by considering the JIVE results as a parameterized model with normally distributed error:

$$X_i = U_i S + W_i S_i + E_i,$$

where $X_i : d_i \times n$ are data for source i and the entries of E_i (the residual noise) are assumed to be independent and normally distributed with mean 0 and variance σ_i^2. In the factorized decomposition, $U_i : d_i \times r$ gives the loadings for the joint structure, $S : r \times n$ is the joint score matrix, $W_i : d_i \times r_i$ gives the loadings for the individual structure, and $S_i : r_i \times n$ is the individual score matrix. Under this framework, the BIC for the given model depends on the number of entries in each data matrix N_i (because data are compressed via SVD beforehand, $N_i = n^2$ if $d_i > n$ in the original data), the sum of squared error for each data matrix $\text{SSE}_i = ||X_i - U_i S - W_i S_i||_F^2$, and the total number of free parameters p. The number of free parameters p is essentially the total number of entries in the score and loading matrices, but accounts for orthogonality restrictions:

$$p = \sum_{j=0}^{r-1}(n-j) + \sum_{i=1}^{k}\sum_{j=0}^{r-1}(d_i - j) + \sum_{i=1}^{k}\sum_{j=0}^{r_i-1}[(n-r-j)+(d_i-j)].$$

The formula for BIC is then

$$\sum_{i=1}^{k} N_i \text{log}(\text{SSE}_i/N_i)) + p \cdot \log\left(\sum_{i=1}^{k} N_i\right).$$

The algorithm for estimating ranks based on BIC is a forward selection procedure:

1. Initialize joint rank r and all individual ranks r$_i$ to 0.

2. Calculate BIC for the JIVE decomposition with ranks r and r$_1$, ..., r$_k$. Set this as the current BIC value.

3. Calculate BIC for the JIVE decomposition with ranks (r+1) and r$_1$, ..., r$_k$ and the decomposition with rank r and r$_1$, ..., r$_i$ + 1, ..., r$_k$ for each i.

4. If any of the BIC values from step 3 are less than the current BIC value, set the ranks to those of the model with the lowest BIC. Set the current BIC to that value.

5. Repeat steps 3 and 4 until the current BIC value is less than all of the BIC values calculated in step 3.

6. Return the JIVE decomposition of the chosen model, as well as the chosen ranks.

2.5 Rank Selection Simulation

We randomly generated 100 simulated datasets to compare the rank selection procedures under varying conditions. For each simulation, the dimensions of each data source (n: number of shared columns, d_1: number of variables in source 1, d_2: number of variables in source 2) were simulated from a discrete uniform$(10, 100)$ distribution using the `sample` function in R. The joint rank and individual ranks for each source were similarly generated from a discrete uniform$(0, 4)$ distribution. The error variance (σ^2) was simulated from a continuous uniform$(0, 2)$ distribution.

Low rank structure was generated by simulating score and loading matrices, using the dimensions given above. Each element of the score and loading matrices was generated from a standard normal distribution. Each element of the error matrices (E_1 and E_2) was drawn from a normal(0,σ^2) distribution. The final simulated data sets X_1 and X_2 were generated as follows:

$$X_1 = U_1S + W_1S_1 + E_1$$
$$X_2 = U_2S + W_2S_2 + E_2.$$

The simulations were analyzed using the permutation and BIC rank selection methods, both with and without enforcing orthogonality among the individual structure. We compared the accuracy between methods using the following metric for relative error (where J$_{est}$ and A$_{est}$ are the joint and individual structure estimates, respectively, and J and A are the true structure matrices):

$$\text{Error}_{\text{JIVE}} = \frac{||J_{est} - J||^2 + ||A_{est} - A||^2}{||J||^2 + ||A||^2}.$$

We also consider the total squared error of the selected ranks:

$$\text{Error}_{\text{Rank}} = (r_{est} - r)^2 + (r_1 - r_{1,est})^2 + (r_2 - r_{2,est})^2.$$

The results are shown in Table 1.

JIVE error	BIC	Perm	BIC⊥	Perm⊥
Mean	**0.631**	**0.463**	**0.640**	**0.365**
Minimum	0.011	0.009	0.011	0.011
1st quartile	0.116	0.190	0.186	0.157
Median	0.766	0.402	0.703	0.308
3rd quartile	1.000	0.580	1.000	0.497
Maximum	1.362	1.999	1.476	1.099

Rank error	BIC	Perm	BIC⊥	Perm⊥
Mean	**11.4**	**3.4**	**12.2**	**3.5**
Minimum	0.0	0.0	0.0	0.0
1st quartile	0.0	1.0	1.0	0.0
Median	8.5	2.0	9.0	2.0
3rd quartile	20.3	3.0	21.0	3.0
Maximum	48.0	26.0	48.0	29.00

Table 2.1: Relative squared error for the JIVE estimates, and squared error for the selected ranks. The mean and five number summary are shown over 100 simulations. For each simulation permutation testing and BIC are used to select the ranks without orthogonality between individual structure (BIC, Perm) and with orthogonality between individual structure (BIC⊥, Perm⊥).

The selected ranks under permutation testing are generally better than the selected ranks under BIC, and in both cases the accuracy of the selected ranks is not substantially better or worse after enforcing orthogonality. With orthogonality enforced, permutation testing gave correct ranks for all joint and individual structure matrices in 36% of simulations whereas BIC gave all correct ranks in 23% of simulations. Both methods were generally conservative in estimation of joint structure. BIC underestimated joint rank in 52% of simulations and overestimated in just 11% of simulations, whereas permutation testing underestimated in 37% of simulations and overestimated in 15% of simulations. For BIC individual ranks were underestimated in 49.5% of cases and overestimated in 11% of cases, whereas for permutation testing individual ranks were underestimated in 19.5% of cases and overestimated in 25.0% of cases.

Overall, permutation testing gave more accurate estimates of joint and individual structure than BIC, reflecting the more accurate rank estimation of permutation testing. Furthermore, estimates with orthogonality enforced between individual structure were generally more accurate than estimates without orthogonality enforced, and this improvement is especially evident under permutation testing. The enforcement of orthogonality ensures that structure that is truly joint will not be estimated as individual when the ranks are misspecified, and this is likely responsible for its improved performance.

Because of its low error in comparison to the other methods, the default rank selection method for the **r.jive** package is the permutation test, and orthogonality is enforced among individual structures by default.

2.6 Scalability

For applications that involve high-dimensional genomics or other large-scale data, computational efficiency is very important. Computing time for the JIVE algorithm can depend on several factors, including the data dimensions, sample size, ranks of joint and individual structure, and number of iterations required for convergence. The application presented in Section 2.7 requires approximately 15 minutes to estimate the ranks via permutation, and 2 minutes to run the JIVE algorithm with given ranks.

The algorithms implemented in the `r.jive` package include some features that help to reduce computation time substantially. Prior to estimating the JIVE decomposition, if the row dimension is larger than the shared column dimension for a given data source, the data are compressed via an initial SVD (see Step 4 of the pseudocode in Sections 2.4.1 and 2.4.3). This will convert a $d_i \times n$ dimensional matrix to an $n \times n$ dimensional matrix with no loss of information for determining the JIVE decomposition. Thus the computational speed of the JIVE algorithm using fixed ranks, or estimating the ranks via BIC, is relatively robust to the dimensionality of each data source. However, use of permutation testing to select the ranks requires permuting the rows and computing the singular values of the original data several times, and therefore the computational speed under permutation testing is more dependent on the data dimensions. Computation time for the JIVE algorithm is also considerably improved by avoiding computing the singular vectors except where they are needed, which is particularly helpful when using the permutation test for rank selection because only the singular values are needed, not the singular vectors.

Table 2.2 gives the results of a simple simulation to illustrate the effects of the data dimensions on computing speed. For this simulation, all of the data were simulated with rank 1 joint structure and rank (1,1) individual structure. Data were simulated at three different shared column dimensions (50, 200, and 1000) and four different row dimensions (100, 1000, 10000, and 100000). Scores, loadings, and errors were simulated from a standard normal distribution, as in Section 2.5. As expected, running the algorithm with given ranks was substantially faster than estimating the ranks. Because of the initial SVD dimension reduction step in the algorithm, the computation time of the BIC rank selection method was predominantly dependent on only the column dimension (i.e., the sample size). The computation time of the permutation rank selection method, however, increased with the row dimensions and the column dimension.

Dimension			Time (Minutes)		
n	d1	d2	Given	Perm	BIC
50	100	100	0.036	0.128	0.495
50	1000	1000	0.039	0.310	2.141
50	10000	10000	0.059	1.901	2.102
50	100000	100000	0.208	17.015	1.349
200	100	100	0.061	0.378	4.064
200	1000	1000	0.096	1.758	5.258
200	10000	10000	0.203	12.860	2.403
200	100000	100000	1.412	108.825	3.920
1000	100	100	0.165	1.305	11.379
1000	1000	1000	4.478	38.251	287.569
1000	10000	10000	6.313	210.315	208.038
1000	100000	100000	32.056	2177.514*	87.655

Table 2.2: Timing of the JIVE algorithm for various dimensions under different rank selection methods, on a laptop with a 2.5 GHz Intel Core i7 processor and 16 GB RAM. All of the computation times are average run times from 5 simulations, with the exception of the permutation test at the highest dimensions (*), which was only run once.

In cases where a direct application of the JIVE algorithm is computationally prohibitive, it may be useful to reduce the dimensionality of the data prior to running JIVE. One such approach is to perform dimension reduction via PCA beforehand, and estimate the JIVE decomposition based on the first few principal components of each data source. Another option is feature selection, in which features are filtered by some criteria (for example, removing genes with less variable expression) before running JIVE.

2.7 Vignette: BRCA Application

The following vignette illustrates `r.jive` using publicly available multi-source genomic data for 348 breast cancer (BRCA) tumor samples from The Cancer Genome Atlas data freeze [Cancer Genome Atlas Network, 2012]. We applied the JIVE algorithm to the mRNA expression, DNA methylation, and miRNA data, processed as previously described [Lock and Dunson, 2013]. There were 654 genes in the mRNA expression data, 574 loci in the DNA methylation data, and 423 loci in the miRNA expression data. All three data sources are shared by the 348 tumor samples.

2.7.1 Loading the data

First, if you have not done so already, install and load the package via CRAN:

```
install.packages("r.jive")
library(r.jive)
```

The package requires two dependencies: **SpatioTemporal** for inputing missing values, and **gplots** for some plotting functions.

To load the BRCA data, enter
`data(BRCA_data)`
These data consist of a single list object `Data` that contains the data. The list has one entry for each of three different molecular sources:

- `Data[[1]]` (`Data$Expression`): gene expression matrix for 654 genes (rows) and 348 samples (columns)
- `Data[[2]]` (`Data$Methylation`): DNA methylation matrix for 574 cg sites (rows) and 348 samples (columns)
- `Data[[3]]` (`Data$miRNA`): miRNA expression matrix for 423 cg sites (rows) and 348 samples (columns).

The 348 columns are shared by the data sources (here, they correspond to tumor samples), and must be in the same order. This is the general data format recognized by `jive`.

These data were originally obtained from the data freeze for TCGA's flagship BRCA publication [Cancer Genome Atlas Network, 2012]. The data were filtered and processed as described in [Lock and Dunson, 2013]. Lock and Dunson [2013] describe an analysis of these data to identify jointly present sample clusters, and these clusters are loaded here as the vector `clusts`. Here we will only use the cluster labels to visualize and interpret results.

We estimate the decomposition on the BRCA data with these and other defaults. (This can take some time to complete, about 15 min, to compute the same estimates more quickly with given ranks see below.)

```
Results = jive(Data)
```

```
> Estimating joint and individual ranks via permutation...
Running JIVE algorithm for ranks:
joint rank: 2 , individual ranks: 21 14 20
JIVE algorithm converged after 113 iterations.
Re-estimating joint and individual ranks via permutation...
Running JIVE algorithm for ranks:
joint rank: 2 , individual ranks: 20 12 19
JIVE algorithm converged after 131 iterations.
Re-estimating joint and individual ranks via permutation...
Running JIVE algorithm for ranks:
joint rank: 2 , individual ranks: 20 12 18
JIVE algorithm converged after 113 iterations.
Re-estimating joint and individual ranks via permutation...
Running JIVE algorithm for ranks:
joint rank: 2 , individual ranks: 20 12 18
JIVE algorithm converged after 113 iterations.
```

The output shown are the ranks used for each call to the base JIVE algorithm (here, 4 calls) and the number of iterations until convergence for each call. The method ends when the ranks are the same for two consecutive calls. For this example, the chosen ranks are

- rank 2 joint structure
- rank 32 structure individual to gene expression
- rank 22 structure individual to methylation, and
- rank 23 structure individual to miRNA.

The results are given as an object of the S3 class JIVE, with values including

- `Results$data`: A list of the three data matrices used for JIVE estimation. These are centered and scaled versions of the original data matrices (by default, each data matrix is scaled by its total variation).
- `Results$joint`: A list of three data matrices giving estimated joint structure for each source.
- `Results$indiv`: A list of three data matrices giving estimated individual structure for each source.

Each list is labeled with the names given in the original input data (e.g., `Results$data$Expression` gives the scaled expression data). These are also used as labels for figures that depict the JIVE results.

To simply estimate the JIVE decomposition with the ranks determined above, enter
`Results = jive(Data,method="given",rankJ=2,rankA=c(32,22,23))`
these results are identical to those obtained after running the method with rank selection.

2.7.2 Visualization and interpretation

The **r.jive** package provides three functions that generate figures with very different views of the JIVE results, and we briefly illustrate each function here. All three take an object of class JIVE as input, with additional optional parameters.

The most simple function is `showVarExplained`, which displays a barchart of the amount of variation explained by joint and individual estimates in each data source. Here we save the resulting figure for the BRCA data as a .png file, shown in Figure 2.1.

```
png("VarExplained.png",height=300,width=450)
showVarExplained(Results)
dev.off()
```

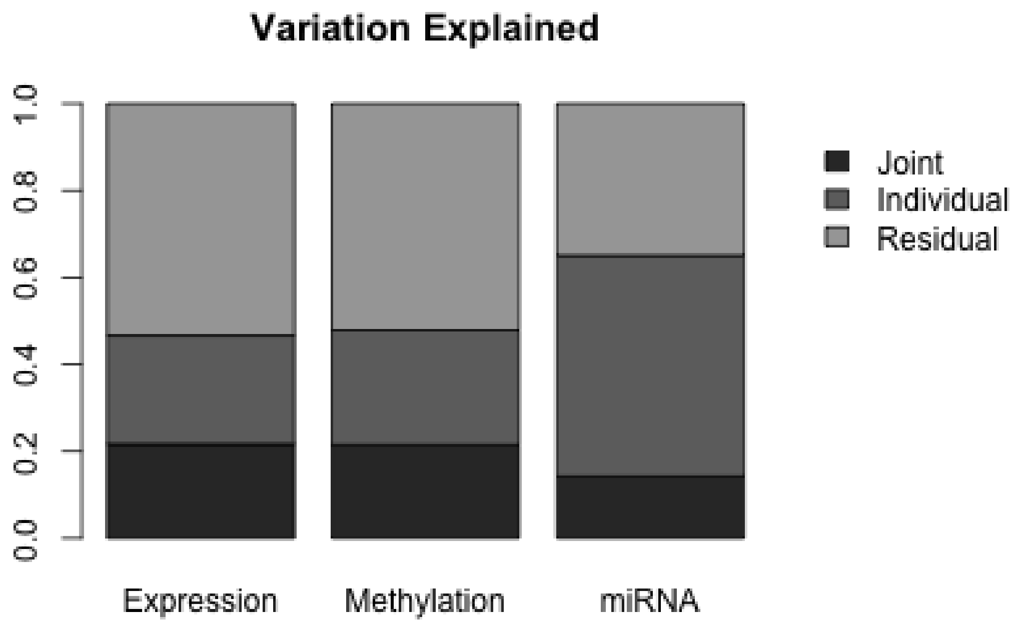

Figure 2.1: Decomposition of variation for each data source in the BRCA data.

The variation explained by joint structure is similar to that for individual structure for gene expression and methylation data, despite the much higher rank of individual structure (e.g., rank 32 individual vs. rank 2 joint for expression). The overall decomposition is remarkably similar between the expression and methylation data. The estimated joint structure explains slightly less variability in the miRNA data, which has more individual structure.

We can display the actual JIVE estimates, in the form of low-rank matrix approximations, as heatmaps using the `showHeatmaps` function. By default, this will create heatmaps of the full JIVE decomposition (Data = Joint + Individual + Residual) and order the rows and columns of all matrices by complete linkage clustering of the joint structure. For larger datasets, performing the clustering and rendering the images can take some time; for these data the process takes less than a minute. You may need to fiddle with the image dimensions a bit for the results to look nice (if the dimensions are too small, the text may be cut-off).

```
png("HeatmapsBRCA.png",height=465,width=705)
showHeatmaps(Results)
dev.off()
```

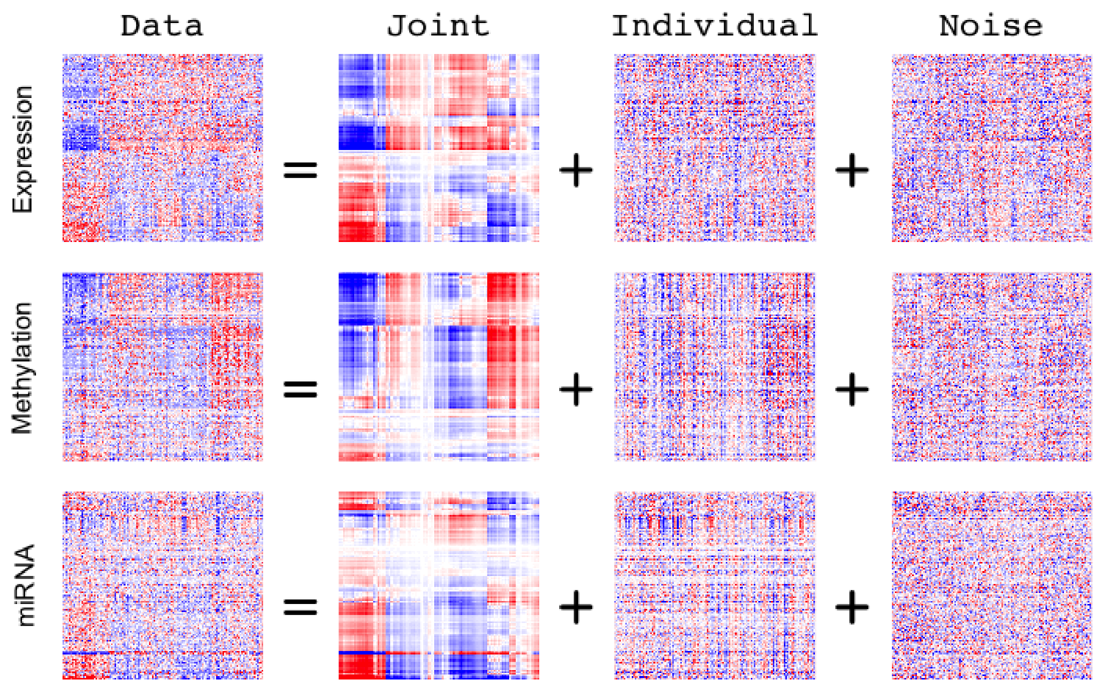

Figure 2.2: Heatmaps of JIVE estimates for the BRCA data, with row and column ordering determined by joint structure.

The heatmaps for the BRCA data are shown in Figure 2.2. The columns (samples) are vertically aligned for all heatmaps, with red correspoding to higher values and blue lower values. The estimated joint structure shows clear joint patterns that can also be seen in three of the original data heatmaps. However, note that the joint structure appears less prominently in the miRNA data heatmap, as the joint structure explains less of the miRNA variability.

In addition, the `showHeatmaps` function includes options to specify how to order rows and columns, and which matrices to display; for details enter `?showHeatmaps`. For example, we can order by the individual methylation structure (data source 2) and show only this heatmap (Figure 2.3).

```
png("HeatmapIndivMeth.png",height=400,width=400)
showHeatmaps(Results,order_by=2,show_all=FALSE)
dev.off()
```

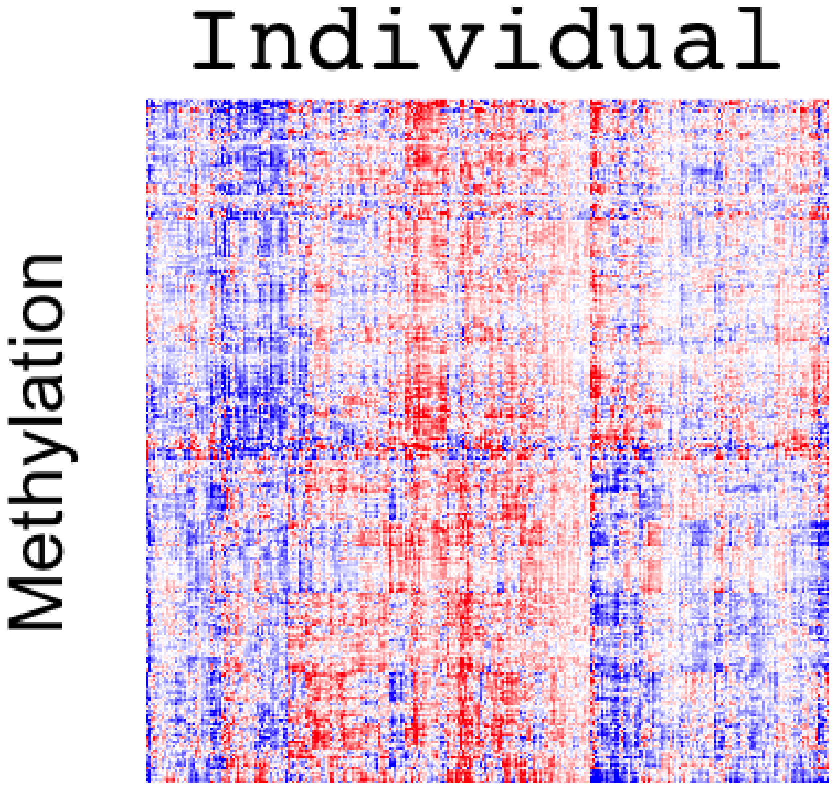

Figure 2.3: Heatmap of the individual structure of the methylation data source, with the rows and columns ordered based on clustering of the individual structure of this data source.

The dominating factor here appears to be a mean effect, distinguishing those samples with relatively high methylation genome-wide from those with relatively low methylation.

To further examine the biological relevance of the estimated joint structure, we consider the “point cloud” view provided by the `showPCA` function. This shows the patterns in the column space that maximize variability of joint or individual structure, analogous to principal components. Any number of components from the joint or different individual structure estimates can be shown in multi-panel scatterplots. First, we view the two components of joint structure (note: because the joint structure is rank 2, it has only 2 principal components). We color these components by the clusters defined in the `clusts` variable.

```
Colors = rep('black',348)
Colors[clusts==2] = 'green'
Colors[clusts==3] = 'purple'
png("JointPCA.png",height=400,width=400)
showPCA(Results,n_joint=2,Colors=Colors)
dev.off()
```

We see in Figure 2.4 that the estimated joint corresponds well to the three previously identified clusters. Specifically, one pattern distinguishes Basal-like tumor samples (cluster 1) from other samples; among the remaining samples a subgroup of Luminal A tumors with a low fraction of genomic alteration and improved clinical prognosis (cluster 2) is distinguished.

For a broader view, we show the first component of joint structure with the first component of each of the three individual structures (Figure 2.5).

```
png("MorePCA.png",height=600,width=600)
showPCA(Results,n_joint=1,n_indiv=c(1,1,1),Colors=Colors)
dev.off()
```

A clustering effect is not apparent in the individual components shown, besides a very slight distinction between clusters 2 and 3 in the expression individual component. This suggests that the coordinated expression, methylation, and miRNA activity in

BRCA tumors is primarily driven by the cluster effects mentioned above, whereas the activity specific to each data source is driven by other biological components.

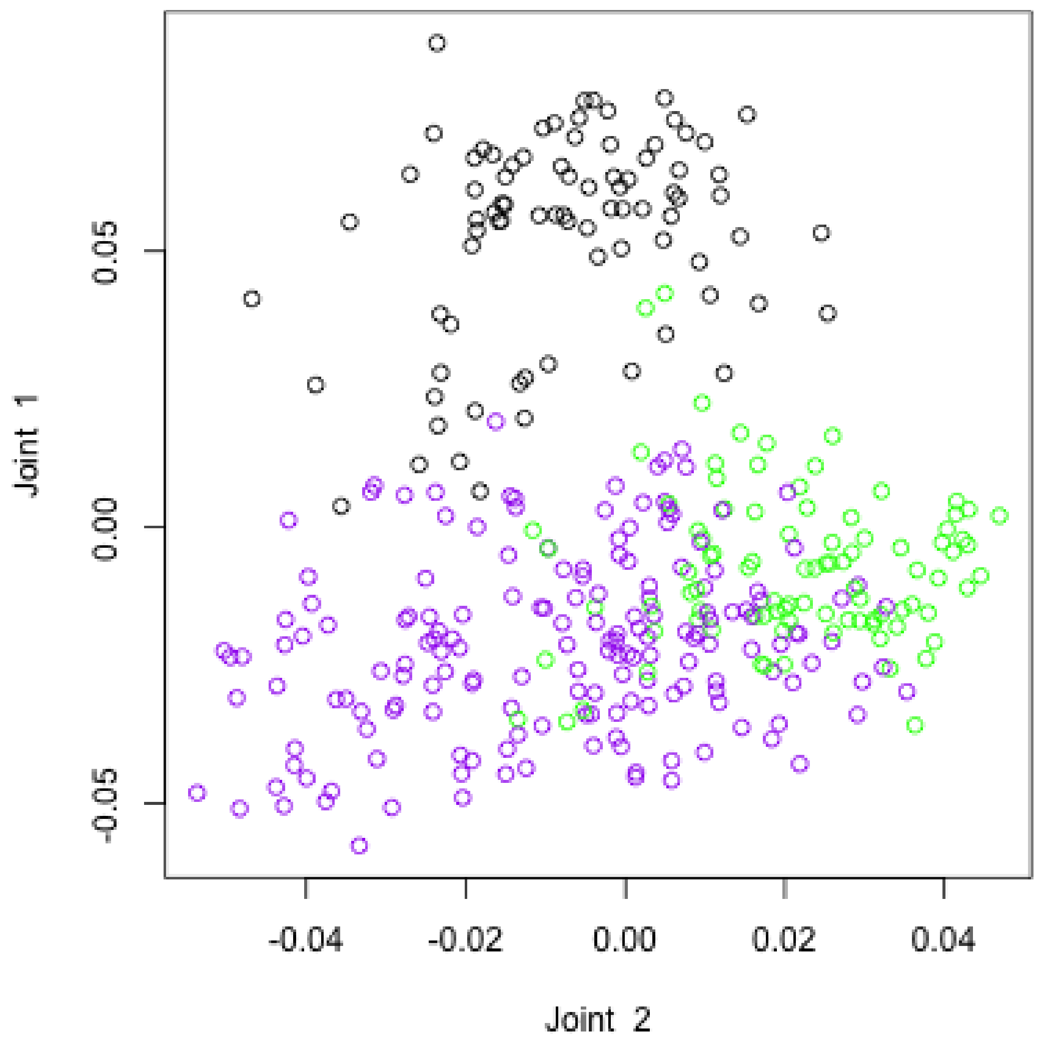

Figure 2.4: The two principal components of joint structure of the BRCA data, colored by previously determined integrative clusters; cluster 1 is at the top, cluster 2 is lower left, and cluster 3 is lower right.

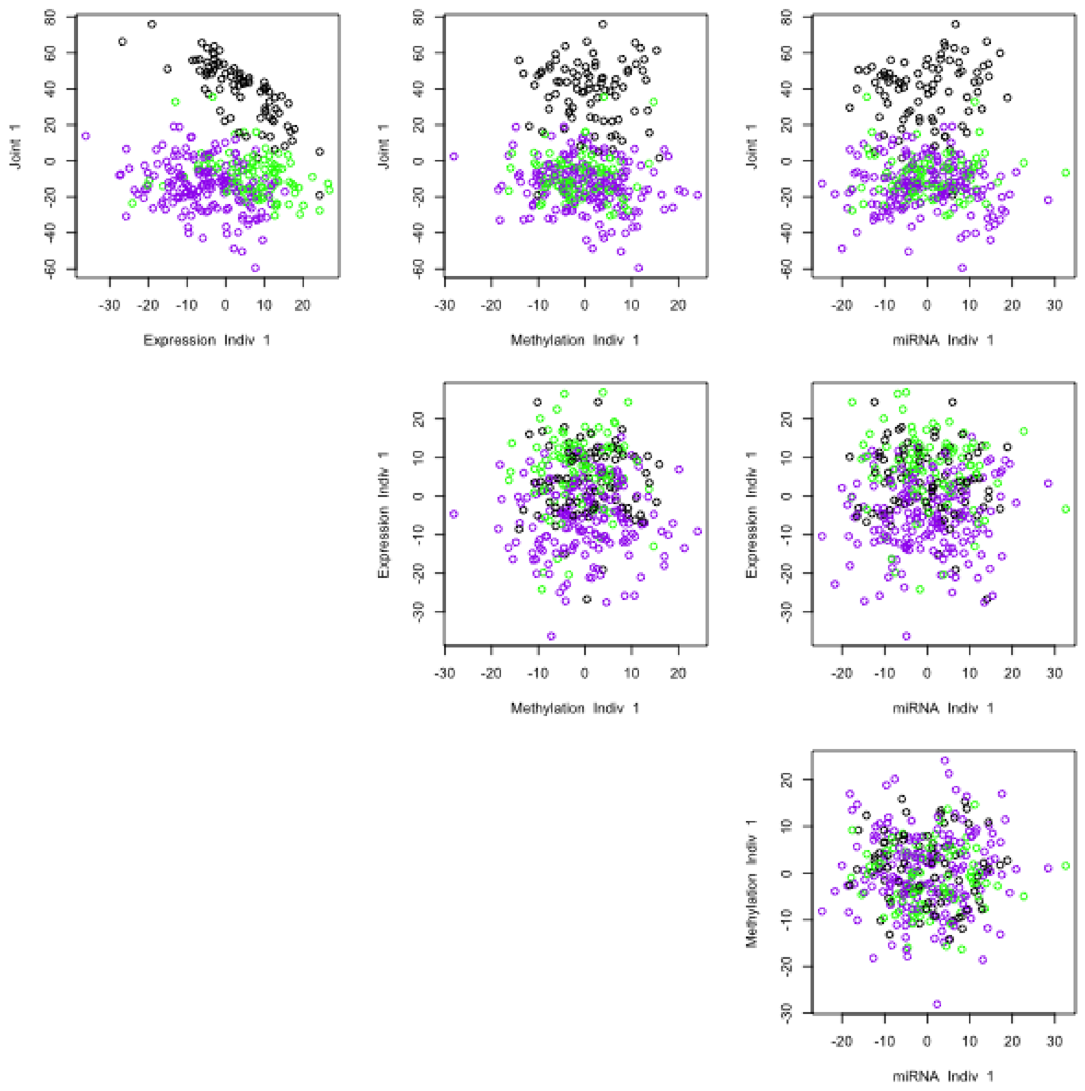

Figure 2.5: The first joint and the first individual components of the JIVE decomposition of the BRCA data plotted against each other, colored again by tumor type.

Chapter 3

Linked Matrix Factorization

3.1 Introduction

Recent technological advances in biomedical research have led to a growing number of platforms for collecting large amounts of health data. Molecular profiling modalities such as genetic sequencing and gene expression microarrays, and imaging modalities such as MRI scans, yield high-dimensional data with complex structure. Methods that simplify such data by identifying latent patterns that explain most of the variability are very useful for exploratory visualization of systematic variation, dimension reduction, missing data imputation, and other tasks. For a single data matrix $X : m \times n$, this simplification can be accomplished via a principal components analysis (PCA) or via other approaches to low-rank matrix factorization [Wall et al., 2003]. For example, X may represent a microarray with expression measurements for m genes for n biological samples. It is also increasingly common to have multiple linked high-dimensional data matrices for a single study, e.g.,

$$X_1 : m_1 \times n, X_2 : m_2 \times n, \dots, X_k : m_k \times n \tag{3.1}$$

with n shared columns. In the *multi-source* context (3.1) X_1 may represent expression for m_1 genes, X_2 may represent abundance of m_2 proteins, and X_3 may represent abundance of m_3 metabolites, for a common set of n samples. For such data a straightforward ad-hoc approach is to perform a separate PCA of each matrix X_i. However, patterns of systematic variability may be shared between blocks; for example, it is reasonable to expect that some sample patterns that are present in gene expression data are also present in proteins. Thus, separate factorizations can be inefficient and underpowered to accurately recover these joint signals, and they also provide no insight into the connections between data matrices that are often of scientific interest. An alternative approach is to perform a single joint PCA analysis of the concatenated data $X : (m_1 + m_2 + \cdots + m_k) \times n$, and this approach has been referred to as consensus PCA [Westerhuis et al., 1998]. However, a consensus PCA approach assumes that all systematic patterns are shared across data matrices, and lacks power to accurately recover signals that may exist in only one data matrix.

The recent ubiquity of high-dimensional multi-source data has motivated more flexible methods for scenario (3.1). A guiding principle for several such methods is to simultaneously model features that are shared across multiple sources (i.e., *joint*) and features that are specific to each source (i.e., *individual*). Methods that follow this strategy have been developed that extend well-established exploratory techniques such as partial least squares [Löfstedt and Trygg, 2011a], non-parametric Bayesian modeling [Ray et al., 2014], non-negative factorization [Yang and Michailidis, 2016], and simultaneous components analysis [Schouteden et al., 2014]. The Joint and Individual Variation Explained (JIVE) method [Lock et al., 2013] is a direct extension of PCA, distinguishing components that explain covariation (joint structure) among sources and components that explain variation that is individual to each data source. This distinction simplifies interpretation, and also improves accuracy to recover underlying signals because structured individual variation can interfere with finding important joint signal, just as joint structure can obscure important signal that is individual to a data source.

Multi-source data integration (3.1) has been termed *vertical integration* [Tseng et al., 2015]. Related dimension reduction and pattern recognition methods have also been developed specifically for the *horizontal integration* of a single data source measured for multiple sample groups [Kim et al., 2017, Huo et al., 2016]:

$$X_1 : m \times n_1, X_2 : m \times n_2, \ldots, X_k : m \times n_k. \tag{3.2}$$

Other approaches have been developed for a collection of matrices that share both dimensions, such as the population value decomposition (PVD) [Crainiceanu et al., 2011]:

$$X_1 : m \times n, X_2 : m \times n, \ldots, X_k : m \times n. \tag{3.3}$$

PVD was designed for the analysis of aligned image populations and produces a joint low-rank factorization for scenario (3.3) with shared row and column components; similar techniques have also been developed in the computer science literature [Ye, 2005]. Another approach for scenario (3.3) is to treat the data as a single multi-way array (i.e., tensor) $\mathbb{X} : m \times n \times k$ and apply well-established tensor factorizations such as the CANDECOMP/PARAFAC [Harshman, 1970] and Tucker [Tucker, 1966] factorization that extend PCA and related matrix dimension reduction methods to higher-order arrays. Neither PVD nor a tensor factorization approach distinguishes joint and individual structure among the constituent data matrices. The Linked Tucker2 Decomposition [Yokota and Cichocki, 2014] and Bayesian Multi-view tensor factorization [Khan and Kaski, 2014] methods do allow for the decomposition of joint and individual structure, under an extended scenario for (3.3) where the collection of $m \times n$ matrices can be grouped into sets of size d_i:

$$\mathbb{X}_1 : m \times n \times d_1, \mathbb{X}_2 : m \times n \times d_2, \ldots, \mathbb{X}_k : m \times n \times d_k. \tag{3.4}$$

Acar et al. [2011] describe a method for the joint factorization of a matrix and a tensor, but their approach does not allow for the decomposition of joint and individual structure.

In this chapter we address the simultaneous low-rank factorization and decomposition of joint and individual structure for the novel context of bidimensionally integrated data, linked data in which there is both vertical and horizontal integration. Our motivating example is a large-scale cytotoxicity study [Abdo et al., 2015] that consists of three interlinked high-content data matrices:

1. $X : m_1 \times n_1$: A cytotoxicity matrix with a measure of cell death for n_1 chemicals across a panel of m_1 genetically distinct cell lines,

2. $Y : m_2 \times n_1$: A chemical attribute matrix with m_2 molecular attributes measured for the n_1 chemicals, and

3. $Z : m_1 \times n_2$: A genomic matrix with n_2 single nucleotide polymorphisms (SNP) measured for each of the m_1 cell lines.

Note that X shares its row set with Z and its column set with Y, as illustrated in Figure 3.1. These data were made public as part of a DREAM challenge for open science [Eduati et al., 2015]. We are particularly interested in investigating the interaction between chemical toxicity, genomics, and measurable chemical attributes, i.e., what systematic variability in X is shared by Y and Z?

We introduce a method called Linked Matrix Factorization (LMF), that gives a unified and parsimonious low-rank factorization of these three data matrices. We also extend the framework of the JIVE method to allow for the decomposition of joint and individual structure in this context with LMF-JIVE. This extension requires new approaches to estimation and new theoretical results concerning the uniqueness, identifiability, and minimal parametrization of the decomposition. We illustrate how the results can facilitate the visual exploration of joint and individual systematic variation. We also describe how the factorization can be used in conjunction with an Expectation-Maximization (EM) algorithm [Dempster et al., 1977] for the principled imputation of missing values and complete analysis of multi-source data, even when entire rows or columns are missing from the constituent data matrices.

In what follows we first describe a novel joint low-rank factorization of these data in Section 3.2, before describing the extension to joint and individual structure in Section 3.3. In Section 3.5 we discuss missing data imputation, and in Section 3.6 we discuss different approaches to select the joint and individual ranks (i.e., number of components) in the factorization. Sections 3.2, 3.3, 3.5, and 3.6 each include simulation studies, to assess each facet of the proposed methodology. In Section 3.7 we describe the results of the cytotoxicity application, and in Section 3.8 we give some concluding remarks.

3.2 Joint Factorization

3.2.1 Model

We will refer to the three matrices involved in the LMF as X, Y, and Z, where X shares its column space with Y and its row space with Z (Figure 3.1). Let the dimensions of

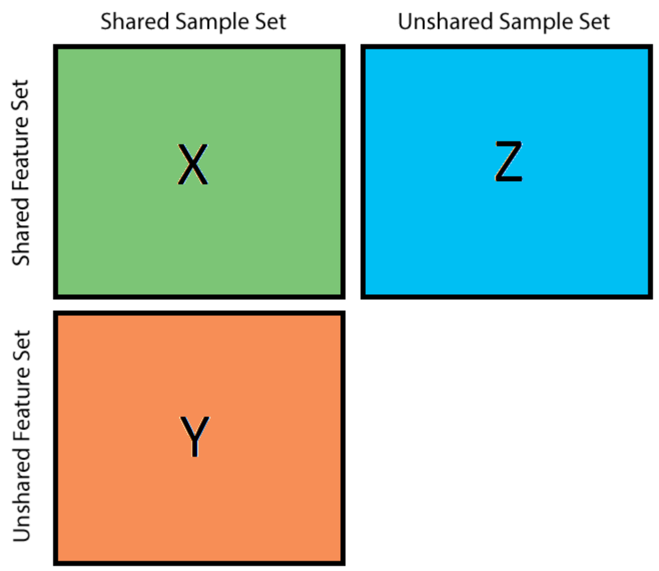

Figure 3.1: The structure of data for which the LMF algorithm was designed to analyze. The X and Y matrices share a sample set and have a common column space. Similarly, the X and Z matrices share a feature set and have a common row space.

X be $m_1 \times n_1$, the dimensions of Y be $m_2 \times n_1$, and the dimensions of Z be $m_1 \times n_2$. Our task is to leverage shared structure across X, Y, and Z in a simultaneous low-rank factorization. We define a joint rank r approximation for the three data matrices as follows:

$$X = US_xV^T + E_x$$
$$Y = U_yS_yV^T + E_y$$
$$Z = US_zV_z^T + E_z$$

- U is an $m_1 \times r$ matrix representing the row structure shared between X and Z
- V is an $n_1 \times r$ matrix representing the column structure shared between X and Y
- U_y is an $m_2 \times r$ matrix representing how the shared column structure is weighted over the rows of Y
- V_z is an $n_2 \times r$ matrix representing how the shared row structure is weighted over

the columns of Z

- S_x, S_y, and S_z are $r \times r$ scaling matrices for X, Y, and Z, respectively, and
- E_x, E_y and E_z are error matrices in which the entries are independent and have mean 0.

For the remainder of this chapter we will subsume the scaling matrices S_y and S_z into the U_y and V_z matrices, respectively, for a more efficient parameterization: $Y \approx U_y V^T$ and $Z \approx UV_Z^T$. A graphical representation of this model is given in Figure 3.2. For identifiability of the components it suffices to assume that the columns of U and V are orthonormal and S_x is diagonal (see Section 3.2.3).

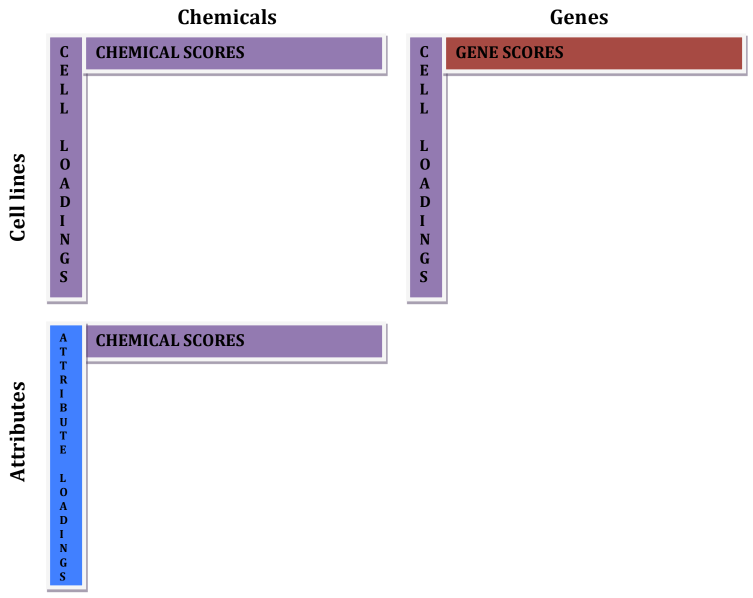

Figure 3.2: The factorized from of the LMF decomposition. X and Z share a common set of cell loadings U, and X and Y share a common set of chemical scores V.

3.2.2 Estimation

For notational convenience we denote the following concatenated matrices that span shared dimensions : $\tilde{U} = [US_x \ \ U_y], \tilde{V} = [VS_x \ \ V_z], \tilde{Y} = [X^T \ \ Y^T]^T, \tilde{Z} = [X \ \ Z]$. To estimate the joint structure, we iteratively minimize the sum of squared residuals

$$\text{SSE} = ||E_x||_F^2 + ||E_y||_F^2 + ||E_z||_F^2,$$

using an alternating least squares algorithm, where $|| \cdot ||_F$ defines the Frobenius norm. Given initial values, the algorithm proceeds by iteratively updating the components U, V, U_y, V_y, and S_x. In practice we first center and scale the three data sets, which prevents any of the matrices from having a disproportionately large influence on the joint components. Next, we initialize $\tilde{V}$ as the first r right singular vectors of the singular value decomposition (SVD) of $\tilde{Z}$. We initialize S_x as the identity matrix, so that V is the first n_1 columns of $\tilde{V}$. We then repeatedly cycle through the following local least-squares minimization steps with other components held fixed:

1. Update U_y via ordinary least squares: $U_y = (V^TV)^{-1}V^TY$
2. Update U via ordinary least squares: $U = (\tilde{V}^T\tilde{V})^{-1}\tilde{V}^T\tilde{Z}$
3. Scale U by dividing each column by its Frobenius norm
4. Update $\tilde{U}$: $\tilde{U} = [US_x \ \ U_y]$
5. Update V via ordinary least squares: $V = (\tilde{U}^T\tilde{U})^{-1}\tilde{U}^T\tilde{Y}$
6. Update V_z via ordinary least squares: $V_z = (U^TU)^{-1}U^TZ$
7. Scale V by dividing each column by its Frobenius norm
8. Update S_x via least squares; define $W : np \times r$ such that the i'th column of W is the vectorization of the product of the ith columns of U and V, $W[, i] = \text{vec}(U[, i]V[, i]^T)$, then the diagonal entries of S_x are $(W^TW)^{-1}W^T\text{vec}(X)$
9. Update $\tilde{U}$ and $\tilde{V}$ to incorporate the new S_x.

The algorithm will improve the SSE at each step until convergence, resulting in the following rank r estimates for the joint structure:

$$\begin{aligned} J_x &= US_xV^T \\ J_y &= U_yV^T \\ J_z &= UV_z^T. \end{aligned} \tag{3.5}$$

3.2.3 Diagonalizing $\mathbf{S}_x$

Importantly, by constraining S_x to be a diagonal matrix we do not limit our solution space. This result is given in Proposition 1 below, and the proof is given in Web Appendix F. This simplifies the parameterization, facilities the identifiability of the components, and improves computation time by only estimating the diagonal elements of S_x. This property does not extend to other scenarios where constituent data matrices share both rows and columns (3.3), such as the PVD factorization [Crainiceanu et al., 2011].

Proposition 1. *Assume we have a decomposition in the form of Equation (3.5). Then the components of the decomposition can be rotated such that S_x is a diagonal matrix.*

Proof. Let S_x^* be a matrix of rank r. Suppose a set of matrices has the decomposition $U^*S_x^*V^{*T}$, $U_y^*V^{*T}$, and $U^*V_z^{*T}$. Then we can take a rank r SVD of S_x^*. For this SVD, let P = the left singular vectors, Q = the right singular vectors, and S_x = the singular values. If we let $U = U^*P$ and $V = V^*Q$, then, we can rewrite this matrix decomposition as US_xV^T, where S_x is a diagonal matrix because it represents the singular values of S_x^*. Then we can set $U_y = U_y^*P$ and $V_z = V_z^*Q$. This gives us a decomposition in the form of Equation 3.5 in which S_x is diagonal. □

3.2.4 Simulation Study

We ran a simulation to assess the accuracy of the LMF algorithm to recover underlying joint structure. The results suggest that the LMF algorithm does a good job of recovering the underlying joint structure even in the presence of random noise. They also demonstrated that with increasing noise the reconstruction error, a measure representing the algorithm's ability to estimate the true underlying joint structure of the data, and the residual error, the estimated random noise, both increase linearly, but the reconstruction error remains small. A detailed description of this simulation and its results is given in Web Appendix B.

We ran a simulation to test the ability of the LMF algorithm to recover the true underlying joint structure of the data. To generate the data, we simulated from the model, generating random matrices for the joint components. There were 100 simulated data sets. All elements of the joint structure components, U, S_x, V, U_y, and V_z, were simulated from a Normal$(0, 1)$ distribution. Error matrices were simulated from a Normal$(0, 1)$ distribution. The simulated data sets had 50 rows and 50 columns, and the rank of the joint structure was 2. The LMF algorithm was run with a convergence threshold of .00001 and a maximum 5000 iterations.

We evaluated the performance of the LMF algorithm in these simulations by two criteria. First, we calculated the relative reconstruction error, which measures the algorithm's ability to retrieve the true joint structure of the data. We scale the reconstruction residuals by the total structure to get the relative reconstruction error:

$$E_{rec} = \frac{\|J_x - J_x^{true}\|_F^2 + \|J_y - J_y^{true}\|_F^2 + \|J_z - J_z^{true}\|_F^2}{\|J_x^{true}\|_F^2 + \|J_y^{true}\|_F^2 + \|J_z^{true}\|_F^2}$$

where

$$J_x^{true} = US_xV^T, J_y^{true} = U_yV^T, \text{ and } J_z^{true} = UV_z^T.$$

For the second criterion we consider the relative residual error, which measures the amount of variability in the data that is captured by the estimated joint structure:

$$E_{res} = \frac{\|J_x - X\|_F^2 + \|J_y - Y\|_F^2 + \|J_z - Z\|_F^2}{\|X\|_F^2 + \|Y\|_F^2 + \|Z\|_F^2}.$$

Over the 100 simulations, the mean reconstruction error was 0.122 with a standard deviation of 0.042. The mean residual error was 0.588 with a standard deviation of 0.074. This relatively small value for the mean reconstruction error in contrast to the mean residual error suggests that the LMF algorithm does a good job of recovering the underlying joint structure even in the presence of random noise.

We also ran a simulation with varying error variance to see how differences in error variance affect the accuracy of the decomposition. This simulation was similar to the above simulation, but instead of the error matrices being drawn from a normal(0,1) distribution, they were drawn from a normal(0, σ^2) distribution, where σ^2 took on a value of $\frac{i}{100}$ for the i^{th} simulated data set, varying between 0.01 and 1. The relationship between error variance (σ^2) and reconstruction error is shown in Figure 3.3a, and the relationship between error variance and residual error is shown in Figure 3.3b. These results demonstrate that in the presence of minimal noise the data are exactly captured with the converged low rank approximation; with increasing noise both the reconstruction error and residual error increase linearly, but the reconstruction error remains small.

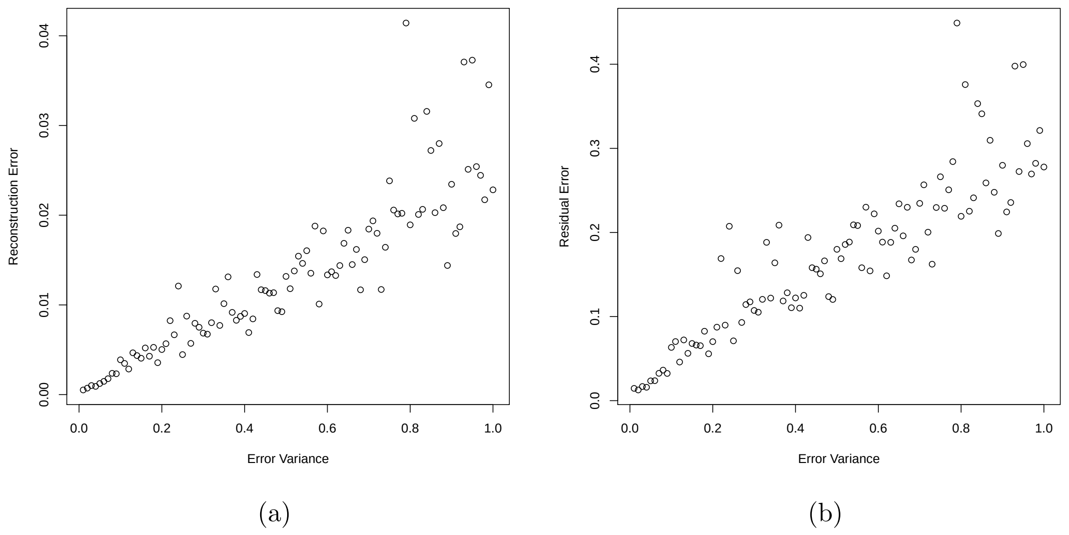

Figure 3.3: The relationship between the error variance of a data set and the reconstruction error (a) or residual error (b) in the LMF decomposition.

3.3 Joint and Individual Decomposition

3.3.1 Model

We extend the joint factorization approach of Section 3.2 to allow for structured low rank variation that is individual to each data matrix. For this model, we define the joint structures as in Section 3.2:

$$J_x = US_xV^T, \;\; J_y = U_yS_yV^T, \;\; J_z = US_zV_z^T. \tag{3.6}$$

The individual structure is defined as follows:

$$A_x = U_{ix}S_{ix}V_{ix}^T, \;\; A_y = U_{iy}S_{iy}V_{iy}^T, \;\; A_z = U_{iz}S_{iz}V_{iz}^T \tag{3.7}$$

where the individual components are given by

- U_{ij} for $j = \{x, y, z\}$, matrices representing the row structure unique to each matrix
- S_{ij} for $j = \{x, y, z\}$, matrices representing the scaling for the individual structure of each matrix, and
- V_{ij} for $j = \{x, y, z)\}$, matrices representing the column structure unique to each matrix.

The full model with joint and individual structure is :

$$\begin{aligned} X &= J_x + A_x + E_x \\ Y &= J_x + A_x + E_y \\ Z &= J_x + A_x + E_z. \end{aligned} \tag{3.8}$$

We define this joint and individual factorization as *LMF-JIVE*. In practice, we do not estimate the scaling matrices S_{ix}, S_{iy}, and S_{iz}, and instead allow them to be subsumed into the row and column structure matrices.

3.3.2 Estimation

To estimate the joint and individual structure, we extend the alternating least squares algorithm from Section 3.2.2. Define the low rank approximations for individual structure

$$\begin{aligned} A_x &= U_{ix} S_{ix} V_{ix}^T \\ A_y &= U_{iy} S_{iy} V_{iy}^T \\ A_z &= U_{iz} S_{iz} V_{iz}^T. \end{aligned} \tag{3.9}$$

Let r be the rank of joint structure as defined in Section 3.2.2, and let r_x, r_y, and r_z be the ranks of the individual structure A_x, A_y, and A_z, respectively. The estimation algorithm proceeds by iteratively updating the components of $\{J_x, J_y, J_z\}$ and $\{A_x, A_y, A_z\}$ to minimize the total sum of squared residuals until convergence. Thus, to estimate joint structure we define the partial residuals $X^J = X - A_x$, $Y^J = Y - A_y$, and $Z^J = Z - A_z$. We similarly define $X^I = X - J_x$, $Y^I = Y - J_y$, and $Z^I = Z - J_z$. In practice we center and scale the three data sets, as for the joint LMF model. We also initialize $\tilde{V}$, S_x, and U_y as in the joint LMF model (see Section 3.2.2). For LMF-JIVE, we must also initialize A_x, A_y, and A_z to matrices of zeros. We then repeat the following steps until convergence (when the total sum of squares for the joint and individual estimates between the current iteration and the previous iteration is less than a chosen threshold) or until we reach the maximum number of iterations:

1. Set $X^J = X - A_x$, $Y^J = Y - A_y$, and $Z^J = Z - A_z$.
2. Define concatenations $\tilde{Y} = [(X^J)^T \;\; (Y^J)^T]^T$ and $\tilde{Z} = [X^J \;\; Z^J]$.
3. Update U via ordinary least squares: $U = (\tilde{V}^T \tilde{V})^{-1} \tilde{V}^T \tilde{Z}$
4. Scale U by dividing each column by its Frobenius norm.
5. Update $\tilde{U} = [U S_x \;\; U_y]$
6. Update V via ordinary least squares $V = (\tilde{U}^T \tilde{U})^{-1} \tilde{U}^T \tilde{Y}$
7. Update V_z via ordinary least squares: $V_z = (U^T U)^{-1} U^T Z^J$
8. Scale V by dividing each column by its Frobenius norm.

9. Update U_y via ordinary least squares: $U_y = (V^T V)^{-1} V^T Y^J$

10. Update S_x via least squares; define $W : np \times r$ such that the i'th column of W is the vectorization of the product of the ith columns of U and V, $W[,i] = \text{vec}(U[,i]V[,i]^T)$, then the diagonal entries of S_x are $(W^T W)^{-1} W^T \text{vec}(X)$

11. Recalculate $\tilde{U}$ and $\tilde{V}$ with the newly updated S_x.

12. Set $X^I = X - A_x$, $Y^I = Y - A_y$, and $Z^I = Z - A_z$.

13. Update A_x via a rank r_x SVD of X^I, wherein U_{ix} gives the right singular vectors, V_{ix} gives the left singular vectors, and S_{ix} gives the singular values.

14. Update A_y via a rank r_y SVD of Y^I, and update A_z via a rank r_z SVD of Z^I

After convergence, we suggest applying the following transformation to assure that the joint and individual structures are orthogonal for Y and Z:

$$\begin{aligned} J_y^{\perp} &= J_y + A_y P_J^R, \ A_y^{\perp} = A_y - A_y P_J^R \\ J_z^{\perp} &= J_z + P_J^C A_z, \ A_z^{\perp} = A_z - P_J^C A_z. \end{aligned} \tag{3.10}$$

Here, $P_J^C = UU^T$ is the projection onto the column space of J_x, and $P_J^R = VV^T$ is the projection onto the row space of J_x. This transformation makes the joint and individual structures identifiable; this and other properties are investigated in Sections 3.4 and 3.4.3. The main scientific rationale for the orthogonalizing transformation is that any structure in the individual matrices that is in the column or row space of J_x should reasonably be considered joint structure.

In the preceding algorithm we begin with the estimation of joint structure, with A_x, A_y and A_z initialized to matrices of zeros. We could alternatively estimate the individual structure first, by initializing J_x, J_y, and J_z to matrices of zeros. We also consider this alternative approach in Section 3.4.3.

3.4 Theoretical Results

3.4.1 Uniqueness

Below we show that the LMF-JIVE decomposition for the structure of matrix X is unique and identifiable. The proof rests on the defining assumption that shared row and column spaces are the same for joint structure, but different for individual structure.

Theorem 1. *Assume $\tilde{X} = J_x + A_x$, $\tilde{Y} = J_y + A_y$, and $\tilde{Z} = J_z + A_z$, the structured components of the LMF model, where*

$$row(J_x) = row(J_y)\ ,\ row(A_x) \cap row(A_y) = \{\mathbf{0}\}\ ,\ row(J_x) \cap row(A_x) = \{\mathbf{0}\}$$

and

$$col(J_x) = col(J_z)\ ,\ col(A_x) \cap col(A_z) = \{\mathbf{0}\}\ ,\ col(J_x) \cap col(A_x) = \{\mathbf{0}\}.$$

If also $\tilde{X} = J_x^* + A_x^*$, $\tilde{Y} = J_y^* + A_y^*$, and $\tilde{Z} = J_z^* + A_z^*$ where

$$row(J_x^*) = row(J_y^*)\ ,\ row(A_x^*) \cap row(A_y^*) = \{\mathbf{0}\}\ ,\ row(J_x^*) \cap row(A_x^*) = \{\mathbf{0}\}$$

and

$$col(J_x^*) = col(J_z^*)\ ,\ col(A_x^*) \cap col(A_z^*) = \{\mathbf{0}\}\ ,\ col(J_x^*) \cap col(A_x^*) = \{\mathbf{0}\},$$

then $J_x = J_x^*$ and $A_x = A_x^*$.

Proof. We first show that J_x and J_x^* have the same row and column spaces. By Theorem 1.1 in the supplement of Lock et al. [2013] there exists a unique orthogonal decomposition of $\tilde{X}$ and $\tilde{Y}$:

$$\begin{aligned} \tilde{X} &= J_x^{\perp} + A_x^{\perp} & \qquad J_x^{\perp} A_x^{\perp^T} &= 0_{m_1 \times m_1} \\ \tilde{Y} &= J_y^{\perp} + A_y^{\perp} & \qquad J_y^{\perp} A_y^{\perp^T} &= 0_{m_2 \times m_2} \end{aligned}$$

such that $row(J_x) = row(J_x^{\perp})$ and $row(J_x^*) = row(J_x^{\perp})$. Thus $row(J_x) = row(J_x^{\perp})$, and by a symmetric argument $col(J_x) = col(J_x^*)$.

We next show that A_x and A_x^* have the same row and column spaces, by showing that they have the same null spaces. Define the nullspace of A_x, $N(A_x) = \{\mathbf{v} \in \mathbb{R}^{n_1} :$

$A_x v = \mathbf{0}\}$, and take $\mathbf{v} \in N(A_x)$. If $\mathbf{v} \in N(J_x)$, then

$$Xv = J_x v + A_x v = \mathbf{0} + \mathbf{0} = \mathbf{0},$$

and $\mathbf{v} \in N(J_x^*)$ because $row(J_x) = row(J_x^*)$. So, $\mathbf{v} \in N(A_x^*)$ because

$$\mathbf{0} = J_x^* v + A_x^* v = A_x^* v.$$

If $\mathbf{v} \notin N(J_x)$, then

$$Xv = J_x v \in col(J_x).$$

So, because $col(J_x) = col(J_x^*)$,

$$J_x^* v + A_x^* v \in col(J_x^*).$$

Thus $A_x^* v = \mathbf{0}$, because $col(J_x) \cap col(A_x) = \{\mathbf{0}\}$, and we again conclude $v \in N(A_x^*)$. It follows that $N(A_x) \subseteq N(A_x^*)$; symmetric arguments show $N(A_x^*) \subseteq N(A_x)$, $N(A_x^T) \subseteq N(A_x^{*^T})$, and $N(A_x^{*^T}) \subseteq N(A_x^T)$. Thus $N(A_x) = N(A_x^*)$ and $N(A_x^T) = N(A_x^{*^T})$, so $row(A_x) = row(A_x^*)$ and $col(A_x) = col(A_x^*)$.

Define $J_{\text{diff}} = J_x - J_x^*$ and $A_{\text{diff}} = A_x - A_x^*$, and consider that

$$J_{\text{diff}} + A_{\text{diff}} = \tilde{X} - \tilde{X} = 0_{m_1 \times n_1}.$$

Note that $row(J_{\text{diff}}) \subseteq row(J_x)$ because $row(J_x) = row(J_x^*)$, and $row(A_{\text{diff}}) \subseteq row(A_x)$ because $row(A_x) = row(A_x^*)$, so $row(J_{\text{diff}}) \cap row(A_{\text{diff}}) = \{\mathbf{0}\}$. For any $\mathbf{v} \in \mathbb{R}^{n_1}$,

$$J_{\text{diff}}\, v + A_{\text{diff}}\, v = \mathbf{0},$$

and therefore $J_{\text{diff}}\, v = \mathbf{0}$ and $A_{\text{diff}}\, v = \mathbf{0}$. It follows that $J_{\text{diff}} = A_{\text{diff}} = 0_{m_1 \times n_1}$, and thus $J_x = J_x^*$ and $A_x = A_x^*$. $\square$

By the inherent identifiability of the decomposition for X, it follows that the entire LMF-JIVE decomposition is identifiable under the orthogonal transformations of structured variability in Y and Z in Equation (3.10).

Corollary 1. *Assume* $\{J_x, A_x, J_y, A_y, J_z, A_z\}$ *and* $\{J_x^*, A_x^*, J_y^*, A_y^*, J_z^*, A_z^*\}$ *satisfy the conditions of Theorem 1. Assume, in addition, that the joint and individual structures for* Y *and* Z *are orthogonal for both decompositions:*

$$J_y A_y^T = 0_{m_2 \times m_2},\ J_z^T A_z = 0_{n_2 \times n_2},\ \ J_y^* {A_y^*}^T = 0_{m_2 \times m_2},\ {J_z^*}^T A_z^* = 0_{n_2 \times n_2}.$$

Then $\{J_x, A_x, J_y, A_y, J_z, A_z\} = \{J_x^*, A_x^*, J_y^*, A_y^*, J_z^*, A_z^*\}$.

Proof. By Theorem 1, $J_x = J_x^*$ and $A_x = A_x^*$. Define P_J^R as the projection onto the row space of X. Then,

$$\begin{aligned} J_y + A_y &= J_y^* + A_y^* \\ \rightarrow (J_y + A_y) P_J^R &= (J_y^* + A_y^*) P_J^R \\ \rightarrow J_y P_J^R + 0 &= J_y^* P_J^R + 0 \\ \rightarrow J_y &= J_y^*. \end{aligned}$$

Thus, $A_y = A_y^*$, and an analogous argument shows that $J_z = J_z^*$ and $A_z = A_z^*$. $\square$

3.4.2 Orthogonality

In the classical JIVE algorithm for vertical integration, the joint and individual structures were restricted to be orthogonal for identifiability and a parsimonious decomposition. For LMF-JIVE we restrict the joint and individual structures for Y and Z to be orthogonal, but not for X, which shares a joint row space and column space. As shown in Section 3.4.1, orthogonality between joint and individual structure in X, in either the row space or the column space, is not needed for identifiability of the decomposition. Thus, variation in Y and Z is parsimoniously decomposed into joint, individual and residual variability, e.g., $||Y||_F^2 = ||J_y||_F^2 + ||A_y||_F^2 + ||E_y||_F^2$; however, this is not the case for X in general: $||X||_F^2 < ||J_x||_F^2 + ||A_x||_F^2 + ||E_x||_F^2$. Interestingly, no such decomposition exists in general, and so no post-hoc transformation of the converged result for X that would yield an orthogonal and parsimonious decomposition. This result is given in Theorem 2.

Theorem 2. *Assume* $\tilde{X} = J_x + A_x$, $\tilde{Y} = J_y + A_y$, *and* $\tilde{Z} = J_z + A_z$ *where (ii) row*$(J_x) =$

$row(J_y)$, (iii) $col(J_x) = col(J_z)$, and (iv) $rank(J_x) = rank(J_y) = rank(J_z) = r$. Assume also J^*_x, J^*_y, J^*_z, A^*_x, A^*_y, and A^*_z also satisfy properties (i), (ii) and (iii) above and that in addition their joint and individual matrices for shared subspaces are all orthogonal: $row(J^*_x) \perp row(A^*_x)$, $col(J^*_x) \perp col(A^*_x)$ $row(J^*_y) \perp row(A^*_y)$ and $col(J^*_z) \perp col(A^*_z)$. Then, $J^*_x + A^*_x \neq J_x + A_x = \tilde{X}$.

Proof. We claim that there exist $\tilde{X} = J^*_x + A^*_x$, $\tilde{Y} = J^*_y + A^*_y$, and $\tilde{Z} = J^*_z + A^*_z$ that meet the following conditions:

$$row(J^*_x) = row(J^*_y)$$

$$col(J^*_x) = col(J^*_z)$$

$$rank(J^*_x) = rank(J^*_y) = rank(J^*_z) = r$$

We also claim that the joint and individual estimates J^* and A^* are orthogonal:

$$row(J^*_x) \perp row(A^*_x)$$

$$col(J^*_x) \perp col(A^*_x)$$

$$row(J^*_y) \perp row(A^*_y)$$

$$col(J^*_z) \perp col(A^*_z)$$

Using a result from Lock et al. [2013], the only estimate that satisfies $row(J^*_x) = row(J^*_y)$ and $row(J^*_x) \perp row(A^*_x)$ under these conditions is $J^*_x = J_x + A_x P^R_J$, where P^R_J is the orthogonal projection onto the row space of J_x. Similarly, the conditions $col(J^*_x) = col(J^*_z)$ and $col(J^*_x) \perp col(A^*_x)$ imply that the estimate is $J^*_x = J_x + P^C_J A_x$. However, this implies that $A_x P^R_J = P^C_J A_x$, which is not necessarily true in general. Therefore, such an estimate does not exist. □

3.4.3 Simulation Study

Here we present a simulation study to test the ability of the LMF-JIVE algorithm to recover the true structure of the data. Our simulation design is analogous to that in Section 3.2.4, but allows for simulated data sets to have both joint and individual structure. There were 300 simulated data sets, with three different variability settings representing equal joint and individual variability, higher joint variability, and higher individual variability. For the first 100 simulations, all elements of the joint structure components, U, S_x, V, U_y, and V_z, and the individual structure components, U_{ix}, V_{ix}, U_{iy}, V_{iy}, U_{iz}, and V_{iz}, were simulated from a distribution with mean 0 and variance 1: $\mathrm{N}(0,1)$. For the next 100 simulations, the joint components were simulated from a $\mathrm{N}(0,9)$ distribution, while the individual structure components were simulated from $\mathrm{N}(0,1)$. In the last 100 simulations, the joint components were simulated from a $\mathrm{N}(0,1)$ distribution, and individual components were simulated from $\mathrm{N}(0,9)$. Error matrices were simulated from a $\mathrm{N}(0,1)$ distribution. As with the previous simulation, the simulated data sets had 50 rows and 50 columns, and the rank of the joint structure was 2. The rank of the individual structure was also 2 for all three matrices. We chose rank 2 for all structures in the simulation for simplicity, but the algorithm is not limited to situations of equal rank and rank selection is explored in Section 3.6.

For these simulations, we applied three different versions of the LMF or LMF-JIVE algorithms. First, we used the joint LMF algorithm, with joint rank equal to the sum of all joint and individual ranks to capture all structured variation in the data. We also applied LMF-JIVE with true tranks, initializing by estimating joint structure first (JF) or individual structure first (IF). We evaluated these algorithms by reconstruction error and residual error, as follows:

$$E_{rec} = \frac{\sum_{i=x,y,z}(\|J_i - J_i^{true}\|_F^2 + \|A_i - A_i^{true}\|_F^2)}{\|J_x^{true}\|_F^2 + \|J_y^{true}\|_F^2 + \|J_z^{true}\|_F^2 + \|A_x^{true}\|_F^2 + \|A_y^{true}\|_F^2 + \|A_z^{true}\|_F^2}$$

$$E_{res} = \frac{\|J_x + A_x - X\|_F^2 + \|J_y + A_y - Y\|_F^2 + \|J_z + A_z - Z\|_F^2}{\|X\|_F^2 + \|Y\|_F^2 + \|Z\|_F^2}.$$

For the LMF algorithm, A_x, A_y, and A_z were set to matrices of zeroes, and we used a rank of 8 (the sum of the joint and individual ranks). For comparison with the LMF algorithm, we also included a measure of overall reconstruction error:

$$E_{ovr} = \frac{\sum_{i=x,y,z}(\|J_i - J_i^{true} + A_i - A_i^{true}\|_F^2)}{\|J_x^{true} + A_x^{true}\|_F^2 + \|J_y^{true} + A_y^{true}\|_F^2 + \|J_z^{true} + A_z^{true}\|_F^2}$$

Table 3.1: Reconstruction error and residual error for the LMF algorithm and 2 different variations of the LMF-JIVE algorithm under equal joint and individual variance, higher joint variance, and higher individual variance. Key: JF = LMF-JIVE with Joint structure estimated First; IF = LMF-JIVE with Individual structure estimated First.

$\mathbf{E_{rec}}$ **Mean (St. Dev.)**	LMF	JF	IF
Equal Variance	0.9932 (0.1488)	0.1846 (0.2003)	0.7079 (0.7783)
Higher Joint	0.0194 (0.0037)	0.0017 (0.0008)	1.997 (0.0085)
Higher Ind	1.968 (0.0143)	0.9870 (0.2284)	0.0505 (0.0259)
$\mathbf{E_{res}}$ **Mean (St. Dev.)**	LMF	JF	IF
Equal Variance	0.1405 (0.0189)	0.1646 (0.0205)	0.1787 (0.0201)
Higher Joint	0.0030 (0.0004)	0.0038 (0.0005)	0.0041 (0.0007)
Higher Ind	0.0049 (0.0005)	0.0096 (0.0018)	0.0057 (0.0007)
$\mathbf{E_{ovr}}$ **Mean (St. Dev.)**	LMF	JF	IF
Equal Variance	0.0608 (0.0076)	0.0323 (0.0063)	0.0503 (0.0179)
Higher Joint	0.0016 (0.0002)	0.0007 (0.0001)	0.0011 (0.0004)
Higher Ind	0.0016 (0.0002)	0.0049 (0.0016)	0.0010 (0.0003)

The results are given in Table 3.1. In general either of the LMF-JIVE settings performed better than the LMF-only settings in terms of E_{ovr} and E_{rec}, demonstrating the value of distinguishing joint and individual structure. However, the relative performance of joint-first or individual-first estimation for LMF-JIVE depended on the context, with individual-first estimation performing better in scenarios with higher individual signal and joint-first estimation performing better in scenarios with higher joint signal. This demonstrates that the algorithm does not always converge to a global least-squares solution. Thus, the results can be explained because the initial iteration will capture more variability with whichever component is estimated first. For this reason, we recommend using whichever order estimates the structure with the largest variance first or using both models and comparing them in terms of converged SSE. Between the two joint and individual models, the mean residual error is similar but the better performing approach in terms of signal recovery tends to have a lower residual error. The LMF model has lower residual error than the LMF-JIVE models because it has a joint rank that also subsumes the individual ranks of the LMF-JIVE models; it is therefore more prone to over-fitting, and the LMF model has higher reconstruction error than the LMF-JIVE models.

3.5 Imputation

In this section, we use the LMF and LMF-JIVE frameworks to introduce an imputation method for linked data with various forms of missingness, including missing rows and columns. Our approach extends similar methods that have been created for data imputation with a single matrix using an SVD. We extend an EM algorithm to iteratively impute the missing values using the SVD and compute the SVD given the imputed values [Kurucz et al., 2007]. Thus, our method proceeds by iteratively updating missing values with successive applications of LMF-JIVE. Here we focus on missing data in X because of our interest in the imputation of cytotoxicity data; however, it is straightforward to extend the approach to impute missing data in Y or Z as well.

3.5.1 Algorithm

The imputation algorithm begins by initiating all of the missing values in the data. For single missing values, the entries are set to the mean of the row and column means for that position. If a full row is missing, each entry is set to the column mean for each column. Similarly if a full column is missing, each entry is set to its respective row mean. Finally, for the entries where both the row and the column are entirely missing, the entries are set to the full matrix mean. For a matrix X:

$$\hat{X}_{ij} = \bar{X} \text{ if row i and column j are both missing}$$

$$\hat{X}_{ij} = \bar{X}_{i.} \text{ if row i is non-missing but column j is missing}$$

$$\hat{X}_{ij} = \bar{X}_{.j} \text{ if row i is missing but column j is non-missing}$$

$$\hat{X}_{ij} = (\bar{X}_{i.} + \bar{X}_{.j})/2 \text{ if row i and column j are both non-missing}$$

After initializing the matrix, we do the following:

1. Compute the LMF-JIVE decomposition using the imputed matrix $\hat{X}$. Let J_x be the joint component estimate for X and A_x be the individual component estimate for X.

2. Impute the missing values using the decomposition in step 1. For missing entries

in X, set $X_{ij} = \hat{X}_{ij}$.

3. Repeat steps 1 and 2 until the algorithm converges. We used the squared Frobenius norm of the difference between the current and previous estimates of X as our convergence criterion, with a threshold of 0.0001. For the tth iteration, we stop if $\|\hat{X}^{(t)} - \hat{X}^{(t-1)}\|_F^2 < 0.0001$.

This imputation strategy can be considered an EM algorithm, under a normal likelihood model. To formalize this, let $\mu_x = USV + U_{ix}S_{ix}V_{ix}$, $\mu_y = U_ySV + U_{iy}S_{iy}V_{iy}$, and $\mu_z = USV_z + U_{iz}S_{iz}V_{iz}$ give the mean for each entry in a random matrix. Assume that the residuals from this model are independent and normally distributed with means μ_x, μ_y, and μ_z for X, Y, and Z, respectively, and variance σ^2. The values in X, Y, and Z are conditionally independent given the parameter space

$$\{U, S, V, U_y, V_z, U_{ix}, S_{ix}, V_{ix}, U_{iy}, S_{iy}, V_{iy}, U_{iz}, S_{iz}, V_{iz}\},$$

so the joint likelihood is a product of independent normal likelihoods

$$\text{logL}(U, S, V, U_y, V_z, U_{ix}, S_{ix}, V_{ix}, U_{iy}, S_{iy}, V_{iy}, U_{iz}, S_{iz}, V_{iz}; X, Y, Z)$$

$$\propto \log \prod_{i,j}[e^{-\frac{1}{2\sigma^2}(X_{ij}-[\mu_x]_{ij})^2}] \prod_{i,j}[e^{-\frac{1}{2\sigma^2}(Y_{ij}-[\mu_y]_{ij})^2}] \prod_{i,j}[e^{-\frac{1}{2\sigma^2}(Z_{ij}-[\mu_z]_{ij})^2}]$$

$$= -\frac{1}{2\sigma^2}[\sum_{i,j}(X_{ij} - [\mu_x]_{ij})^2 + \sum_{i,j}(Y_{ij} - [\mu_y]_{ij})^2 + \sum_{i,j}(Z_{ij} - [\mu_z]_{ij})^2]$$

This likelihood is maximized when the total sum of squared residuals is minimized, which is accomplished by the alternating least squares method implemented by the LMF algorithm. Thus, step (1) in the algorithm above corresponds to an M-step. Step (2) corresponds to an E-step, where the expected values for X, Y, and Z are given by their means μ_x, μ_y, and μ_z.

In practice the residual variance σ^2 may not be the same across data matrices. However, when X, Y and Z are normalized to have the same Frobenius norm, the M-step (1) can be considered to optimize a weighted log-likelihood in which the likelihood for each data matrix are scaled to contribute equally.

3.5.2 Simulation

We generated 100 data sets using a simulation scheme analogous to that in Section 3.4.3 under each of 6 settings (2 different matrix dimensions for Y and Z and 3 different noise variances). In each setting, the X matrix dimensions were 50×50. The Y matrix was $m_2 \times 50$, where m_2 was 30 or 200. Similarly, the Z matrix was $50 \times n_2$, where n_2 was 30 or 200. In all simulations, m_2 and n_2 were equal. We varied the non-shared dimensions of Y and Z to see the effect of having more data in Y and Z on the imputation accuracy for X. We also varied the variance of the noise matrices among 0.1, 1, and 10. The ranks (r, r_x, r_y, and r_z) were each chosen from a Uniform($\{0, 1, 2, 3, 4, 5\}$) distribution. For each simulation we randomly set 3 rows, 3 columns, and up to 50 additional entries to missing. We compared 3 different methods for imputation: LMF-JIVE, LMF with only joint structure (LMF), and SVD. We then evaluated the methods using two different error calculations: the sum of squared errors for the imputed X matrix compared to the simulated X matrix and the sum of squared errors for the imputed X matrix compared to the true underlying structure of the X matrix ($X_{true} = US_xV^T + U_{ix}S_{ix}V_{ix}^T$):

$$\begin{aligned}\text{Error}(X) &= \|(X_{est} - X)[\text{missing values}]\|_F^2 / \|X[\text{missing values}]\|_F^2 \qquad (3.11)\\ \text{Error}(X_{true}) &= \|(X_{est} - X_{true})[\text{missing values}]\|_F^2 / \|X_{true}[\text{missing values}]\|_F^2\end{aligned}$$

These values were computed separately for the two types of missing entries (those missing an entire row/column and those missing single entries).

A summary of the results are given in Table 3.2. Generally, the LMF-JIVE imputations performs better than the SVD imputation, indicating that incorporating information from Y and Z can improve accuracy. This is especially helpful for the full row/column missing values, where the SVD imputation can not do better than mean imputation because there is no information present for estimating those entries with an SVD (so the SVD error is always greater than 1). An exception is when there is high noise and the dimensions of Y and Z are small, in which case LMF and LMF-JIVE may be overfitting. LMF with joint structure universally performed worse than LMF-JIVE for this study, demonstrating the benefit of decomposing joint and individual structure.

Table 3.2: Results of the imputation simulations. Error(X) shows how well each method imputed the simulated data set, while Error(X_{true}) shows how well each method imputed the true underlying structure of the simulated data set when excluding random noise. Each value is the mean from 100 simulations. The oracle is calculated as $\|E_x[\text{missing values}]\|_F^2/\|X[\text{missing values}]\|_F^2$ and represents the best the imputation can do because of the random noise.

Individual Missing		Error(X)			Error(X_{true})			
m_2, n_2	Var(Noise)	SVD	LMF	LMF-JIVE	SVD	LMF	LMF-JIVE	Oracle
30	0.1	0.025	0.031	0.024	0.007	0.013	0.005	0.018
200	0.1	0.028	0.028	0.032	0.007	0.006	0.010	0.022
30	1	0.232	0.274	0.218	0.068	0.120	0.051	0.174
200	1	0.222	0.215	0.198	0.067	0.058	0.039	0.165
30	10	1.061	1.185	0.931	1.144	1.495	0.803	0.640
200	10	1.139	0.927	0.85	1.288	0.815	0.580	0.631
Row/Column Missing		Error(X)			Error(X_{true})			
m_2, n_2	Var(Noise)	SVD	LMF	LMF-JIVE	SVD	LMF	LMF-JIVE	Oracle
30	0.1	1.022	0.968	0.572	1.022	0.965	0.563	0.019
200	0.1	1.023	1.698	0.594	1.024	1.731	0.584	0.023
30	1	1.022	3.967	0.667	1.027	4.788	0.601	0.175
200	1	1.018	7.971	0.618	1.021	9.639	0.536	0.175
30	10	1.016	7.652	1.124	1.050	22.804	1.328	0.650
200	10	1.014	5.399	0.877	1.038	14.088	0.663	0.632

3.6 Rank Selection Algorithms

Selection of the joint and individual ranks for LMF-JIVE must be considered carefully, to avoid misallocating joint and individual structure. Several rank selection approaches have been proposed for JIVE and related methods for vertical integration, including permutation testing [Lock et al., 2013], Bayesian information criterion (BIC) [O'Connell and Lock, 2016], and likelihood cross-validation [Li and Jung, 2017]. Here we propose and implement two approaches to rank selection in the LMF-JIVE context: (i) a permutation testing approach extending that used for JIVE (Section 3.6.1), and (ii) a novel approach based on cross-validated imputation accuracy of missing values (Section 3.6.2). Our results suggest that the two approaches give similar overall performance regarding rank recovery; approach (i) is well-motivated if a rigorous and conservative statistical approach to the identification of joint structure is desired, whereas approach (ii) is well-motivated if missing value imputation is the primary task.

3.6.1 Permutation Testing

For the permutation test rank selection algorithm, we generate a null distribution for joint structure by randomly permuting the rows of Z and the columns of Y to break any associations between the matrices while maintaining the structure within each matrix. We generate a null distribution for individual structure by an appropriate permutation of the entries within a matrix, under the motivation that individual low-rank approximations should give correlated structure that is not explained by the joint approximation. We iterate between selecting the joint rank from $\{X^J, Y^J, Z^J\}$, updating the LMF-JIVE decomposition, and selecting the individual ranks from $\{X^I, Y^I, Z^I\}$, until the selected ranks remain unchanged.

The joint rank is estimated as follows:

1. Let r_{max} be the maximum possible (or plausible) joint rank (r) for the data.

2. Initialize $\tilde{X} = X^J$, $\tilde{Y} = Y^J$, and $\tilde{Z} = Z^J$.

3. Compute the sum of squared residuals for a rank 1 LMF approximation of the data:

$$\text{SSR} = \|J_x - \tilde{X}\|_F^2 + \|J_y - \tilde{Y}\|_F^2 + \|J_z - \tilde{Z}\|_F^2$$

4. Subtract the joint structure from step 3 from X^J, Y^J, and Z^J. Set $\tilde{X} = \tilde{X} - J_x$, $\tilde{Y} = \tilde{Y} - J_y$, and $\tilde{Z} = \tilde{Z} - J_z$.
5. Repeat steps 3 and 4 $(r_{max} - 1)$ times.
6. Permute the columns of Y and the rows of Z by sampling without replacement, yielding Y_{perm} and Z_{perm}.
7. Set $\tilde{Y} = Y_{perm}$, and $\tilde{Z} = Z_{perm}$.
8. Repeat steps 3 and 4 r_{max} times using the simulated data set.
9. Repeat steps 6 through 8 for 99 more permutations.
10. Select r as the highest rank such that the SSR of the true data is higher than the 95th percentile of the SSRs for the permuted data.

To estimate the individual ranks of the data sets, we permute all entries of X^I, the entries within each row in Y^I, and the entries within each column in Z^I. We compute an SVD of rank $r_{i,max}$ for each of the true data matrices $i = X, Y, Z$, and also for each of the permuted matrices. For each matrix, r is then chosen to be the highest rank such that the r^{th} singular value of the true data is higher than the 95th percentile of the r^{th} singular values for the permuted data sets.

We used a simulation to test the permutation rank selection algorithm. We simulated 100 data sets with randomly chosen joint and individual ranks from independent Uniform($\{0, 1, 2, 3, 4, 5\}$) distributions. All elements of the joint structure components, U, S_x, V, U_y, and V_z, and the individual structure components, U_{ix}, V_{ix}, U_{iy}, V_{iy}, U_{iz}, and V_{iz}, were simulated from a Normal$(0, 1)$ distribution.

The simulation showed that the permutation test tends to underestimate joint rank, and it tends to overestimate the individual rank. It underestimated joint rank in 76% of the simulations. It overestimated the individual ranks for X, Y, and Z by 51%, 59%, and 68%, respectively. The individual rank of X was estimated better than the other ranks, with 43% of simulations getting the correct rank and a mean absolute deviation of 0.85.

3.6.2 Cross-validation

We used a forward selection algorithm based on cross-validation missing value imputation to choose the ranks. The algorithm works by setting a portion of the X matrix to missing at random, then it uses the imputation method described in Section 3.5 for various rank combinations, choosing the model with the lowest sum of squared error ($SSE_0 = \|(X_{est}[\text{missing values}] - X[\text{missing values}])\|_F^2$). The forward selection starts with the model with ranks $r, r_x, r_y, r_z = 0, 0, 0, 0$ and tests models when adding one to any of the ranks. Below is a detailed description of the algorithm:

We consider the following forward selection algorithm based on cross-validation missing value imputation to choose the ranks:

1. Randomly set a portion (full rows, full columns, and some individual entries) of the matrix entries to missing.

2. Use the imputation algorithm described in Section 4.3.1 to estimate the full matrix, X_{est}.

3. Compute the sum of squared error ($SSE_0 = \|(X_{est}[\text{missing values}] - X[\text{missing values}])\|_F^2$) for the null imputation with ranks $r, r_x, r_y, r_z = 0, 0, 0, 0$.

4. Add 1 to each rank and compute the SSE for each resulting imputed matrix.

5. If no models were better than SSE_0, then choose that model's ranks and stop.

6. Otherwise, choose the model with the lowest SSE and set SSE_0 to that model's SSE.

7. Repeat steps 4 through 6 until adding 1 to any rank does not decrease the SSE of the imputation.

We tested this cross-validation approach using 100 simulated data sets generated as in Section 3.6.1. In contrast to the permutation test, the cross-validation method tended to overestimate the joint rank and underestimate the individual ranks. It correctly estimated joint rank in 30% of simulations and overestimated it in 58%. The mean absolute deviation for the estimated rank from the true joint rank was 1.17. The estimates for the individual structure of X were closer, with 58% capturing the true rank and an mean absolute deviation of 0.74. The Y and Z individual ranks were underestimated 61% and 62% of the time, respectively. They had a mean absolute deviation of 1.26 (for Y) and 1.21 (for Z).

While we chose forward selection here, stepwise selection and an exhaustive search of all possible combinations are also possible. We tested the stepwise selection and got almost equivalent results, indicating that the algorithm was seldom taking backwards steps. A more exhaustive search would have probably yielded more accurate results, as the SSE was higher for our selected ranks than when using the true ranks in many of the cases; however, this would have required running the algorithm for 6^4 rank combinations, which is computationally infeasible.

3.7 Application to Toxicology Data

We applied the LMF-JIVE algorithm to the toxicity (X), chemical attribute (Y), and genotype data (Z). Previous investigations of these data [Abdo et al., 2015, Eduati et al., 2015] and similar data [Lock et al., 2012] have found clear evidence of systematic cell-line variability in toxicity for several chemicals, but the genetic drivers of this variability have not been well characterized. Various analyses presented in Eduati et al. [2015] show that molecular chemical attributes are significantly predictive of overall (mean) chemical toxicity. We are interested in assessing the combined predictive power of chemical attributes and genetics on toxicity using a fully multivariate and unified approach, and we are also interested more generally in exploring patterns of systematic variability within and across these three linked data matrices.

For 751 cell lines, the cytotoxic effect of each of 105 different chemicals was quantified by $\log(\mathrm{EC}_{10})$, where EC_{10} is the dose concentration that results in a 10% decrease in cell viability; these values make up the toxicity matrix $X : 751 \times 105$. Approximately 1.3 million SNPs were available for each of the 751 cell lines. SNP values were of the form $z = \{0, 1, 2\}$, where z gives the number of minor alleles. We first removed all SNPs with missing values across any of the 751 cell lines, and removed those SNPS with a minor allele frequency less than or equal to $5/751 \approx 0.007$. To identify a set of SNPs with potential relevance to cell toxicity, we performed a simple additive linear regression to predict toxicity from SNP for each (SNP, chemical) pair. Those SNPs with an association p-value less than 10^{-8} for any chemical were included, resulting in 441 SNPs $Z : 751 \times 441$. For each of the 105 chemicals, data were available for 9272 quantitative structural attributes defined using the Simplex representation for molecular structure (SIRMS) [Kuz'min et al., 2008]. Attributes with value 0 for at least 100 chemicals were removed, leaving 2092 attributes. These values were log-transformed ($y = log(y + 1)$ and centered so that each attribute had mean 0. To identify a set of attributes with potential relevance to cell toxicity, we performed a simple additive linear regression to predict toxicity for each (attribute, cell line) pair. Those attributes with an association p-value less than 10^{-3} for any cell line were included, resulting in 105 attributes $Y : 105 \times 105$. All three matrices (X, Y, and Z) were centered to have overall mean 0, and scaled to have the same Frobenius norm.

We conducted a robust 20-fold cross-validation study to assess the accuracy of recovering underlying structure via missing data imputation. For each fold 5% of the columns of X (chemicals) were withheld as missing, 5% of the rows of X were withheld as missing, and 5% of the entries of X from the remaining rows and columns are randomly selected to be withheld as missing. The folds were non-overlapping, so that each entry of X has its column missing exactly once, its row missing exactly once, and its value missing (with most other values in its row and column present) exactly once, over the 20 folds. For each fold we impute all missing values as described in Section 4.3.1, using either a joint-only LMF for $\{X, Y, Z\}$, an SVD approach for X only, or a joint and individual LMF-JIVE approach. For further comparison we also imputed missing values in X using a nuclear norm penalty via the R package softImpute [Hastie and Mazumder, 2015].

We select the model ranks via the forward-selection approach of Section 3.6.2, using the mean squared error for imputed values (averaged over row-missing, column-missing, and entrywise-missing imputations) as the selection criteria. This results in a joint factorization with rank $r = 4$, an SVD factorization with rank $r_X = 5$, and a joint and individual factorization with ranks $r = 3, r_X = 4, r_Y = 2$, and $r_Z = 6$. The relative imputation accuracy (Equation (3.11) for each method is shown in Table 3.3, broken down by accuracy in imputing entry-wise missing values, column-wise missing, row-wise missing, and values that are missing their entire row and column. The joint and individual factorization approach performed comparatively well for all types of

Table 3.3: Relative error for missing data imputation under different factorization approaches.

	LMF	SVD	softImpute	LMF-JIVE
Missing chemical and cell line	0.878	1.02	1.00	0.854
Missing chemical	0.898	1.02	1.00	0.875
Missing cell line	0.203	0.208	1.00	0.201
Missing entry	0.164	0.112	0.113	0.114

missing data, demonstrating that it is a flexible compromise between the joint only and individual only (SVD) approaches. SVD and softImpute imputation performed better than joint imputation for entry-wise missing data, and similarly to joint and individual imputation. This suggests that the attributes (Y) and genetics (Z) provide little additional information on the toxicity for a given [cell line, chemical] pair, given other values in X. However, where no toxicity data are available for a given cell line and chemical, imputation accuracy is substantially improved with the joint approaches that use Y and Z (here SVD imputation using X only can perform no better than a relative error of 1, because there is no data for the given row and column).

Figure 3.4 shows heatmaps for each of linked the linked matrices, and heatmaps from their low-rank approximations resulting from the joint and individual factorizations with the selected ranks. The toxicity data has several chemicals that are universally more toxic across the cell line panel and several chemicals that are universally less toxic, but there are also some chemicals that demonstrate clear and structured heterogeneity across the cell lines. The chemical attribute heatmap shows distinctive patterns corresponding to more toxic and less toxic chemicals. Patterns in the SNP data that are associated with toxicity are less visually apparent. However, a plot of the cell line scores for the first two joint components in Figure 3.5 reveals a prominent racial effect; cell lines that are derived from native African populations are distinguished from cell lines derived from non-African populations. The effect of race on toxicity is not strongly detectable when considering each chemical independently; under independent t-tests for African vs. non-African populations, the smallest FDR-adjusted [Benjamini and Hochberg, 1995] p-value is 0.05. However, a permutation-based test using Distance Weighted Discrimination [Wei et al., 2016] for an overall difference between African and non-African populations across the 105 chemicals is highly significant (p-value< 0.001), reinforcing the finding that toxicity profiles differ by race.

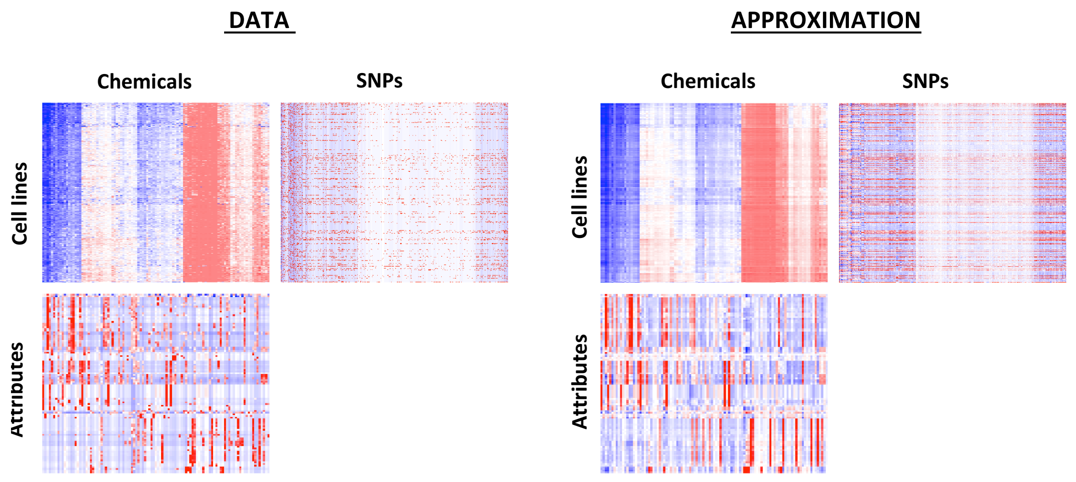

Figure 3.4: Heatmaps of the toxicity, attribute, and genotype data matrices (left) and their low-rank approximations (right).

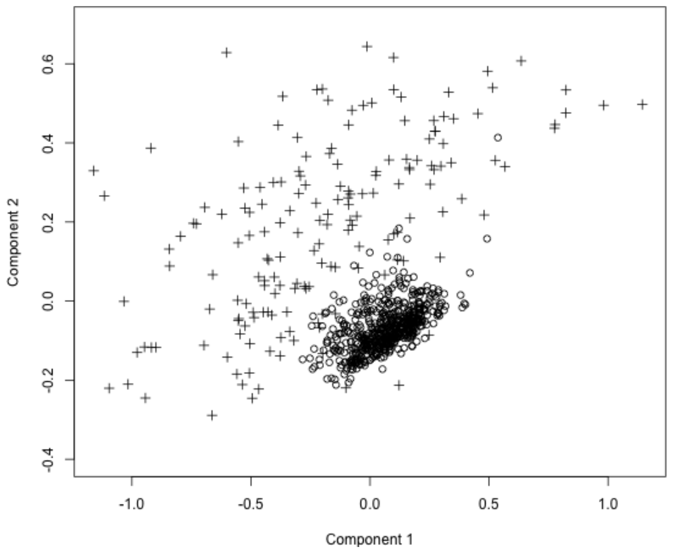

Figure 3.5: First two cell-line components of joint structure; '+' denotes a cell line derived from a native African population, 'O' denotes a cell line derived from a non-African population.

3.8 Discussion

With a dramatically increasing number of modalities for collecting high-content and multi-faceted data efficiently, large multi-source and interlinked data sets are becoming increasingly common. Methods that appropriately address these data without simplifying their structure are needed. Although our development for this chapter was motivated by a cytotoxicity study, the methodology is relevant to potential applications in a wide range of other fields. Most notably, new methods for simultaneous horizontal and vertical integration are needed for integrating molecular "omics" data across multiple disparate sample sets, e.g., the integration of multi-omics molecular data across multiple disparate types of cancer (pan-omics pan-cancer). LMF improves on previous methods for horizontal or vertical integration only, to allow for bi-dimensional integration. The LMF and LMF-JIVE decompositions can be used for exploratory analyses, missing data imputation, dimension reduction, and for creating models using the joint or individual components.

Although we have proposed two rank selection methods for LMF in this paper, both of them tend to be overly conservative. Other approaches may be more accurate; for example, some form of efficient search may perform better than forward selection for the cross-validation rank selection approach. Model-based approaches such as BIC may also be used, but are generally less accurate in the JIVE context [Section 2.5; O'Connell and Lock [2016]].

Because the LMF-JIVE algorithm does not always reach a global optimum, it may be beneficial to try both estimating joint structure first and estimating individual structure first, as in Section 3.4.3. Alternatively, one can try several randomly generated starting values.

The LMF algorithm is estimated via a squared residual loss function, which is best motivated under the assumption that the data are normally distributed. Adjusting the LMF algorithm to accommodate other likelihoods is an area for future research. For example, for categorical or count data, a model-based approach with alternative likelihoods for each dataset may be more appropriate. This idea is explored for the context of vertical integration in Li and Gaynanova [2017]. In Chapter 4, we introduce an exponential family model for bidimensionally integrated data.

For this chapter we have described an application to 3 interlinked data matrices. All results are trivially extended to data with more than 2 vertically-aligned matrices $(Y_1, Y_2, \ldots)$ and with more than 2 horizontally-aligned matrices $(Z_1, Z_2, \ldots)$. However, results concerning uniqueness and identifiability do not readily extend to the context of multiple matrices that share both their row and column dimensions $(X_1, X_2, \ldots)$, which is a subject for future research. Another exiting direction of future work are extensions to higher-order tensors (i.e., multi-way arrays). Current multi-source decomposition methods, including LMF, are limited to two-dimensional arrays (matrices). However, multi-source data sets may involve data with more than two dimensions. For example, for the cytotoxicity data analyzed here, the toxicity matrix is a summary of toxicity data for multiple concentrations of each chemical. However, we could avoid this summary and work with the individual concentrations, which would give us a 3-dimensional tensor for toxicity (cell lines $\times$ chemicals $\times$ concentrations). Then we would have a multi-way, multi-source problem that we could solve using LMF if we had a method for handling joint and individual tensor decompositions. Additionally, a multi-source data decomposition for higher-order tensors could have potential applications in MRI studies and personalized medicine. LMF provides an important first step toward more general higher-order data integration, because it allows multiple dimensions to be shared, which is is likely to be encountered for multi-source tensor data.

Chapter 4

Generalized Linked Matrix Factorization

4.1 Introduction

The methods described in Chapters 2 and 3 are based on models that use the squared error loss function. The LMF algorithm in particular is reliant on ordinary least squares to estimate the scores and loadings. Such an approach requires that the data are quantitative. Moreover, it is best motivated by the assumption that data are normally distributed, because the least squares estimate maximizes the likelihood under an assumption of normality. Thanks to the central limit theorem, this assumption is often reasonable to make when using statistics that are based on the mean. In other situations, the distribution of interest may approximate the normal distribution well enough. However, some contexts may be much better served with more accurate distributional assumptions. Many biological outcomes are represented as binary variables, such as disease or exposure status. In genetics one might be interested in including binary SNP data in a multi-source data set. We may also be interested in studying proportions, such as batting averages, as we use in our motivating example in this section. These data would be better represented by a binomial distribution.

In the context of regression, there are several alternative models to ordinary least

Table 4.1: Common exponential family distributions and their corresponding canonical links.

Distribution	f(x)	μ	Canonical Link
Normal	$(2\pi\sigma^2)^{-\frac{1}{2}}e^{(x-\mu)^2}$	θ	Identity ($g(\mu) = \mu$)
Binomial	$\binom{N}{x}\theta^x(1-\theta)^{N-x}$	$\frac{e^\theta}{1+e^\theta}$	Logit ($g(p) = \log \frac{p}{1-p}$)
Poisson	$\frac{1}{\theta}e^{-\theta x}$	e^θ	Log ($g(\lambda) = \log \lambda$)

squares that can account for other distributions. Some methods, such as Generalized Estimating Equations (GEE) [Hardin and Hilbe, 2002], are robust to model misspecification. Others are designed for estimating other specific parametric models. Generalized Linear Models (GLMs), for example, are used to estimate regression models when the assumed distribution of the response is within the exponential family.

GLMs are used when data are assumed to follow some exponential family distribution. Exponential familiy distributions are distributions that can be parameterized as follows:

$$f(x|\theta) = h(x)\exp\{x\theta - b(\theta)\}.$$

In this parameterization, $h(x)$ is some function of x that does not depend on θ, and $b(\theta)$ is some function of θ that does not depend on x. Common examples of exponential family distributions are the normal, binomial, and Poisson distributions. A key component of the GLM model is the link function, which is a function that relates the natural parameter θ and the mean $\mu = E[x]$. Under the exponential family model, $\mu = b'(\theta)$ and $\text{Var}[x] = b''(\theta)$. The canonical link function is defined as $g(\mu) = b'^{-1}(\theta)$. Commonly used canonical link functions are shown in Table 4.1.

GLMs are fit using Iteratively Reweighted Least Squares (IRLS) [Green, 1984]. IRLS is a method that maximizes the likelihood of a GLM by alternating between fitting a weighted least squares regression and adjusting the weights.

There have been some methods developed to handle matrix decompositions of non-Gaussian data. An exponential family version of PCA was developed to maximize the likelihood under exponential family models [Collins et al., 2002]. In the exponential PCA model, we assume each entry x_{ij} of a matrix X follows a particular exponential distribution given Θ_{ij}. Exponential PCA decomposes Θ into loadings U and scores V:

$$\Theta = UV^T$$

In the multi-source context, Li and Gaynanova [2017] introduced the Generalized Association Study (GAS), which is used for heterogeneous multi-source data with one shared dimension. Heterogenous data are data that follow different distributions. In the case of GAS, the different matrices can follow different distributional assumptions. Their method models a decomposition of the natural parameter space rather than a decomposition of the mean. They used an IRLS algorithm to estimate either the scores and loadings for the decomposition at each step. As they mention in their paper, GAS is identical to JIVE if the data are normally distributed.

For our motivating example for this project, we analyzed a baseball data set with the intention of estimating and predicting batting averages for particular batters against particular pitchers. The primary data of interest was a batter vs. pitcher data set, which had the number of at-bats (AB) and hits (H) between each specific batter and pitcher combination in Major League Baseball (MLB). We assume that these data can be represented with a binomial distribution: $H \sim Bin(AB, p)$, where p represents the hypothetical true batting average for a particular batter against a particular pitcher. We also have the pitching stats for all of the pitchers, as well as the batting stats for all of the batters, so this data set is a multi-source data set in which 2 matrices share the batters and 2 matrices share the pitchers. This is the same data structure as we used in the LMF algorithm (the batter vs. pitcher data is X, the pitching data is Y, and the batting data is Z) (see Chapter 3). The complication is that we now have data that are heterogenously distributed, as X follows a binomial distribution, while we assume Y and Z are normally distributed.

In this chapter, we will discuss how we combined the ability to handle multiple shared dimensions of LMF with the ability to handle heterogeneously distributed matrices of GAS to create an algorithm for data sets that are heterogeneously distributed with multiple shared dimensions, which we call Generalized Linked Matrix Factorization (GLMF).

4.2 Joint Factorization

Model

As in Chapter 3, we will refer to the three matrices involved in the GLMF as X, Y, and Z, where X shares its row space with Y and its column space with Z. The only difference in the structure of the data is that we assume X, Y, and Z can follow any exponential family distribution, and each of them can follow different distributions. Let the dimensions of X be $m_1 \times n_1$, the dimensions of Y be $m_2 \times n_1$, and the dimensions of Z be $m_1 \times n_2$. Our task is to leverage shared structure across X, Y, and Z in a simultaneous low-rank factorization.

We begin by assuming exponential family distributions (F_x, F_y, and F_z) on each of the data sets.

$$X \sim F_X(\Theta_X)$$
$$Y \sim F_Y(\Theta_Y)$$
$$Z \sim F_Z(\Theta_Z)$$

In the context of the baseball data, we assume

$$X \sim Bin(N, P)$$
$$Y \sim N(\mu_Y, \sigma_Y^2)$$
$$Z \sim N(\mu_Z, \sigma_Z^2)$$

where $P = InverseLogit(\Theta_X)$, $\mu_Y = \Theta_Y$, and $\mu_Z = \Theta_Z$.

We define a joint rank r approximation for the three data matrices as follows:

$$\Theta_X = UV^T$$
$$\Theta_Y = U_y V^T$$
$$\Theta_Z = UV_z^T$$

where each Θ represents the corresponding natural parameter matrix.

- U is an $m_1 \times r$ matrix representing the row structure shared between Θ_X and Θ_Z

- V is an $n_1 \times r$ matrix representing the column structure shared between Θ_X and Θ_Y
- U_y is an $m_2 \times r$ matrix representing how the shared column structure is weighted over the rows of Θ_Y
- V_z is an $n_2 \times r$ matrix representing how the shared row structure is weighted over the columns of Θ_Z

4.2.1 GLMF Estimation

To estimate the underlying decomposition, we use an approach similar to the alternating least squares approach that was used for LMF. We alternate between estimating the loadings U and U_y and the scores V and V_z. At each step, the estimation of the scores or loadings are done using the IRLS algorithm described in [Li and Gaynanova, 2017]. This algorithm iteratively operates to maximize the likelihood of Θ_x, Θ_Y, and Θ_Z given X, Y, and Z, shown below.

$$L(\Theta|X, Y, Z) = \prod_{x_i \in X} f_X(x_i|\theta_{X_i}) \prod_{y_i \in Y} f_Y(y_i|\theta_{Y_i}) \prod_{z_i \in Z} f_Z(z_i|\theta_{Z_i}) \tag{4.1}$$

In this likelihood, f_X, f_Y, and f_Z are the probability density functions for the exponential family distributions assumed for X, Y, and Z. If we express this in the general form for exponential family distributions, we have the following lkelihood for the X matrix:

$$L(\Theta_X|X) = \prod_{x_i \in X} h_X(x_i) \exp\{x_i\theta_{X_i} - b(\theta_{X_i})\}.$$

The likelihoods for Y and Z can be written similarly.

IRLS Algorithm

The following steps describe the IRLS procedure for estimating the scores V for a matrix X given the loadings U. Let $g()$ be the link function.

1. By default, all starting weights w are set to 1.

2. Set the starting values for $\mu = X$. A small correction is made to binomial data so that the link function does not involve taking the log of zero, which is undefined.

3. Set the natural parameter matrix $\Theta = g(\mu)$. For heterogeneous data, the link function may vary across rows or columns. In that case, if we split the data into n partitions, $i = 1, .., n$, then we can set each $\Theta_i = g_i(\mu_i)$, where $g_i()$ is the link function corresponding to the distribution of the i^{th} partition.

4. Generate an induced response matrix $S = \Theta + \frac{X-\mu}{\frac{d\mu}{d\Theta}\Theta}$.

5. Set the weights $\tilde{w} = \sqrt{\frac{w[\frac{d\mu}{d\Theta}\Theta]^2}{Var(\mu)}}$.
   Note: For heterogeneous data, steps 4 and 5 are partitioned by distribution (as in step 3).

6. For each row i, compute the weighted least squared estimate for $V_{i\cdot}$:

$$\hat{V_{i\cdot}} = (U^T \tilde{w}^2 U)^{-1} U^T \tilde{w}^2 S_{\cdot i}$$

7. Update $\Theta = X\hat{V}^T$.

8. Repeat Steps 4 through 7 until convergence

A similar procedure is used for estimating V given U. For simplicity in this case, X^T is used as the response instead of X to avoid transpositions within the algorithm, and the least squares estimate for each row i of $\hat{U}$ is $\hat{U_{i\cdot}} = (V^T \tilde{w}^2 V)^{-1} V^T \tilde{w}^2 S^*_{\cdot i}$, where S^* is the induced response matrix based on X^T.

GLMF Algorithm

Given initial values, the algorithm proceeds by iteratively updating the components U, V, U_y, and V_y. We first initialize $\tilde{V} = [V^T V_Z^T]^T$ as the first r right singular vectors of the singular value decomposition (SVD) of $\tilde{Z}$. The initial estimate for V is the first n_1 rows of $\tilde{V}$. We then repeatedly cycle through the following steps to maximize the likelihood in Equation 4.1:

1. Update U_y using the IRLS algorithm detailed above, given Y and scores V.

2. Update U via IRLS, given $\tilde{Z}$ and scores $\tilde{V}$.

3. Update $\tilde{U}$: $\tilde{U} = [U^T \;\; U_y^T]^T$

4. Update V via IRLS, given $\tilde{Y}$ and loadings $\tilde{U}$.

5. Update V_z via IRLS, given Z and loadings U.

6. Update $\tilde{V} = [V^T \;\; V_z^T]^T$.

7. Iterate Steps 1 through 6 until the estimates of μ converge.

We also update the variance estimate $\hat{\sigma}^2$ to maximize the likelihood for the normally distributed data at each stage. The algorithm results in the following rank r estimates for the joint structure natural parameters:

$$\begin{aligned} \Theta_x &= UV^T \\ \Theta_y &= U_y V^T \\ \Theta_z &= UV_z^T. \end{aligned} \tag{4.2}$$

4.2.2 Illustrative Simulation

To test the GLMF algorithm's ability to recover the underlying parameters, we simulated a single data set with three data sources X, Y, and Z. We simulated rank 3 scores and loadings from a N(0,0.4) distribution. We used these scores and loadings to construct the natural parameters Θ for the simulated X, Y, and Z matrices. A matrix N was generated from integer values between 1 and 8, inclusive. An inverse logit was used to calculate p for X, and X was drawn from a Bin(N, p) distribution. Y and Z were drawn from N(Θ_Y,0.1) and N(Θ_Z,0.1), respectively. We fit a rank 3 GLMF model to the data. We can see in Figure 4.1 that this low-rank approximation of the data does a fairly good job of recovering the true values for p, $\mu_y = \Theta_y$, and $\mu_z = \Theta_z$.

4.3 Imputation

With the LMF algorithm, we imputed values for the X matrix by alternating between imputing X using the LMF decomposition scores and loadings and estimating the LMF

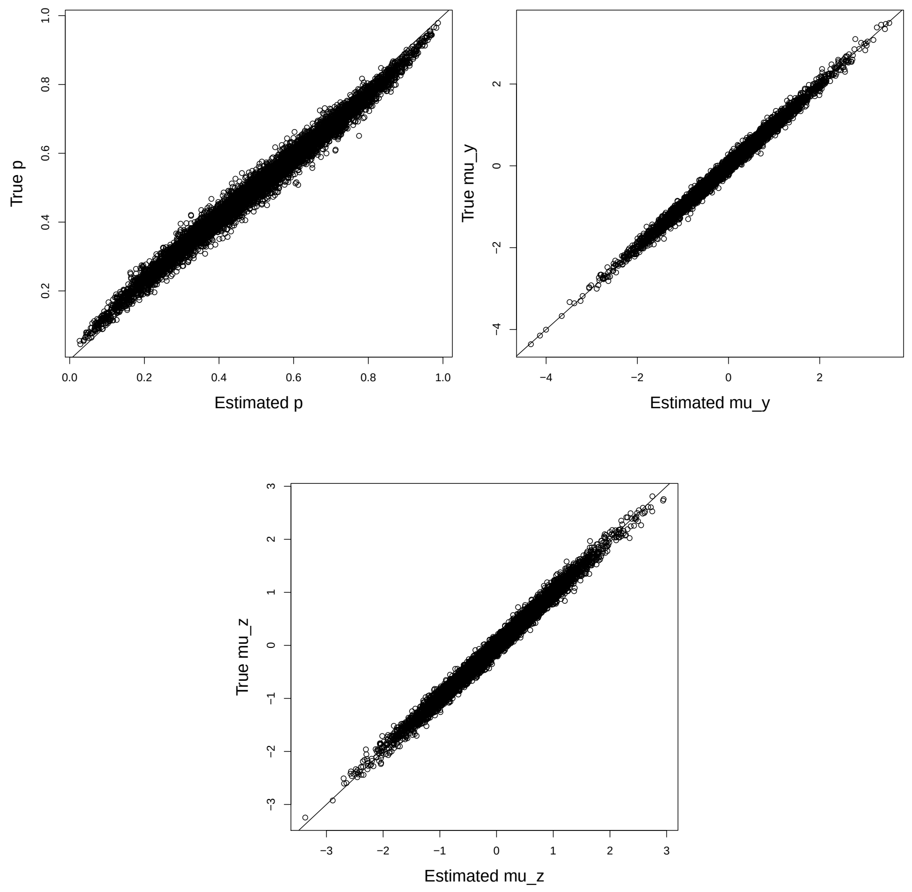

Figure 4.1: Estimated parameters of the GLMF decomposition of a simulated data set compared to the true parameter values.

decomposition using the imputed values (Section 3.5). In this case, our X matrix contains binomial data instead of normally distributed data, so instead of using the estimated means to impute the missing values, we use the estimated binomial probabilities to sample the values. In order to impute the binomial missing values in the X, we alternate between simulating imputed values from the estimated binomial probability matrix p for X and estimating the p matrix through a GLMF of the imputed X matrix.

4.3.1 Algorithm

The imputation algorithm begins by initializing the missing values. To initialize $\hat{X}$, the imputed matrix based on X, we begin by initializing the $\hat{p}$ matrix, the working binomial probability matrix. To do this, we average the observed probabilities over the rows and over the columns. To compute each estimated $\hat{p_{ij}}$, we take the average of $\hat{p_{i\cdot}}$ and $\hat{p_{\cdot j}}$. Then we use $\hat{p}$ to impute the missing values in X. For each missing value X_{ij}, we simulate $\hat{X}_{ij}$ using a single draw from a Bernoulli($\hat{p_{ij}}$) distribution. Next, we fit the GLMF model for $\hat{X}$ to re-estimate $\hat{p_{ij}}$. Using this new estimate of $\hat{p_{ij}}$, we create a measure $\bar{p_{ij}}$ which is the average over $\hat{p_{ij}}$ from all of the iterations. We repeat the estimation of $\hat{X}$ and $\hat{p}$ until the successive estimates of $\bar{p}$ converge.

4.3.2 Simulation

We simulated 36 data sets of different ranks 1 through 4 to test the imputation algorithm. We also varied the dimensions of X, Y, and Z among 50,100, and 150, to test the peformance of the algorithm when Y and Z are larger than X. We compared the GLMF algorithm to four other imputation methods: mean imputation, SVD imputation, LMF imputation, and a rank 1 GLMF. For mean imputation, we simply set $\hat{p}$ to X/N for all of the non-missing values and set it to the mean of X/N for all of the missing values. SVD and LMF imputation were performed using X/N with the methods described in Section 3.5. To compare the methods, we used a measure which we refer to as imputation error, given by the formula below, where p_{true} represents the true probability matrix from which X was simulated, and $\hat{p}$ is the imputed estimate for p, equal to $\hat{X}/N$ for mean, SVD, and LMF imputation.

$$Err_{Imputation} = \frac{||p_{true} - \hat{p}||_F^2}{||p_{true}||_F^2}$$

Results from this simulation are shown in Figure 4.2. Compared to each of the other methods, the GLMF algorithm had smaller errors on average. Notably, the LMF imputation, which assumes all matrices are normally distributed, compares very poorly to the GLMF imputation in many of the simulations. All of these simulations (in which the LMF imputation error was greater than 0.8), the dimensions of Y and Z were equal

Table 4.2: Median imputation errors for the rank r GLMF algorithm compared to mean, rank r SVD, rank r LMF, and rank 1 GLMF imputation methods, where r was the true rank of the simulated data before introducing error. Median was used so that the LMF algorithm results would not be skewed as heavily by the portion of cases in which the error was unusully high.

Rank	GLMF	Mean	SVD	LMF	GLMF1
1	0.010	0.026	0.036	0.061	-
2	0.008	0.048	0.040	0.071	0.031
3	0.012	0.071	0.047	0.061	0.051
4	0.013	0.089	0.051	0.054	0.066

to or larger than the dimensions of X. This may suggest that LMF is not very robust to violations of the normality assumption when the dimensions of Y and Z are larger than the dimensions of X. In contrast, the GLMF algorithm does not seem to do poorly when X has smaller dimensions than Y and Z. The GLMF algorithm failed to converge for one of the simulated data sets. Table 4.2 compares the median imputation error across the different methods. The GLMF algorithm consistently had the lowest median imputation error. Even a rank 1 GLMF performed better than using a rank 2 LMF or a rank 2 SVD when the true rank was 2, suggesting that it may be important to have the correct assumption about the distribution of the data than about the rank of the data.

4.4 Baseball Application

We obtained batting data, pitching data, and batter vs. pitcher data for the 2017 season from MLB.com. We removed pitchers with 20 or fewer innings pitched and batters with 50 or fewer at-bats. The resulting data set contained 516 pitchers and 508 batters. We used the GLMF algorithm to impute the missing batting averages. To assess the ability of the algorithm to recover the missing averages accurately, we used 5-fold cross-validation. For each fold, we set one fifth of the non-missing entries in the data to missing. Then, we estimated all missing entries using the GLMF algorithm with rank 2. To assess the accuracy, we computed the log likelihoods of the entries in X that were set to missing using the imputed values as p. We then averaged this measure over the five folds. We compared this to mean imputation, rank 2 SVD imputation, and

Table 4.3: Comparison across different imputation methods of the likelihood for X based on imputed values of the true batting average p.

Method	$logL(p\|X)$
GLMF	-10780.29
Mean	-10782.86
SVD	-11037.48
LMF	-10983.59

rank 2 LMF imputation, computed as in Section 4.3.2. In the cases of SVD and LMF impution, we set a lower bound of 0.05 for the estimated batting averages because the normal model allows values to be zero or negative, which causes the likelihood to be undefined. The relative performances of these algorithms are summarized in Table 4.3. GLMF consistently performed better than all three other methods, as it had the highest likelihood averaged over the cross-validation folds. However, the naive mean imputation approach was not much worse than GLMF. Mean imputation actually performed better than SVD and LMF imputation. There are a couple of likely reasons for this. First, since values could be very small or negative under the normal error distribution, some of the values in SVD and LMF imputation are very small, and these have a very low likelihood. Additionally, the mean imputation value of 0.250 is likely a reasonable estimate for most batting averages because batting averages in the long run tend to fall in the range of 0.200 to 0.300, which is a fairly narrow range.

4.5 Discussion

As we have illustrated through simulations, the GLMF algorithm can be used to generate a low-rank approximation or impute missing values of a multi-source, heterogeneously distributed data set. However, although the underlying probability matrix $\hat{p}$ is identifiable in our model, the scores and loadings are not identifiable without any other constraints. If the goal is to recover the scores and loadings, some transformation needs to be done on the scores and loadings in order to make them identifiable.

We focused on a specific context in this Chapter, in which X followed a binomial distribution, and Y anf Z followed a normal distribution. But the GLMF algorithm is applicable to any exponential family distributions in any combination. Although it is primarily useful for heterogeneously distributed data, it can also be used for homogeneously distributed data, such as all normally distributed or all binomially distributed data sets. In the case of normally distributed data sets, this is an LMF model, which is a special case of GLMF.

Although we only used three matrices in our analyses in this section, with the entries of each following a single distribution, these methods are not limited to this strict data structure. The heterogeneous IRLS algorithm used to compute joint structure allows for any number of different distributions. This allows for both modeling more than two horizontally or vertically integrated data sets simultaneously even if they all follow different distributions. It also allows the entries within a matrix to follow different distributions when one of the matrices has variables that follow different distributions. However, the current algorithm may not be readily applicaple to data with more than one matrix that is both horizontally and vertically integrated with other matrices.

GLMF extends the LMF algorithm to contexts in which the distribution is assumed to be in the exponential family. However, there may be cases in which the data follow distributions that are not within the exponential family. It might be possible to create a model that uses GEE instead of GLM for joint structure estimation, which could accommodate such data.

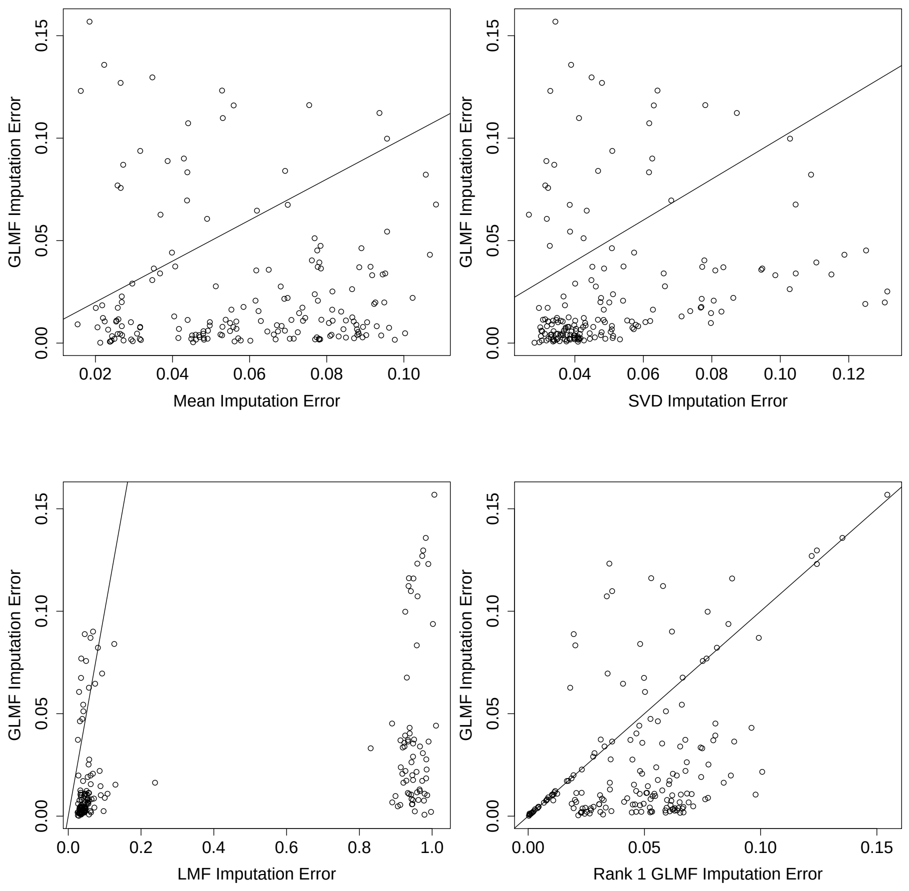

Figure 4.2: Comparisons between the imputation accuracy of the GLMF model imputation and other imputation methods. The line on the each plot corresponds to equal imputation errors between the methods. Points above the line suggest that the alternative imputation methods were more accurate for particular simulated data sets, while points below the line suggest that the GLMF imputation was more accurate (and therefore had smaller imputation error).

Chapter 5

Conclusion and Discussion

The supply of multi-source data is ever increasing as researchers continue to collect data from multiple sources. Traditionally, approaches for these data were naive and did not account for the relationships between data sources. More methods are being developed to handle such data. One of the biggest challenges in such a fast advancing field is bridging the gap between the methods development and implementing the methods in practice. Through the r.jive package, we made multi-source data methods accessible through a widely used platform. In the future, this package will be updated with functions to run the LMF and GLMF algorithms as well, putting multiple methods for analyzing different types of multi-source data in one software package. This will help others easily apply these methods to their own data sets, allowing multi-source methods to start replacing more naive approaches to these data in practice.

These methods have potential impact across copious fields of study. In this paper, we explored applications in genetics, toxicology, and baseball. Genetics data often encompasses multiple data sources, such as gene expression, miRNA expression, DNA methylation, and copy number. JIVE allows the analyses of all of these data sources together, and also allows them to be analyzed alongside other data sources.

One of the most impactful potential uses of these methods is in precision medicine. LMF or GLMF could be used to study relationships between treatment outcomes and both patient characteristics and treatment characteristics. In particular, GLMF would be useful for studies with a binary treatment outcome.

There are a lot of future directions for research in multi-source data methods. A number of methods have been developed for decomposition and dimension reduction of higher-order tensors, data arrays with more than two dimensions. A multi-source method for linked tensors would be useful for data with several dimensions, such as imaging data. There may also be use for methods that do not require assumptions on the distribution of the errors. JIVE and LMF are based on the assumption that the errors are normally distributed, while GLMF is based on the assumption that they follow some exponential family distribution.

References

Nour Abdo, Menghang Xia, Chad C Brown, Oksana Kosyk, Ruili Huang, Srilatha Sakamuru, Zhou Yi-Hui, John R Jack, Paul Gallins, Kai Xia, et al. Population-based in vitro hazard and concentration-response assessment of chemicals: the 1000 genomes high-throughput screening study. *Environmental Health Perspectives (Online)*, 123 (5):458, 2015.

Evrim Acar, Tamara G Kolda, and Daniel M Dunlavy. All-at-once optimization for coupled matrix and tensor factorizations. *arXiv preprint arXiv:1105.3422*, 2011.

Y Benjamini and Y Hochberg. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society: Series B*, 57(1):289–300, 1995.

Cancer Genome Atlas Network. Comprehensive molecular portraits of human breast tumours. *Nature*, 490(7418):61–70, 2012.

Michael Collins, Sanjoy Dasgupta, and Robert E Schapire. A generalization of principal components analysis to the exponential family. In *Advances in neural information processing systems*, pages 617–624, 2002.

Ciprian M Crainiceanu, Brian S Caffo, Sheng Luo, Vadim M Zipunnikov, and Naresh M Punjabi. Population value decomposition, a framework for the analysis of image populations. *Journal of the American Statistical Association*, 106(495):775–790, 2011.

Jan De Leeuw, Forrest W Young, and Yoshio Takane. Additive structure in qualitative data: An alternating least squares method with optimal scaling features. *Psychometrika*, 41(4):471–503, 1976.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

Federica Eduati, Lara M Mangravite, Tao Wang, Hao Tang, J Christopher Bare, Ruili Huang, Thea Norman, Mike Kellen, Michael P Menden, Jichen Yang, et al. Prediction of human population responses to toxic compounds by a collaborative competition. *Nature biotechnology*, 2015.

Peter J Green. Iteratively reweighted least squares for maximum likelihood estimation, and some robust and resistant alternatives. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 149–192, 1984.

James W Hardin and Joseph M Hilbe. *Generalized estimating equations*. Chapman and Hall/CRC, 2002.

Richard A Harshman. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.

Trevor Hastie and Rahul Mazumder. *softImpute: Matrix Completion via Iterative Soft-Thresholded SVD*, 2015. URL `https://CRAN.R-project.org/package=softImpute`. R package version 1.4.

Kristoffer Hellton and Magne Thoresen. Integrative clustering of high-dimensional data with joint and individual clusters, with an application to the metabric study. *arXiv preprint arXiv:1410.8679*, 2014.

Harold Hotelling. Relations between two sets of variates. *Biometrika*, 28(3/4):321–377, 1936.

Zhiguang Huo, Ying Ding, Silvia Liu, Steffi Oesterreich, and George Tseng. Meta-analytic framework for sparse k-means to identify disease subtypes in multiple transcriptomic studies. *Journal of the American Statistical Association*, 111(513):27–42, 2016.

Shashank Jere, Justin Dauwels, Muhammad Tayyab Asif, Nikola Mitro Vie, Andrzej Cichocki, and Patrick Jaillet. Extracting commuting patterns in railway networks through matrix decompositions. In *13th IEEE International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 541–546. IEEE, 2014.

Suleiman A Khan and Samuel Kaski. Bayesian multi-view tensor factorization. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 656–671. Springer, 2014.

Henk AL Kiers and Jos MF ten Berge. Alternating least squares algorithms for simultaneous components analysis with equal component weight matrices in two or more populations. *Psychometrika*, 54(3):467–473, 1989.

Sunghwan Kim, Dongwan Kang, Zhiguang Huo, Yongseok Park, and George C Tseng. Meta-analytic principal component analysis in integrative omics application. *Bioinformatics*, 2017.

Julia Kuligowski, David Pérez-Guaita, Ángel Sánchez-Illana, Zacarías León-González, Miguel de la Guardia, Máximo Vento, Eric F Lock, and Guillermo Quintás. Analysis of multi-source metabolomic data using joint and individual variation explained (JIVE). *Analyst*, 140(13):4521–4529, 2015.

Miklós Kurucz, András A Benczúr, and Károly Csalogány. Methods for large scale svd with missing values. In *Proceedings of KDD cup and workshop*, volume 12, pages 31–38, 2007.

VE Kuz'min, Anatoly G Artemenko, and Eugene N Muratov. Hierarchical QSAR technology based on the simplex representation of molecular structure. *Journal of computer-aided molecular design*, 22(6-7):403–421, 2008.

Gen Li and Irina Gaynanova. A general framework for association analysis of heterogeneous data. *arXiv preprint arXiv:1707.06485*, 2017.

Gen Li and Sungkyu Jung. Incorporating covariates into integrated factor analysis of multi-view data. *Biometrics*, 2017.

Johan Lindstrom, Adam Szpiro, Paul D. Sampson, Silas Bergen, and Assaf P. Oron. *SpatioTemporal: Spatio-Temporal Model Estimation*, 2018. URL `https://CRAN.R-project.org/package=SpatioTemporal`. R package version 1.1.9.

Eric F Lock and David B Dunson. Bayesian consensus clustering. *Bioinformatics*, 29 (20):2610–2616, 2013.

Eric F Lock, Nour Abdo, Ruili Huang, Menghang Xia, Oksana Kosyk, Shannon H O'Shea, Yi-Hui Zhou, Alexander Sedykh, Alexander Tropsha, Christopher P Austin, et al. Quantitative high-throughput screening for chemical toxicity in a population-based in vitro model. *Toxicological Sciences*, 126(2):578–588, 2012.

Eric F Lock, Katherine A Hoadley, James Stephen Marron, and Andrew B Nobel. Joint and individual variation explained (JIVE) for integrated analysis of multiple data types. *The annals of applied statistics*, 7(1):523, 2013.

Tommy Löfstedt and Johan Trygg. OnPLS – a novel multiblock method for the modelling of predictive and orthogonal variation. *Journal of Chemometrics*, 25(8):441–455, 2011a.

Tommy Löfstedt and Johan Trygg. Onpls—a novel multiblock method for the modelling of predictive and orthogonal variation. *Journal of Chemometrics*, 25(8):441–455, 2011b.

Roger E Millsap and William Meredith. Component analysis in cross-sectional and longitudinal data. *Psychometrika*, 53(1):123–134, 1988.

Michael J O'Connell and Eric F Lock. R.JIVE for exploration of multi-source molecular data. *Bioinformatics*, 32(18):2877–2879, 2016.

Michael J. O'Connell and Eric F. Lock. *r.jive: Perform JIVE Decomposition for Multi-Source Data*, 2017. URL `https://CRAN.R-project.org/package=r.jive`. R package version 2.1.

R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2018. URL `https://www.R-project.org/`.

Priyadip Ray, Lingling Zheng, Joseph Lucas, and Lawrence Carin. Bayesian joint analysis of heterogeneous genomics data. *Bioinformatics*, 30(10):1370–1376, 2014.

Martijn Schouteden, Katrijn Van Deun, Sven Pattyn, and Iven Van Mechelen. Sca with rotation to distinguish common and distinctive information in linked data. *Behavior research methods*, 45(3):822–833, 2013.

Martijn Schouteden, Katrijn Van Deun, Tom F Wilderjans, and Iven Van Mechelen. Performing disco-sca to search for distinctive and common information in linked data. *Behavior research methods*, 46(2):576–587, 2014.

Jos MF ten Berge, Henk AL Kiers, and Véronique Van der Stel. Simultaneous components analysis. *Statistica Applicata*, 4(4):277–392, 1992.

Johan Trygg. O2-pls for qualitative and quantitative analysis in multivariate calibration. *Journal of chemometrics*, 16(6):283–293, 2002.

George C Tseng, Debashis Ghosh, and Xianghong Jasmine Zhou. *Integrating Omics Data*. Cambridge University Press, 2015.

Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

Katrijn Van Deun, Age K Smilde, Mariët J van der Werf, Henk AL Kiers, and Iven Van Mechelen. A structured overview of simultaneous component based data integration. *Bmc Bioinformatics*, 10(1):246, 2009.

Michael E Wall, Andreas Rechtsteiner, and Luis M Rocha. Singular value decomposition and principal component analysis. In *A practical approach to microarray data analysis*, pages 91–109. Springer, 2003.

Susan Wei, Chihoon Lee, Lindsay Wichers, and JS Marron. Direction-projection-permutation for high-dimensional hypothesis tests. *Journal of Computational and Graphical Statistics*, 25(2):549–569, 2016.

JA Westerhuis, T Kourti, and JF MacGregor. Analysis of multiblock and hierarchical pca and pls models. *Journal of Chemometrics*, 12(5):301–321, 1998.

Herman Wold. Partial least squares. *Encyclopedia of statistical sciences*, 9, 2004.

Wei Yang, Nicole M Warrington, Sara J Taylor, Eduardo Carrasco, Kyle W Singleton, Ningying Wu, Justin D Lathia, Michael E Berens, Albert H Kim, Jill S Barnholtz-Sloan, et al. Clinically important sex differences in gbm biology revealed by analysis of male and female imaging, transcriptome and survival data. *bioRxiv*, page 232744, 2017.

Zi Yang and George Michailidis. A non-negative matrix factorization method for detecting modules in heterogeneous omics multi-modal data. *Bioinformatics*, 32(1):1–8, 2016.

Jieping Ye. Generalized low rank approximations of matrices. *Machine Learning*, 61 (1-3):167–191, 2005.

Tatsuya Yokota and Andrzej Cichocki. Linked Tucker2 decomposition for flexible multi-block data analysis. In *International Conference on Neural Information Processing*, pages 111–118. Springer, 2014.

G Zhou, A Cichocki, Y Zhang, and DP Mandic. Group component analysis for multi-block data: Common and individual feature extraction. *IEEE transactions on neural networks and learning systems*, 2015.