2014

Biclustering methods and a Bayesian approach to fitting Boltzmann machines in statistical learning

Jing Li
Iowa State University

Biclustering methods and a Bayesian approach to fitting Boltzmann machines in statistical learning

by

Jing Li

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Statistics

Program of Study Committee:

Stephen B. Vardeman, Major Professor
Karin S. Dorman
Max D. Morris
Dan Nettleton
Huaiqing Wu

Iowa State University

Ames, Iowa

2014

DEDICATION

To my husband Xuesong, my son Peter and my parents.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Page

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere thanks to everyone who gave me help through the Ph.D. journey.

First and foremost, I would like to thank my advisor, Prof. Stephen Vardeman for his persistent support, inspiring guidance, enthusiastic encouragement, insightful advice and immense knowledge during my Ph.D. study and research work. I have been really fortunate to have Prof. Vardeman as my advisor.

My sincere thanks also goes to my committee members Prof. Max Morris, Prof. Karin Dorman, Prof. Dan Nettleton and Prof. Huaiqing Wu for many thought-provoking discussions, insightful suggestions and enormous effort.

Also, I would like to thank Prof. Alyson Wilson for all her guidance and support in my Master's study in Statistics at Iowa State University. The knowledge and experience that I learned from her is very beneficial in my Ph.D. study.

I would also like to thank Department of Statistics for providing me a lot of resources and opportunities that enriched my knowledge, widened my horizon and helped me in career development.

I am also very grateful to all my friends for their support and help.

Last but not least, I would like to thank my family who are always there supporting me.

ABSTRACT

This disertation focuses on two topics in Statistical Learning. One is biclustering, and the other is deep learning. The whole dissertation has three chapters, where Chapter 1 and 2 focus on biclustering; Chapter 3 focuses on the deep learning topic.

Biclustering is a Statistical Learning technique that simultaneously partitions the set of samples and the set of their attributes into homogeneous subsets. In Chapter 1, motivated by movie rating data, we firstly propose a Bayesian model and an MCMC algorithm for model estimation. Because this algorithm is too slow to be of practical use with current computation power, we next propose a simplified model and design a Genetic Algorithm for maximizing the likelihood function. This approach works well on a small data set. However, due to the NP-hard nature of the problem, both approaches fail to be practically useful with current computation power. Nonetheless, they provide principled ways of solving a biclustering problem for future use as computation power develops.

Also motivated by movie rating data, where missing values need to be addressed, in Chapter 2 we propose a new Prototype-based Biclustering method. We evaluate our method on test cases with various percentages of missing values in terms of the Rand Index between our result and "true" partitions. In fact, our method has good performance on test cases even with a large missing value percentage. We further evaluate our method on a gene expression data set, that contains no missing values. Our method outperforms an existing biclustering method, i.e., Spectral Biclustering, in terms of a Mean Squared Error criterion.

Deep Learning is a Statistical Learning topic which involves a "deep" network architecture mimicing the information representation structure in human brain. In Chapter 3, motivated by a hand-written digit classification problem, we propose a Bayesian framework for fitting Boltzmann machine models that can be used in deep learning contexts. The proposed approach surpasses the previously available methods in that it provides a principled fitting method using an MCMC algorithm. The approach presented is shown to provide a reasonably effective way to extract features from multivariate data for use in classification.

CHAPTER 1. MODEL-BASED BICLUSTERING

A paper to be submitted

Jing Li

Stephen B. Vardeman

Department of Statistics

Iowa State University

Ames, IA, 50010

Abstract

Biclustering is a Statistical Learning technique that simultaneously partitions the set of samples and the set of their attributes into subsets. In this paper, motivated by movie rating data, we first propose a Bayesian model and an MCMC algorithm for model estimation. Because this algorithm is too slow to have much practical use with current computation power, we next propose a simplified model and designed a genetic algorithm for maximizing a corresponding likelihood function. This approach works well on a small data set. However, due to the NP-hard nature of the problem, both approaches presented here fail to be effective for most practical problems with current computation power. Nonetheless, these approaches provide principled ways of solving a biclustering problem for future use when computation power develops to an appropriate level.

Keywords. statistical learning, biclustering, model-based, genetic algorithm, global optimum

1.1 Introduction

Clustering is the segmenting of heterogeneous collections of objects into homogeneous groups. It is an exploratory tool for data analysis. It was first developed in the 1950s (e.g., Sneath (1957); Sokal et al. (1958)), and has a long and rich history. The goal of this study is to develop a model-based biclustering method producing results in a "checkerboard" structure as defined in Chapter 2. The motivating application of this chapter is the same as that in Chapter 2. That is, the work presented here is motivated by the movie rating data discussed in Chapter 2. Therefore, we here describe our methods in terms of "movies" and "raters" as we do in Chapter 2. The difference between this chapter and Chapter 2 is that we here develop model-based biclustering approaches in contrast with the heuristic prototype-based approach that we developed in Chapter 2. We discuss the NP-hardness of this problem in Chapter 2, and conclude that it is unrealistic to solve the problem by directly seeking a global optimization. However, the approaches we present in this chapter are not heuristic and do seek a global optimization. As we may expect based on our later discussion of the NP-hard nature of this problem, the model-based approaches presented here have limited practical use. However, they represent an effort at finding a global optimization solution using principled approaches. In future, if computation power develops to the point that NP-hard problems are not computationally infeasible, the approaches developed here could be useful for solving the biclustering problem. If we could obtain a solution through the approach here, it would probably be better than one obtained from a heuristic approach because the approaches in this chapter are principled methods.

This chapter is divided into two parts. In the first part, we present a Bayesian model for the movie rating data and develop an MCMC algorithm for it. Unfortunately, this MCMC algorithm is too slow to be used in practice. Therefore, in the second part, we simplify the model and use a genetic algorithm to search for a global solution to maximize

a likelihood. We can see from our example that this genetic algorithm works well for small data sets. However, it can only be used in relatively small problems.

1.2 An Initial Model

Suppose Y is a movie rating data set for biclustering. Let y_{ij} denote the rating score that movie i gets from rater j. Let y_{ij} be 0 if movie i is unrated by rater j and 1 if it is rated by rater j, i.e., we only consider the data as "rated" or "unrated" in this model. Suppose further that the movie index set $\mathcal{I} = \{1, 2, \ldots, I\}$ is partitioned into mutually exclusive and exhaustive sets $S_1, S_2, \ldots, S_{N^{\mathcal{P}}}$ and that the rater index set $\mathcal{J} = \{1, 2, \ldots, I\}$ is partitioned into mutually exclusive and exhaustive sets $T_1, T_2, \ldots, T_{M^{\mathcal{Q}}}$. Denote the partition of the movie index set as $\mathcal{P}$; the partition of the rater index set as $\mathcal{Q}$.

Define for every non-empty $T \subset \mathcal{J}$ a parameter $\lambda^T \in (0, 1)$ that more or less indexes the "rate" at which a rater group's loyalty to movie classes drops off across classes.

Suppose that there are positive weights $w_1, w_2, \ldots, w_I$ on movies that are related to the likelihoods of a rater watching and rating them.

For every non-empty $T \subset \mathcal{J}$ and partition $\mathcal{P}$ of movies, let $i_{T,\mathcal{P}}$ map $\mathcal{P}$ one-to-one onto $\{1, 2, \ldots, N^{\mathcal{P}}\}$. (This function will identify in decreasing order of preference movie types as viewed by people in the rater type.)

Then, if rater $j \in T$ rates $n_j = \sum_i I[y_{ij} \neq 0]$ movies in $\{i_1^j, i_2^j, \ldots, i_{n_j}^j\} \subset \mathcal{I}$, we'll let

$$S^{\mathcal{P}}(i_l^j) = \text{the element of } \mathcal{P} \text{ containing } i_l^j$$

and suppose that the subset of movies the person rates is selected with probability proportional to

$$\prod_{l=1}^{n_j} w_{i_l^j} (\lambda^{T_j})^{i_{T_j,\mathcal{P}}(S^{\mathcal{P}}(i_l^j))}, \tag{1.1}$$

where T_j denotes the set of raters that includes the rater j. Of interest here are

1. $\mathcal{P}$, the partition of the movie index set $\mathcal{I}$

2. $\mathcal{Q}$, the partition of the rater index set $\mathcal{J}$

and secondarily,

3. the λ^T values

4. the vector of weights $\mathbf{w} = (w_1, w_2, ..., w_I)$, and

5. the ordering of the elements of any $\mathcal{P}$ for a given rater class T represented by $i_{T,\mathcal{P}}(\cdot)$ (this is essentially just a permutation of an index set for the elements of $\mathcal{P}$).

1.2.1 Likelihood

In using expression 1.1, it will be convenient to consider the transforms of λ^T defined by

$$\gamma^T \equiv \ln(\frac{\lambda^T}{1-\lambda^T}).$$

The likelihood function will be written in terms of the γ^T as

$$\begin{aligned} L(\mathcal{P}, \mathcal{Q}, \gamma^T, \mathbf{w}, i_{T,\mathcal{P}}(\cdot)) &= \prod_{j=1}^{J} p_j \\ &\propto \prod_{j=1}^{J}\prod_{l=1}^{n_j} w_{i_l^j}(\lambda^{T_j})^{i_{T_j,\mathcal{P}}(S^{\mathcal{P}}(i_l^j))}, \end{aligned}$$

where

$$\lambda^T = \frac{e^{\gamma^T}}{1+e^{\gamma^T}}.$$

1.2.2 Priors

Let $\Delta_1(\mathcal{P})$ be a prior distribution over partitions of the movie index set $\mathcal{I}$. Let $\Delta_2(\mathcal{Q})$ be a prior distribution over partitions of the rater index set $\mathcal{Q}$. (We could start with uniform priors or perhaps more appropriately, we might make weights on partitions

related to entropy.) The algorithm developed below leaves the choice of these priors completely unconstrained. Further calculation is easy to make when we decide which distributions to use.

We use the following prior distribution for γ^T:

$$\pi_\gamma(\gamma^T|\theta) \propto \prod_{T_i,T_j \text{ are neighbors}} \exp\{-\theta(\gamma^{T_i} - \gamma^{T_j})^2\}, \quad \theta > 0.$$

In order to define "neighbors", we consider the $\mathcal{J}$-vector of 0's and 1's

$$\mathbf{e}^T = (I[1 \in T], I[2 \in T], \ldots, I[J \in T]).$$

Then, T_i and T_j are "neighbors" if $\mathbf{e}^{T_i}$ and $\mathbf{e}^{T_j}$ differ only in one coordinate.

We consider θ as a fixed parameter. One might do sensitivity analysis on θ to see if it affects the results very much.

For each combination of T and $\mathcal{P}$, we will use a uniform distribution on permutations of $\{1, 2, \ldots, N^{\mathcal{P}}\}$ as the prior distribution for $i_{T,\mathcal{P}}(\cdot)$. Denote the joint prior distributions for all the pairs of T and $\mathcal{P}$ as $\pi_i(i_{T,\mathcal{P}}(\cdot))$. Assume $i_{T,\mathcal{P}}(\cdot)$'s are independent for different pairs of T and $\mathcal{P}$. Then, $\pi_i(i_{T,\mathcal{P}}(\cdot))$ is the product of uniform densities for each $i_{T,\mathcal{P}}(\cdot)$.

Also, for simplicity, because $w_1 + w_2 + \cdots + w_I = 1$, it is convenient to use a Dirichlet distribution for the prior distribution of $\mathbf{w}$. Because we do not have any knowledge about the likelihood of a rater watching and rating a particular movie before we see the data, we will use a symmetric Dirichlet distribution, $\text{Dir}(\alpha)$ for a positive real parameter α, as the prior for $\mathbf{w}$. Denote the prior distribution of $\mathbf{w}$ as $\pi(\mathbf{w})$.

According to above modeling structure, the posterior distribution has density

$$p_o(\mathcal{P}, \mathcal{Q}, \mathbf{w}, \gamma^T, i_{T,\mathcal{P}}(\cdot)) \propto \Delta_1(\mathcal{P}) \cdot \Delta_2(\mathcal{Q}) \cdot \pi(\mathbf{w}) \cdot \pi_\gamma(\gamma^T|\theta) \cdot \pi_i(i_{T,\mathcal{P}}(\cdot)) \cdot L(\mathcal{P}, \mathcal{Q}, \mathbf{w}, \gamma^T, i_{T,\mathcal{P}}(\cdot)) \tag{1.2}$$

1.2.3 MCMC algorithm

A possible MCMC algorithm for sampling from the posterior specified by 1.2 is as follows.

(a) Update $\mathcal{P}$.

1. Obtain a proposed partition $\mathcal{P}^*$ from the Du-process transition mechanism (work by Chuanlong Du, to be published.) (See Appendix A for more details.), i.e., sample

$$\mathcal{P}^* \sim J_{\mathrm{Du}}(\mathcal{P}^*|\mathcal{P}^{(s)}),$$

where $\mathcal{P}^{(s)}$ is the current value for $\mathcal{P}$.

2. Compute the acceptance ratio

$$\begin{aligned} r_1 &= \frac{p_o(\mathcal{P}^*, \mathcal{Q}^{(s)}, \mathbf{w}^{(s)}, \gamma^{T(s)}, i^{(s)}_{T,\mathcal{P}^*}(\cdot))}{p_o(\mathcal{P}^{(s)}, \mathcal{Q}^{(s)}, \mathbf{w}^{(s)}, \gamma^{T(s)}, i^{(s)}_{T,\mathcal{P}^{(s)}}(\cdot))} \cdot \frac{J_{\mathrm{Du}}(\mathcal{P}^{(s)}|\mathcal{P}^*)}{J_{\mathrm{Du}}(\mathcal{P}^*|\mathcal{P}^{(s)})} \\ &= \frac{\Delta_1(\mathcal{P}^*)}{\Delta_1(\mathcal{P}^{(s)})} \cdot \frac{L(\mathcal{P}^*, \mathcal{Q}^{(s)}, \mathbf{w}^{(s)}, \gamma^{T(s)}, i^{(s)}_{T,\mathcal{P}^*}(\cdot))}{L(\mathcal{P}^{(s)}, \mathcal{Q}^{(s)}, \mathbf{w}^{(s)}, \gamma^{T(s)}, i^{(s)}_{T,\mathcal{P}^{(s)}}(\cdot))} \\ &= \frac{\Delta_1(\mathcal{P}^*)}{\Delta_1(\mathcal{P}^{(s)})} \cdot \frac{\prod_{j=1}^{J}\prod_{l=1}^{n_j} w^{(s)}_{i^j_l} (\lambda^{T^{(s)}_j})^{i^{(s)}_{T_j,\mathcal{P}^*}(S^{\mathcal{P}^*}(i^j_l))}}{\prod_{j=1}^{J}\prod_{l=1}^{n_j} w^{(s)}_{i^j_l} (\lambda^{T^{(s)}_j})^{i^{(s)}_{T_j,\mathcal{P}^{(s)}}(S^{\mathcal{P}^{(s)}}(i^j_l))}} \\ &= \frac{\Delta_1(\mathcal{P}^*)}{\Delta_1(\mathcal{P}^{(s)})} \cdot \prod_{j=1}^{J}\prod_{l=1}^{n_j} (\lambda^{T^{(s)}_j})^{i^{(s)}_{T_j,\mathcal{P}^*}(S^{\mathcal{P}^*}(i^j_l)) - i^{(s)}_{T_j,\mathcal{P}^{(s)}}(S^{\mathcal{P}^{(s)}}(i^j_l))} \end{aligned}$$

Note that the acceptance ratios in this document are computed in their log form in practice for efficiency.

3. Set $\mathcal{P}^{(s+1)}$ to $\mathcal{P}^*$ or $\mathcal{P}^{(s)}$, the first with probability $\min(1, r_1)$.

(b) Update $\mathcal{Q}$.

1. Obtain a proposed partition $\mathcal{Q}^*$ from the Du-process transition mechanism, i.e., sample

$$\mathcal{Q}^* \sim J_{\mathrm{Du}}(\mathcal{Q}^*|\mathcal{Q}^{(s)}),$$

where $\mathcal{Q}^{(s)}$ is the current value for $\mathcal{Q}$.

2. Compute the acceptance ratio

$$r_2 = \frac{p_o(\mathcal{P}^{(s+1)}, \mathcal{Q}^*, \mathbf{w}^{(s)}, \gamma^{T(s)}, i^{(s)}_{T,\mathcal{P}^{(s+1)}}(\cdot))}{p_o(\mathcal{P}^{(s+1)}, \mathcal{Q}^{(s)}, \mathbf{w}^{(s)}, \gamma^{T(s)}, i^{(s)}_{T,\mathcal{P}^{(s+1)}}(\cdot))} \cdot \frac{J_{\mathrm{Du}}(\mathcal{Q}^{(s)}|\mathcal{Q}^*)}{J_{\mathrm{Du}}(\mathcal{Q}^*|\mathcal{Q}^{(s)})}$$

$$
\begin{aligned}
&= \frac{\Delta_2(\mathcal{Q}^*)}{\Delta_2(\mathcal{Q}^{(s)})} \cdot \frac{L(\mathcal{P}^{(s+1)}, \mathcal{Q}^*, \mathbf{w}^{(s)}, \gamma^{T(s)}, i^{(s)}_{T,\mathcal{P}^{(s+1)}}(\cdot))}{L(\mathcal{P}^{(s+1)}, \mathcal{Q}^{(s)}, \mathbf{w}^{(s)}, \gamma^{T(s)}, i^{(s)}_{T,\mathcal{P}^{(s+1)}}(\cdot))} \\
&= \frac{\Delta_2(\mathcal{Q}^*)}{\Delta_2(\mathcal{Q}^{(s)})} \cdot \frac{\prod_{j=1}^{J}\prod_{l=1}^{n_j} w^{(s)}_{i^j_l} (\lambda^{T^*_j})^{i^{(s)}_{T^*_j,\mathcal{P}^{(s+1)}}(S^{\mathcal{P}^{(s+1)}}(i^j_l))}}{\prod_{j=1}^{J}\prod_{l=1}^{n_j} w^{(s)}_{i^j_l} (\lambda^{T^{(s)}_j})^{i^{(s)}_{T^{(s)}_j,\mathcal{P}^{(s+1)}}(S^{\mathcal{P}^{(s+1)}}(i^j_l))}} \\
&= \frac{\Delta_2(\mathcal{Q}^*)}{\Delta_2(\mathcal{Q}^{(s)})} \cdot \frac{\prod_{j=1}^{J}\prod_{l=1}^{n_j} (\lambda^{T^*_j})^{i^{(s)}_{T^*_j,\mathcal{P}^{(s+1)}}(S^{\mathcal{P}^{(s+1)}}(i^j_l))}}{\prod_{j=1}^{J}\prod_{l=1}^{n_j} (\lambda^{T^{(s)}_j})^{i^{(s)}_{T^{(s)}_j,\mathcal{P}^{(s+1)}}(S^{\mathcal{P}^{(s+1)}}(i^j_l))}}
\end{aligned}
$$

3. Set $\mathcal{Q}^{(s+1)}$ to $\mathcal{Q}^*$ or $\mathcal{Q}^{(s)}$, the first with probability $\min(1, r_2)$.

(c) Update $\mathbf{w}$. The full conditional distribution of $\mathbf{w}$ is

$$
\begin{aligned}
p(\mathbf{w}|\mathcal{P}, \mathcal{Q}, \gamma^T, i_{T,\mathcal{P}}(\cdot), \mathbf{y}) &\propto \prod_{i=1}^{I} w_i^{\alpha-1} \cdot \prod_{j=1}^{J}\prod_{l=1}^{n_j} w^j_{i_l} \\
&\propto \prod_{i=1}^{I} w_i^{\alpha-1} \cdot \prod_{i=1}^{I} w_i^{n_i} \\
&\propto \prod_{i=1}^{I} w_i^{\alpha+n_i-1} \\
&\sim \mathrm{Dir}(\boldsymbol{\beta}),
\end{aligned}
$$

where $\boldsymbol{\beta} = (\alpha + n_1 - 1, \alpha + n_2 - 1, \ldots, \alpha + n_I - 1)'$, where n_i is the number of ratings on movie i. Thus, we want to generate $\mathbf{w}^{(s+1)}$ from $\mathrm{Dir}(\boldsymbol{\beta})$.

(d) Update γ^T.

1. Obtain a new candidate value for γ^T from the proposal density

$$
J_\gamma(\gamma^{T^*}|\gamma^{T^{(s)}}) \sim N(\gamma^{T^{(s)}}, \sigma^2_\gamma).
$$

Denote the proposed value as γ^{T^*}. (J_γ is symmetric, i.e., $J_\gamma(\gamma^{T^*}|\gamma^{T^{(s)}}) = J_\gamma(\gamma^{T^{(s)}}|\gamma^{T^*})$.)

2. Compute the acceptance ratio

$$
r_4 = \frac{p_o(\mathcal{P}^{(s+1)}, \mathcal{Q}^{(s+1)}, \mathbf{w}^{(s+1)}, \gamma^{T*}, i^{(s)}_{T,\mathcal{P}^{(s+1)}}(\cdot))}{p_o(\mathcal{P}^{(s+1)}, \mathcal{Q}^{(s+1)}, \mathbf{w}^{(s+1)}, \gamma^{T(s)}, i^{(s)}_{T,\mathcal{P}^{(s+1)}}(\cdot))}
$$

$$
\begin{aligned}
&= \frac{\pi_\gamma(\gamma^{T^*}|\theta)}{\pi_\gamma(\gamma^{T^{(s)}}|\theta)} \cdot \frac{L(\mathcal{P}^{(s+1)}, \mathcal{Q}^{(s+1)}, \mathbf{w}^{(s+1)}, \gamma^{T*}, i^{(s)}_{T,\mathcal{P}^{(s+1)}}(\cdot))}{L(\mathcal{P}^{(s+1)}, \mathcal{Q}^{(s+1)}, \mathbf{w}^{(s+1)}, \gamma^{T(s)}, i^{(s)}_{T,\mathcal{P}^{(s+1)}}(\cdot))} \\
&= \frac{\prod_{j=1}^{J} \exp\left[-\theta(\gamma^{T^*} - \gamma^{T_j})^2\right]}{\prod_{j=1}^{J} \exp\left[-\theta(\gamma^{T^{(s)}} - \gamma^{T_j})^2\right]} \cdot \frac{\prod_{j\in T}\prod_{l=1}^{n_j} w_{i_l^j} (\lambda^{T^*})^{i^{(s)}_{T,\mathcal{P}^{(s+1)}}(S^{\mathcal{P}^{(s+1)}}(i_l^j))}}{\prod_{j\in T}\prod_{l=1}^{n_j} w_{i_l^j} (\lambda^{T^{(s)}})^{i^{(s)}_{T,\mathcal{P}^{(s+1)}}(S^{\mathcal{P}^{(s+1)}}(i_l^j))}} \\
&= \prod_{j=1}^{J} \exp\left\{-\theta\left[(\gamma^{T^*} - \gamma^{T_j})^2 - (\gamma^{T^{(s)}} - \gamma^{T_j})^2\right]\right\} \\
&\quad \cdot \prod_{j\in T}\prod_{l=1}^{n_j} \left[\frac{\lambda^{T^*}}{\lambda^{T^{(s)}}}\right]^{i^{(s)}_{T,\mathcal{P}^{(s+1)}}(S^{\mathcal{P}^{(s+1)}}(i_l^j))} \\
&= \prod_{j=1}^{J} \exp\left\{-\theta(\gamma^{T^*} - 2\gamma^{T_j} + \gamma^{T^{(s)}})(\gamma^{T^*} - \gamma^{T^{(s)}})\right\} \\
&\quad \cdot \prod_{j\in T}\prod_{l=1}^{n_j} \left[\frac{\lambda^{T^*}}{\lambda^{T^{(s)}}}\right]^{i^{(s)}_{T,\mathcal{P}^{(s+1)}}(S^{\mathcal{P}^{(s+1)}}(i_l^j))},
\end{aligned}
$$

where $T_j's$ are the index sets of raters that are neighbors of T.

3. Set $\gamma^{T^{(s+1)}}$ to γ^{T^*} or $\gamma^{T^{(s)}}$, the first with probability $\min(1, r_4)$.

(e) Update $i_{T,\mathcal{P}^{s+1}}(\cdot)$.

1. Sample $i^*_{T,\mathcal{P}}(\cdot)$ from $J_i\left(i^*_{T,\mathcal{P}^{(s+1)}}(\cdot)|i^{(s)}_{T,\mathcal{P}^{(s+1)}}(\cdot)\right)$, where this jumping kernel specifies random choice of 2 adjacent entries to switch, i.e., switch the k^{th} and $k+1^{st}$ entries only for randomly chosen k, where k is between 1 and $N^{\mathcal{P}} - 1$.

2. Compute the acceptance ratio

$$
\begin{aligned}
r_5 &= \frac{p_o(\mathcal{P}^{(s+1)}, \mathcal{Q}^{(s+1)}, \mathbf{w}^{(s+1)}, \gamma^{T(s+1)}, i^*_{T,\mathcal{P}^{(s+1)}}(\cdot))}{p_o(\mathcal{P}^{(s+1)}, \mathcal{Q}^{(s+1)}, \mathbf{w}^{(s+1)}, \gamma^{T(s+1)}, i^{(s)}_{T,\mathcal{P}^{(s+1)}}(\cdot))} \cdot \frac{J_i\left(i^*_{T,\mathcal{P}^{(s+1)}}(\cdot)|i^{(s)}_{T,\mathcal{P}^{(s+1)}}(\cdot)\right)}{J_i\left(i^{(s)}_{T,\mathcal{P}^{(s+1)}}(\cdot)|i^*_{T,\mathcal{P}^{(s+1)}}(\cdot)\right)} \\
&= \frac{\pi_i(i^*_{T,\mathcal{P}^{(s+1)}}(\cdot))}{\pi_i(i^{(s)}_{T,\mathcal{P}^{(s+1)}}(\cdot))} \cdot \frac{L(\mathcal{P}^{(s+1)}, \mathcal{Q}^{(s+1)}, \mathbf{w}^{(s+1)}, \gamma^{T(s+1)}, i^*_{T,\mathcal{P}^{(s+1)}}(\cdot))}{L(\mathcal{P}^{(s+1)}, \mathcal{Q}^{(s+1)}, \mathbf{w}^{(s+1)}, \gamma^{T(s+1)}, i^{(s)}_{T,\mathcal{P}^{(s+1)}}(\cdot))} \\
&= \frac{\prod_{j\in T}\prod_{l=1}^{n_j} w^{(s+1)}_{i_l^j} (\lambda^{T^{(s+1)}})^{i^*_{T,\mathcal{P}^{(s+1)}}(S^{\mathcal{P}^{(s+1)}}(i_l^j))}}{\prod_{j\in T}\prod_{l=1}^{n_j} w^{(s+1)}_{i_l^j} (\lambda^{T^{(s+1)}})^{i^{(s)}_{T,\mathcal{P}^{(s+1)}}(S^{\mathcal{P}^{(s+1)}}(i_l^j))}} \\
&= (\lambda^{T^{(s+1)}})^{\sum_{j\in T}\sum_{l=1}^{n_j}\left\{i^*_{T,\mathcal{P}^{(s+1)}}(S^{\mathcal{P}^{(s+1)}}(i_l^j)) - i^{(s)}_{T,\mathcal{P}^{(s+1)}}(S^{\mathcal{P}^{(s+1)}}(i_l^j))\right\}}
\end{aligned}
$$

3. Set $i^{(s+1)}_{T,\mathcal{P}^{(s+1)}}(\cdot)$ to $i^{*}_{T,\mathcal{P}^{(s+1)}}(\cdot)$ or $i^{(s)}_{T,\mathcal{P}^{(s+1)}}(\cdot)$, the first with probability $\min(1, r_5)$.

1.2.4 Results and discussion

We started with uniform priors as $\Delta_1(\mathcal{P})$ and $\Delta_2(\mathcal{Q})$. Also, we chose $\theta = 3, \alpha = 1$ and $\sigma^2_\gamma = 1$. These priors and parameter values can easily be changed if more reasonable ones are available in practice. We started with them in order to see if the algorithm developed above works or not. (We could conduct sensitivity analysis on θ and α to check if they have big impact on the results of partitions. Also, σ^2_γ is adjustable to make better proposals for γ^T's.)

There are two series of parameters, γ^T's and $i_{T,\mathcal{P}}(\cdot)$'s that need to be treated specially when implementing the algorithm. Firstly, there are 2^J such γ^T's in the model. Even for a moderately large J, this will be a number much much larger than that of iterations we could obtain. Also, it will not be practical to keep track of all of them in each iteration. Moreover, if T is not a subset of the current $\mathcal{Q}$, and we update its γ^T, we are not going to sample it from a posterior, but only from its prior distribution. Thus, we chose not to update a γ^T unless its T is a subset of the current $\mathcal{Q}$. We did the following when applying the algorithm to the small data set. We initialized all the γ^T's with 0. For the current $\mathcal{Q}$, we checked for each subset of $\mathcal{Q}$, denoted as T, if T was visited in the previous iterations or not. If T has been visited, we found the latest γ^T for that T and used it as our current value for γ^T. If T had not been visited before, the initial value for γ^T was used as the current value. One thing worth noticing is that when we update γ^T, in addition to finding the current value for γ^T, we also need to find the current values for its J neighbors in the same way as we find the current value for γ^T.

Secondly, we find current values for $i_{T,\mathcal{P}}(\cdot)$'s as we do for γ^T's. The reason is as follows. Surely we could generate a new permutation for $i_{T,\mathcal{P}}(\cdot)$ from its uniform prior every time we did not visit $T, \mathcal{P}$ in the last iteration. However, it makes sense that a randomly selected $i_{T,\mathcal{P}}(\cdot)$ will not be as good as the permutation such that the j^{th} position

in the permutation corresponds to the j^{th} movie set ordered decreasingly by the number of ratings of the set T of raters on each movie set. Thus, we initialized $i_{T,\mathcal{P}}(\cdot)$'s by such permutations. Whenever we need to update an $i_{T,\mathcal{P}}(\cdot)$ for the current $\mathcal{P}$ and $T \in \mathcal{Q}$, we checked if $T, \mathcal{P}$ were visited together in previous iterations. If $T, \mathcal{P}$ were visited, we used that corresponding $i_{T,\mathcal{P}}(\cdot)$ as the current value to be updated. Otherwise, we used its initial value as the current one.

The way we find current values for γ^T's and $i_{T,\mathcal{P}}(\cdot)$'s suggests that it would take longer and longer to compute per iteration as the iteration number increases, because the list of previous iterations that we need to check gets longer and longer. This is verified by applying the algorithm on the small data set with only 10 ratings. The trial data set is shown in Table 1.1. This data set was used to test if the MCMC algorithm previously was able to work efficiently or not. The algorithm was coded in `R`. For example, it took several minutes to obtain 1000 iterations; however, it took a few hours to obtain 10000 iterations.

Table 1.1 Trial Data Set for Model 1.4

Rater ID	Movie ID	Rating
1	20	4
1	33	4
1	61	4
1	117	3
1	155	2
2	13	4
2	50	5
2	251	5
2	280	3
2	281	3

The above example on the small data set indicates that it would take even longer to apply the algorithm to a larger real data set. Therefore, improvements need to be made to make this method useful for a real data set. Firstly, we may simplify the model such

that the number of parameters is within the range that we can keep track of. Secondly, we may think of a better way to obtain current values of γ^T's and $i_{T,\mathcal{P}}(\cdot)$'s instead of checking previous iterations. We may also need to change the model. Finally, we could treat this problem as an optimization problem in the sense that we find the partitions for movies and raters such that they maximize the posterior density. We could restart the algorithm every small number of iterations such that it would not take too long to obtain the iterations and we would use the most updated parameter values as the starting values to obtain a new set of iterations. We would eventually have many "small" sets of iterations for each of which we would keep track of the posterior density value. Then, we would be able to find a set of partitions for movies and raters such that their posterior probability is the largest. In this way, we may be able to get close to the posterior mode. Other optimization algorithms may also be helpful in finding a posterior mode.

In summary, the MCMC algorithm developed for the model for biclustering the movie rating data works but is very slow given current computation power. We need to make improvements in the model, the algorithm, the programming or the computational power in order for this new methodology to be useful for a real data set.

1.3 Modeling Movie Ratings on a 5-Star Scale

Because the initial model has too many parameters, we now look at the problem from another perspective, and try to construct a model that is more computationally feasible. Let r_{ij} be the rating that movie i gets from rater j. The movie rating was actually done on a 5 star scale, and thus, r_{ij} can take values of $1, 2, 3, 4$ and 5. If movie i was not rated by rater j, then r_{ij} takes a missing value and is here set to 0. Note that r_{ij} is not "real-valued" in the sense that the scores $1, 2, 3, 4, 5$ are probably not "evenly spaced", i.e., are only ordinal-level values. Let $\tilde{r}_{ij}$ be an unobserved real rating score that movie

i would get from rater j. That is, assume

$$r_{ij} = \begin{cases} 1 & \text{if } \tilde{r}_{ij} \in [0.5, 1.5) \\ 2 & \text{if } \tilde{r}_{ij} \in [1.5, 2.5) \\ 3 & \text{if } \tilde{r}_{ij} \in [2.5, 3.5) \\ 4 & \text{if } \tilde{r}_{ij} \in [3.5, 4.5) \\ 5 & \text{if } \tilde{r}_{ij} \in [4.5, 5.5) \end{cases}$$

As before, suppose further that the movie index set $\mathcal{I} = \{1, 2, \ldots, I\}$ is partitioned into mutually exclusive and exhaustive sets $S_1, S_2, \ldots, S_{N^{\mathcal{P}}}$ and that the rater index set $\mathcal{J} = \{1, 2, \ldots, I\}$ is partitioned into mutually exclusive and exhaustive sets $T_1, T_2, \ldots, T_{M^{\mathcal{Q}}}$. Denote the partition of the movie index set as $\mathcal{P}$; the partition of the rater index set as $\mathcal{Q}$.

Let $\bar{r}_{S_i,T_j}$ be the sample mean of the available rating scores for all the raters in the subset of T_j to rate movies in the subset of S_i. Note that $\bar{r}_{S_i,T_j}$ is between 1 and 5 and may not be an integer.

Then, it is perhaps reasonable to model $\tilde{r}_{ij}$ as

$$\tilde{r}_{ij} \ \sim \ \text{Truncated Normal}(\mu_{S_i,T_j}, \sigma^2), \ \tilde{r}_{ij} \in [0.5, 5.5], \tag{1.3}$$

where σ^2 is the variance and a tuning parameter of the model and μ_{S_i,T_j} is the mean (before truncation) rating score for all the raters in the subset T_j to rate movies in the subset S_i. μ_{S_i,T_j} can be estimated as $\bar{r}_{S_i,T_j}$. Taking $\bar{r}_{S_i,T_j}$ as μ_{S_i,T_j}, a likelihood function might be

$$L(\mathcal{P}, \mathcal{Q}) = \prod_{r_{ij} \neq 0} F_{TN}\left(\frac{r_{ij} + 0.5 - \bar{r}_{S_i,T_j}}{\sqrt{\sigma^2}}\right) - F_{TN}\left(\frac{r_{ij} - 0.5 - \bar{r}_{S_i,T_j}}{\sqrt{\sigma^2}}\right),$$

where $F_{TN}(x)$ is the truncated standard normal cdf for which x ranges from $\frac{0.5-\bar{r}_{S_i,T_j}}{\sigma}$ to $\frac{5.5-\bar{r}_{S_i,T_j}}{\sigma}$. As before, we let $\Delta_1(\mathcal{P})$ be a prior distribution over partitions of the movie index set $\mathcal{I}$. Let $\Delta_2(\mathcal{Q})$ be a prior distribution over partitions of the rater index set $\mathcal{Q}$.

Then, the posterior distribution has density

$$p_o(\mathcal{P}, \mathcal{Q}) \propto \Delta_1(\mathcal{P}) \cdot \Delta_2(\mathcal{Q}) \cdot L(\mathcal{P}, \mathcal{Q}). \tag{1.4}$$

Although we could get posterior samples, it would not be very useful for two reasons. Firstly, it is difficult to tell if an MCMC chain for partitions converges to stationarity or not. Secondly, with the large partition space we will have, it is likely not possible to visit enough of it to get a good posterior sample to represent the distribution. Thus, in order to make a decision on the point estimate of partitions, we could choose an objective function such as the likelihood function penalized with a complexity function of the number of parameters in the model. By optimizing the objective function, we can in theory get a point estimate of the partitions.

The space consisting of all combinations of possible partitions for users and for items, is very large and therefore, it is important to choose an algorithm that is able to explore the space in an efficient way. Genetic Algorithms (GAs) are known for the capability to search in a complex and large space. They are stochastic search and optimization methods inspired by principles of evolution and genetics. See Michalewicz (1996).

1.3.1 Genetic algorithms

Genetic Algorithms work with a population of individuals which represent solutions to an optimization problem. Each individual has a fitness value which represents a measure of how good the solution is. The population of solutions evolves towards a better population. In each generation, the population undergoes mutation and crossover to generate a new population.

The following is a sketch of the basic structure of a general genetic algorithm.

1. $i = 0$, select M individuals as the initial population P_0.

2. Evaluate fitness for individuals from P_i.

3. If the stop criterion is satisfied, stop and return the best individual.
4. Let P_{i+1} be an empty set.
5. Select a pair of individuals I_1, I_2 from P_i by some selection scheme.
6. With probability p_c, cross over I_1 and I_2 to get I_1^* and I_2^*.
7. With probability p_m, I_1^* and I_2^* undergo mutation to form I_1^{**} and I_2^{**}
8. Include I_1^{**} and I_2^{**} into P_{i+1}.
9. If P_{i+1} has less than M individuals, goto 5.
10. $i = i + 1$ and goto 2.

The technical meanings of "fitness", "selection", "crossover" and "mutation" are problem specific

1.3.2 Clustering using genetic algorithms

GAs for clustering are based on the framework above. To implement a GA for clustering, we need to make decisions on the representation of the problem, the selection scheme, crossover and mutation operators and probabilities, and the size of the population. They are described in the following.

1.3.2.1 Representation

Each individual represents one solution to the biclustering problem. That is, each individual consists of a partition of users and a partition of items. There are some possible ways of representing partitions in the literature of GAs Michalewicz (1996), where all use a string of integers as a representation of partitions. Here, for the ease of implementing the crossover operator that we designed, we chose to use a binary matrix to represent a partition. Suppose $\mathcal{P}$ is a partition for N movies/items and has K clusters/elements,

then, $\mathcal{P}$ will be represented as an $N \times K$ matrix, for which the ij^{th} entry is 1 if item i is in the j^{th} subset of $\mathcal{P}$ and 0 otherwise.

1.3.2.2 Selection

A number of selection schemes are available in the literature. The most widely used selection mechanisms are the *roulette-wheel selection* and *tournament selections*. We chose to use the *deterministic tournament selection* that randomly choose two individuals/solutions from the population and then selects the one with the better fitness value.

1.3.2.3 Crossover

Suppose we have two partitions $\mathcal{P}_1$ and $\mathcal{P}_2$ with K_1 and K_2 clusters. Then, the following are the steps to obtain a pair of "child" partitions.

1. Randomly select c_1 and c_2 such that $c_1 \in \{1, \ldots, K_1\}$ and $c_2 \in \{1, \ldots, K_2\}$.

2. Suppose $S_{c_1}^{\mathcal{P}_1}$ is the subset of $\mathcal{P}_1$ numbered with c_1 and $S_{c_2}^{\mathcal{P}_2}$ is the subset of $\mathcal{P}_2$ numbered with c_2. Then, under the current representation, $S_{c_1}^{\mathcal{P}_1}$ and $S_{c_2}^{\mathcal{P}_2}$ are length N vectors of 0's and 1's. Compute ${S_{c_1}^{\mathcal{P}_1}}^T S_k^{\mathcal{P}_2}$ for $k = 1, \ldots, K_2$, then, find

$$k_{c_1} = \underset{k}{\text{argmax}} {S_{c_1}^{\mathcal{P}_1}}^T S_k^{\mathcal{P}_2}.$$

   Find k_{c_2} in a similar way.

3. Then, let $S_{c_1}^* \equiv S_{c_1} \cup S_{k_{c_1}}$. Let $\bar{S}_{c_1} \equiv S_{c_1}^* \backslash S_{c_1}$. Create $\mathcal{P}_1^*$ based on $\mathcal{P}_1$ such that for $k = 1, \ldots, K_1$

$$S_k = \begin{cases} S_{c_1}^* & \text{if } k = c_1 \\ S_k^{\mathcal{P}_1} \backslash \{S_k^{\mathcal{P}_1} \cap \bar{S}_{c_1}\} & \text{Otherwise} \end{cases}$$

   Similarly, $\mathcal{P}_2^*$ can be obtained.

4. Delete empty sets in $\mathcal{P}_1^*$ and $\mathcal{P}_2^*$ if there is any. Now we have the child partitions $\mathcal{P}_1^*$ and $\mathcal{P}_2^*$.

1.3.2.4 Mutation

Du-Process is used as the mutation scheme. In particular, for the simulation study, one-step Du-Process is used.

1.3.2.5 Fitness

The fitness function measures how good an individual is in the problem. Under our statistical model framework for the biclustering problem, it is reasonable to choose the log likelihood as our objective function. It is also natural to penalize the log likelihood with the complexity, which is a function of the number of parameters in the model. The following is a version of possible fitness function. Other versions may be considered in other situations, however, the following works well for the simulation study later. The fitness function we will use is

$$f(\mathcal{P}, \mathcal{Q}) = -2 \ln L(\mathcal{P}, \mathcal{Q}) + \frac{1}{2} K_{\mathcal{P}} \cdot K_{\mathcal{Q}} \cdot \ln(N),$$

where $\mathcal{P}$ and $\mathcal{Q}$ are the partitions for items and users respectively; $K_{\mathcal{P}}$ and $K_{\mathcal{Q}}$ are the number of subsets in $\mathcal{P}$ and $\mathcal{Q}$ respectively; N is the number of ratings in the data set.

1.3.3 Simulation study

1.3.3.1 A simulation without missing values

In this simulation study, we simulated a data set from model 1.3. Let $\mathcal{P}$ and $\mathcal{Q}$ be the true partitions from which the data set will be generated. Here, $\mathcal{P}$ is taken to be the following partition for 15 items. For simplicity, we write in terms of indices of items/movies. In our computation, we used the binary matrix representation form for partitions. Here, $\mathcal{P} = \{\{1,2,3\},\{4,5,6\},\{7,8,9\},\{10,11,12\},\{13,14,15\}\}$; $\mathcal{Q} = \{\{1,2,3\},\{4,5,6\},\{7,8,9,10\}\}$. We chose σ to be 0.5 in this case. The matrix for the mean values is shown in Table 1.2. The simulated data set is shown in Figure 1.1, where ratings were shown in five different colors indicating five levels of ratings.

We chose the crossover and mutation probabilities as 0.8 and 0.2 respectively; we also chose the size of population to be $M = 6$. For the initial population, we first chose the partitions with singletons as subsets, then, we used the Du-process to generate other individuals in the initial population. The reason that we chose the initial population this way is that the crossover operator generates a child partition that have less than or equal to the number of clusters of the parent based on which the child partition is generated.

Then, we applied the GA for this data set. After 1000 iterations/generations, the population converges to the "true" partitions from which the data set was generated. The fitness value for the best individual in each generation was plotted in Figure 1.2. The above implementation was done twice for different starting populations. The results are the same for the two implementations.

Table 1.2 Matrix of Pre-Truncation Means for Simulation

	S_1	S_2	S_3	S_4	S_5
T_1	1	2	3	4	5
T_2	5	4	3	2	1
T_3	2	3	1	4	5

1.3.4 Conclusions and discussion

Note that the simulated data set contains $15 \times 10 = 150$ ratings, which is still a relatively small number. For larger data sets, we found that the GA proposed here is still not fast enough to find a global minimum for the fitness function. This is because of the problem is NP-hard, which was mentioned in Busygin et al. (2008). NP-hard is a concept in computation complexity theory. A problem A is NP-hard if every problem B in NP can be reduced in polynomial time to A, or intuitively speaking, an NP-hard problem is at least as complex as those hardest problems in NP. Therefore, it is unrealistic to look

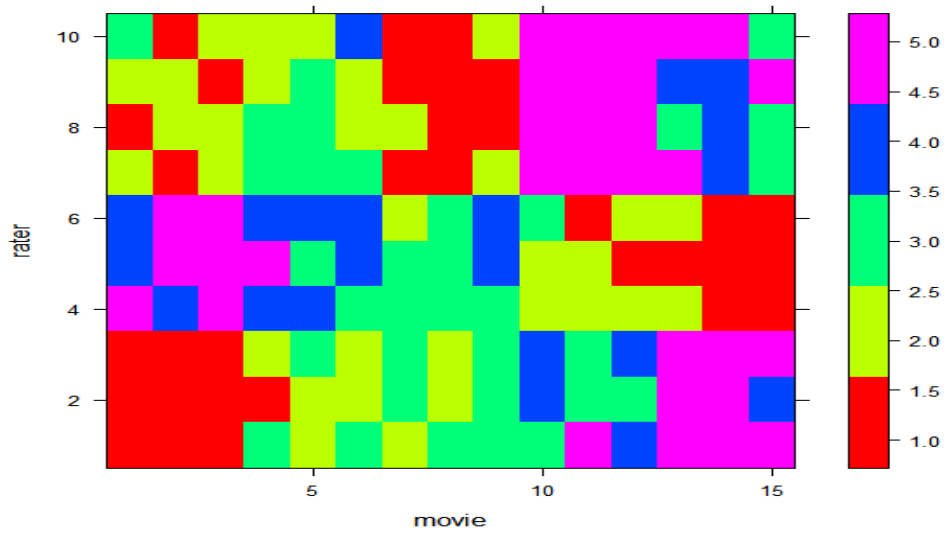

Figure 1.1 Simulated Data Set

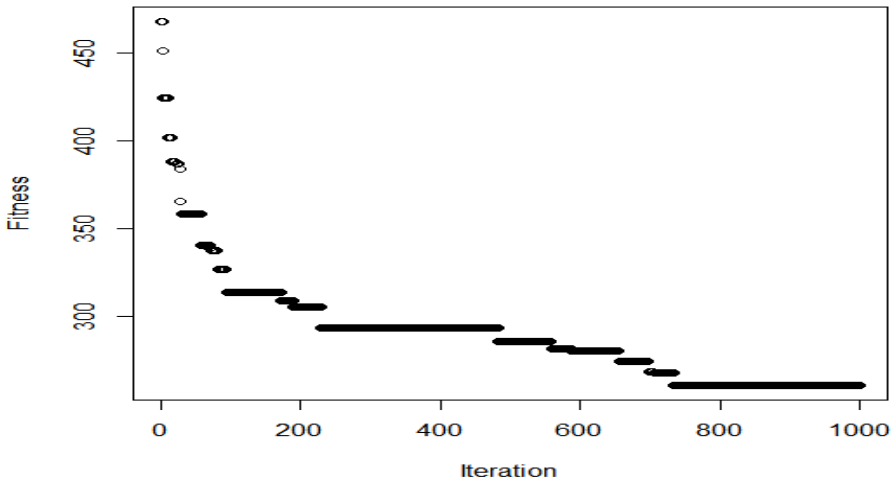

Figure 1.2 Fitness Values for Best Individuals

for a global optimum for these problems when the sample size is moderately large. Here in the context of our biclustering problem, the total number of partitions of an n-element set is the Bell number B_n. Thus, for the simulated data set, the problem size is 15×10, and therefore, the total number of partitions is $B_{10} \times B_{15} = 1.6 \times 10^{14}$. And we found the true solution within 1500×6 checks, which means we only checked $\frac{1500 \times 6}{1.6 \times 10^{14}} = 5.625 \times 10^{-11}$ part of the whole solution space. The GA is efficient in this sense, however, for a larger problem, for instance, if we consider a problem with sample size 150×100, we will need to find an optimum in a space of partitions of size $B_{100} \times B_{150} \approx 10^{308.5}$, which is infeasible with current computational power. Therefore, by the complexity of the problem, we stop looking for a global solution and will look for an ad-hoc algorithm that finds local optimization solutions. We propose and look into a prototype-based biclustering method in detail in the next chapter.

CHAPTER 2. A PROTOTYPE-BASED TWO-WAY CLUSTERING METHOD

A paper to be submitted

Jing Li

Stephen B. Vardeman

Department of Statistics

Iowa State University

Ames, IA, 50010

Abstract

Biclustering is a statistical learning technique that simultaneously partitions the set of samples and a set of their attributes into subsets. Biclustering is known to be an NP-hard problem. Therefore, various heuristic approaches have been pursued in literature. Motivated by movie rating data, where missing values need to be addressed, we propose a new prototype-based Biclustering method. We evaluated our method on test cases with various percentages of missing values in terms of the Rand Index between our result and the "true" partitions. Our method has good performance on test cases even with large missing-value percentage. We further evaluate our method on a gene expression data set with no missing values. Our method outperforms a competing biclustering method from the literature, i.e., spectral biclustering, in terms of a Mean Squared Error criterion.

Keywords. statistical learning, biclustering, mean squared error criterion, prototype-based, Rand Index

2.1 Introduction

Biclustering is a statistical learning technique that simultaneously partitions a set of samples and the set of their attributes into subsets. In contrast, a traditional one-way clustering approach only partitions the set of samples. There is a large literature on biclustering methods, most of which are heuristic approaches. The reason that researchers take a heuristic approach is that biclustering problem is known to be NP-hard (see Busygin et al. (2008)). Therefore, it is unrealistic to search for a global optimum for this problem. We can illustrate the complexity of this problem by calculating the number of possible solutions of a biclustering problem. For instance, suppose we have a data set of size 150 by 100, i.e., there are 150 samples and 100 attributes to be partitioned simultaneously. The size of this data set is only moderate by current standards. The total number of partitions of an $n-$element set is the Bell number B_n. Then this example of dimensions 150 by 100, the total number of partitions is $B_{100} \times B_{150} = 1.6 \times 10^{308.5}$, which is infeasible to search directly with current computation power. Therefore, the NP-hard level complexity of this problem demands that we look for a heuristic approach for finding good biclusterings.

There are a large number of applications of biclustering. The review of biclustering in data mining by Busygin et al. (2008) points to applications in many fields such as biomedicine, text mining and marketing.

In biomedicine, microarray data analysis is a very important application for biclustering. The objective of biclustering a microarray data set is not only to identify similar genes, but also to identify similar conditions in which a sample of a gene was taken. We will provide an implementation of our proposed algorithm on a cancer microarray data

set later in this paper. Biclustering has been applied in other biomedical problems. For example, in Liu et al. (2003), biclustering was used in the analysis of drug activity data in order to cluster chemical compounds and their features simultaneously. In Lazzeroni et al. (2002), biclustering was used in the analysis of nutritional data, where foods and their nutritional attributes were clustered simultaneously.

In marketing, biclustering is used to simultaneously partition customers into groups and products into groups. For example, for each rating score that is available in the famous Netflix problem (see Toscher et al. (2009)), the corresponding rater and movie are known. The goal of a biclustering algorithm on a movie rating data set is to obtain both rater groups, in each of which raters are similar in their overall rating profile on each movie group, and movie groups, in each of which movies are similar in their rating profile by each rater group. In fact, the approach presented in this paper was initially motivated by the famous Netflix movie rating data. Note that such data sets usually have a high percentage of missing values, because usually a rater does not rate all movies in the list. In fact, in the movie rating data sets that are available to us, the percentage of missing values is at least 94%. Therefore, we test the performance of our proposed algorithm on simulated test data sets with missing values.

Although we defined the term biclustering briefly at the beginning of this section, biclustering is a very broad term. That is, the partitioning results of various biclustering algorithms in literature can be quite different in nature from one to another (Madeira et al. (2004)). For example, in Kluger et al. (2003), the authors proposed a Spectral Biclustering method such that the biclustering result is actually a result of rearranging the raw data matrix as illustrated in Figure 2.1. Suppose the rows i denote genes and columns j denote conditions (e.g., patients or cell types). Suppose the colour of each entry denotes the gene expression level for the corresponding gene and condition. The matrix on the left side is the raw data, which we denote by X. After row and column rearranging, we have the result matrix on the right side in a checkerboard-like structure

with blocks of various gene expression levels. Note that both genes and conditions were partitioned into disjoint subsets. There exist other types of biclustering algorithms. For instance, the biclustering approach proposed in Cheng et al. (2000) seeks to divide a gene expression data matrix X into a pre-specified number of "biclusters", which are not necessarily disjoint. However, in this paper, we pursue a biclustering approach that produces results with checkerboard structure.

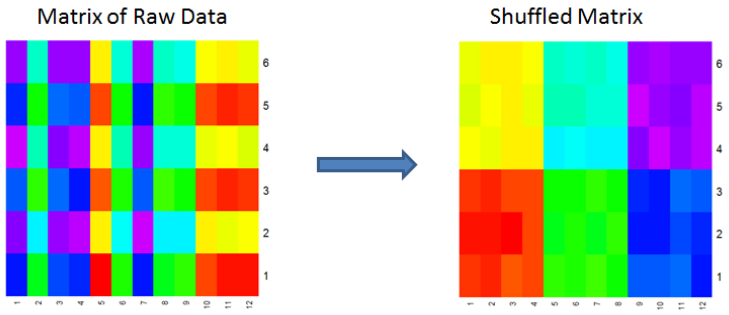

Figure 2.1 Biclustering with "Checkerboard" Structure Results

2.2 A New Biclustering Algorithm

In this section, we present the new biclustering algorithm. Because this approach was initially motivated by movie rating data like the famous Netflix data, we here discuss our algorithm in terms of "movies" and "raters." Suppose X is the movie rating data set for biclustering. Let raters index rows and movies index columns. Suppose further that the movie index set $\mathcal{J} = \{1, 2, \ldots, J\}$ is partitioned into mutually exclusive and exhaustive sets $S_1, S_2, \ldots, S_{N^{\mathcal{P}}}$ and that the rater index set $\mathcal{I} = \{1, 2, \ldots, I\}$ is partitioned into mutually exclusive and exhaustive sets $T_1, T_2, \ldots, T_{M^{\mathcal{Q}}}$. Denote the partition of the movie index set as $\mathcal{P}$; the partition of the rater index set as $\mathcal{Q}$. Let X_{ij} be the rating score of movie j by rater i, for $i \in \{1, 2, \ldots, I\}$ and $j \in \{1, 2, \ldots, J\}$.

As we mentioned in Section 2.1, our goal of biclustering is to generate a rearranged data matrix with a checkerboard structure such that each block of the matrix is as coherent as possible, where we need to define what we mean by block being "coherent." Depending on the purpose in a real problem, "coherent" can have different meanings, and hence different objective functions to be optimized can be appropriate. For illustration purposes, we present our algorithm here with the goal of optimizing a within-cluster sum of squares (WCSS) given both the number of row groups and that of column groups. That is, suppose after rearranging the rows and columns, we obtain a result matrix as illustrated on the right side of Figure 2.1. Using the above notations, we have $M^{\mathcal{Q}}$ row groups and $N^{\mathcal{P}}$ column groups, i.e., $M^{\mathcal{Q}}$ groups for raters and $N^{\mathcal{P}}$ groups for movies. Then, let A denote the prototype matrix with entry

$$A_{ij} \equiv \frac{1}{|\{X_{i'j'} : \ i' \in T_i; \ j' \in S_j; \ X_{i'j'} \neq NA\}|} \sum_{\{X_{i'j'}: \ i' \in T_i; \ j' \in S_j; \ X_{i'j'} \neq NA\}} X_{i'j'}$$

, for $i \in 1, 2, \ldots, M^{\mathcal{Q}}$ and $j \in 1, 2, \ldots, N^{\mathcal{P}}$, where $|(\cdot)|$ is the cardinal number function and missing values are denoted as "NA." Then, the WCSS function to be minimized is

$$\text{WCSS} \equiv \sum_{\substack{X_{i'j'} \neq NA; \\ i' \in \{1,2,\ldots,I\}; \\ j' \in \{1,2,\ldots,J\}}} (X_{i'j'} - A_{ij})^2 \tag{2.1}$$

Our biclustering algorithm is the following. For simplification of notation purpose, let $a \equiv M^{\mathcal{Q}}$ and $b \equiv N^{\mathcal{P}}$. (Note that our algorithm already accommodates data sets with missing values by ignoring them in the computation.)

1. Select prototypes for raters $P_{u_1}, P_{u_2}, \ldots, P_{u_a}$ such that $P_{u_i} \in \mathcal{U}$ for $i = 1, 2, \ldots, a$, where $\mathcal{U}$ is the set of rating records for all raters. Each element of $\mathcal{U}$ is a vector of length J, which is the total number of movies. The number of elements in $\mathcal{U}$ is I, which is the total number of raters.

2. Select prototypes for movies $P_{i_1}, P_{i_2}, \ldots, P_{i_b}$ such that $P_{i_j} \in \mathcal{M}$ for $j = 1, 2, \ldots, b$, where $\mathcal{M}$ is the set of rating records for all movies. Each element of $\mathcal{M}$ is a vector

of length I, which is the total number of raters. The number of elements in $\mathcal{M}$ is J, which is the total number of movies.

3. For given a and b, let the two-way prototype table be denoted as A, where $A_{ij} \equiv P_{u_i}P_{i_j}$, where $P_{u_i}P_{i_j}$ is the rating of rater prototype i on movie prototype j. This is for the initial calculation of A_{ij}. For the next iterations, A_{ij} is the block mean of rater group i on the movie group j ignoring missing values as calculated above. If all scores corresponding to a rater group and a movie group are missing, we must choose an arbitrary value to use in place of a cell mean. For the movie rating data, because the rating scores ranges from 1 up to 5, we chose 3 because a score 3 means the rater does not like or dislike a movie very much. When no data is available for a rater group's opinion on a movie group, it seems reasonable to assume the rater group does not like or dislike the movie group very much.

4. Initial partitions for movies and raters are generated randomly with pre-specified numbers of groups.

5. Set $s = 0$. We start with $\mathcal{P}^{(s)} = \mathcal{P}^{(0)}$ and $\mathcal{Q}^{(s)} = \mathcal{Q}^{(0)}$.

6. For a row $u_{i'}$, which corresponds to rater i', for $i' = 1, 2, \ldots, I$, where I is the number of raters, find the distance between the prototype P_{u_i} and rater i' defined as

$$d(P_{u_i}, u_{i'}) \equiv \sum_{j=1}^{b} (A_{ij} - \bar{u}_{i'j})^2 \cdot |\{l : l \in Q_j\}|, \tag{2.2}$$

where $\bar{u}_{i'j} = \frac{1}{|Q_j|} \sum_{\substack{l \in Q_j \\ u_{i'l} \neq NA}} u_{i'l}$. Find the prototype $P_u(u_{i'})$ such that

$$P_u(u_{i'}) \equiv \underset{P_{u_i}}{\operatorname{argmin}}\, d(P_{u_i}, u_{i'}).$$

Move $u_{i'}$ to $P_{P_u(u_{i'})}$, i.e., the subset of $\mathcal{P}$ including prototype $P_u(u_{i'})$.

7. Do step 6 for column $i_{j'}$, for $j' = 1, 2, \ldots, J$, where J is the number of movies, then we will end up with new partitions $\mathcal{P}^{(s+1)}$ and $\mathcal{Q}^{(s+1)}$.

8. Do steps 6 and 7 S times or until the memberships of clusters do not change, i.e., the algorithm converges.

2.3 Results

In order to assess the performance of the proposed biclustering algorithm, we firstly applied it to simulated data. These results will be presented in Section 2.3.1. Then, we applied the algorithm on a real microarray data set. Then, we compared our biclustering result with that obtained using the Spectral Biclustering method Kluger et al. (2003), which, as mentioned in Section 2.1, is a benchmark approach that seeks to generate biclustering results with a checkerboard structure. The comparison criterion used here is Mean Squared Error (MSE)

$$\text{MSE} \equiv \frac{1}{N}\text{WCSS}, \tag{2.3}$$

where WCSS is defined as in Equation 2.1; $N \equiv I \cdot J$, where I is the number of rows of the data matrix and J the number of columns. For data sets of the same dimensions, this MSE criterion is equivalent with the WCSS criterion. We can see that MSE= 0 for trivial or constant biclusters where each entry forms one bicluster. These trivial biclusters are usually not very interesting. We will ignore these trivial biclusters by choosing the number of groups for rows and columns respectively.

2.3.1 Performance on simulated data

Our goal is to compare the biclustering results on simulated data sets with different levels of missing values, thus, we at first simulate a test case with no missing values and then create patterns of missingness in principled ways.

Test cases were generated in the following way. We firstly needed to decide the numbers of prototypes/groups of movies and raters. Suppose we decide to look into a test case with 5 groups of movies and 5 groups of raters. Thus, the prototype matrix

is 5 by 5, where the rows correspond to the rater groups and columns to movie groups. Each entry A_{ij} in the prototype matrix A corresponds to the rating score of rater group i on movie group j. Then, we will randomly select from $\{1, 2, 3, 4, 5\}$ as the value for each cell in the prototype matrix.

Firstly, we tested the performance of the biclustering algorithm with test cases derived from a fixed prototype matrix. For example, we generated a 5 by 5 prototype matrix in Table 2.1. Then, based on this prototype matrix, we firstly obtain the data set with no missing values by expanding the prototype matrix to a 100×150 data set. We here assume equal group sizes for the raters and for the movies. That is, if rater $i' \in T_i$, a rater group, and movie $j' \in S_j$, a movie group, then, let $X_{i'j'} = A_{ij}$. Then, we obtained the data matrix X with no missing values.

Next, we also generated data sets with missing values at $10\%, 50\%$ and 90%. Here, we assume that the probability of a rating score is not missing is proportional to the score value. This is a reasonable assumption because a movie with a higher rating score would usually get watched by more people and hence more ratings. Let k be a rating score in the data set with no missing values, for $k = 1, 2, 3, 4, 5$. Let $\Pr(k) \propto k$ be the probability that the score k is not missing in the data set. Then, we could solve for each total missing percentage, i.e., $10\%, 50\%$ and 90%, the values of $\Pr(k)$ for $k = 1, 2, 3, 4, 5$. We simulated data sets with missing values $S = 1000$ times. Then, we implemented our biclustering algorithm on each simulated data set and also on the data set with no missing values.

In order to compare the biclustering results with the partitions for rows and columns that actually generated the test case, we used the Rand Index Rand (1971) to measure the similarity between the "true" clustering and the result of applying our algorithm. The definition of an RI follows. Given a set of n elements $S = \{o_1, \ldots, o_n\}$ and two partitions of S to compare, $P = \{P_1, \ldots, P_r\}$, a partition of S into r subsets, and $Q = \{Q_1, \ldots, Q_t\}$, a partition of S into t subsets, define the following:

1. a = the number of pairs of elements in S that are in the same set in P and in the same set in Q

2. b = the number of pairs of elements in S that are in different sets in P and in different sets in Q

3. c = the number of pairs of elements in S that are in the same set in P and in different sets in Q

4. d = the number of pairs of elements in S that are in different sets in P and in the same set in Q

Rand Index is defined as RI $= \frac{a+b}{a+b+c+d}$. Based on this definition, we can see that RI $\in [0, 1]$, with 0 indicating that the two clusterings do not agree on any pair of points and 1 indicating that the clusterings are exactly the same.

The summary and histograms of Rand Indices between the true row clusterings and row clustering results from our biclustering algorithm are shown in Table 2.2 and Figure 2.2 for data simulated with one common prototype matrix with no missing values or $10\%, 50\%$ and 90% missing values. The results related to the columns are shown in Table 2.3 and Figure 2.3. We can see that in general, our algorithm has good performance because most of the Rand Indices between the true clusterings and our results are close to 1. It is worth of noticing that the performance of our algorithm seems to be the best in terms of Rand Index when the missing percentage is 10% or 50%. The performance of our algorithm seems to be not as good when there were no missing values in the simulated data. This is because our algorithm is a heuristic approach and hence, it seeks a local optimum. In our test case construction, it is easy to have prototypes that are very similar. For example, the first two columns of the prototype matrix in Table 2.1 only differs at the fourth entry. Therefore, in the non-missing case the algorithm easily converged to a local optimum. In fact, the non-missing cases converged within fewer iterations than missing ones did. Similar observations can be found for results obtained by

implementing our algorithm on data sets simulated with a random prototype matrix at each run. The results are shown in Table 2.4 and 2.5; Figure 2.4 and 2.5. In addition to the above observations, we can also see that for all the results shown here, our algorithm performs the worst when the missing percentage is at its highest, 90%. This is common for almost all the clustering algorithms and is completely reasonable.

We have seen that our algorithm has good performance on simulated data sets marginally. That is, the row clustering results and column clustering results well match the true row and column clusterings respectively. Lastly, in order to see how well our algorithm performs jointly, we computed the percentage of perfect reconstructions, shown in Table 2.6. We can see that in general our algorithm has good performance in that starting from totally random initial prototypes, our biclustering results matched the original two-way grouping perfectly for over 90% of cases when the data sets, with 10% or 50% missing values, were simulated using a common prototype matrix; over 80% if simulated using random prototype matrices. Also, as we observed previously for the algorithm on data marginally, it performs not as well when the simulated data sets have no missing values.

We observed missing values slowed down the algorithm computationally. In addition, we found that at 90% missing percentage, the perfect reconstruction rate for data simulated with one common prototype matrix in Table 2.1 was lower than if simulated with random prototype matrices. As we discussed previously, this is probably because it is easier for the algorithm to converge to a local optimum when data was simulated using our chosen prototype matrix in Table 2.1.

2.3.2 Results on a real microarray data set

In order to see the performance of the proposed algorithm on a real data set, we firstly tried the algorithm on a movie rating data set with 100,000 ratings, in which 94% of data were missing. The data set has 943 raters and 1682 movies. We found the

Table 2.1 Example of a Prototype Matrix

1	1	1	4	5
3	3	5	5	2
4	4	3	4	3
3	2	4	4	1
5	5	3	5	1

Table 2.2 RI Summary Table for Clustering Results of Rows for Data Simulated with One Common Prototype Matrix

	Missing Percentage			
	0%	10%	50%	90%
Min	0.76	0.92	0.92	0.40
$1^{st}Q$	0.92	1	1	0.98
Median	1	1	1	1
Mean	0.96	1	1	0.96
$3^{rd}Q$	1	1	1	1
Max	1	1	1	1
Stdev	0.06	0.01	0.004	0.11

Table 2.3 RI Summary Table for Clustering Results of Columns for Data Simulated with One Common Prototype Matrix

	Missing Percentage			
	0%	10%	50%	90%
Min	0.76	0.86	0.92	0.67
$1^{st}Q$	0.92	1	1	0.95
Median	1	1	1	0.96
Mean	0.94	1	1	0.95
$3^{rd}Q$	1	1	1	0.97
Max	1	1	1	0.97
Stdev	0.10	0.02	0.01	0.04

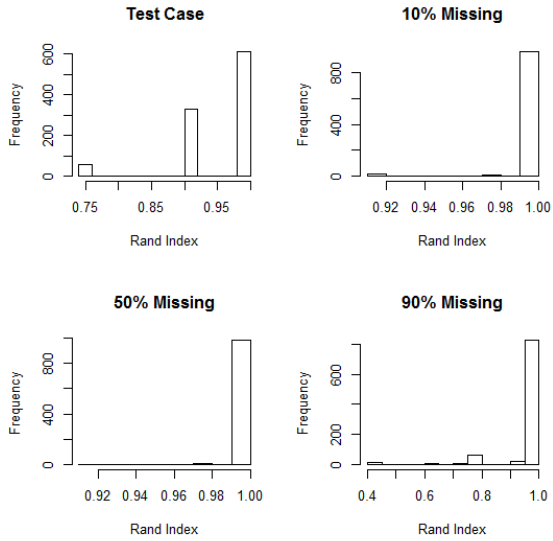

Figure 2.2 RI Histograms for Row Partitioning for Data Simulated with One Common Prototype Matrix

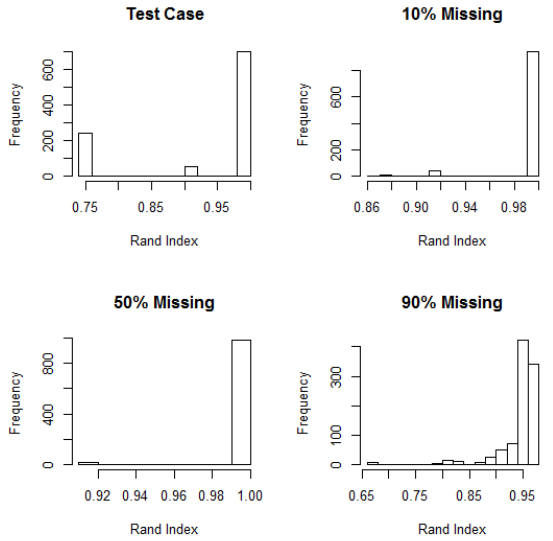

Figure 2.3 RI Histograms for Column Partitioning for Data Simulated with One Common Prototype Matrix

Table 2.4 RI Summary Table for Clustering Results of Rows for Data Simulated with Random Prototype Matrices

	Missing Percentage			
	0%	10%	50%	90%
Min	0.19	0.19	0.19	0.19
$1^{st}Q$	0.92	1	1	0.55
Median	1	1	1	0.92
Mean	0.92	0.96	0.94	0.76
$3^{rd}Q$	1	1	1	1
Max	1	1	1	1
Stdev	0.17	0.15	0.18	0.30

Table 2.5 RI Summary Table for Clustering Results of Columns for Data Simulated with Random Prototype Matrices

	Missing Percentage			
	0%	10%	50%	90%
Min	0.19	0.19	0.19	0.19
$1^{st}Q$	1	1	1	0.79
Median	1	1	1	0.94
Mean	0.94	0.96	0.95	0.86
$3^{rd}Q$	1	1	1	0.99
Max	1	1	1	1
Stdev	0.15	0.13	0.16	0.17

Table 2.6 Percent of Perfect Reconstructions

	Fixed Prototype Matrix	Random Prototype Matrix
0% missing	61.2%	63.7%
10% missing	94.5%	87.3%
50% missing	98.3%	80.7%
90% missing	0%	12.6%

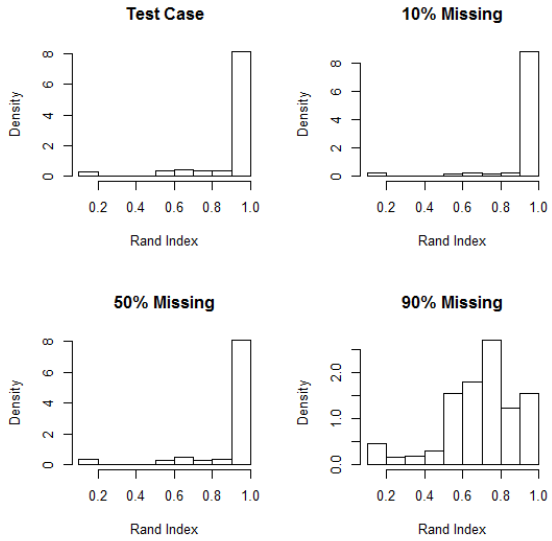

Figure 2.4 RI Histograms for Row Partitioning for Data Simulated with Random Prototype Matrices

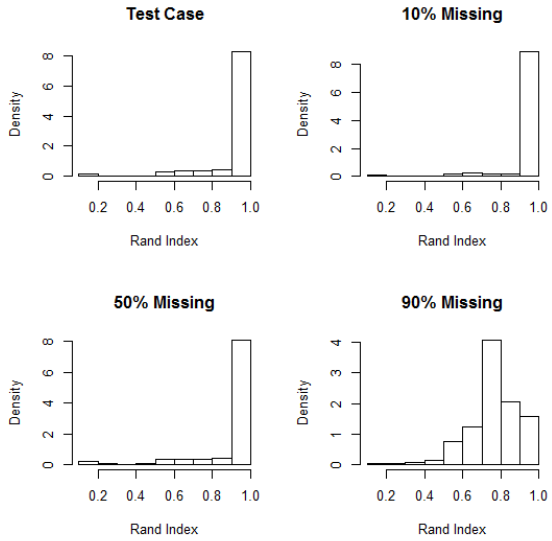

Figure 2.5 RI Histograms for Column Partitioning for Data Simulated with Random Prototype Matrices

algorithm was not fast enough for application on this data set with the high missing percentage. We then applied it on a real microarray data set that has no missing values. In Ross et al. (2000), the NCI60 cancer microarray data set was reported. It has 6830 genes and 64 samples. The raw data matrix X is presented in Figure 2.6, where the rows i index the genes; columns j index the samples; entry X_{ij} indexes the ratio of ratio of induction/repression such that the magnitude is indicated by the intensity of the colors displayed. If the color is black, it means the ratio of control to experimental cDNA is equal to 1. In all cases red indicates an increase in mRNA abundance while green indicates a decrease in abundance in the experimental sample with respect to the control.

We implemented both our prototype-based algorithm and the Spectral Biclustering method on this raw microarray data set. We are aware that there are various data preprocessing methods available for microarray data biclustering. However, we here applied our algorithm on this raw data set instead of preprocessing the data first. The reason is that we would like to illustrate with this example that our proposed algorithm is applicable to a real data set. The biclustering result from the Spectral Biclustering method is shown in Figure 2.7 and that from our algorithm in Figure 2.8. That is, the rows and columns were reordered according to the biclustering results from each method. We also computed the Mean Squared Error (MSE) for results from both methods. Given fixed number of groups for genes and number of groups for samples, we observed about 3% decrease in MSE using our method. The number of groups for genes is 100 and that for samples is 10. We chose these numbers of groups for genes and samples for illustration purposes here. In future, other numbers of groups can be used. The initial clusters were obtained using K-means on both genes and samples.

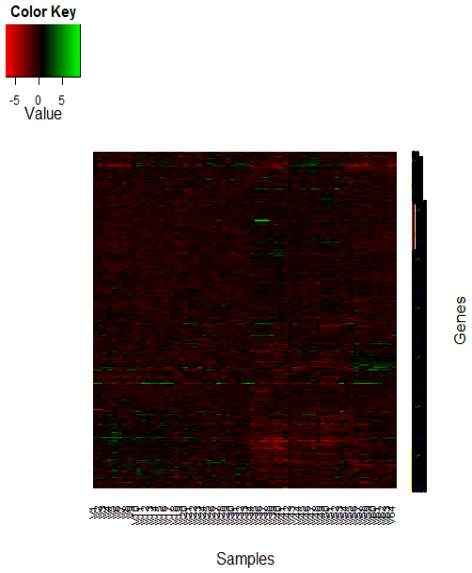

Figure 2.6 NCI60 Cancer Microarray Data Set Heat Map

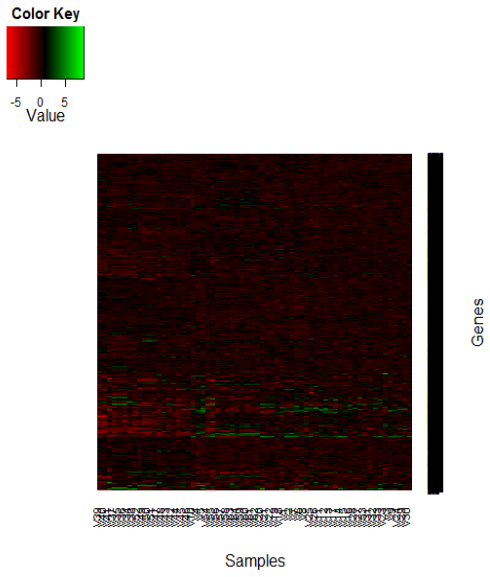

Figure 2.7 Results of SVD method

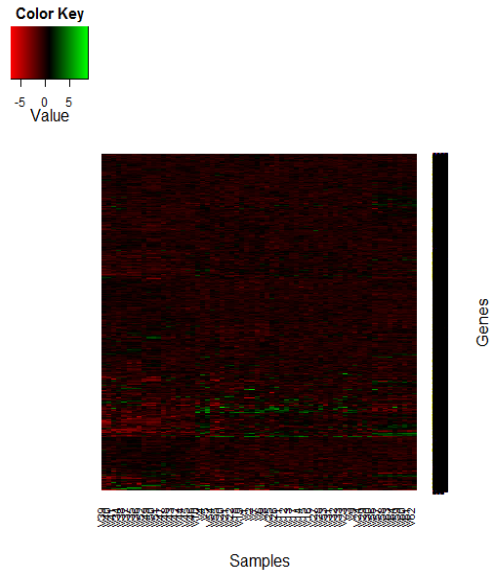

Figure 2.8 Results of Our Method

2.4 Conclusions and Discussion

In this paper, we presented a prototype-based biclustering method. We found that it has good consistency and performance evaluated using simulated data. Also, it outperforms the Spectral Biclustering method (which is an SVD-based method and a comparable benchmark) in terms of Mean Squared Error criterion.

In addition, the algorithm presented in Section 2.2 can be generalized to accommodate other analysis purposes. For example, in the analysis of microarray data such as the NCI60 data set in Section 2.3.2, a distance function commonly used in literature is the Pearson Correlation Coefficient. We may change the objective function in Equation 2.2 to a Pearson Correlation Coefficient between a prototype and rater/movie.

Another observation about this algorithm is that this algorithm would end up with empty clusters for some "bad" choices of initial clusters. This phenomenon is similar with empty clusters possible in K-means algorithm. That is, the algorithm determines the new membership for movies and raters in the following way. Suppose $\mathcal{P}^{(s)}$ and $\mathcal{Q}^{(s)}$ are the current partitions for movies and raters respectively. The new partition $\mathcal{Q}^{(s+1)}$ for raters is determined by firstly finding for each rater, the objective function value in Equation 2.2 between a rater prototype P_{u_i} and the rater. Then, move each rater to his/her "best" prototype group simultaneously. Then, we get an update for partition of raters, $\mathcal{Q}^{(s+1)}$. Similarly, $\mathcal{P}^{(s+1)}$ can be obtained conditional on $\mathcal{P}^{(s)}$ and $\mathcal{Q}^{(s+1)}$. Therefore, empty subsets may exist in $\mathcal{P}^{(s+1)}$ and $\mathcal{Q}^{(s+1)}$. These empty subsets are to be deleted.

According to this observation, it is important that we choose good initial clusters for both the rows and columns. We recommend that one could run a one-way clustering algorithm on both rows and columns as we did for the analysis of microarray data. Then, we could use the one-way clustering results as initial clusters.

We have seen in Section 2.2 that here we moved rows or columns all at once. In future, we could consider a variant of this proposed algorithm such that moves rows or columns one at a time.

Also, since our proposed algorithm only produces a partition of a two-dimensional data matrix for given numbers of groups of rows and columns, another possible direction of this work is to modify our proposed algorithm for a hierarchical variant.

Another comment about this algorithm is that missing values in data can slow down the algorithm substantially. In fact, the computation time increases roughly linearly as the missing percentage increases. This is illustrated in Figure 2.9, where the computation time for $S = 1000$ runs on test cases with various missing value percentages simulated with one prototype matrix in Table 2.1 was plotted against the missing value percentage. One must be cautious when implementing this algorithm if data has a high percentage of missing values. However, a linearly increasing computation time is still within a practical range in many cases when the data matrix is not too large. Also, in Section 2.3.1, we have seen that the proposed approach has very good performance on data with missing values in terms of correctly reconstruct group membership for simulated data. As computation power improves, the algorithm presented in this paper has potential on analysis of data with missing values.

To sum up, the work presented here has shown that the proposed prototype-based algorithm has good performance on both simulated and real data. If one wants to bicluster a data matrix possibly with missing values into a checkerboard structure, the algorithm here is a good option. In particular, if a minimum Mean Squared Error is the goal to be achieved, the algorithm presented here can outperform a comparable benchmark approach, the Spectral Biclustering method.

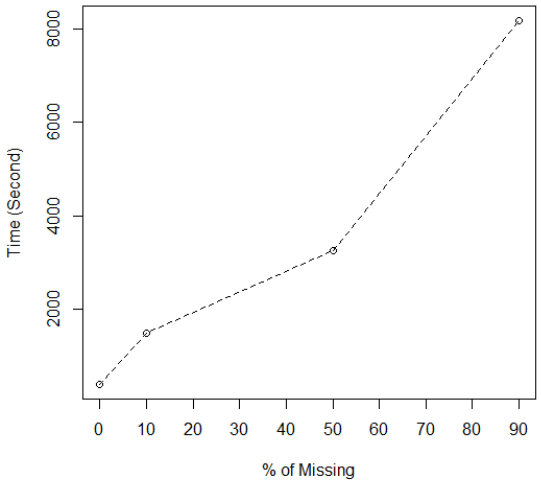

Figure 2.9 Computer Time versus Missing Value Percentage for $S = 1000$ Runs on Test Cases with a Common Prototype Matrix

CHAPTER 3. A BAYESIAN FRAMEWORK FOR FITTING BOLTZMANN MACHINES

A paper to be submitted

Jing Li

Stephen B. Vardeman

Max D. Morris

Department of Statistics

Iowa State University

Ames, IA, 50010

Abstract

Deep Learning is a Statistical Learning topic which involves a "deep" network architecture mimicing the information representation structure in human brain. In this paper, motivated by a hand-written digit classification problem, we proposed a Bayesian framework for fitting Boltzmann machine models. The proposed approach improves on previous available methods because it provides a principled fitting method using an MCMC algorithm. We show that the approach presented here provides a reasonably effective way to extract features from multivariate data for classification.

Keywords. statistical learning, deep learning, Boltzmann machine, feature extraction, model degeneracy

3.1 Introduction

Deep Learning is a field in Statistical Machine Learning. It includes methods that have a "deep" architecture mimicing the information representation structure in human brain. In the realm of Deep Learning, Convolutional Neural Networks (CNNs) and Deep Belief Networks (DBNs) are well established (see Arel et al. (2010)). In DBNs, models consist of several layers of Restricted Boltzmann Machines (RBMs). A RBM is a type of network structure that has only two layers and only allows inter-layer connections. (prohibiting intra-layer connections) For example, Figure 3.1 shows a graph illustrating a RBM, where the blue nodes denote the unobservable/hidden layer; the white nodes denote the observable layer.

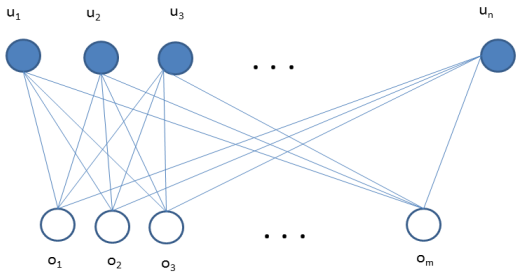

Figure 3.1 A Restricted Boltzmann Machine

Let $\mathbf{o}_i$, for $i \in 1, 2, \ldots, m$, be the i^{th} observable part of $\mathbf{x}$, a realization of a Restricted Boltzmann Machine; let $\mathbf{u}_j$, for $j \in 1, 2, \ldots, n$, be the j^{th} unobservable/hidden part of $\mathbf{x}$. We can see that according to the restriction of RBMs, only inter-layer connections are allowed. Hinton (2002) proposed a by now quite popular estimation method for RBMs, involving minimizing what is called "contrastive divergence". However, this approach is

heuristic and can only train a RBM or stacked RBMs, which exclude connections within a layer of nodes (or between non-adjacent layers of nodes). In this chapter, we will focus on developing a principled estimation method for DBNs using a Bayesian framework. Once we develop a Bayesian framework for this type of problem, we can estimate parameters in the model using an MCMC algorithm. We obtain the posterior samples through Gibbs sampling. Also, our proposed approach can accommodate broader variants of Boltzmann Machine models, e.g., a fully connected Boltzmann Machine that allows every node to be connected with every other node in the model.

Firstly, we motivate a general version of the model with a hand-written digits data set. This is a standard example in DBNs. The goal of a hand-written digits recognition problem is to classify the images of digits 0 up to 9 into a digit class. For each image, the data available for classification purpose are pixel intensities of the image.

3.1.1 Hand-written digits data

In a typical hand-written digits data set, the rows encode individual hand-written images; the columns give intensities for a given pixel from the images. The data sets that we use in this chapter are from the `ElemStatLearn` package in `R`. There are two hand-written digits data sets, of which one is a training set and the other a test set.

The training set contains 7291 observations, each of which is a vector of length 257, where the first element indicates the digit class that the image belongs to; the rest elements are the intensities of the pixels of the 16×16 image of a hand-written digit, for which the intensities are real numbers ranging from -1 to 1. A sample of the images of the digits is shown in Figure 3.2. The frequency table for each digit class is shown in Table 3.1.

Similarly, the test set has 2007 images. The column variables are in the same order as in the training set.

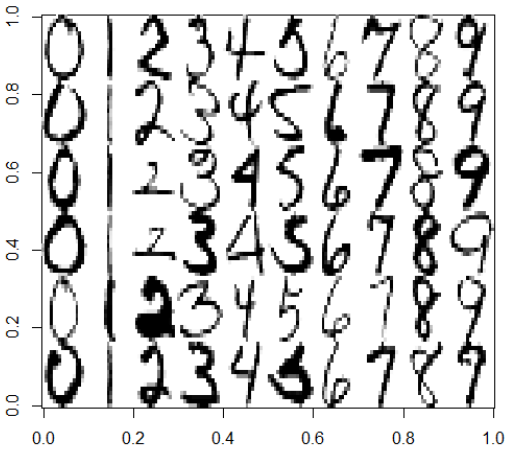

Figure 3.2 Sample of Images of Hand-Written Digits

Table 3.1 Frequency Table for the Data Set by Digit Classes in the Training Set

	0	1	2	3	4	5	6	7	8	9
Frequency	1194	1005	731	658	652	556	664	645	542	644

3.2 A Bayesian Boltzmann Machine Approach

3.2.1 A general version of Bayesian Boltzmann machine model

Now let us focus on solving the digit classification problem using a Boltzmann Machine Model in a Bayesian framework. Suppose we have K classes of characters that we would like to classify. In this case, $K = 10$. Then, in the context of modeling with Boltzmann Machines, one assumes that images from a simple digit class k, for $k \in 1, 2, ...K$, were generated from a common Boltzmann Machine model that shares the same set of parameters, denoted as $\boldsymbol{\theta}_k$. (for $k \in 1, 2, ...K$)

Without loss of generality, let us focus on the case of a single digit class that shares the set of parameters, denoted as $\boldsymbol{\theta}$. Suppose that $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ are independent Boltzmann Machine realizations in this digit class with the same set of parameters $\boldsymbol{\theta}$. We will suppose that some sub-vectors of the $\mathbf{x}_i$, for $i \in 1, 2, \ldots, N$, $\mathbf{o}_i$ are observable and let $\mathbf{u}_i$ stand for the parts of the $\mathbf{x}_i$ that are unobserved. In particular, the observable part in $\mathbf{x}_i$, denoted as $\mathbf{o}_i$, for $i \in 1, 2, \ldots, N$, is actually the observable pixel intensity data for the i^{th} observation in this digit class. Suppose the number of observable nodes, i.e., the number of elements in $\mathbf{o}_i$, is n_o, which is 256 in the hand-written digit data sets in `ElemStatLearn`; also suppose the number of unobservable nodes, i.e., the number of elements in $\mathbf{u}_i$, is p_1. Further, let $p_1 + n_o = p_2$. Then, $\mathbf{x}_i$ can be written as $\mathbf{x}_i = (\mathbf{o}_i', \mathbf{u}_i')' = (o_{i1}, \ldots, o_{in_o}, u_{i1}, \ldots, u_{ip_1})' = (x_{i1}, x_{i2}, \ldots, x_{ip_2})'$, for $i \in 1, 2, \ldots, N$.

In the context of Boltzmann Machine modeling, the parameter vector $\boldsymbol{\theta}$ includes two types of parameters. One group of parameters are associated with single nodes; the other group are those that are associated with edges between two nodes. Suppose we denote the vector of the first type of parameters in $\boldsymbol{\theta}$ as $\boldsymbol{\alpha}$; the second one as $\boldsymbol{\beta}$. Because the total number of nodes is p_2, the total number of edges is $\binom{p_2}{2}$, denoted in what follows as p_3. Then, the parameter vector $\boldsymbol{\alpha}$ can be written as $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_{p_2})'$. Similarly, the set of elements in parameter vector $\boldsymbol{\beta}$ can be written as $\{\beta_{jl} : j < l, \ \forall j, l \in \{1, 2, \ldots, p_2\}\}$.

Then, the joint density of $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ can be written as

$$\prod_{i=1}^{N} f(\mathbf{x}_i|\boldsymbol{\alpha}, \boldsymbol{\beta}) = \gamma(\boldsymbol{\alpha}, \boldsymbol{\beta})^{-N} \prod_{i=1}^{N} \exp\left[\sum_{j=1}^{p_2} \alpha_{0j} x_{ij} + \sum_{j<l} \beta_{jl} x_{ij} x_{il}\right]. \tag{3.1}$$

Now the normalizing constant $\gamma(\boldsymbol{\alpha}, \boldsymbol{\beta})$ involves integration and summation, and is difficult to compute in practice for problems of interesting size. Thus, we propose a prior for $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ of the form

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}) \propto \gamma(\boldsymbol{\alpha}, \boldsymbol{\beta})^N \exp\left(-c\left(\sum_{j=1}^{p_2} \alpha_{0j}^2 + \sum_{j<l} \beta_{jl}^2\right)\right), \tag{3.2}$$

where c is a tuning parameter for this model. This prior for $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ is chosen such that it cancels out the intractable normalizing constant in the likelihood function when multiplying the two terms together to produce a form for the posterior distribution. Then, the joint posterior for $\boldsymbol{\alpha}, \boldsymbol{\beta}$ and $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ is proportional to

$$\exp\left[\sum_{i=1}^{N}\left(\sum_{j=1}^{p_2} \alpha_{0j} x_{ij} + \sum_{j<l} \beta_{jl} x_{ij} x_{il}\right)\right] \exp\left[-c\left(\sum_{j=1}^{p_2} \alpha_{0j}^2 + \sum_{j<l} \beta_{jl}^2\right)\right], \tag{3.3}$$

i.e.,

$$\exp\left[-c\sum_{j}^{p_2} \alpha_{0j}^2 + \sum_{j}^{p_2} \alpha_{0j}\left(\sum_{i}^{N} x_{ij}\right) - c\sum_{j<l} \beta_{jl}^2 + \sum_{j<l} \beta_{jl}\left(\sum_{i}^{N} x_{ij} x_{il}\right)\right]. \tag{3.4}$$

3.2.2 MCMC algorithm for the general model

It seems that we only have three types of variables, $\boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{u}_i$, for $i \in 1, 2, \ldots, N$, for the posterior of the general version of Bayesian Boltzmann Machine model, where $\mathbf{u}_i$ is the vector of unobservable latent variables for $\mathbf{x}_i$, which is the i^{th} realization generated from the Boltzmann Machine corresponding to this digit class. Because in the handwritten digit data sets, the observable variables are continuous ranging from -1 to 1, we derive the conditional posterior distribution for $\mathbf{o_i}$, the observable nodes for $\mathbf{x}_i$, assuming $o_{ij} \in [-1, 1] \; \forall \; j \in 1, 2, \ldots, n_o$. But as a matter of convenience, we assume that the hidden nodes in $\mathbf{u}_i$ are binary, i.e., $u_{ij} \in \{-1, 1\} \; \forall \; j \in 1, 2, \ldots, p_1$. We assume that hidden nodes

are binary because we found that even if we assume hidden nodes are continuous, the posterior samples of the hidden nodes tend to pile up -1 and 1. Then, based on the joint posterior distribution in Equation 3.4, we here provide a Gibbs sampling algorithm (a type of Markov Chain Monte Carlo (MCMC) algorithm) in order to get posterior samples from the joint posterior distribution for the parameters. At every iteration of the MCMC algorithm, we draw samples conditional on the last iteration and current updates. The MCMC algorithm that we develop for the general version of the BM model is presented below.

Firstly, the conditional posterior distribution for α_{0j}, for $j = 1, 2, \ldots, p_2$, is $\mathrm{N}\left(\frac{\sum_{i=1}^{N} x_{ij}}{2c}, \frac{1}{2c}\right)$, where $\mathrm{N}(a, b)$ denotes the Normal distribution with mean a and variance b. Thus conditionally, we update α_{0j}, for $j = 1, 2, \ldots, p_2$ by sampling from this distribution.

Secondly, the conditional posterior distribution for β_{jl}, for all $j < l$, $j, l \in \{1, \ldots, (p_2)\}$, is $\mathrm{N}\left(\frac{\sum_{i=1}^{N} x_{ij}x_{il}}{2c}, \frac{1}{2c}\right)$. Thus, conditionally, we update β_{jl}, for all $j < l$, $j, l \in \{1, \ldots, (p_2)\}$, by sampling from this distribution.

Thirdly, for $i = 1, 2, \ldots, N$ and $j = 1, \ldots, p_1$, the conditional posterior distribution for u_{ij}, i.e., the hidden nodes, is a Bernoulli distribution with probability p, i.e., Bernoulli(p), where $p = \frac{a}{1+a}$, where

$$a = \frac{\Pr(u_{ij} = 1 | \mathbf{x} \backslash u_{ij}, \boldsymbol{\alpha}, \boldsymbol{\beta})}{\Pr(u_{ij} = -1 | \mathbf{x} \backslash u_{ij}, \boldsymbol{\alpha}, \boldsymbol{\beta})} = \exp\left[2\left(\alpha_{0j} + \sum_{l<k; k \in \{1,\ldots,p_2\}} \beta_{lk} x_{ik}\right)\right],$$

where $\mathbf{x} \backslash u_{ij}$ denotes the combined observable and hidden variables in the Boltzmann Machine without the hidden variable u_{ij}; $l = j + n_o$, which is the identification number in $\mathbf{x}_i$ adjusted for the observable variables.

Lastly, for $i = 1, 2, \ldots, N$ and $j = 1, \ldots, n_o$, the posterior distribution for x_{ij}, i.e., o_{ij}, the observable nodes, is proportional to

$$\propto \exp\left\{\left(\alpha_{0j} + \sum_{k<j;k\in\{1,\ldots,p_2\}} \beta_{jk}x_{ik}\right)x_{ij}\right\}.$$

This form of distribution can be sampled by the method of sampling in the Appendix B for a distribution with a Probability Density Function (pdf) proportional to $\exp(ax)$, where $a = \alpha_{0j} + \sum_{k\neq j;k\in\{1,\ldots,p_2\}} \beta_{jk}x_{ik}$ in this context and $x = x_{ij}$. Then, conditionally, we get a sample for x_{ij}, i.e., o_{ij}, for $i = 1, 2, \ldots, N$ and $j = 1, \ldots, n_o$, by sampling from this distribution by inverting the Cumulative Distribution Function (CDF) of x_{ij}. See the Appendix for more details of sampling from this distribution.

If the purpose of the MCMC algorithm is only to obtain posterior samples for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, the last step for sampling an observable node is not needed. If the purpose of the MCMC algorithm is to simulate a distribution for observable variables, i.e., to simulate images of hand-written digits based on parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, the first two steps are not needed.

Therefore, we could either obtain posterior sample draws for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ or simulate images of hand-written digits based on the above conditional posterior distributions,

3.2.3 Restrictions for RBMs

Although the above MCMC algorithm draw samples based on a posterior distribution of a general version of Boltzmann Machine, it is easy to adapt the above algorithm for other versions of Boltzmann Machine by putting restrictions on the parameters. For instance, if we want to get posterior samples from a Restricted Boltzmann Machine model, we need to make a few restrictions. For the purpose of illustration, we will show the restrictions for a RBM with one hidden layer. Again let $\mathbf{o}_i$ be the observable part of $\mathbf{x}_i$; let $\mathbf{u}_i$ be the unobservable/hidden part of $\mathbf{x}_i$. Then, the joint posterior distribution involves their interaction terms such as $\beta_{jk}o_{ij}o_{ik}$ and $\beta_{lm}u_{il}u_{im}$, where β_{jk} and β_{lm} need

to be set to 0 for all $j < k, \forall\ j, k \in 1, 2, \ldots, n_o$, where n_o is the number of observable nodes, and for all $l < m, \forall\ l, m \in 1, 2, \ldots, p_1$, where p_1 is the number of hidden nodes.

3.3 Results

Based on the MCMC algorithm above, we obtained posterior samples of the parameters for different variants of the general version of Boltzmann Machine model presented in Section 3.2. For illustration purposes, we consider three different variants of Boltzmann Machine model, that is, we consider a Restricted Boltzmann Machine model with one hidden layer, a RBM with two hidden layers and a Boltzmann Machine model that fully connected, i.e., in which every node is connected with every other node.

3.3.1 RBM with one hidden layer

For a RBM with one hidden layer as illustrated in Figure 3.1, we estimated parameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ through the MCMC algorithm presented in Section 3.2 for each digit class separately based on the hand-written digit data set `zip.train`. Here, we fitted a model with 100 hidden nodes in the hidden layer. The number of hidden nodes was arbitrarily chosen for illustration. If we use notation presented in Section 3.2, we have $\boldsymbol{\theta} = (\boldsymbol{\alpha}', \boldsymbol{\beta}')'$. Let $\boldsymbol{\theta}_i$, for $i \in 0, 1, \ldots, 9$, denote the parameters corresponding to the training data of digit i. That is, we trained our parameters on each digit class separately. Then, we obtained the posterior samples for parameters $\boldsymbol{\theta}_i$, for $i \in 0, 1, \ldots, 9$. Using the posterior means, denoted as $\hat{\boldsymbol{\theta}}_i$, for each parameter vector of $\boldsymbol{\theta}_i$ for $i \in 0, 1, \ldots, 9$, we computed the un-normalized log probability scores for each observation in the test data set `zip.test` using trained parameters from each digit class. (much as Hinton (2002) did) Let $S(\mathbf{x}|\boldsymbol{\theta})$ be the un-normalized log probability score function. Then, based on Equation 3.1, we have the following equation for computing the un-normalized scores,

$$S(\mathbf{x}|\hat{\boldsymbol{\theta}}) = \sum_{i=1}^{N} \left(\sum_{j}^{p_2} \hat{\alpha}_{oj} x_{ij} + \sum_{j<l} \hat{\beta}_j x_{ij} x_{il} \right), \tag{3.5}$$

where the posterior means for $\boldsymbol{\theta}$ were used for computation of the un-normalized scores. That is, we computed for each observation in the test set `zip.test`, the un-normalized scores $S(\mathbf{x}|\boldsymbol{\theta})$, for $i \in 0, 1, \ldots, 9$.

Because the un-normalized scores can include negative values and have a large magnitude in most cases, we adopted the transformation proposed in Reising et al. (2011),

$$h(\theta) = \begin{cases} \ln(\theta) & \text{for } \theta \geq 2 \\ \theta(\ln(2)/2) & \text{for } 2 < \theta < 2 \\ -\ln(|\theta|) & \text{for } \theta \leq -2 \end{cases} \tag{3.6}$$

The scatter plot for the transformed un-normalized log probability scores for classes 0 and 1 is shown in Figure 3.3. Observations from class 1 were in red; those from class 0 black. We can see that the two classes are separable with some error. For illustration purposes, only the scatter plot for classes 0 and 1 is shown here. Scatter plots for other comparisons can be generated in a similar way. All of them look similar with Figure 3.3 in the sense that two classes in consideration are separable with some error, where the overlap between the two classes may vary from one comparison to another. More plots from other comparisons will be shown in Section 3.3.3.

3.3.2 RBM with two hidden layers

Similarly with the RBM model with one hidden layer fitted above, we also considered a RBM model with two hidden layers, where the first hidden layer has 100 nodes and the second 50 nodes. The scatter plot for the transformed un-normalized log probability scores for classes 0 and 1 is shown in Figure 3.4. Observations from class 1 were in red; those from class 0 black. Comparing with results using a RBM model with one hidden layer shown in Figure 3.3, we can see that similarly as the previous results, the two classes are separable with some error. However, it seems that the two classes separate better if using a RBM with two hidden layers. For the same illustration purposes as in Section 3.3.1, only the scatter plot for classes 0 and 1 is shown here.

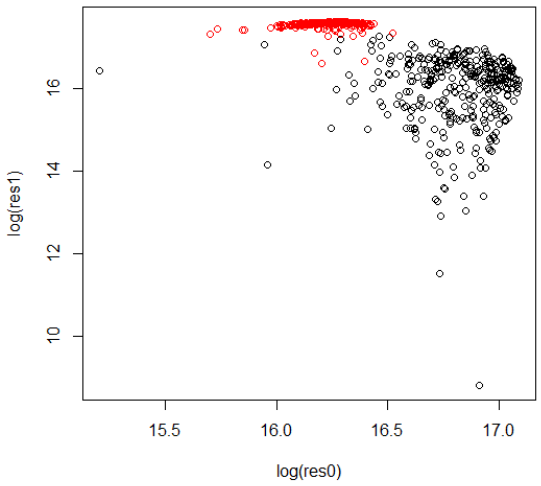

Figure 3.3 Transformed Un-normalized log Probability Scores for Test Images from Classes 0 and 1 Computed with Parameters of RBMs with One Hidden Layer Trained on Classes 0 and 1

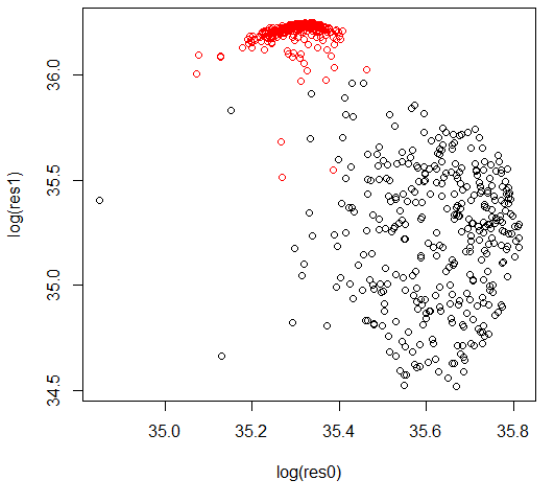

Figure 3.4 Transformed Un-normalized log Probability Scores for Test Images from Classes 0 and 1 Computed with Parameters of RBMs with Two Hidden Layer Trained on Classes 0 and 1

Next, we simulated images using the MCMC algorithm presented in Section 3.2 based on trained parameters for each digit class. A sample of the simulated images for class 0 are shown in Figure 3.5. We can see that they look almost identical to each other, i.e., there is not enough variability in the simulated images in order to look like real handwritten images. In particular, they all look like the image produced if we take the mean of the class 0 images in the training set. The cause of this phenomenon is that the model we consider in this paper seems to be degenerate according to Schweinberger (2011). In fact, model degeneracy is closely related to the performance of models used in this paper. (which are common exponential family models that have been used for quite a while). We will discuss the model degeneracy issue further in the Section 3.4.

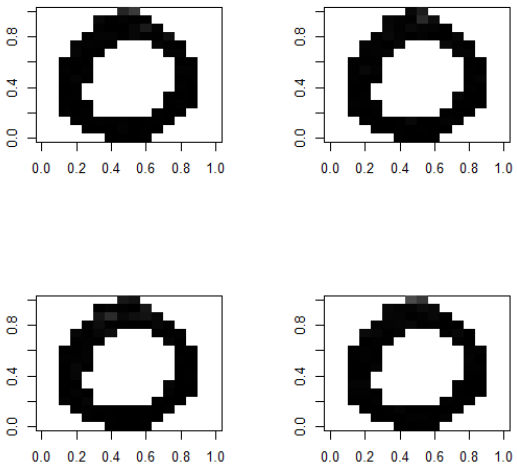

Figure 3.5 Simulated Images for Class 0 Based on a RBM with 2 Hidden Layers

3.3.3 Full BMs

As we mentioned before, in a full Boltzmann Machine model, every node connects with every other node. Here, we considered a full Boltzmann Machine model with 10

hidden nodes, which is comparable to a RBM with one hidden layer of 143 hidden nodes in terms of the number of non-zero entries of $\boldsymbol{\theta} = (\boldsymbol{\alpha}', \boldsymbol{\beta}')'$. The scatter plots for the transformed un-normalized log probability scores for classes 0 and 1 is shown in Figure 3.6. The plot on the left is the whole scatter plot; the one on the right shows the part of the scatter plot that has the most points, with outliers deleted. Observations from class 1 were in red; those from class 0 black. We can see that the scatter plot looks similar to the ones that compare classes 0 and 1; two classes are separable with some error.

Similarly, we present the scatter plots for the transformed un-normalized log probability scores for classes 7 vs 9 in Figure 3.7; 2 vs 8 in Figure 3.8. We can see that the two classes in each comparision are separable with some error. However, if we compare scatter plots in Figure 3.6, Figure 3.7 and Figure 3.8, we can see that classes 0 and 1 separate the best in all three comparisons; 7 and 9 separate the worst in the sense that the scatter plot in Figure 3.7 has the most "overlap points"; separation between classes 2 and 8 is at the middle level. The high error rate between classes 7 and 9 is well known, for instance, Hinton (2002) pointed out this fact. Also, it is easy to understand this because these two digits are similar to each other when people write them.

Next, as in Section 3.3.2, we simulated images using the MCMC algorithm presented in Section 3.2 based on trained parameters for each digit class. A sample of the simulated images for class 0 are shown in Figure 3.9. Similar to what we have seen in Section 3.3.2, the simulated images look almost identical to each other. In fact, we also found that the simulated images look similar to those shown here even when we adjusted the tuning parameter c in the model presented in Equation 3.4.

Although the models presented in this paper all have the degeneracy issue, we still observed that the un-normalized log probability scores seem to be able to provide a reasonably good separation between classes. Therefore, we treated the un-normalized log probability scores as "features" for classification purposes. That is, we computed

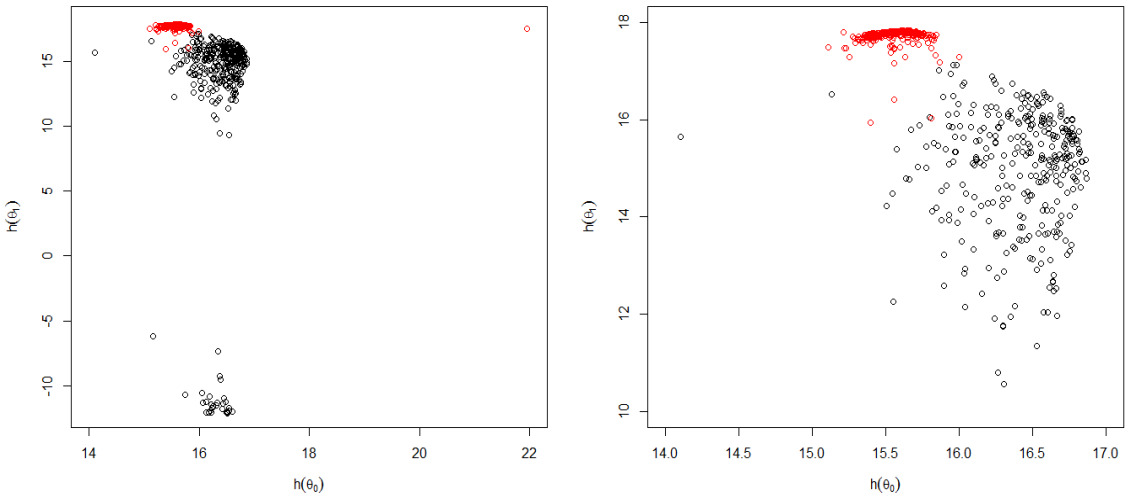

Figure 3.6 Transformed Un-normalized log Probability Scores for Test Images from Classes 0 and 1 Computed with Parameters of Full BMs Trained on Classes 0 and 1

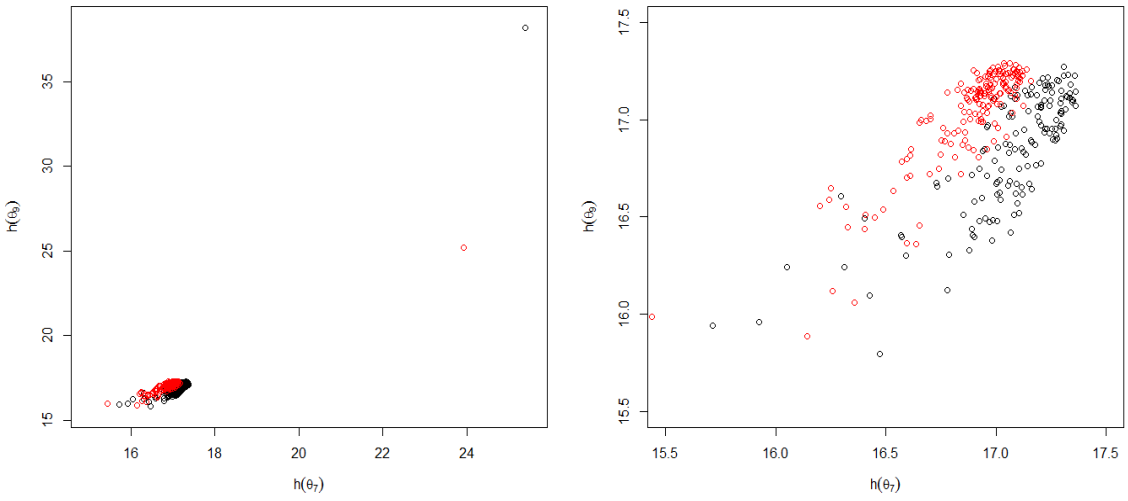

Figure 3.7 Transformed Un-normalized log Probability Scores for Test Images from Classes 7 and 9 Computed with Parameters of Full BMs Trained on Classes 7 and 9

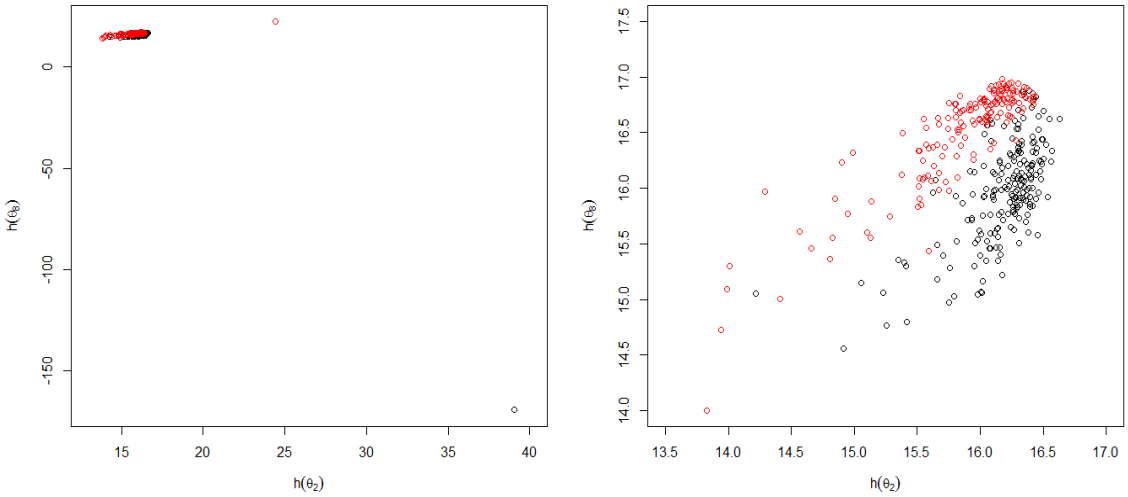

Figure 3.8 Transformed Un-normalized log Probability Scores for Test Images from Classes 2 and 8 Computed with Parameters of Full BMs Trained on Classes 2 and 8

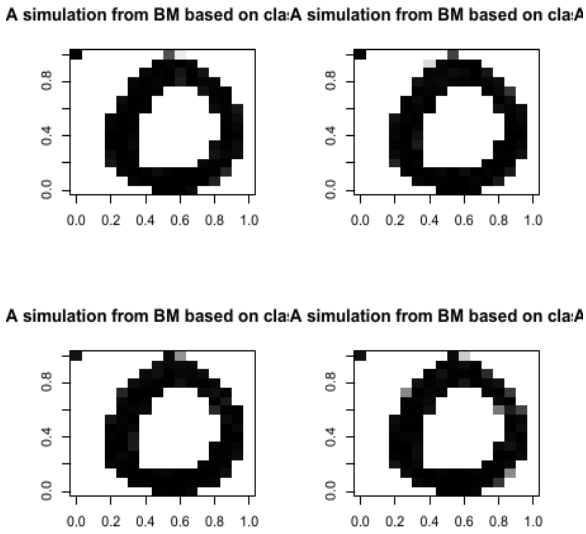

Figure 3.9 Simulated Images for Class 0 Based on a Full BM

the un-normalized log probability scores for each observation, in both the training set `zip.train` and the test set `zip.test`, under the trained parameters set from each digit class. Then, we applied the Random Forest method to the feature set based on the training set. Then, we applied the resulting classification rule to the test set. We obtained the classification confusion matrix shown in Table 3.2, for the results on the test set. In contrast, we also applied the Random Forest method to the original data sets directly. The classification confusion matrix is shown in Table 3.3, for the results on the test set. We can see that the classification error rate ranges from 2% to 10% for various digit classes based on Random Forest results on the original data with 256 pixel intensity feature variables. Also, the classification error rates are around twice those for Random Forest results based on the un-normalized log probability scores. In fact, pooling all observations from all the classes together, we have that the Random Forest on the original data gives a 94.2% correct classification rate, while the rate is 87.6% employing only the 10 un-normalized log probability scores features. That is, Random Forest classification gave more prediction errors on the test set when using the only the un-normalized log probability scores as features in place of the 256-dimensional raw data. However, we can still see that the feature extraction through the un-normalized log probability scores is reasonably effective.

3.4 Conclusions and Discussion

Motivated by the hand-written digits classification problem, the work presented in this paper focuses on developing a Bayesian framework for Boltzmann Machine models and thus a principled estimation method. In contrast, previous literature such as Hinton (2002) developed only an ad-hoc fitting method. Moreover, our approach can accommodate Boltzmann Machine models with an arbitrary connection structure. In

Table 3.2 Confusion Matrix for Test Set Results Based on Random Forest Classification for Un-normalized Log Probability Scores

		Predicted Class										
		0	1	2	3	4	5	6	7	8	9	Class Error
Actual Class	0	345	0	2	1	2	1	8	0	0	0	0.039
	1	1	249	0	1	4	1	6	0	0	2	0.057
	2	6	0	169	5	8	2	1	2	5	0	0.15
	3	5	0	6	131	0	17	0	1	5	1	0.21
	4	1	1	6	0	170	1	4	1	3	13	0.15
	5	9	0	0	8	3	134	1	0	3	2	0.16
	6	7	1	4	0	5	4	148	0	1	0	0.13
	7	1	0	2	0	7	0	0	124	3	10	0.16
	8	5	0	1	8	3	6	0	1	139	3	0.16
	9	0	3	0	1	11	1	0	8	4	149	0.16

Table 3.3 Confusion Matrix for Test Set Results Based on Random Forest Classification for Original Data

		Predicted Class										
		0	1	2	3	4	5	6	7	8	9	Class Error
Actual Class	0	353	0	2	0	2	0	1	0	0	1	0.017
	1	0	255	0	0	4	0	4	1	0	0	0.034
	2	2	0	182	4	2	1	1	1	5	0	0.081
	3	0	0	4	150	0	11	0	0	1	0	0.096
	4	0	2	4	0	190	0	2	0	0	2	0.05
	5	3	0	0	5	1	147	0	0	1	3	0.081
	6	0	0	3	0	3	3	159	0	2	0	0.065
	7	0	0	2	0	6	0	0	137	1	1	0.068
	8	3	0	2	4	1	3	0	0	149	4	0.10
	9	0	2	0	0	4	0	0	0	3	168	0.051

contrast, the ad hoc approach in Hinton (2002) can only handle models with layered structures.

However, as we mentioned previously in the Section 3.3, according to Schweinberger (2011), the type of models presented here (which have been widely used in decades) have a serious defect, that is, they tend to put on most of their probability mass on modes. Indeed, we observed this unfavorable property by seeing that simulated hand-written digit images, such as those shown in Figure 3.5, basically show no variation from one simulation to another. Also, they all look similar with the image produced if we take the mean of all available data. Tuning the fitting method would not help because of this model degeneracy issue.

Nonetheless, BMs still captured a part of features in the data. We have seen that although classification using un-normalized log probability scores alone as features is not as good as using the whole data vector, the prediction in this case still has a high level of correctness. And also, we here only considered the Random Forest method for classification. It is possible that these features will perform better paired with other classification methods, such as Support Vector Machines and Boosting, to name a few. In fact, it is a good future direction to study the performance of these features with these other classifier methods.

It is also interesting to observe here that the un-normalized probability scores for a RBM either with one hidden layer or two hidden layers are all positive and have no outliers. In contrast, a full BM can have negative values with some outliers. This may still be explained by the instability characteristic of the model presented in Equation 3.4 according to Schweinberger (2011). Models taking this form are called "2-star" models in Schweinberger (2011), where a 2-star model includes stable sufficient statistic $\sum_i \sum_j x_{ij}$ and unstable sufficient statistic $\sum_i \sum_{j<l} x_{ij}x_{il}$. It is the unstable statistic that causes the instability and model degeneracy problem. Apparently, a full BM has more unstable statistic terms (which are actually the interaction terms) in proportion compared with

a RBM with the same number of parameters in $\boldsymbol{\theta}$. It seems that based on the empirical results we have here, we could conclude that a discrete exponential model, such as the one presented in Equation 3.4, is more unstable in performance if it has more unstable sufficient statistic terms, holding the total number of parameters constant. This might be an interesting direction for future study.

In order to improve the prediction, we could also consider model averaging. That is, we could take a weighted average of the un-normalized log probability scores based on different models, such as models with different number of hidden nodes and connecting structures. Hinton (2002) reported model averaging in this way to be effective.

To sum up, the approach outlined here provides a reasonably effective way to extract features from multivariate data for classification. It also surpasses the previous available method in terms of fitting because it provides a principled fitting method for Boltzmann Machine models.

APPENDIX A. DU-PROCESS

Here we describe a construction due to Du (2014). For partition $\mathcal{P} = \{S_1^{\mathcal{P}}, S_2^{\mathcal{P}}, \ldots, S_{k^{\mathcal{P}}}^{\mathcal{P}}\}$, a one-step Markov transition to a new partition $\mathcal{Q} = \{S_1^{\mathcal{Q}}, S_2^{\mathcal{Q}}, \ldots, S_{k^{\mathcal{Q}}}^{\mathcal{Q}}\}$ (where $k^{\mathcal{Q}}$ is of necessity one of $k^{\mathcal{Q}} - 1$, $k^{\mathcal{Q}}$ or $k^{\mathcal{Q}} + 1$) is made as follows.

1. Pick an index from the index set $I = \{1, 2, \ldots, N\}$ at random, call this index i_0.
2. Find the element of $\mathcal{P}$ to which i_0 belongs, call this set $S_{k_0}^{\mathcal{P}}$.
3. If $S_{k_0}^{\mathcal{P}}$ is a singleton, move index i_0 to one of the sets $S_k^{\mathcal{P}}$ for $k \neq k_0$ chosen at random with equal probabilities $(k^{\mathcal{P}} - 1)^{-1}$ to create the new partition.
4. If $S_{k_0}^{\mathcal{P}}$ is not a singleton, move index i_0 to one of the sets $S_k^{\mathcal{P}}$ for $k \neq k_0$ or to a new singleton set $\{i_0\}$, where the choice is made at random with equal probabilities $(k^{\mathcal{P}})^{-1}$ to create the new partition.

It is remarkable that this simple prescription produces an irreducible, aperiodic, positive recurrent Markov Chain (on the space of partitions).

APPENDIX B. SAMPLING FROM A DISTRIBUTION PROPORTIONAL TO EXPONENTIAL FUNCTION BY INVERTING A CDF

Suppose a random variable X has a Probability Density Function (pdf) proportional to $\exp(ax)$, that is,

$$f_X(x) = k\exp(ax), x \in [-1, 1], k > 0,$$

where $f_X(x)$ is the pdf of X. Then, the Cumulative Density Function (cdf) is

$$\begin{aligned} F_X(x) &= \int_{-1}^{x} f_X(t)dt \\ &= \int_{-1}^{x} k\exp(at)dt \\ &= k\int_{-1}^{x} \exp(at)dt \\ &= k\frac{1}{a}\exp(at)|_{-1}^{x} \\ &= k\frac{1}{a}[\exp(ax) - \exp(-a)]. \end{aligned}$$

We also know that $F_X(1) = 1$, that is

$$k\frac{1}{a}[\exp(a) - \exp(-a)] = 1.$$

Thus, $k = \frac{a}{\exp(a)-\exp(-a)}$. Further, as is well known, the cdf $F_X(x)$ is a Standard Uniform random variable. Suppose y is a sample from the Standard Uniform distribution, then, we could get a sample x from the distribution of X by solving the following equation,

$$y = k\frac{1}{a}[\exp(ax) - \exp(-a)] = \frac{1}{\exp(a) - \exp(-a)}[\exp(ax) - \exp(-a)].$$

Then, we get $x = \frac{1}{a} \log \left(y(\exp(a) - \exp(-a)) + \exp(-a) \right)$, which is the solution we could use for sampling from the distribution of a random variable X, of which the pdf is proportional to $\exp(ax)$ on $[-1, 1]$.

BIBLIOGRAPHY

I. Arel, D. C. Rose and T. P. Karnowski, Deep Machine Learning - A New Frontier in Artificial Intelligence Research, *IEEE Computational Intelligence Magazine*, November 2010

S. Busygin, O. Prokopyev and P. Pardalos, Biclustering in Data Mining, *Computers and Operations Research 35*, 2964-2987, 2008

Y. Cheng and G. M. Church, Biclustering of Expression Data, *Proc. Eighth Int'l Conf. Intelligent Systems for Molecular Biology*, pp. 93-103, 2000.

C. Du, Modeling, inference and clustering for equivalence classes of 3-D orientations, *Graduate Theses and Dissertations*, Paper 13738, http://lib.dr.iastate.edu/etd/13738

G. Hinton, Training Products of Experts by Minimizing Contrastive Divergence, *Neural Computation 14*, 1771-1800, 2002

Y. Kluger, R. Basri, J. T. Chang and M. Gerstein, Spectral Biclustering of Microarray Data, *Genome Res.*, 2003 13: 703-716

L. Lazzeroni and A. Owen, Plaid Models for Gene Expression Data, *Statistica Sinica*, Vol. 12, No. 1, 61-86, 2002

J. Liu and W. Wang, OP-cluster: Clustering by Tendency in High Dimensional Space, *Proceedings of the Third IEEE International Conference on Data Mining*, 187-194, 2003

S. C. Madeira and A. L. Oliveira, Biclustering Algorithms for Biological Data Analysis: A Survey, *IEEE Transactions on Computational Biology and Bioinformatics*, Vol 1, NO. 1, January-March 2004

Z. Michalewicz, Genetic Algorithms + Data Structure = Evolution Programs, *Springer*, New York, 1996

W. M. Rand, Objective Criteria for the Evaluation of Clustering Methods, *Journal of the American Statistical Association*, Vol. 66, No. 336, Dec., 1971

M. Reising, M. Morris, S. Vardeman and S. Higbee, Modeling Spectral-Temporal Data From Point Source Events, *Technometrics*, 53:2, 183-195, DOI: 10.1198/TECH.2011.09014

D. T. Ross, U. Scherf, M. B. Eisen, C. M. Perou, C. Rees, P. Spellman, V. Iyer, S. S. Jeffrey, M. Van de Rijn, M. Waltham, A. Pergamenschikov, J. C. Lee, D. Lashkari, D. Shalon, T. G. Myers, J. N. Weinstein, D. Botstein, P. O. Brown, Systematic Variation in Gene Expression Patterns in Human Cancer Cell Lines, *Nature Genetics*, 24, 227 - 235 (2000), DOI:10.1038/734

M. Schweinberger, Instability, Sensitivity, and Degeneracy of Discrete Exponential Families, *Journal of the American Statistical Association*, 106:496, 1361-1370, DOI: 10.1198/jasa.2011.tm10747

P. H. A. Sneath, The Application of Computers to Taxonomy, *Journal of General Microbiology*, 1957

R. R. Sokal and C. D. Michener, A Statistical Method for Evaluating Systematic Relationships, *University of Kansas Scientific Bulletin*, 1958

A. Toscher, M. Jahrer and R. M. Bell, The BigChaos Solution to the Netflix Grand Prize, 2009