Network selection, Information filtering and Scalable computation

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

Changqing Ye

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

Xiaotong Shen, Adviser

March 2014

Acknowledgements

There are a number pf people without whom this thesis might not have been written and to whom Iam greatly indebted.

I would like to express the deepest appreciation to my thesis advisor, Xiaotong-Shen, who continues to guide and support me. His expertise in statistics and machine learning improved my research skills and prepared me for future challenged in the big data field. As a software engineer, I feel really lucky received comprehensive training under Professor Shen in statistics, which make me special in the team.

I would like to thank my committee members, Professor Dennis Cook, Professor Yuhong Yang and Professor Wei Pan, who carefully reviewed my thesis and provided many advice.

I would like to thank my wife Qian for her personal support and great patience at all times. My parents have given me their unequivocal support throughout, as always, for which my mere expression of thanks likewise does not suffice.

Dedication

I dedicate this thesis to my wife, Qian, who did more than her share around the house as I sat at the computer.

Abstract

This dissertation explores two application scenarios of sparsity pursuit method on large scale data sets. The first scenario is classification and regression in analyzing high dimensional structured data, where predictors corresponds to nodes of a given directed graph. This arises in, for instance, identification of disease genes for the Parkinson's diseases from a network of candidate genes. In such a situation, directed graph describes dependencies among the genes, where direction of edges represent certain causal effects. Key to high-dimensional structured classification and regression is how to utilize dependencies among predictors as specified by directions of the graph. In this dissertation, we develop a novel method that fully takes into account such dependencies formulated through certain nonlinear constraints. We apply the proposed method to two applications, feature selection in large margin binary classification and in linear regression. We implement the proposed method through difference convex programming for the cost function and constraints. Finally, theoretical and numerical analyses suggest that the proposed method achieves the desired objectives. An application to disease gene identification is presented. The second application scenario is personalized information filtering which extracts the information specifically relevant to a user, predicting his/her preference over a large number of items, based on the opinions of users who think alike or its content. This problem is cast into the framework of regression and classification, where we introduce novel partial latent models to integrate additional user-specific and content-specific predictors, for higher predictive accuracy. In particular, we factorize a user-over-item preference matrix into a product of two matrices, each representing a user's preference and an item preference by users. Then we propose a likelihood method to seek a sparsest latent

factorization, from a class of over-complete factorizations, possibly with a high percentage of missing values. This promotes additional sparsity beyond rank reduction. Computationally, we design methods based on a "decomposition and combination" strategy, to break large-scale optimization into many small subproblems to solve in a recursive and parallel manner. On this basis, we implement the proposed methods through multi-platform shared-memory parallel programming, and through Mahout, a library for scalable machine learning and data mining, for mapReduce computation. For example, our methods are scalable to a dataset consisting of three billions of observations on a single machine with sufficient memory, having good timings. Both theoretical and numerical investigations show that the proposed methods exhibit significant improvement in accuracy over state-of-the-art scalable methods.

Contents

List of Tables

List of Figures

Chapter 1

Classification and regression over networks defined by directed graphs

1.1 Introduction

Given a huge amount of information gathered in investigations, scientific knowledge is managed and explored in a structured manner. For instance, in a study for identifying disease-causing genes for Parkinson's disease, expression profiles of 22283 genes are collected from 105 patients with 55 disease cases versus 50 control cases; see the Gene Expression Omnibus dataset—GSE6613 (http://www.ncbi.nlm.nih.gov/geo/) and [40] for more details. In a situation as such, the number of candidate genes $p = 22,283$ is still much higher than the size of sample $n = 105$. To battle "curse of dimensionality", additional problem structures must be exploited, especially dependency structures from gene interactions, grouping and casual relationships. The central issue this article addresses is how to utilize dependency structures described by a directed graph for high-dimensional regression and classification.

Knowledge is commonly expressed in terms of graphs, which loosely describes dependencies among all connecting nodes of a graph. In biology, for instance, many

biological processes can be represented and modeled by directed graphs, such as regulatory networks or metabolic pathways. Here we focus our attention to direct graphs, where edges between two nodes represent certain causal-effect relationships. In general, the problem can be formulated as a prediction problem with n observations having outcome $y_i, i = 1, 2, \ldots, n$ and predictors $\boldsymbol{x}_i, i = 1, 2, \ldots, n$, $\boldsymbol{x}_i$ is a vector with dimension p. Moreover, dependencies are defined on the p dimensional predictors.

For unstructured data, many regularization methods have been proposed in regression, including least absolute shrinkage and selection operator ([42]), smoothly clipped absolute deviation([35]), elastic net ([51]), least angle regression ([34]) and some extensions such as adaptive Lasso ([50]) and group Lasso ([47]). However, when the dimension is much larger than the sample size, regularization is not enough. For structured data, to take account the dependency of the predictors, a few methods such as network-constrained regularization ([38]) and network-based SVM ([49]), have been proposed. Despite progress, how to utilize a directed graph has rarely been explored in regression and classification. In this article, we develop a novel method to utilize information on this kind of graphs and networks as well as a non-convex penalty. In our method, we introduce the graph constraints which fully capture the causal-effect relationship defined by the edges. By enforcing the proposed constraints on the coefficients, the method achieves pathway extraction and thus has better model selection and prediction accuracy.

For implementation, we consider two large margin classifiers SVM ([45]) and ψ-learning([39]), ordinary least square regression. To handle non-convex cost functions and non-convex constraints, we employ difference convex (DC) programming ([32]). Finally, we approximate the original non-convex optimization problem by a convex problem and it can be showed that by solving this convex problem iteratively, we can obtain the desired results.

We perform several simulation studies with the proposed method. The result

shows the graph information has strong effect on model selection when the graph has long pathways and the dimension of the predictors is much larger than the sample size. In addition to the simulation studies, we apply the proposed methods to a public gene expression data sets, Parkinson's disease data([40]). A lot of work has been done to use the network information to improve the prediction and disease gene discovery, such as [38] and [49], where the networks are treated as non-directed graphs. In our study, we treat the network as directed graphs. The direction information comes from the hierarchical structure of the gene network and the known disease genes. Both simulation and real data results show that the proposed methods can achieve the desired goal and improve both prediction and gene discovery if the right network information is given.

This part is organized in six sections. Section 2 introduces the general method for network analysis. Section 3 the implements the proposed method. Section 4 presents some theoretical analysis of the method. Sections 5 and 6 present some simulation results and an application to two real micro-array data respectively.

1.2 Proposed method

1.2.1 Formulation for directed graphs

Consider a classification or regression problem, where pairs of observations are given. Suppose $(\boldsymbol{x}_i, y_i)_{i=1}^n$ are the observation pairs, where $\boldsymbol{x}_i \in \mathcal{R}^n$ is the predictor and $y_i \in \mathcal{R}$ is the response, let $\boldsymbol{f} = (f_1, f_2, \ldots, f_m)$ be a vector of decision or regression functions with its m component $f_m(\boldsymbol{x}) = \sum_{j=1}^p \beta_j h_{mj}(\boldsymbol{x})$. This generic representation includes a flexible representation through basis $\{h_j(\boldsymbol{x})\}_{j=1}^p$ and a linear representation when $h_j(\boldsymbol{x}) = x_j$. Usually the dimension p can be much larger than the sample size n and a major task for statistical learning is to do model selection, i.e, select predictors

with strong effects on the response. In our formulation, the important predictors are those with nonzero coefficients. For both classification and regression, our goal is to select important predictors and build appropriate model. However, when the p is much larger than n, extra information among the predictors or the responses should be used to improve model selection accuracy.

In our setting, dependencies of predictors are defined by a directed graph, with each edge of the graph representing a causal relationship between the associated predictors. For example, a direction from node A to node B means A is a cause and B is an effect; in other words node A regularizes node B. A good model selection procedure should maintain this relationship. If the effect predictor B is presented in the model, the cause predictor A must be included in model. However the existence of a cause variable B does not imply the effect variable A is included in model. A typically example is analysis of variance situation. Usually the existence of two way interaction effects implies the existence of the main effects. However, the inverse is not true. Our goal is to find a way to make use of the dependencies of the predictors in model selection procedure.

Dependencies defined by the directed graph can effectively influence model selection. Figure 1(a) is a directed graphs with 11 nodes, where each node represents a predictor. The edge starting from node 1 and ending at node 2 indicates that node 1 will regularize node 2, which in turn means including predictor 2 in the model will enforce the model also includes predictor 1. In other words, any informative predictor implies that its parents are informative. By enforcing this restriction when building the model, if predictor 8 is included in the model, predictors 3 and 1 should also be included, which yields the whole pathway from the leave node 14 to the node 1. We call this a pathway extraction procedure. Moreover, node 1 itself is regularized by node 6 and a loop occurs through nodes 1, 5 and 6, where the graph constraints indicates that these three predictors should be included or excluded from the model

together, which indicates graph constraints can be applied to graphs with loops and achieve grouping effects. Furthermore, one node may have more than one parent nodes, see node 3 in Figure 1(a).

To make use of the available information, we propose the constraints: $I(\beta_{par(j)} \neq 0) \geq I(\beta_j \neq 0)$; $j = 1, \cdots, p$, for connectivity from $par(j)$ to j. The constraints fully capture the characteristics of the graph information and hence can achieve the goal of pathway extraction. If a loop occurs among some nodes, the constraints imply that the corresponding nodes are included in or excluded from a model all together as discussed in the above example which indicates the constraints can also be applied to achieve grouping effects. For efficient computation, we approximate these nonlinear constraints by $\lambda_2^{-1} g(\beta_{par(j)}) \geq \lambda_2^{-1} g(\beta_j)$ as $\lambda_2 \to 0^+$; $j = 1, \cdots, p$. Here $g(z) = \lambda_2$ if $|z| > \lambda_2$ and $g(z) = |z|$ otherwise, with $\lambda_2 > 0$ a thresholding parameter, see Figure 1(b). This leads to our constraints for a directed graph:

$$g(\beta_{par(j)}) - g(\beta_j) \geq 0; par(j) \neq \phi, j = 1, \ldots, p, \tag{1.1}$$

where ϕ indicates the empty set.

Placing the constraints in the frame work of regularization, we propose our regularized loss function as:

$$\min_{\boldsymbol{f}} \quad \sum_{i=1}^{n} L(y_i, \boldsymbol{f}(\boldsymbol{x}_i)) + \lambda J(\boldsymbol{f}), \tag{1.2}$$

subject to certain graph constraints over $\boldsymbol{f}$, where $L(\cdot, \cdot)$ is a negative likelihood or a loss function, $J(\boldsymbol{f})$ is a non-negative penalty functions, and λ is a vector of non-negative regularization coefficients. In our setting, $J(\boldsymbol{f}) = g(\boldsymbol{f})$, where g is truncated Lasso penalty([41]). Function g is piecewise linear but has two concave points at $f = \pm\lambda_2$. As a penalty function, it has many advantages over the convex

L_1 penalty(Lasso) and the non-convex Scad penalty.

The region defined by (1.1) is non-convex, see Figure 1.2(a), which is a challenge for efficient computation. To concur this problem, we find a convex subset of the non-convex region and do optimization within this subset instead of the whole region. With the estimates, we refine the subset and do optimization again until the value of the loss function converges. In other words, we try to solve a non-convex optimization problem by solving a convex problem iteratively. Suppose we are in iteration $k+1$ and the current estimation is $\hat{\boldsymbol{\beta}}^{(k)}$. Through a DC decomposition of g, the constraint can be written as $g_1(\beta_i) - g_2(\beta_i) - (g_1(\beta_{par(i)}) - g_2(\beta_{par(i)})) \leq 0$, which is equivalent to $g_1(\beta_i) + g_2(\beta_{par(i)}) \leq g_2(\beta_i) + g_1(\beta_{par(i)})$, where g_1 is linear in (β^+, β^-) and both g_1 and g_2 are convex in (β^+, β^-). However the region defined by this inequality is non-convex. To overcome the concavity, we linearize $g_2(\beta_i)$ in current estimation $\hat{\beta}_i^{(k)}$. By convexity of g_2, we can obtain a lower bound of the right hand side and the new feasible region is a convex, which is a subset of the original non-convex feasible region. The linearizion of g_2 has two situations according to the current value of $\hat{\beta}$. If $|\hat{\beta}_i^{(k)}| \leq \lambda$ be the current estimation, the new feasible region is $|\beta_i| + (|\beta_{par(i)}| - \lambda)_+ \leq 0 + |\beta_{par(i)}|$, which is $(|\beta_{par(i)}| - \lambda)_+ \leq 0 + |\beta_{par(i)}| - |\beta_i|$. Since the left hand side is nonnegative, this is equivalent to $|\beta_{par(i)}| \geq |\beta_i|$ and $|\beta_{par(i)}| \leq |\beta_{par(i)}| - |\beta_i| + \lambda$, which yield $|\beta_i| \leq \lambda$, see Figure 1.2(b). If $\hat{\beta}_i^{(k)} > \lambda$, then $|\beta_i| + (|\beta_{par(i)}| - \lambda)_+ \leq \beta_i - \lambda + |\beta_{par(i)}|$, which is $(|\beta_{par(i)}| - \lambda)_+ \leq \beta_i - \lambda + |\beta_{par(i)}| - |\beta_i|$. Again, the right hand side should be nonnegative, so $\beta_i + |\beta_{par(i)}| - |\beta_i| \geq \lambda$. Also $|\beta_{par(i)}| \leq \beta_{par(i)} + |\beta_{par(i)}| - |\beta_i| - \lambda + \lambda$, which is equivalent to $\beta_i - |\beta_i| \geq 0$, and so $\beta_i \geq 0$. Plug this in $\beta_i + |\beta_{par(i)}| - |\beta_i| \geq \lambda$, we have $|\beta_{par(i)}| \geq \lambda$, see Figure 1.2(c). Similarly, if $\hat{\beta}_i^{(k)} < -\lambda$, we have $|\beta_{par(i)}| \geq \lambda$ and $\beta_i \leq 0$, see Figure 1.2(d). Since g_1 is linear in (β^+, β^-), the constructed convex subset is a polyhedral in the space $(\boldsymbol{\beta}^+, \boldsymbol{\beta}^-)$, which is very important for efficient computing. Now we obtain a convex subset of the region induced by (1.1) and turn the non-convex optimization problem into a convex one.

1.2.2 Implementation

For classification, we examine two large margin classifiers: support vector machine(SVM) and ψ-learning. For binary classification, let $\boldsymbol{x}_i \in \mathcal{R}^p$, $i = 1, 2, \ldots, n$, be a vector of predictors or input and $\boldsymbol{y} \in \{-1, 1\}$ be a label variable or output, where p can be much larger than n. The classification rule $G(x) = \text{sign}[\boldsymbol{f}(\boldsymbol{x}_i) = \boldsymbol{x}_i^T\boldsymbol{\beta} + \beta_0]$ assigns $\boldsymbol{x}_i$ to the positive class if sign $\boldsymbol{f}(\boldsymbol{x}_i) > 0$. In the case of SVM, $L(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x})) = \sum_{i=1}^n \psi_{SVM}(y_i f(x_i))$, where $\psi_{SVM}(z) = (1 - z)_+$ is the hinge loss. In the case of ψ-learning, $L(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x})) = \sum_{i=1}^n \psi(y_i f(x_i))$, where $\psi(z) = -2z$ if $0 < z < 1$, 2 if $z \leq 0$ and 0 otherwise. For least square regression, $\boldsymbol{x}_i \in \mathcal{R}^p (i = 1, 2, \ldots, n)$, $\boldsymbol{y} \in \mathcal{R}$ and $L(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x})) = \frac{1}{2n}\sum_{i=1}^n (y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2$.

Let $\boldsymbol{S}(\boldsymbol{f}) = \boldsymbol{L}(\boldsymbol{y}, \boldsymbol{f}(\boldsymbol{x})) + \lambda_1 \boldsymbol{f}$ and it can be written as

$$\boldsymbol{S}(\boldsymbol{\beta}) = \boldsymbol{S}_1(\boldsymbol{\beta}) - \boldsymbol{S}_2(\boldsymbol{\beta}), \tag{1.3}$$

where both $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$ are convex function. In the case of SVM, $\boldsymbol{S}_1 = \sum_{i=1}^n \psi_{SVM}(y_i f(x_i)) + \lambda_1 \sum_{j=1}^p g_1(\beta_j)$ and $\boldsymbol{S}_2 = \lambda_1 \sum_{j=1}^p g_2(\beta_j)$. In the case of ψ-learning, $\boldsymbol{S}_1 = S_{1,1} + S_{2,1} = \sum_{i=1}^n \psi_1(y_i f(x_i)) + \lambda_1 \sum_{j=1}^p g_1(\beta_j)$ and $\boldsymbol{S}_2 = S_{1,2} + S_{2,2} = \sum_{i=1}^n \psi_2(y_i f(x_i)) + \lambda_1 \sum_{j=1}^p g_2(\beta_j)$. For regression, $\boldsymbol{S}_1 = \frac{1}{2n}\sum_{i=1}^n (y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2 + \lambda_1 \sum_{j=1}^p g_1(\beta_j)$ and $\boldsymbol{S}_2 = \lambda_1 \sum_{j=1}^p g_2(\beta_j)$. In all three cases, $\psi_1(z)$ is 0 if $z \geq 1$ and $-2(z-1)$ otherwise; $\psi_2(z)$ is 0 if $z \geq 0$ and $-2z$ otherwise; $g_1(z) = |z|$ and $g_2(z) = (z - \lambda_2)_+$.

With these decompositions and refined convex feasible regions, we treat the non-convex minimization (1.3) by solving a sequence of convex problem iteratively. In iteration step $k + 1$, we solve

$$\min_{\boldsymbol{\beta} \in C(\hat{\boldsymbol{\beta}}^{(k)})} \boldsymbol{S}_1(\boldsymbol{\beta}) - \langle \boldsymbol{\beta}, \nabla \boldsymbol{S}_2(\hat{\boldsymbol{\beta}}^{(k)}) \rangle, \tag{1.4}$$

where $\langle ., . \rangle$ is the inner product, $\nabla \boldsymbol{S}_2(\hat{\boldsymbol{\beta}}^{(k)})$ is the gradient vector of $\boldsymbol{S}_2(\boldsymbol{\beta})$ at the

k-th step solution $\hat{\boldsymbol{\beta}}^{(k)}$, and $C(\hat{\boldsymbol{\beta}}^{(k)})$ is the feasible region induced by $\hat{\boldsymbol{\beta}}^{(k)}$. The cost function of SVM and ψ-learning are piecewise linear in $\boldsymbol{\beta}$ and the feasible region is a finite intersect of polyhedrons. Thus the problem can be solved through linear programming(LP). For regression, the cost function is quadratic and can be solved through quadratic programming(QP). The algorithm is given as follows.

Algorithm 1:

Step 1: Supply an initial value $\hat{\boldsymbol{\beta}}^{(0)}$, compute $\hat{\boldsymbol{\beta}}^{(1)}$ and $\boldsymbol{S}(\hat{\boldsymbol{\beta}}^{(1)})$. Specify precision tolerance level $\epsilon > 0$.

Step 2: At iteration $k+1$, compute $\hat{\boldsymbol{\beta}}^{(k+1)}$, $C(\hat{\boldsymbol{\beta}}^{(k+1)})$ and $\boldsymbol{S}(\hat{\boldsymbol{\beta}}^{(k+1)})$ by solving ((1.4)) based on $\hat{\boldsymbol{\beta}}^{(k)}$.

Step 3: Terminate if $\boldsymbol{S}(\hat{\boldsymbol{\beta}}^{(k)}) - \boldsymbol{S}(\hat{\boldsymbol{\beta}}^{(k+1)}) < \epsilon$ and the estimate $\hat{\boldsymbol{\beta}} = \hat{\boldsymbol{\beta}}^{(k+1)}$ is the final solution, otherwise,

Step 4: Go to Step 2.

In **Algorithm 1**, we solve a serials of QP problem in regression, which is not efficient if the dimension is large and will lead to a large quadratic term matrix. For linear regression, we suggest using the coordinate-wise descent method instead of QP. Suppose $Y \in \mathcal{R}$ is response, a vector $X \in \mathcal{R}^p$ are the predictors, the regression function can be approximate by a linear model $E(Y|X = x) = \beta_0 + x^T\boldsymbol{\beta}$. If n observation pairs (x_i, y_i) are available, we have the following optimization problem in **step 2** of **Algorithm 1**.

$$\min_{(\beta_0,\boldsymbol{\beta})\in C(\hat{\boldsymbol{\beta}}^{(k)})} R_\lambda(\beta_0,\boldsymbol{\beta}) = \min_{(\beta_0,\boldsymbol{\beta})\in C(\hat{\boldsymbol{\beta}}^{(k)})} (\frac{1}{2n}\sum_{i=1}^{n}(y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2 - \langle\boldsymbol{\beta}, \nabla\boldsymbol{S}_2(\hat{\boldsymbol{\beta}}^{(k)})\rangle) + \lambda_1\sum_{j=1}^{p}|\beta_j| \tag{1.5}$$

where $\boldsymbol{S}_2 = \lambda_1\sum_{j=1}^{p} g_2(\beta_j)$ and $C(\hat{\boldsymbol{\beta}}^{(k)})$ is the feasible region induced by current estimation of $\boldsymbol{\beta}$. (1.5) is a convex problem with constraints, which can be solved by coordinate-wise descent method. Suppose the current estimates are $\hat{\boldsymbol{\beta}}$, we can

partially optimize (1.5) with respect to β_j. Without the constraints, the minimizer of this one dimensional problem is $\tilde{\beta}_j = \frac{S(\frac{1}{n}\sum_{i=1}^{n} x_{ij}(y_i - \tilde{y}_i^{(j)}), \lambda_1)}{\frac{1}{n}\sum_{i=1}^{n} x_{ij}^2}$, where $S(z, \gamma)$ is the soft-thresholding operator and $\tilde{y}_i^{(j)}$ is the current fitted value of y_i without using predictor j. Combining this one dimensional update with the constraints on this dimension, we obtain the updates subject to the constraints. If $\max_{i \in chi\{j\}}(|\hat{\beta}_i^{(k)}|) \geq \lambda_2$, then the constraint is $|\beta_j| \geq \lambda_2$ and the update subject to constraint is $\hat{\beta}_j^{(k+1)} \leftarrow \text{Sgn}(\tilde{\beta}_j^{(k)}) max(|\tilde{\beta}_j^{(k)}|, \lambda_2)$. If $\max_{i \in chi\{j\}}(|\hat{\beta}_i^{(k)}|) < \lambda_2$, the constraint is $|\beta_j| > \max_{i \in chi\{j\}}(|\hat{\beta}_i^{(k)}|)$ and the update subject to constraint is $\hat{\beta}_j^{(k+1)} \leftarrow \text{Sgn}(\tilde{\beta}_j) max(|\tilde{\beta}_j|, \max_{i \in chi\{j\}}(|\hat{\beta}_i^{(k)}|))$. With the new updates, we have the following algorithm.

Algorithm 2:

Step 1: Let $\hat{\boldsymbol{\beta}}^{(0)}$ be the original estimator(lasso), compute $\hat{\boldsymbol{\beta}}^{(1)}$ and $R(\hat{\boldsymbol{\beta}}^{(1)})$. Specify precision tolerance level $\epsilon > 0$.

Step 2: At iteration $k+1$,

Loop: j from 1 to p, update $\hat{\beta}_j^{(k)}$

Compute $R(\hat{\boldsymbol{\beta}}^{(k+1)})$ based on $\hat{\boldsymbol{\beta}}^{(k+1)}$.

Step 3: Stop if $R(\hat{\boldsymbol{\beta}}^{(k)}) - R(\hat{\boldsymbol{\beta}}^{(k+1)}) < \epsilon$, and set $\hat{\boldsymbol{\beta}} = \hat{\boldsymbol{\beta}}^{(k+1)}$, otherwise,

Step 4: Go to Step 2.

Algorithm 2 has several advantages over **Algorithm 1**. First, it is easy to implement. In **Algorithm 1**, it is difficult to specify the QP problem when there are thousands of constraints. In **step 2** of **Algorithm 2**, we only need to solve a one dimensional optimization problem with a simple one dimensional constraint, which leads to a easy updating formula. Second, if the dimension is extremely high or if we have thousands of edge on the graph, **Algorithm 1** may break down because we need to maintain a large matrix in the QP problem. Even with the best QP solver, Cplex for example, it is also time consuming passing such a big matrix between our program and the QP solver. In the coordinate-wise descent algorithm, we solve a one dimensional problem in each time and there is no need of large memory. This make

our algorithm works for a graph with thousands of nodes and thousands of edges. Third, the coordinate-wise descent usually converges fast especially if a good starting point is given. [37] shows that the coordinate-wise descent method is comparable to the homotopy method.

1.3 Theory

This section establishes the finite termination property of **Algorithm 1**. For regression, we prove that $\hat{\boldsymbol{\beta}}(\boldsymbol{\lambda})$ can achieve consistency with regard to model selection.

Lemma 1.1

lemma1 The *Algorithm*1, $\hat{\boldsymbol{\beta}}^{(k)} \in C(\hat{\boldsymbol{\beta}}^{(k)})$ for k until converges. □

Theorem 1.1

In **Algorithm 1**, $\boldsymbol{S}(\hat{\boldsymbol{\beta}}^{(k)}) < \boldsymbol{S}(\hat{\boldsymbol{\beta}}^{(k-1)})$, for k until converges and the algorithm converges in a finite number of steps. □

In each iteration of **Algorithm 1**, we solve a quadratic programming problem. The refined convex region is obtained by a DC decomposition of the nonconvex region based on the current estimation. **Lemma 1** shows that the refined convex region contains the current estimation. This conclusion make sense and is critical for the algorithm to converge.

Theorem 1 shows the value of the loss function decreases between each iteration. For a non-constrained DC optimization problem, this property follows directly from the design of the algorithm. In the constraint scenario, the refined region contains the current estimation and the minimizer in the new region will make the value of loss function smaller than the current estimation by intuition. Once the value of loss function does not change between two iterations, the algorithm converges. A strict proof is provided in Section 7.

Since the value of the loss function is strictly decreasing, the algorithm will converge in finite step, as showed in Theorem 1. However the speed of converge depends on the initial value. In the implementation, we choose the solution of SVM as the starting point of constrained SVM, and choose the solution of constrained SVM as the starting point of constrained ψ-learning. For regression, we choose the solution of lasso as our starting points. The choice of starting point only has influence on the speed of the algorithm, but not the solution. Coordinate descent is an easy way to solve a convex minimization problem. However, it may not converge to the global minimizer. [36] examines the fused lasso and showed that the coordinate-wise descent method does not work in this case. However coordinate-wise algorithms work fine for the lasso, the grouped lasso and elastic net etc. Key to validity of coordinate-wise algorithms is the separability of penalty function $\sum_{j=1}^{p} g_i(\beta_j)$, a sum of functions of each individual parameter. In **step 2** of **Algorithm 2**, the loss function is separable except that we have constraints. Fortunately, all the constraints are separable and we can show the coordinate-wise descend algorithm also works in our method, see section 7.

For regression, our method is consistent in variables selection. Let $A = \{j : |\beta_j|\} = 0$ and $A^0 = \{j : |\beta_j| > 0\}$ where A^0 is the true set of nonzero coefficients with $|A^0| \leq n$. Let $c_{min}(A) > 0$ be the smallest eigenvalue of $\boldsymbol{X}_A^T \boldsymbol{X}_A / n$ and $\gamma_{min} = \min\{|\beta_k^0| : k \in A^0\}$. γ_{min} is the resolution level or level of difficulty of feature selection. A small value of γ_{min} means difficulty, see [41]. Here γ_{min} and A^0 are allowed to dependent on n, p. The following result is established for the estimate obtained from **Algorithm 1** with the Lasso initial estimate.

Theorem 1.2 (Consistency)

For least square regression, assume that the error is distributed according to $N(0, \sigma^2)$,

if $\min_{|A|\leq|A^0|} c(A) > \frac{3\lambda_2}{2}$, then for any n and p,

$$P(\hat{A}(\boldsymbol{\lambda}) \neq A^0) \leq P(\hat{\boldsymbol{\beta}}(\boldsymbol{\lambda}) \neq \hat{\boldsymbol{\beta}}^{(ols)}) \leq |A^0|\Phi\left(\frac{n^{1/2}(\lambda_2 - \gamma_{min})}{2\sigma c_{min}(A^0)^{-1/2}}\right) + |A|\Phi\left(\frac{n\lambda_1}{\sigma \max_{j=1,\ldots,p}\|x\|_j}\right) \quad (1.6)$$

where $\hat{\boldsymbol{\beta}}(\boldsymbol{\lambda})$ is the estimate obtained by **Algorithm 1**, $\hat{\boldsymbol{\beta}}^{(ols)} = (\hat{\beta}_1^{(ols)}, \ldots, \hat{\beta}_p^{(ols)}, \underbrace{0, \ldots, 0}_{p-p_0})$ is the unbiased least square estimate base on A^0, $\Phi(z) = \int_\infty^z \exp(-u^2/2)du$ is the cumulative distribution function of $N(0,1)$, and $\|\boldsymbol{x}_j\|$ is the L_2-norm of $\boldsymbol{x}_j$. Moreover, if

(i) $\frac{nc_{min}(A^0)(\gamma_{min}-\lambda_2)^2}{2\sigma^2} - \log|A^0| \to \infty, \quad \gamma_{min} > \lambda_2;$

(ii) $\frac{n\lambda^2}{2\sigma^2\|\boldsymbol{x}_j\|/n} - \log p \to \infty,$

then

$$P(\hat{A} \neq A^0) \leq P(\hat{\boldsymbol{\beta}} \neq \hat{\boldsymbol{\beta}}^{ols}) \to 0, \quad as \quad n, p \to +\infty. \quad (1.7)$$

□

Theorem 2 shows that the solution $\hat{\boldsymbol{\beta}}$ can recover the true model as well as $\hat{\boldsymbol{\beta}}^{ols}$, without knowing the true model A^0 when $p, n \to \infty$, where $p = O(\exp(n\lambda_1^2))$, or $n^{-1}\log p \to 0$. The non-convex penalty function enables to reconstruct the unbiased least squares estimate based on true model ([41]). However, our method takes account the structure information which is important when p is larger than n. Section 1.4.1 and Section 1.4.2 give some examples that indicate the constrained regression gain extra power in variable selection over the non-convex penalized regression.

1.4 Simulation studies

For classification, we compare the performance of L_1-SVM, SVM with their non-convex counterpart, CL_1-SVM and $C\psi$. All the methods are implemented in R

using package **Rcplex**. For regression, we compare the performance of the Lasso and the proposed constrained penalized regression.

1.4.1 Example 1

Consider a simple network defined by a tree consisting 26 nodes and 25 edges, which mimics a regulatory gene network centered at gene 1, as displayed in Figure 3. The data is generated as: X_1 follows $N(0,1)$; assume gene s and each of its regulated genes follow a bivariate normal distribution with correlation 0.7. Thus, the expression level of each regulated gene is distributed as $N(0.7X_s, 0.51)$. For classification, the response Y is generated according to a logistic regression model: $\log \frac{Pr(Y=1|X)}{1-Pr(Y=1|X)} = X^T\boldsymbol{\beta} + \beta_0$, $\beta_0 = 2$, where X is the vector of predictors.

Three sets of informative genes are characterized by three subnetworks. The three sets of true coefficients, $\boldsymbol{\beta}$'s, are specified as:

1. Genes $1, 2, 5, 6, 14, 15, 16$ are informative.

$$\boldsymbol{\beta} = (5, 5/\sqrt{3}, 0, 0, 5/3, 5/3, \underbrace{0, \ldots, 0}_{7}, 5/\sqrt{2}, 5/\sqrt{3}, 5/\sqrt{3}, \underbrace{0, \ldots, 0}_{10}).$$

2. Genes $1, 3, 7, 17$ are informative.

$$\boldsymbol{\beta} = (5, 0, 5/\sqrt{3}, 0, 0, 0, 5/\sqrt{12}, \underbrace{0, \ldots, 0}_{9}, 5/\sqrt{24}, \underbrace{0, \ldots, 0}_{9}).$$

3. Genes $1, 2, 5, 14$ are informative.

$$\boldsymbol{\beta} = (5, 5/\sqrt{3}, 0, 0, 5/3, \underbrace{0, \ldots, 0}_{8}, 5/3\sqrt{2}, \underbrace{0, \ldots, 0}_{12}).$$

For classification, 50 observations are generated for training, 50 for tuning, and $10,000$

for testing under each case. We search a wide range of tuning parameter λ_1, but for λ_2 we only search 12 of them. So the computing effort is not much more than the SVM with L1 penalty. After obtaining a classifier from the training set, we apply it to the tuning set, and identify $\hat{\lambda} = (\hat{\lambda}_1, \hat{\lambda}_2)$ that produces the minimal classification error on the tuning set. Then we use the classifier corresponding to $\hat{\lambda}$ to find the classification error on the testing data. Repeat the entire process 100 times. The critical value is defined as the symmetric difference of selected genes and the true informative genes.

In Table 1.1, the mean of the testing classification error and the critical value are reported. SVM with non-convex penalty generates sparser model than L_1SVM and also has better prediction accuracy. Constrained SVM has higher prediction accuracy than SVM with non-convex penalty but the drawback is it includes too many noisy genes and the selection is relatively bad. Finally constrained ψ-learning correct this problem and gain an improvement for both prediction accuracy and discovery of disease genes. Thus the result shows the proposed method can achieve the desired goal for all 3 cases.

In regression, consider the network containing three replications of the network described in Figure 1.4. The new network contains three directed trees centered at 1, 27, 53, see Figure 1.3. Three sets of informative genes are described by three subnetworks. The three sets of true coefficients, $\boldsymbol{\beta}$'s, are specified as:

1. Genes $1, 2, 5, 6, 14, 15, 16, 27, 28, 31, 32, 40, 41, 42, 53, 54, 57, 65, 66, 67$ are informative.

$$\begin{aligned}\boldsymbol{\beta} \;=\; & (5/\sqrt{3}, 5/2, 0, 0, 5/\sqrt{2}, 5/3, \underbrace{0, \ldots, 0}_{7}, 5/3, 5/\sqrt{3}, 5, \underbrace{0, \ldots, 0}_{10}, \\ & 5/\sqrt{3}, 5/2, 0, 0, 5/\sqrt{2}, 5/3, \underbrace{0, \ldots, 0}_{7}, 5/3, 5/\sqrt{3}, 5, \underbrace{0, \ldots, 0}_{10}, \\ & 5/\sqrt{3}, 5/2, 0, 0, 5/\sqrt{2}, 5/3, \underbrace{0, \ldots, 0}_{7}, 5/3, 5/\sqrt{3}, 5, \underbrace{0, \ldots, 0}_{10})\end{aligned}$$

2. Genes $1, 3, 7, 17, 27, 29, 33, 43, 53, 55, 59, 69$ are informative.

$$\begin{aligned}\boldsymbol{\beta} = (&5/\sqrt{24}, 0, 5/\sqrt{12}, 0, 0, 0, 5/\sqrt{3}, \underbrace{0, \ldots, 0}_{9}, 5, \underbrace{0, \ldots, 0}_{9},\\ &5/\sqrt{24}, 0, 5/\sqrt{12}, 0, 0, 0, 5/\sqrt{3}, \underbrace{0, \ldots, 0}_{9}, 5, \underbrace{0, \ldots, 0}_{9},\\ &5/\sqrt{24}, 0, 5/\sqrt{12}, 0, 0, 0, 5/\sqrt{3}, \underbrace{0, \ldots, 0}_{9}, 5, \underbrace{0, \ldots, 0}_{9})\end{aligned}$$

3. Genes $1, 2, 5, 14, 27, 28, 31, 40, 53, 54, 57, 66$ are informative.

$$\begin{aligned}\boldsymbol{\beta} = (&5/\sqrt{18}, 5/3, 0, 0, 5/\sqrt{3}, \underbrace{0, \ldots, 0}_{8}, 5, \underbrace{0, \ldots, 0}_{12},\\ &5/\sqrt{18}, 5/3, 0, 0, 5/\sqrt{3}, \underbrace{0, \ldots, 0}_{8}, 5, \underbrace{0, \ldots, 0}_{12},\\ &5/\sqrt{18}, 5/3, 0, 0, 5/\sqrt{3}, \underbrace{0, \ldots, 0}_{8}, 5, \underbrace{0, \ldots, 0}_{12})\end{aligned}$$

The response Y is generated according to $Y = X^T\boldsymbol{\beta} + 2 + \sigma N(0, 1)$, where X is the vector of predictors and $\sigma = 4$.

For regression, we simulate 50 observations for training, 50 for tuning, and $10,000$ for testing under each scenario. So the problem now is a $n < p$ problem and we compare the performance of our method to the lasso. The result is summarized in Table 1.2. It shows that in the $n < p$ case, the proposed method is much better than the lasso which does not consider the network information. Furthermore our method performs better in discovering informative genes if the correct network information is supplied. This may due to the following reasons. On the one hand it hard to miss informative genes on the pathway due to the constraints, especially if the informative genes at the end of the pathway have strong effects. On the other hand, it is also hard to select non-informative genes. Because selecting non-informative genes may

result in many redundant genes being selected along the pathway, which will lead to poor models in prediction. And the tuning procedure will leave out these kind of models automatically. So the proposed method can make use of the information from both informative and non-informative genes, and thus gain extra power in variable selection as well as prediction.

1.4.2 Example 2

Consider a more complex network originating from gene 1 as displayed in Figure 1.5, where multiple parents exist. For example, gene 32 has parent genes 23 and 3. Our goal is to identify disease genes that play a critical role in mediating other genes in multiple biological processes even if their direct effect on the outcome is weak. The data is generated largely as in the simple network. Here, two cases are considered: (1) genes 1, 2, and 3 have weak effects ($\beta_{gene1} = \beta_{gene2} = \beta_{gene3} = 0.1$), and leaf gene 32 has strong effect ($\beta_{gene32} = 10$); (2) the three disease genes have the same effect as in scenario (1) whereas leaf gene 34 had strong effect ($\beta_{gene34} = 10$).

Table 1.3 indicates that SVM with con-convex penalty has the best prediction accuracy. In contrast, constrained ψ-learning and constrained SVM are better in discovering disease genes. Although the disease genes have very weak effects on the outcome, the proposed method can discover almost all of them. In this situation, SVM is better in prediction due to sparser model but misses more disease genes with weak effects. By enforcing the network constraints in SVM and ψ-learning, the genes tend to be selected along the pathway. That is if one gene is selected, all the genes along the pathway should also be selected, even though some of them may have very weak effects on the outcome. The simulation results indicate that the proposed method definitely achieve this goal and can perform pathway extraction on the directed graphs.

1.4.3 Example 3 (Analysis of variance)

In a situation as in ANOVA, each of the explanatory factor may be represented by a group of derived input variables. Then variable selection amounts to selecting important factors(groups of variables) rather than individual derived variables. [47] propose a grouped variable selection procedure. However there is a drawback with this approach. The grouped variable selection procedure treat the groups equally, which is usually not true. For example, the existence of group of two-way interaction effects imply that the main effect groups exist. So there is a natural hierarchical structure between the groups of variables. Both ANOVA and the grouped variable selection procedure can not capture this characteristics. The model generated by these methods can not guarantee the hierarchical structure in the variables are maintained.

Our graph based methods can easily take account the hierarchical relationship between the groups of variables. We treat all the variables as nodes and the hierarchical relationship as edges. So there is an edge starting from main effect nodes and ending at two-way interaction nodes. In this way, we can easily set up the graphs for ANOVA problems. In this section, we set up two simulation examples to show how our method can capture the hierarchical structure of the predictors and achieve improvement in prediction.

(1) In model 1, 8 variables $Z_1, Z_2, \ldots, Z_8$ are first simulated according to a centered normal distribution with covariance between Z_i and Z_j being $0.5^{|i-j|}$. Then Z_i is trichotomized as $0, 1, 2$ if it is smaller than $\Phi^{-1}(1/3)$, larger than $\Phi^{-1}(2/3)$ or

between. The true regression equation is

$$\begin{aligned} Y &= 0.5I(Z_1=1)+0.5I(Z_1=0)+0.5I(Z_2=1)+0.5I(Z_2=0)+0.5I(Z_3=1) \\ &\quad +0.5I(Z_3=0)+1I(Z_1=1,Z_2=1)+1.5I(Z_1=1,Z_2=0) \\ &\quad +2I(Z_1=0,Z_2=1)+2.5I(Z_1=0,Z_2=0)+1I(Z_1=1,Z_3=1) \\ &\quad +1.5I(Z_1=1,Z_3=0)+2I(Z_1=0,Z_3=1)+2.5I(Z_1=0,Z_3=0) \\ &\quad +1I(Z_2=1,Z_3=1)+1.5I(Z_2=1,Z_3=0)+2I(Z_2=0,Z_3=1) \\ &\quad +2.5I(Z_2=0,Z_3=0)+3(I(Z_1=1,Z_2=1,Z_3=1),\ldots, \\ &\quad +I(Z_1=0,Z_2=0,Z_3=0))+\epsilon \end{aligned}$$

where $\epsilon \sim N(0, 2^2)$. 300 observations are collected for each simulation data set.

(2) In model 2, 30 random variables $Z_1, Z_2, \ldots, Z_{30}$ and W are independently generated from a standard normal distribution. The covariates are defined as $X_i = (Z_i + W)/\sqrt{2}$. The response follows

$$Y = X_3+X_3^2+X_3^3+3X_3^4+\frac{1}{3}X_6-X_6^2+\frac{2}{3}X_6^3+2X_6^4+\frac{1}{6}X_9+\frac{1}{3}X_9^2+2X_9^3+4X_9^4+\epsilon, \quad (1.8)$$

where $\epsilon \sim N(0, 2^2)$. 100 observations are collected for each run.

1.5 Applications to micro-array data

To evaluate the performance of the proposed method in the real world, we applied it to two data sets for breast cancer metastasis ([46] and [33])(available at NCBI Genbank GEO database (`http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE2034`)) and Parkinson's disease ([40]) (Gene Expression Omnibus: GSE6613; `http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi`).

1.5.1 Parkinson's disease

Parkinsons disease (PD) progresses relentlessly and affects five million people worldwide. Laboratory tests for PD are critically needed for developing treatments designed to slow or prevent progression of the disease. [40] performed a transcriptome-wide scan in 105 individuals to interrogate the molecular processes perturbed in cellular blood of patients with early-stage PD. Their data set includes disease status and expression levels of $22,283$ genes from 105 patients, 50 patients with Parkinson's disease, 33 with neurodegenerative diseases other than PD, and 23 as healthy controls.

The gene network information is obtained from two sources: (1) the network structure from [38] ,which combines 33 Kyoto Encyclopedia of Genes and Genomes(KEGG) regulatory pathways and contains $1,523$ genes and $6,865$ edges; (2) the Parkinson's disease KEGG pathway(PD-KEGG,) which uncovers the interactions of 27 PD disease genes that lead to the disease . A total of 12 out of 27 PD disease genes fall into the Li and Li network structure. They are UBB, $UBE1$, $CASP9$, $CASP3$, $APAF1$, $CYCS$, $PARK2$, $GPR37$, $SEPT5$, $SNCAIP$, $SNCA$, and TH. We focus our analysis on the subnetwork grown from the 12 disease genes by using the Li and Li network information, named as PD-net, which consists of four components: (1) the 6th-order-neighbor-subnetwork of UBB(A direct neighbor of UBB is defined as a 1st-order-neighbor; a direct neighbor of a direct neighbor of UBB is defined as 2nd-order-neighbor; and so on.); (2) the 3rd-order-neighbor-subnetwork of $CASP9$; (3) the isolated four-gene-subnetwork including $PARK2, GPR37, SEPT5$ and $SNCAIP$; and (4) the isolated two-gene-subnetwork including $SNCA$ and TH. The network is showed in Figure 1.8, which include a total of 181 genes. The direction information is obtained by the hierarchical structure in Figure 1.9. In Figure 1.9, known disease gene are in blue. Also the genes on the first level are considered as centers. So there are 5 centers on this network. The direction is defined from higher level to lower level. Since there are 7 levels at most, the longest pathway has length

6.

We apply L1-SVM, SVM with non-convex penalty, constrained SVM and constrained ψ-learning over PD network for 50 runs. In each run, the data set is randomly split into training, tuning, and test set with 40, 20, and 45 observations respectively. For the training data, half of them are from the case group and the other half are from control group as well as the tuning set. The expression level of each gene is normalized to have mean 0 and standard deviation 1 across samples. The performance of each method was evaluated on each test set by the classification error, the selection of PD genes, and critical values averaged over 50 runs. We calculate the classification error corresponding to each pair of tuning parameters. The value that generated the minimal averages error is used to fit the final model on the test data. The result was summarized in Table 1.5. The table shows that by enforcing the constraints on L1SVM, the prediction error decrease from 45.38% to 42.49% and if we further combine the constraints with ψ-learning, the prediction error decrease to 38.58%. Also constrained ψ-learning discover 5.80 disease genes with a critical value 30.36. Although SVM with non-convex penalty has a critical value 21.64, it misses about 2 more disease genes than constrained ψ-learning. Thus a smaller critical value for SVM with non-convex penalty is probably due to a sparser model. In summary, this real data example shows the proposed method can efficiently capture the biology information on the directed gene network and achieve improvement for both prediction and gene discovery.

1.5.2 Breast cancer metastasis

[46] develop a gene-expression-based algorithm and use it to provide quantitative predictions on the disease outcome for patients with lymph-node-negative breast cancer. Through this algorithm, they reveal a 76-gene signature that can predicts distant tumor recurrence. This signature can be applied to all lymph-node-negative patients

independently of age, tumor size and grade, and ER status. The 76 genes in the signature belong to many functional classes, which suggests that different pathways could lead to disease progression. Three tumor suppressor genes, $TP53, BRCA1$, and $BRCA2$ are included in the 76-gene signature. They are known to prevent uncontrolled cell proliferation, and to play a critical role in repairing the chromosomal damage. The malfunction of these genes leads to increasing risk of breast cancer. The data they used consists expression levels of $8,141$ genes from 286 patients, 107 of whom developed metastasis within a 5-year follow-up after surgery. For classification purpose, we include two groups in our study, one group with the patients who developed cancer metastasis and one group with patients had been free of metastasis for more than 8 years. For regression purpose, we include all the observations and use survival time as the response.

The network information is obtained from the protein-protein interaction (PPI) network previously used in ([33]). They obtained the PPI network by assembling a pooled data set comprising $57,235$ interactions among $11,2034$ proteins and curation of the literature. We restrict our analysis to the subnetwork consisting of the direct neighbors of the three tumor suppressor genes, which is called BC-1nb-net in ([48]). A total of 294 genes that belong to BC-1nb-net have observed expression levels and 18 of them also belong to the 76 genes revealed in ([46]). In stead of one center network in ([48]), $TP53, BRCA1$ and $BRCA2$ are considered as the center genes of the network. Among these three genes, we enforce a loop $TP53 \to BRCA1 \to BRCA2 \to TP53$, which is displayed in Figure 1.10.

We run L1-SVM,SVM with non-convex penalty, constrained SVM and constrained ψ-learning over this data for 50 times. In each time, we randomly split the data into training, tuning, and testing set with 95 observations, 95 observations, and 96 observations respectively. The expression level of each gene is normalized to have mean 0 and standard deviation 1 across samples. Given any value from a prespecified

set of wide-ranging values for the tuning parameter λ, we obtain the classifier on the training set and apply $f(\hat{\lambda})$ on tuning data. The value that corresponded to the minimal classification error on the tuning set is identified as $\hat{\lambda}$. Then we apply the classifier $\hat{f}(\hat{\lambda})$ on the test set to evaluate its performance. In practice, we have 100 possible numbers for λ_1 and 8 for λ_2. We compare the performance of the proposed method Cψ with that of CSVM, L1-SVM and SVM with non-convex penalty in terms of classification error, number of selected disease genes and critical number of selected genes averaged over 50 replications.

For regression, we divide the data into training(60), tuning (60) and testing (166). The gene expression matrix is normalized and take log of survival time are centered. From Table 1.6, lasso only discovered 0.25 of them and elastic net discover 0.98 but with much large model size. Under log scale, the difference of predicted MSE is small.

1.6 Discussions

1.7 Technical proofs

Proof of Lemma 1:

If $C(\hat{\beta}_i^{(k-1)})$ has the region in Figure 1.2(b), then $\hat{\boldsymbol{\beta}}^{(k)} \in C(\hat{\boldsymbol{\beta}}^{(k-1)})$ implies $|\hat{\beta}_i^{(k)}| < \lambda_2$ and $|\hat{\beta}_{par(i)}^{(k)}| > |\hat{\beta}_i^{(k)}|$, which indicates $(\hat{\beta}_i^{(k)}, \hat{\beta}_{par(i)}^{(k)}) \in C(\hat{\beta}_i^{(k-1)})$.

If $C(\hat{\beta}_i^{(k-1)})$ has the region in Figure 1.2(c), then $\hat{\boldsymbol{\beta}}^{(k)} \in C(\hat{\boldsymbol{\beta}}^{(k-1)})$ implies $\hat{\beta}_i^{(k)} > 0$ and $|\hat{\beta}_{par(i)}^{(k)}| > \lambda_2$. If $0 < \hat{\beta}_i^{(k)} \leq \lambda_2$, then $C(\hat{\beta}_i^{(k)})$ is Figure 1.2(b) and obviously $(\hat{\beta}_i^{(k)}, \hat{\beta}_{par(i)}^{(k)}) \in C(\hat{\beta}_i^{(k)})$. If $\hat{\beta}_i^{(k)} > \lambda_2$, then $C(\hat{\beta}_i^{(k)})$ is Figure 1.2(c), which implies $C(\hat{\beta}_i^{(k-1)})$ and $C(\hat{\beta}_i^{(k)})$ are the same and the result follows. For the case $C(\hat{\beta}_i^{(k-1)})$ has the region in Figure 1.2(d), a similar argument can apply.

For all three cases, the result holds for all constraints and hence $\hat{\boldsymbol{\beta}}^{(k)} \in C(\hat{\boldsymbol{\beta}}^{(k)})$.

Proof of Theorem 1: First we show that the objective function is decreasing during

each iteration. The objective function can be written as $\boldsymbol{S} = (\boldsymbol{S}_{11}+\boldsymbol{S}_{21})-(\boldsymbol{S}_{12}+\boldsymbol{S}_{22}) = \boldsymbol{S}_1 - \boldsymbol{S}_2$ and in step $k+1$, we solve optimization problem:
$\min_{\boldsymbol{\beta}\in C(\hat{\beta}^{(k)})} \boldsymbol{L}^{(k+1)}(\beta) = \min_{\boldsymbol{\beta}\in C(\hat{\beta}^{(k)})} \left\{ \boldsymbol{S}_1(\boldsymbol{\beta}) - \langle \boldsymbol{\beta} - \hat{\beta}^{(k)}, \nabla \boldsymbol{S}_2(\hat{\beta}^{(k)})\rangle - \boldsymbol{S}_2(\hat{\beta}^{(k)}) \right\}$,where $C(\hat{\beta}^{(k)})$ is the feasible region induced by $\hat{\beta}^{(k)}$. Note that $\hat{\boldsymbol{\beta}}^{(k-1)} \in C(\hat{\boldsymbol{\beta}}^{(k-1)})$ by Lemma 1, which implies $\boldsymbol{L}^{(k)}(\hat{\boldsymbol{\beta}}^{(k-1)}) \geq \boldsymbol{L}^{(k)}(\hat{\boldsymbol{\beta}}^{(k)})$. Now we will show $\boldsymbol{L}^{(k)}(\hat{\boldsymbol{\beta}}^{(k)}) > \boldsymbol{L}^{(k+1)}(\hat{\boldsymbol{\beta}}^{(k)})$. $\boldsymbol{L}^{(k)}(\hat{\boldsymbol{\beta}}^{(k)}, \hat{\boldsymbol{a}}^{(k)}) = \boldsymbol{S}_1(\hat{\boldsymbol{\beta}}^{(k)}) - \langle \hat{\boldsymbol{\beta}}^{(k)} - \hat{\boldsymbol{\beta}}^{(k-1)}, \nabla \boldsymbol{S}_2(\hat{\boldsymbol{\beta}}^{(k-1)})\rangle - \boldsymbol{S}_2(\hat{\boldsymbol{\beta}}^{(k-1)})$. $\boldsymbol{L}^{(k+1)}(\hat{\boldsymbol{\beta}}^{(k)}) = \boldsymbol{S}_1(\hat{\boldsymbol{\beta}}^{(k)}) - \boldsymbol{S}_2(\hat{\boldsymbol{\beta}}^{(k)}) = \boldsymbol{S}(\hat{\boldsymbol{\beta}}^{(k)})$. We consider the difference of these two: $\boldsymbol{L}^{(k)}(\hat{\boldsymbol{\beta}}^{(k)}) - \boldsymbol{L}^{(k+1)}(\hat{\boldsymbol{\beta}}^{(k)}) = \boldsymbol{S}_2(\hat{\boldsymbol{\beta}}^{(k)}) - \boldsymbol{S}_2(\hat{\boldsymbol{\beta}}^{(k-1)}) - \langle \hat{\boldsymbol{\beta}}^{(k)} - \hat{\boldsymbol{\beta}}^{(k-1)}, \nabla \boldsymbol{S}_2(\hat{\boldsymbol{\beta}}^{(k-1)})\rangle \geq 0$, which follows from the convexity of $\tilde{\boldsymbol{S}}_2$. Now we have inequality: $\boldsymbol{S}(\hat{\boldsymbol{\beta}}^{(k)}) = \boldsymbol{L}^{(k+1)}(\hat{\boldsymbol{\beta}}^{(k)}) \leq \boldsymbol{L}^{(k)}(\hat{\boldsymbol{\beta}}^{(k)}) \leq \boldsymbol{L}^{(k)}(\hat{\boldsymbol{\beta}}^{(k-1)}) = \boldsymbol{S}(\hat{\boldsymbol{\beta}}^{(k-1)})$. Since $\boldsymbol{S}(\hat{\boldsymbol{\beta}}^{k})$ is decreasing in each step and it is lower bounded by zero, the algorithm will converge. Also the difference of the objective function between two adjacent steps is the linearization of $\tilde{\boldsymbol{S}}$, where the subderivative can only change in integer values. Thus, the algorithm will stop in finite number of steps.

Proof of Theorem 2: [44] and [43] have established that coordinate descent algorithm works in problem like the following.

$$f(\beta_1, \ldots, \beta_p) = g(\beta_1, \ldots, \beta_p) + \sum_{j=1}^{p} h_j(\beta_j), \tag{1.9}$$

where $g(.)$ is differentiable and convex, and the $h_j(.)$ are convex. He shows that the coordinate descent converges to the minimizer of f. The key to this is the separability of penalty function $\sum_{j=1}^{p} h_j(\beta_j)$, a sum of functions of each individual parameter.

By Tseng's theory, if the objective function has the form of equation (1.9), the coordinate-wise descent method converges to the global optimal solution. In step k

of Algorithm 2, we solve the following problem:

$$\min_{(\beta_0,\boldsymbol{\beta})\in C(\hat{\boldsymbol{\beta}}^{(k)})} R_\lambda(\beta_0,\boldsymbol{\beta}) = \min_{(\beta_0,\boldsymbol{\beta})\in C(\hat{\boldsymbol{\beta}}^{(k)})} (\frac{1}{2n}\sum_{i=1}^{n}(y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2 - \langle \boldsymbol{\beta}, \nabla \boldsymbol{S}_2(\hat{\boldsymbol{\beta}}^{(k)})\rangle) + \lambda_1 \sum_{j=1}^{p}|\beta_j| \quad (10)$$

where $\boldsymbol{S}_2 = \lambda_1 \sum_{j=1}^{p} g_2(\beta_j)$. The feasible region for each β_j is one of the four regions $\beta_j > \lambda_2$, $\beta_j < -\lambda_2$, $\beta_j > \max_{i\in chi\{j\}}(|\hat{\beta}_i^{(k)}|)$ or $\beta_j < -\max_{i\in chi\{j\}}(|\hat{\beta}_i^{(k)}|)$ depending on the current estimations. The first part of the loss function is convex and differentiable. Both the penalty and constraints are separable and convex. So Tseng's theory can be applied here directly.

Proof of Theorem 3:

$$E = \left\{\min_{k\in A^0} |\hat{\beta}_k^{(ols)}| > \frac{3\lambda_2}{2}\right\} \bigcap \left\{\max_{j\in A} |\boldsymbol{x}_j^T(\boldsymbol{Y} - \boldsymbol{X}^T\hat{\boldsymbol{\beta}}^{(ols)})| \le n\lambda_1\right\}$$

The **KKT** condition:

$$\begin{cases} -\boldsymbol{x}_j^T(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}) - n\lambda_1 \nabla g(\beta_j) + \nabla C_{\cdot j} = 0 \quad : \quad \forall j \in A^0, \\ \quad |\boldsymbol{x}_j^T(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{\beta}) - n\lambda_1 \nabla g(\beta_j)| \le n\lambda_1 \quad : \quad \forall j \in A, \end{cases}$$

On the first event in E, $\nabla\phi(\beta_j) = \nabla C_{\cdot j} = 0$ and with least squares property $\boldsymbol{x}_j^T(\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}^{(ols)}) = 0$ implies the first equation hold. the second event guarantee the second inequality. So on event E, $\hat{\boldsymbol{\beta}}^{(ols)}$ is a solution of the optimization problem.

Note that $\hat{\beta}_k \sim N(\beta_k^0, Var(\hat{\beta}_k^{(ols)}))$ with $Var(\hat{\beta}_k^{(ols)}) \le c_{min}^{-1}\sigma^2/n$, and $\boldsymbol{x}_j(\boldsymbol{Y} - \boldsymbol{X}^T\hat{\boldsymbol{\beta}}^{(ols)}) \sim N(0, \sigma^2\|(\boldsymbol{I} - \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T)\boldsymbol{x}_j\|)^2$ with $\|(\boldsymbol{I} - \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T)\boldsymbol{x}_j\|^2 \le \|\boldsymbol{x}_j\|^2$.

It follows that

$$\begin{aligned} P(\hat{A}^0 \neq A^0) &\leq P(\hat{\boldsymbol{\beta}} \neq \hat{\boldsymbol{\beta}}^{(ols)}) \leq P(E^c) \\ &\leq \sum_{k\in A^0} P(|\hat{\beta}_k| \leq \lambda_2) + \sum_{k\in A} P(|\boldsymbol{x}_j^T(\boldsymbol{Y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}^{(ols)})| \geq n\lambda_1) \\ &\leq |A^0|\Phi\left(\frac{n^{1/2}(\lambda_2 - \gamma_{min})}{2\sigma c_{min}^{-1/2}}\right) + |A|\Phi\left(\frac{n\lambda_1}{\sigma \max_{j=1,\ldots,p}\|x\|_j}\right) \end{aligned}$$

It remains to show that $\hat{\boldsymbol{\beta}}^{(ols)}$ is the only local minimizer that satisfied **KKT** condition on E. Define

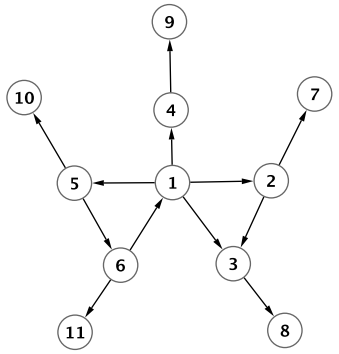
$$\tilde{g}(z) = \begin{cases} g(z;\lambda_2) & : \quad \text{if } |z| \leq \frac{\lambda_2}{2} \text{ or } |z| \geq \frac{3\lambda_2}{2}, \\ -\frac{1}{2\lambda_2}(z-\lambda_2)^2 + \frac{1}{2}(z-\lambda_2) + \frac{7\lambda_2}{8} & : \quad \text{if } |z-\lambda_2| < \frac{\lambda_2}{2}, \\ -\frac{1}{2\lambda_2}(z+\lambda_2)^2 - \frac{1}{2}(z-\lambda_2) + \frac{7\lambda_2}{8} & : \quad \text{if } |z+\lambda_2| < \frac{\lambda_2}{2}. \end{cases}$$

Given any set A with $|A| \leq |A^0|$, $\tilde{\boldsymbol{S}}(\boldsymbol{\beta})$ is a function of $\boldsymbol{\beta}_A = (\beta_1, \beta_2, \ldots, \beta_A)^T$. With $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{|A|}, 0, \ldots, 0)^T$, $\tilde{\boldsymbol{S}}(\boldsymbol{\beta}) = \tilde{\boldsymbol{S}}(\boldsymbol{\beta}_{|A|}) = \frac{1}{2n}\sum_{i=1}^n (Y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2 + \lambda_1 \sum_{j=1}^p \tilde{g}(\beta_j)$ is strictly convex in $\boldsymbol{\beta}_A$ in $\mathcal{R}^{|A|}$ when $\frac{1}{n}\boldsymbol{X}_A^T\boldsymbol{X}_A > \frac{\lambda_1}{\lambda_2}\boldsymbol{I}_{|A|}$. This occurs when $c_{min}(A) > \frac{\lambda_1}{\lambda_2}$. Because $\min_{|A|\leq A^0} c_{min}(A) > \frac{\lambda_1}{\lambda_2}$, it follows that $\hat{\boldsymbol{\beta}}^{(ols)}$ is the unique minimizer of $\tilde{\boldsymbol{S}}(\boldsymbol{\beta})$. Moreover, if A^0 are placed on the pathway of the network, $\hat{\boldsymbol{\beta}}^{(ols)}$ will satisfy all the network constraints on the first event of E. By mean value theorem,

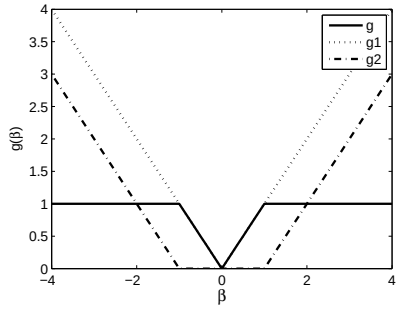
$$\left| (\frac{\partial}{\partial \boldsymbol{\beta}_A}\tilde{\boldsymbol{S}}(\boldsymbol{\beta}) - \frac{\partial}{\partial \boldsymbol{\beta}_A}\tilde{\boldsymbol{S}}(\hat{\boldsymbol{\beta}}^{(ols)}))^T \frac{(\boldsymbol{\beta}_A - \hat{\boldsymbol{\beta}}_A^{(ols)})}{\|\boldsymbol{\beta}_A - \hat{\boldsymbol{\beta}}_A^{(ols)}\|} \right| \tag{1.11}$$

$$\geq \left\{ \min_{|A|<|A^0|} c_{min}(A) - \frac{\lambda_1}{\lambda_2} \right\} \|\boldsymbol{\beta}_A - \hat{\boldsymbol{\beta}}_A^{(ols)}\| > 0. \tag{1.12}$$

Note further that $\tilde{\boldsymbol{S}}(\boldsymbol{\beta}) = \boldsymbol{S}(\boldsymbol{\beta})$ over $F = \{\boldsymbol{\beta} : ||\beta_j| - \lambda_2| > \lambda_2/2 \text{ for all } j \in A\}$. Furthermore, by construction, $\sup_{\boldsymbol{\beta}\neq\boldsymbol{\beta}^{(ols)}} \left| (\frac{\partial}{\partial \boldsymbol{\beta}_A}\boldsymbol{S}(\boldsymbol{\beta}) - \frac{\partial}{\partial \boldsymbol{\beta}_A}\tilde{\boldsymbol{S}}(\boldsymbol{\beta}))^T \frac{(\boldsymbol{\beta}_A - \hat{\boldsymbol{\beta}}_A^{(ols)})}{\|\boldsymbol{\beta}_A - \hat{\boldsymbol{\beta}}_A^{(ols)}\|} \right| \leq \frac{\lambda_1}{2}$ on E, which implies, together with ((1.11)), for any $\boldsymbol{\beta} \in F^c$, $\left| (\frac{\partial}{\partial \boldsymbol{\beta}_A}\boldsymbol{S}(\boldsymbol{\beta}))^T \frac{(\boldsymbol{\beta}_A - \hat{\boldsymbol{\beta}}_A^{(ols)})}{\|\boldsymbol{\beta}_A - \hat{\boldsymbol{\beta}}_A^{(ols)}\|} \right|$ can



(a)



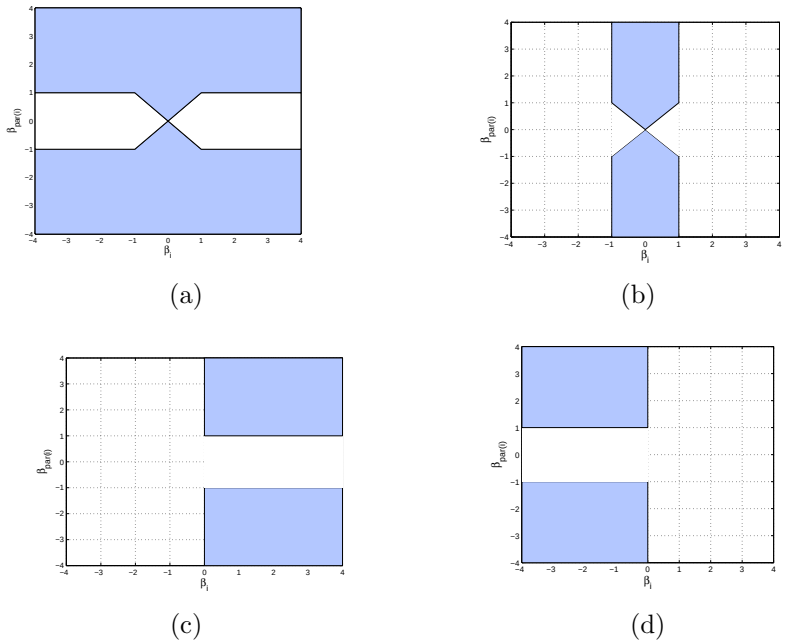
(b)

Figure 1.1: (a) A directed graph with loops; (b) Decomposition of g,$\lambda_2 = 1$.

be written as

$$\begin{aligned}
&\left| \left\{ (\frac{\partial}{\partial \boldsymbol{\beta}_A} \tilde{\boldsymbol{S}}(\boldsymbol{\beta}) - \frac{\partial}{\partial \boldsymbol{\beta}_A} \tilde{\boldsymbol{S}}(\hat{\boldsymbol{\beta}}^{(ols)})) + (\frac{\partial}{\partial \boldsymbol{\beta}_A} \tilde{\boldsymbol{S}}(\boldsymbol{\beta}) - \frac{\partial}{\partial \boldsymbol{\beta}_A} \boldsymbol{S}(\hat{\boldsymbol{\beta}}^{(ols)})) \right\}^T \frac{(\boldsymbol{\beta}_A - \hat{\boldsymbol{\beta}}_A^{(ols)})}{\|\boldsymbol{\beta}_A - \hat{\boldsymbol{\beta}}_A^{(ols)}\|} \right| \\
&\qquad \geq (\min_{|A| \leq |A^0|} c_{min}(A) - \frac{\lambda_1}{\lambda_2}) \|\boldsymbol{\beta}_A - \hat{\boldsymbol{\beta}}_A^{(ols)}\| - \frac{\lambda}{2} \\
&\qquad \geq (\min_{|A| \leq |A^0|} c_{min}(A) - \frac{\lambda_1}{\lambda_2}) \frac{\lambda_2}{2} - \frac{\lambda_1}{2} > 0
\end{aligned}$$

because $\min_{|A| \leq |A^0|} c_{min}(A) > \frac{2\lambda_1}{\lambda_2}$. This implies that $\boldsymbol{S}(\boldsymbol{\beta})$ has no local minimal in F^c on E, and hence it has unique local minimal on F. On the other hand, the proposed estimator $\hat{\boldsymbol{\beta}}$ is a local minimizer of $\boldsymbol{S}(\boldsymbol{\beta})$ on F. Consequently, $\hat{\boldsymbol{\beta}}^{(ols)} = \hat{\boldsymbol{\beta}}$ on F.



(a) (b)

(c) (d)

Figure 1.2: (a) Shaded area is the feasible region for a single constraint; (b)Adjust region $C(\hat{\beta}_i^{(k)})$ when $|\hat{\beta}_i^{(k)}| \leq \lambda_2$;(c) Adjust region $C(\hat{\beta}_i^{(k)})$ when$\hat{\beta}_i^{(k)} > \lambda_2$; (d)Adjust region $C(\hat{\beta}_i^{(k)})$when $\hat{\beta}_i^{(k)} < -\lambda_2$

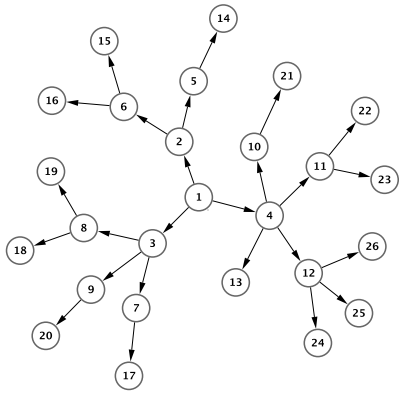


Figure 1.3: Network for Example 1 in classification framework.

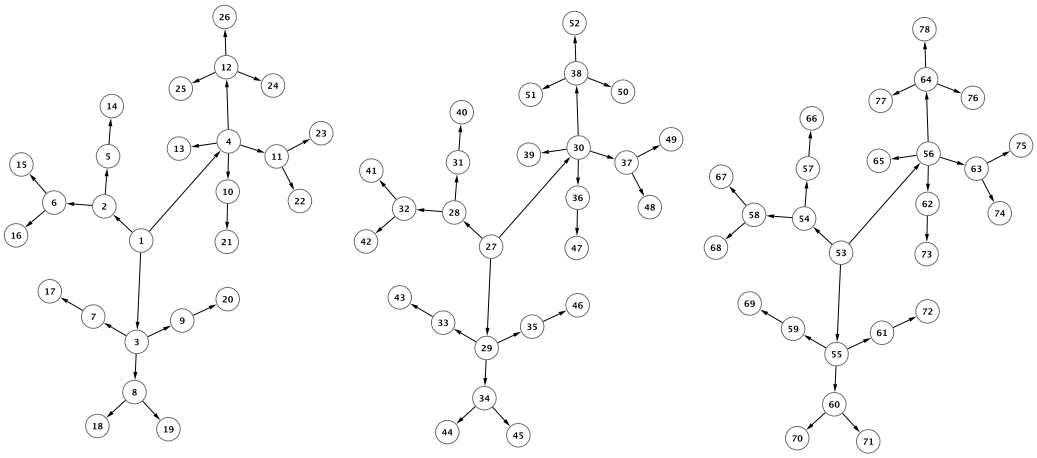


Figure 1.4: Network for Example 1 in regression framework.

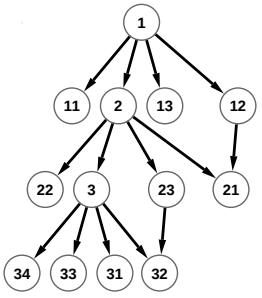


Figure 1.5: Network in Example 2.

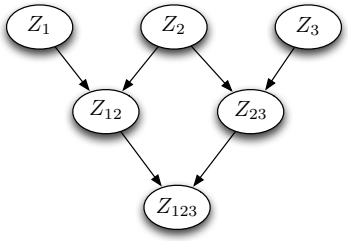


Figure 1.6: Network in Example 3(1).

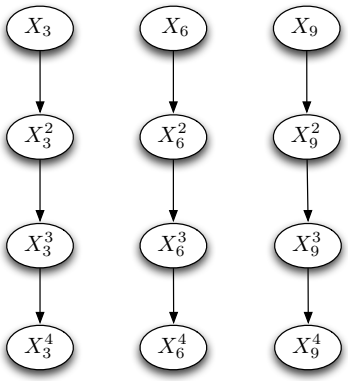


Figure 1.7: Network in Example 3(2).

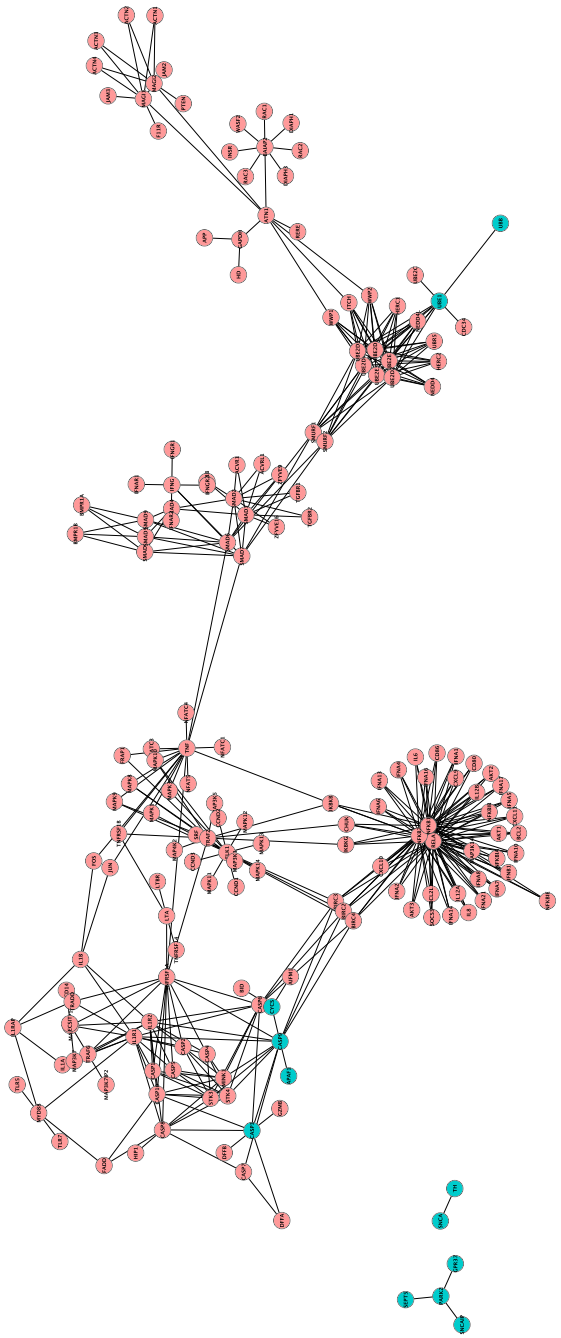


Figure 1.8: **(PD-network)** The hierarchical structure with fourcenters: $UBB, CASP9, PARK2$ and $SNCA$.

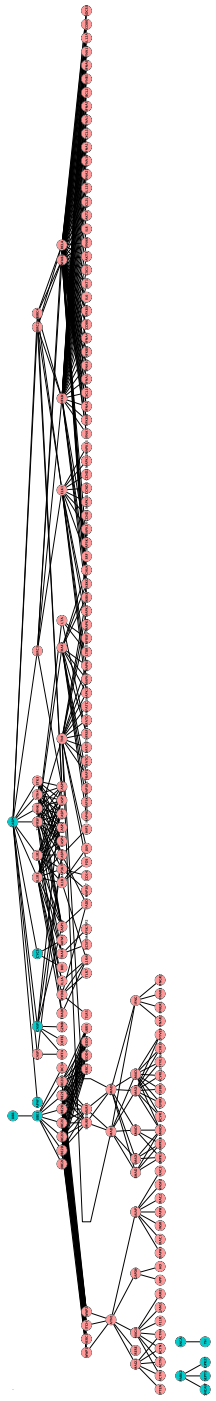


Figure 1.9: **(PD-network)** Gene network involving hierarchicalstructures with four centers: $UBB, CASP9, PARK2$ and $SNCA$.

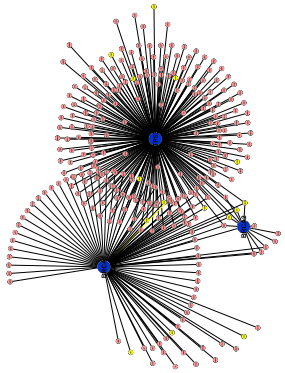


Figure 1.10: **(BC-1nb-net)** Direct neighbors of TP53, BRCA1 andBRCA2 of a breast cancer gene network including 294 genes. TP53, BRCA1 andBRCA2 are marked in blue and the other 15 suspicion genes are marked inyellow.

Table 1.1: Test errors, numbers of selected informative genes and critical numbers of selected genes averaged over 100 simulation replications with $p = 26$. The genes with nonzero coefficients are informative. In cases 1-3, genes $1, 2, 5, 6, 14, 15, 16$, genes $1, 3, 7, 17$ and genes $1, 2, 5, 14$ are informative respectively.

Case	Method	Test Error%(SE)	#Critical
1	L1SVM	12.41(0.24)	7.94(0.27)
	SVM	11.37(0.21)	5.50(0.34)
	CL1SVM	10.52(0.20)	7.53(0.38)
	Cψ	10.00(0.19)	5.18(0.28)
2	L1SVM	12.04 (0.26)	6.84(0.34)
	SVM non-convex	11.47(0.26)	4.89(0.28)
	CL1SVM	10.89(0.29)	6.29(0.45)
	Cψ	9.95(0.20)	3.28 (0.53)
3	L1SVM	12.60(0.30)	5.68(0.36)
	SVM non-convex	11.67(0.26)	5.88(0.26)
	CL1SVM	10.94(0.28)	6.85(0.48)
	Cψ	10.17(0.26)	5.00(0.50)

Table 1.2: Mean square error and critical numbers of selected genes averaged over 100 simulation replications with $p = 78$. The genes with nonzero coefficients are informative. In cases 1-3, genes $1, 2, 5, 6, 14, 15, 16$, genes $1, 3, 7, 17$ and genes $1, 2, 5, 14$ and their duplication are informative respectively.

Case	Method	MSE(SE)	#Critical
1	Lasso	35.77(0.82)	16.56(0.48)
	TLP	35.01(0.76)	15.31(0.45)
	CR	31.33(0.53)	4.29(0.36)
2	Lasso	27.67(0.44)	16.63(0.58)
	TLP	26.78(0.42)	15.10(0.53)
	CR	21.61(0.28)	2.55(0.41)
3	Lasso	28.02(0.51)	15.49(0.60)
	TLP	27.32(0.48)	14.09(0.58)
	CR	21.41(0.24)	2.50(0.40)

Table 1.3: Test errors, numbers of selected informative genes and critical numbers of selected genes averaged over 100 simulation replications with $p = 13$. The genes with nonzero coefficients are informative. In cases 1-2, genes $1, 2, 3, 32$, genes $1, 2, 3, 34$ are informative respectively.

Case	Method	Test Error%(SE)	#Critical
1	L1SVM	10.72(0.25)	6.86(0.20)
	SVM non-convex	9.86(0.24)	6.32(0.16)
	CL1SVM	11.02(0.31)	4.59(0.20)
	Cψ	10.53(0.21)	3.49(0.24)
2	L1SVM	10.88(0.30)	6.87(0.20)
	SVM non-convex	9.86(0.24)	6.22(0.17)
	CL1SVM	11.66(0.32)	4.90(0.18)
	Cψ	10.96(0.20)	3.68(0.15)

Table 1.4: Mean square error and critical numbers of selected effects averaged over 100 simulation replications with $p = 636$ in case 1 and $p = 120$ in case 2. The effects with nonzero coefficients are informative. In case 1, main effects, two way interactions and three way interactions among Z_1, Z_2, Z_3 are informative. In case 2, one up to fourth order effects of X_3, X_6, X_9 are informative.

Case	Method	MSE(SE)	#Critical
1	Lasso	50.65(1.45)	16.56(0.82)
	TLP	49.48(1.38)	15.34(0.76)
	CR	48.12(1.34)	8.20(0.60)
2	Lasso	20.19(0.58)	15.89(0.92)
	TLP	20.04(0.52)	14.43(0.89)
	CR	19.10(0.40)	4.61(0.87)

Table 1.5: Averaged test errors, numbers of selected disease genes and critical number of L1SVM, SVM with non-convex penalty, constrained SVM and constrained ψ-learning over 50 pairs of training and testing samples The estimated standard errors are in parenthesis.

Method	Test Error% (SE)	# selected disease genes	#Critical
L1SVM	45.00(0.64)	2.88(0.22)	62.44(1.43)
SVM non-convex	45.30(0.66)	1.92(0.22)	46.98(2.63)
CL1SVM	44.72(0.72)	5.30(0.29)	55.46(2.85)
Cψ	40.69(0.58)	5.18(0.21)	41.96(2.71)

Table 1.6: Predict Mean squared errors, numbers of selected disease genes and critical numbers of Lasso, elastic net, constrained regression over 100 pairs of training, tuning and testing samples. The estimated standard errors are in parenthesis.

Method	mse% (SE)	# selected disease genes	#Critical
Lasso	0.69(0.01)	0.25(0.06)	21.78(1.07)
Elastic net	0.70(0.01)	0.98(0.29)	37.38(6.15)
CR	0.68(0.01)	2.00(0.15)	18.37(0.58)

Chapter 2

Collaborative filtering

2.1 Introduction

Personalized information filtering automates the process of personalized prediction of a user's preference over a large number of items. It has become increasingly important given today's explosive growth of information, having an array of applications in personalized advertising, on-line news personalization, consumers' recommendation, among others. Personalized information filtering often results in "BIG data" associated with the ever-increasing volume, variety and velocity of information. Analysis of this type of data requires new statistical and computational treatments beyond those for conventional data. In this chapter, we cast the problem of personalized information filtering into the framework of regression and classification, argue that utilizing predictors is essential for personalized prediction, and address core issues towards high predictive accuracy and scalability for massive data with billions of observations.

Personalized information filtering can be regarded as multi-response regression and classification involving a large number of responses and users, where it predicts the preference of a user over a large number of items, called personalized prediction, simultaneously for all relevant users. Personalized prediction is summarized by a preference matrix, whose rows and columns correspond to users and items. This

problem can be phrased as estimating unknown parameters of high-dimensionality with very few observations, in the presence of high percentage of missing values. However, it differs substantially from matrix completion [5], which does not usually utilize predictors.

Two major approaches have emerged, namely, collaborative filtering [12] and content-based filtering [28]. The former pools the information across similar users for a specific item, whereas the latter acts on characteristics of the items that a user prefers, on which two kinds of recommender systems Grooveshark and Pandora [4, 28] are built. These two approaches use either nearest neighbors defined by a similarity metric or matrix factorization [22]. A similarity-based method is intuitive but is difficult to handle newer users and items [14], while a matrix factorization method imputes or to constrains out missing values. Despite success, many challenges remain, among which two salient ones are predictive accuracy and scalability. In reality, many statistical methods are unscalable thus impractical [8], whose predictive accuracy deteriorates rapidly as the missing percentage escalates. Consequently, how to design a method becomes critical, to yield high accuracy for a dataset of hundreds millions of observations, for instance, the famous MovieLens data consisting of about one hundred thousands (10^5) users and ten thousands (10^4) movies, which amounts to one billion (10^9) ratings with only 1% observed values, c.f., http://www.grouplens.org/node/12.

For personalized prediction, predictors may be available such as users' demographic profiles together with content information such as item-related web-browsing history. Utilizing them in prediction is not common in recommender systems, but its importance has been recognized [10, 17]. In what follows, we address the aforementioned issues in a general case. In particular, we develop three novel treatments: (1) partial latent models, (2) sparse latent factorizations, and (3) decomposable likelihood for scalable computation.

Within the framework of partial latent models, we utilize user-specific and content-specific covariates, permitting a treatment of multi-response regression and classification with a large number of variables in the presence of high percentage of missing values. This is in contrast to collaborative filtering and content-based filtering. Specifically, we factorize the preference matrix into a product of a user preference matrix $\boldsymbol{A}$ and an item preference matrix $\boldsymbol{B}^T$, each having the same rank as the original matrix, where $\boldsymbol{A}$ represents overall preference as well as user-specific preference, and $\boldsymbol{B}$ denotes overall item preference as well as item-specific preference, both of which correspond to the main and interaction in factor analysis. As is relevant in many real-world situations, we assume that a preference matrix can be well represented by a product of two sparse matrices $\boldsymbol{A}\boldsymbol{B}^T$. Then we seek a sparsest factorization from a class of such overcomplete factorizations. This yields a parsimonious model for given data at hand, which may be viewed as an analogy of basis pursuit from overcomplete dictionary of bases [6]. In this situation, a sparse factorization suggests that user j's preference score on item i is an inner product of two sparse vectors– the jth row of $\boldsymbol{A}$ and the ith row of $\boldsymbol{B}$. As suggested by our analysis, a sparsest factorization can be identified using a proposed L_0-method, leading to high accuracy of personalized prediction by the means of sparsity pursuit.

Statistically, we propose regularized likelihood methods to pursue a sparsest factorization to minimize the number of nonzero factorization entries. Our primary method is the L_0-regularization method through a continuous surrogate, with the L_1- and L_2- methods as a by-product. For the L_0-method, we prove that it yields higher accuracy than its counterparts. Two somewhat surprising results are noted. First, the L_0-method is capable of identifying the rank of the true preference matrix, whereas its counterparts are not. Moreover, the L_0-method yields a more parsimonious representation, due to sparsity pursuit beyond rank identification, thus leading to higher accuracy than matrix completion through low rank approximation without

predictors. Second, the proposed computational methods are most effective for high missing data, which is unlike an imputation method geared towards low missing data.

Computationally, we decompose the log-likelihood and regularizers over users and items so that non-convex minimization for maximum likelihood estimation is solved by treating many small subproblems recursively, alleviating high storage costs and permitting parallel computation. Then the proposed methods are implemented through multi-platform shared-memory parallel programming (OpenMP, c.f., http://www.openmp.org), and implemented and tested in Mahout, a library for scalable machine learning and data mining. This enables the methods to be scalable to a dataset with one billion (10^9) observations on a single machine, with good timings.

This article is organized in seven parts. Section 2 introduces partial latent models and sparse factorizations, followed by computational methods to identify a sparsest factorization for prediction with missing values in Section 3. Section 4 establishes some theoretical results concerning estimation accuracy of the proposed methods. Section 5 presents some simulated and three real benchmark examples, including 1M MovieLens with movie categorization and demographic information, 10M MovieLens data without demographic information. Section 6 discusses the methodology. Section 7 contains technical proofs.

2.2 Partial latent models and sparse factorizations

2.2.1 Partial latent models

Given an observed preference matrix $\boldsymbol{R} = (r_{ji})_{U\times M}$, with its jith element r_{ji} measuring the jth user on the ith item, we link $r_{ji} = G(\theta_{ji})$ with preference probability or mean parameter $\theta_{ji} = Er_{ji}$ in a generalized linear model, where E denotes the expectation, and $G(\cdot)$ is a link function [15], for instance, the logit function in logistic regression. Associated with each r_{ji} are user-specific and content-specific predictor

vectors $\boldsymbol{x}_j = (x_{j1}, \ldots, x_{jU_0})^T$ and $\boldsymbol{y}_i = (y_{1i}, \ldots, y_{M_0 i})^T$. Now we propose partial latent models to model $\{\theta_{ji}\}$ as a function of predictors and latent factors in an additive fashion:

$$\theta_{ji} = \boldsymbol{x}_j^T \boldsymbol{\alpha} + \boldsymbol{\beta}^T \boldsymbol{y}_i + \boldsymbol{a}_j^T \boldsymbol{b}_i; \tag{2.1}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_{U_o})^T$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{M_o})^T$ are vectors of regression parameters, respectively for $\boldsymbol{x}_j$ and $\boldsymbol{y}_i$, and $\boldsymbol{a}_j$, $\boldsymbol{b}_i$ are K-dimensional unobserved latent vectors, and K is an upper bound of the number of informative latent factors of $\boldsymbol{\Theta}$ explaining variability in $\boldsymbol{R}$, with $\boldsymbol{\Theta}$ a $U \times M$ user-over-item preference matrix defined in (2.2). Importantly, missing is permitted for $\{\boldsymbol{x}_j, \boldsymbol{y}_i\}$ in (2.2), that is, when some components of $\boldsymbol{x}_j$ and/or $\boldsymbol{y}_i$ are missing, the corresponding values are set to zero in (2.1), where missing is assumed to occur at random. Model (2.1) can be expressed in a matrix form

$$\boldsymbol{\Theta} = \boldsymbol{A}\boldsymbol{B}^T, \quad \boldsymbol{A} = \begin{pmatrix} \boldsymbol{x}_1^T & \boldsymbol{\beta}^T & \boldsymbol{a}_1^T \\ \boldsymbol{x}_2^T & \boldsymbol{\beta}^T & \boldsymbol{a}_2^T \\ \vdots & \vdots & \vdots \\ \boldsymbol{x}_U^T & \boldsymbol{\beta}^T & \boldsymbol{a}_U^T \end{pmatrix}, \boldsymbol{B} = \begin{pmatrix} \boldsymbol{\alpha}^T & \boldsymbol{y}_1^T & \boldsymbol{b}_1^T \\ \boldsymbol{\alpha}^T & \boldsymbol{y}_2^T & \boldsymbol{b}_2^T \\ \vdots & \vdots & \vdots \\ \boldsymbol{\alpha}^T & \boldsymbol{y}_I^T & \boldsymbol{b}_I^T \end{pmatrix}. \tag{2.2}$$

In (2.1), $\boldsymbol{R}$ is only partially observed over subset Ω of indices, is indicated by a binary variable $z_{ji} \in \{0, 1\}$, with 1 indicating being observed and $P(z_{ji} = 1) = \delta_{ji}$, where $z'_{ji}s$ are independent when missing is at random. Our goal is to estimate $\boldsymbol{\Theta}$ as well as the number of informative latent factors, based on $\{r_{ji}, z_{ji}, \boldsymbol{x}_j, \boldsymbol{y}_i\}$. Note that selection of informative predictors from a set of candidate predictors $\{\boldsymbol{x}_j, \boldsymbol{y}_i\}$ can be performed as well. However, we will not pursue this direction in this article.

Model (2.1) is highly interpretable in that $\boldsymbol{x}_j$ and $\boldsymbol{y}_i$ represent overall or global information regarding users and items, whereas $\boldsymbol{a}_j$ and $\boldsymbol{b}_i$ reflect specific or local information for the jth user over the ith item. They can be regard as the main ef-

fects and unobserved interactions as in analysis of variance (ANOVA). Model (2.1) is useful in several aspects. First, it leverages additional information from $\{\boldsymbol{x}_j, \boldsymbol{y}_i\}$, which pushes the latent model [27] to the next level by utilizing covariates for prediction. Second, (2.1) yields models for multi-response regression and classification with a large number of responses, for personalized prediction, in lieu of the latent models without covariates. Statistically, regression and classification of this sort are extremely challenging for personalized prediction due to overparametrization and lack of repeated measurements, user-specific and item-specific information, where each r_{ji} may be observed at most once or missing. This is in contrast to traditional statistical analysis such as two-way ANOVA. Third, (2.1) makes personalized prediction possible through pooling information across users and items, which is advantageous over collaborative filtering without content-specific information and content-based filtering without user-specific information.

In (2.2), $\boldsymbol{A}$ and $\boldsymbol{B}$ are called a user preference matrix and an item preference matrix, respectively. Nonzero-columns of $\boldsymbol{A}$ and $\boldsymbol{B}$ can be thought of as features for predicting outcome of $\{r_{ji}\}$. In personalized information filtering, each nonzero-column of $\boldsymbol{A}$ is either a user-specific predictor or an unobserved latent factor governing a user's preference over items, whereas each nonzero-column of $\boldsymbol{B}$ is either a content-specific predictor or an unobserved latent factor governing an item's preference by users. In a sense, matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ work together to yield personalized prediction of a user's preference over an item simultaneously for all users and items, where the number of nonzero columns of $\boldsymbol{A}$ or $\boldsymbol{B}$ leads to an estimated number of informative latent factors.

2.2.2 Sparse latent factorizations

For motivation, consider latent factor models (2.1) without predictors $(\boldsymbol{x}_j, \boldsymbol{y}_i)$. Let $\boldsymbol{\Theta}_0$ be the true parameter matrix. A factorization of $\boldsymbol{\Theta}_0 = \bar{\boldsymbol{A}}\bar{\boldsymbol{B}}^T$ is said to be a latent

factorization if

$$\boldsymbol{\Theta}_0 = \bar{\boldsymbol{A}}\bar{\boldsymbol{B}}^T, \bar{\boldsymbol{A}} = (\boldsymbol{a_1}, \ldots, \boldsymbol{a_U})^T, \bar{\boldsymbol{B}} = (\boldsymbol{b_1}, \ldots, \boldsymbol{b_M})^T, r(\bar{\boldsymbol{A}}) = r(\bar{\boldsymbol{B}}) = r(\boldsymbol{\Theta}_0) \equiv r_0 \leq K, \quad (2.3)$$

where $r(\cdot)$ denotes the rank of a matrix, and $\bar{\boldsymbol{A}}$ and $\bar{\boldsymbol{B}}$ are $U \times K$ and $M \times K$ matrices that have the same locations of zero-columns simultaneously, that is, if the jth column of $\bar{\boldsymbol{A}}$ is identical to zero, so is the corresponding one of $\bar{\boldsymbol{B}}$, and vice versa. Note that representations in (2.3) are overcomplete and involve myriad factorizations, including the one defined by the singular value decomposition (SVD) of $\boldsymbol{\Theta}_0$. For instance, $\boldsymbol{\Theta}_0$ can be expressed as follows.

$$\boldsymbol{\Theta}_0 = \begin{pmatrix} 2 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} -1.376 & -.325 \\ -.851 & .526 \end{pmatrix} \begin{pmatrix} -1.376 & -.851 \\ -.325 & .526 \end{pmatrix}. \quad (2.4)$$

The first latent factorization with six nonzero entries is more sparse than the second one involving eight, which is preferred in estimation of $\boldsymbol{\Theta}_0$ from a point of view of dimension reduction. As showed in Lemma 2.4, the number of nonzero elements in a sparsest factorization is no greater than $(M + U - r(\boldsymbol{\Theta}_0) + 1)r(\boldsymbol{\Theta}_0)$, which is substantially less than the dimension of $\boldsymbol{\Theta}_0$ UM when U and M are large. This is in contrast to principle component analysis (PCA) in a different context, where orthogonality is imposed to $\bar{\boldsymbol{A}}$ and $\bar{\boldsymbol{B}}$ for an unique factorization, that is, $\bar{\boldsymbol{A}}^T\bar{\boldsymbol{A}}$ and $\bar{\boldsymbol{B}}^T\bar{\boldsymbol{B}}$ are diagonal. Therefore, we seek, among many factorizations in (2.3), a sparsest factorization $(\boldsymbol{A}_0, \boldsymbol{B}_0) = \arg\min_{\{\boldsymbol{\Theta}_0 = \bar{\boldsymbol{A}}\bar{\boldsymbol{B}}^T, (\bar{\boldsymbol{A}}, \bar{\boldsymbol{B}}) \text{ satisfies } (2.3)\}} (\|\bar{\boldsymbol{A}}\|_0 + \|\bar{\boldsymbol{B}}\|_0)$ to minimize $\|\bar{\boldsymbol{A}}\|_0 + \|\bar{\boldsymbol{B}}\|_0$, where $\|\boldsymbol{A}\|_q$ denotes the L_q-norm of matrix $\boldsymbol{A}$. This permits a sparser factorization than the SVD factorization imposed by orthogonality. In general, this treatment for (2.3) without $(\boldsymbol{x}_j, \boldsymbol{y}_i)$ is readily generalized to (2.3) with $(\boldsymbol{x}_j, \boldsymbol{y}_i)$. In short, seeking a sparest factorization in (2.3) leads to further dimension reduction thus higher accuracy of prediction.

In (2.3), benefits of pursuit of a sparsest factorization are three-folded. First, a sparser factorization in terms of $(\boldsymbol{A}_0, \boldsymbol{B}_0)$ can be obtained by removing additional redundant entries to realize further sparsity within nonzero-columns of $\boldsymbol{A}_0$ and $\boldsymbol{B}_0$. This leads to a sparser factorization than one without doing so. In this sense, a factorization of this type is more preferable over a matrix factorization defined by low rank approximation and PCA. Second, estimation of $r(\boldsymbol{\Theta}_0)$ can be performed by removing redundant columns of $\boldsymbol{A}_0$ and $\boldsymbol{B}_0$ simultaneously given $\boldsymbol{\Theta}_0 = \boldsymbol{A}_0\boldsymbol{B}_0^T$, which also yields an estimated $r(\boldsymbol{\Theta}_0)$ by-product. This is especially useful in the presence of high percentage of missing values. Third, a general matrix $\boldsymbol{\Theta}_0$, sparse or nonsparse, can be expressed in terms of a sparse factorization as illustrated in (2.4), hence that it can be estimated through pursuit of a sparsest factorization. In summary, pursuit of a sparsest factorization is achieved by the means of identifying zero entries of $\boldsymbol{A}_0$ and $\boldsymbol{B}_0$.

The factorization (2.3) can be also interpreted an undercomplete dictionary $\boldsymbol{A}$ with K atoms, multiplied by $\boldsymbol{B}^T$ whose columns corresponding to decomposition coefficients of columns of $\boldsymbol{R}$ against the dictionary $\boldsymbol{A}$. In this regard, this interpretation brings (2.3) close to sparse dictionary learning in [18], where $\boldsymbol{A}$ is a known dictionary consisting of possibly some base functions, which requires $K > \min(U, M)$ for an overcomplete dictionary.

2.3 Methods for missing values

Given (2.1) and (2.3), assume, without of loss of generality, that the marginal likelihood of $\{z_{ji}, \boldsymbol{x}_j, \boldsymbol{y}_i\}$ is independent of the parameters $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{a}_j, \boldsymbol{b}_i\}$. Assume ignorable missing, or the conditional probability of $(r_{ji}, \boldsymbol{x}_j, \boldsymbol{y}_i)$ given z_{ji} is the same as the unconditional probability of $(r_{ji}, \boldsymbol{x}_j, \boldsymbol{y}_i)$, we obtain the negative log-likelihood or empirical loss function of $\{r_{ji}\}_{j=1,i=1}^{U,M}$ given $\{z_{ji}, \boldsymbol{x}_j, \boldsymbol{y}_i\}_{j=1,i=1}^{U,M}$, after ignoring parameter-

independent terms involving the marginal distribution of $\{z_{ji}, \boldsymbol{x}_j, \boldsymbol{y}_i\}$, can be written as

$$\sum_{(j,i)\in\Omega} l(r_{ji}, \boldsymbol{x}_j^T\boldsymbol{\alpha} + \boldsymbol{\beta}^T\boldsymbol{y}_i + \boldsymbol{a}_j\boldsymbol{b}_i^T) = \sum_{j=1}^{U}\sum_{i=1}^{M} w_{ji} l(r_{ji}, \boldsymbol{x}_j\boldsymbol{\alpha}^T + \boldsymbol{y}_i\boldsymbol{\beta}^T + \boldsymbol{a}_j^T\boldsymbol{b}_i), \quad (2.5)$$

where $l(r_{ji}, \theta_{ji})$ is the negative log-likelihood (loss function) of r_{ji} given $\{z_{ji}, \boldsymbol{x}_j, \boldsymbol{y}_i\}$, and $w_{ji} = 1$ if $(j,i) \in \Omega$, $w_{ji} = 0$ otherwise. For nonignorable missing, modeling the distribution of z_{ji} given $(r_{ji}, \boldsymbol{x}_j, \boldsymbol{y}_i)$ may be required. We refer to [11] for a more discussion about relevant issues.

The choice of $l(\cdot,\cdot)$ depends on models underlying the observed data. If $l(r_{ji}, \theta_{ji}) = (r_{ji} - \theta_{ji})^2$ in (2.5), then it yields the SVD of $\boldsymbol{R}$, which is useful for continuous response r_{ij}. For ordinal response r_{ji}, a model such as the proportional odd model [15] may be useful, where

$$l(r_{ji}, \theta_{ji}) = -\sum_{t=1}^{L} \delta_t(r_{ji}) \log\Big(\frac{\exp(\mu_t(r_{ji}))}{1+\exp(\mu_t(r_{ji}))} - \frac{\exp(\mu_{t-1}(r_{ji}))}{1+\exp(\mu_{t-1}(r_{ji}))}\Big), (j,i) \in \Omega, \quad (2.6)$$

where $\delta_t(r_{ji}) = I(r_{ji} = t)$ and $\mu_t(r_{ji}) = \mu_t + \boldsymbol{x}_j^T\boldsymbol{\alpha} + \boldsymbol{\beta}^T\boldsymbol{y}_i + \boldsymbol{a}_j^T\boldsymbol{b}_i$; $\mu_0(r_{ji}) = 0$, and $P(r_{ji} \le t) = \frac{\exp(\mu_t(r_{ji}))}{1+\exp(\mu_t(r_{ji}))}$; $t = 1, \cdots, L$.

For incomplete data, a common treatment is imputation of missing values to construct the likelihood of pseudo complete data, which is mainly for convenience [11]. Unfortunately, however, such a treatment may not be scalable and feasible with high percentage of missing values, as in our situation. Our strategy is to work with the likelihood of incomplete data (2.5) and adopt an approach of "decomposing and combining" to break (2.5) into many nearly independent subproblems to solve recursively. Details are given next.

2.3.1 Sparsity pursuit

This section introduces our methods for estimating a sparsest factorization as well as reconstruction of $\boldsymbol{\Theta}$, which is a nonconvex problem itself. These methods are designed so that they can be implemented through mapReduce for distributed computation, which is an important consideration for scalability. Other competing methods such as their constrained counterpart will not be considered.

Statistically, we develop methods to achieve three objectives simultaneously: 1) identifying a sparsest factorization of $\boldsymbol{\Theta}$ in terms of $(\boldsymbol{A}, \boldsymbol{B})$; 2) estimating $\boldsymbol{\Theta}$ through sparse $(\boldsymbol{A}, \boldsymbol{B})$; 3) determining the number of informative latent factors. For estimation, the number of parameters amounts to $U_0 + M_0 + UM$, which greatly exceeds the sample size $|\Omega|$ due to a high missing percentage in $\boldsymbol{R}$. To pursue sparsity and prevent overfitting, we regularize (2.5) through row by row regularization:

$$S_1(\boldsymbol{A}, \boldsymbol{B}) = \sum_{(j,i)\in\Omega} l(r_{ji}, \boldsymbol{x}_j^T\boldsymbol{\alpha} + \boldsymbol{\beta}^T\boldsymbol{y}_i + \boldsymbol{a}_j^T\boldsymbol{b}_i) + \lambda\Big(\sum_{j=1}^{U}\|\boldsymbol{a}_j\|_1 + \sum_{i=1}^{M}\|\boldsymbol{b}_i\|_1\Big) \quad (2.7)$$

$$S_0(\boldsymbol{A}, \boldsymbol{B}) = \sum_{(j,i)\in\Omega} l(r_{ji}, \boldsymbol{x}_j^T\boldsymbol{\alpha} + \boldsymbol{\beta}^T\boldsymbol{y}_i + \boldsymbol{a}_j^T\boldsymbol{b}_i) + \lambda\Big(\sum_{j=1}^{U}\|\boldsymbol{a}_j\|_0 + \sum_{i=1}^{M}\|\boldsymbol{b}_i\|_0\Big) \quad (2.8)$$

where $\|\cdot\|_q$ denotes the L_q-norm of a vector of length K for row by row regularization, and λ is a nonnegative tuning parameter. For efficient computation, we replace the L_0-function in (2.8) by its continuous surrogate, the truncated L_1-function $J(u) = \frac{1}{\tau}\min(|u|, \tau)$ [24], to yield our cost function:

$$S_0(\boldsymbol{A}, \boldsymbol{B}) = \sum_{(j,i)\in\Omega} l(r_{ji}, \boldsymbol{x}_j^T\boldsymbol{\alpha} + \boldsymbol{\beta}^T\boldsymbol{y}_i + \boldsymbol{a}_j^T\boldsymbol{b}_i) + \lambda\Big(\sum_{j=1}^{U}\sum_{k=1}^{K} J(|a_{jk}|) + \sum_{i=1}^{M}\sum_{k=1}^{K} J(|b_{ik}|)\Big) \quad (2.9)$$

where τ is a tuning parameter, and $J(u)$ approximates the L_0-function as $\tau \to 0^+$. Minimization of (2.7) and (2.9) with respect to $\{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{a}_j, \boldsymbol{b}_i\}$ yields the regularized maximum likelihood estimates $(\hat{\boldsymbol{A}}^{L_1}, \hat{\boldsymbol{B}}^{L_1})$ and $(\hat{\boldsymbol{A}}^{L_0}, \hat{\boldsymbol{B}}^{L_0})$. Importantly, regulariza-

tion in (2.7)–(2.9) is imposed to $\boldsymbol{A}$ and $\boldsymbol{B}$ in the exactly same fashion so that the latent factorization property in (2.3) is satisfied by our estimates.

Computationally, we develop strategies to solve large-scale nonconvex minimization described in (2.7) and (2.9). For (2.7), as in the foregoing discussion, we decompose it into many small subproblems to solve recursively by a blockwise coordinate decent method with a maximum block improvement [7]. This strategy yields parallelization and mapReduce for fast computation, reducing memory requirement. In particular, we employ a blockwise updating scheme with three blocks, corresponding to the main effects $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ involving all observations, individual user preference vectors $\{\boldsymbol{a}_j\}$, and individual item preference vectors $\{\boldsymbol{b}_j\}$. For each block, the estimates are updated while the other blocks are held fixed at the current values. Then we circle through the three blocks until convergence, where circling proceeds with the largest amount of block improvement.

To update $\hat{\boldsymbol{a}}_j$ given the rest; $j = 1, \cdots, U$, we treat $\boldsymbol{a}_j$ as unknown parameters and minimize, after ignoring terms independent of $\boldsymbol{a}_j$ in (2.7),

$$\sum_{i \in R_{j\cdot}} l(r_{ji}, \boldsymbol{x}_j^T \hat{\boldsymbol{\alpha}} + \hat{\boldsymbol{\beta}}^T \boldsymbol{y}_i + \boldsymbol{a}_j^T \hat{\boldsymbol{b}}_i) + \lambda \|\boldsymbol{a}_j\|_1, \tag{2.10}$$

over a set $R_{j\cdot}$ of items that user j's preference scores have observed, which is a K-dimensional generalized Lasso problem with sample size $|R_{j\cdot}|$. Importantly, (2.10) involves user j alone, which separates this user from the rest, and becomes ideal for parallelization and mapReduce. Similarly, from (2.7), we update $\hat{\boldsymbol{b}}_i$ in (2.11); $i = 1, \cdots, M$, by minimizing

$$\sum_{j \in R_{\cdot i}} l(r_{ji}, \boldsymbol{x}_j^T \hat{\boldsymbol{\alpha}} + \hat{\boldsymbol{\beta}}^T \boldsymbol{y}_i + \hat{\boldsymbol{a}}_j^T \boldsymbol{b}_i) + \lambda \|\boldsymbol{b}_i\|_1, \tag{2.11}$$

over a set $R_{\cdot i}$ of users who have indicated their preferences on item i. From the view of efficient computation, the above decomposition permits distributed computation

over users and items, with each user or each item as one building block. Therefore, (2.10) and (2.11) can be parallelized in j and i, respectively over users and items, without cross-referencing to other users and items.

For the main effects updating, we treat $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ as unknown parameters and minimize

$$\sum_{(j,i)\in\Omega} l(r_{ji}, \boldsymbol{x}_j^T\boldsymbol{\alpha} + \boldsymbol{\beta}^T\boldsymbol{y}_i + \hat{\boldsymbol{a}}_j^T\hat{\boldsymbol{b}}_i) \tag{2.12}$$

to yield estimates $(\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}})$. This is $(U_0 + M_0)$-dimensional regression, which is usually a convex problem when $\{\hat{\boldsymbol{a}}_j\}$ and $\{\hat{\boldsymbol{b}}_i\}$ are held fixed.

For (2.12), we use an analytic formula integrated with sequential updating over observations by regressing $r_{ji} - \hat{\boldsymbol{a}}_j^T\hat{\boldsymbol{b}}_i$ on $\{\boldsymbol{x}_j, \boldsymbol{y}_i\}$ for the l_2-loss, where the process of sequential updating can be parallelized over observations. For a general loss, we consider use a second-order method involving the gradient and the hassen of the cost function in (2.12), which is less efficient than the l_2-case. For (2.10) and (2.11), we use fast Lasso implementation of [13] for the l_2-loss $l(r_{ji}, \theta_{ji}) = (r_{ji} - \theta_{ji})^2$, where standardization of covariates is applied as usual.

For efficient computation of (2.10) and (2.11) and reducing memory requirement, we store data as a row-based sparse matrix and a column-based matrix, respectively for user (row) updating and item (column) updating. This enables efficient updating with searching user- or item-specific information. Algorithm 1 below summarizes our computational strategy for the L_1-method, where parallel computation is implemented inside each of the three blocks through OpenMP.

For alternating updating between $\mathbf{A}$ and $\mathbf{B}$, it seems that there is one identifiability issue due to scaling with respect to the product $\boldsymbol{A}\boldsymbol{B}^T$, that is, $\boldsymbol{A}\boldsymbol{B}^T = (k\boldsymbol{A})(k^{-1}\boldsymbol{B})^T$ for any constant k. To circumvent this difficulty, we propose to a equal-scaling strategy performing an additional scaling step after each alternating updating. This is to

ensure that column-wise scales of $\boldsymbol{A}$ are the same those of $\boldsymbol{B}$. This strategy is performed for each column of $\boldsymbol{A} = (A_1, \cdots, A_K)^T$ and $\boldsymbol{B} = (B_1, \cdots, B_K)^T$. Specifically, we update the solution $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{B}})$ at the present step: $\hat{\boldsymbol{A}}_i = \tilde{\boldsymbol{A}}_i \sqrt{\frac{\|\tilde{\boldsymbol{B}}_i\|_1}{\|\tilde{\boldsymbol{A}}_i\|_1}}$; $\hat{\boldsymbol{B}}_i = \tilde{\boldsymbol{B}}_i \sqrt{\frac{\|\tilde{\boldsymbol{A}}_i\|_1}{\|\tilde{\boldsymbol{B}}_i\|_1}}$, $i = 1, \cdots, K$ such that $\tilde{\boldsymbol{A}}\tilde{\boldsymbol{B}}^T = \hat{\boldsymbol{A}}\hat{\boldsymbol{B}}^T$, and $\|\hat{\boldsymbol{A}}_i\|_1 = \|\hat{\boldsymbol{B}}_i\|_1$, $i = 1, \cdots, K$.

Lemma 2.1

The equal scaling strategy does not increase the cost function value. The same property holds for the L_2-penalty. □

Proof: Note that $\hat{\boldsymbol{A}}\hat{\boldsymbol{B}}^T = \tilde{\boldsymbol{A}}\tilde{\boldsymbol{B}}^T$. For the L_1-penalty function,

$$\begin{aligned} \|\tilde{\boldsymbol{A}}\|_1 + \|\tilde{\boldsymbol{B}}\|_1 &= \sum_{i=1}^{K} \left(\|\tilde{\boldsymbol{A}}_i\|_1 + \|\tilde{\boldsymbol{B}}_i\|_1 \right) \geq 2 \sum_{i=1}^{K} \sqrt{\|\boldsymbol{A}_i\|_1 \|\boldsymbol{B}_i\|_1} \\ &= \sum_{i=1}^{K} \left(\|\hat{\boldsymbol{A}}_i\|_1 + \|\hat{\boldsymbol{B}}_i\|_1 \right) = \|\hat{\boldsymbol{A}}\|_1 + \|\hat{\boldsymbol{B}}\|_1. \end{aligned}$$

As a result, the rescaling does not increase the objective function.

Algorithm 1 Parallel computation for the L_1-method with missing values

Require: Ratings r_{ji}, the upper bound K, tuning parameter λ, initial value for $(\boldsymbol{A}, \boldsymbol{B})$, and specify a simple bound on each entries of $\boldsymbol{A}$ and $\boldsymbol{B}$.

1: Start iteration;
2: Parallel computation: For each user j, solve (2.10) to update $\hat{\boldsymbol{a}}_j$; $j = 1, \cdots, U$. For each item i, solve (2.11) to yield $\hat{\boldsymbol{b}}_i$; $i = 1, \cdots, M$.
3: Parallel computation: Given $\{\hat{\boldsymbol{a}}_i, \hat{\boldsymbol{b}}_i\}$, update sequentially over observations for fast computation of regression estimate $(\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}})$. Solve (2.12) to update $(\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}})$.
4: After each alternating updating, we apply the equal scaling strategy for the L_1-method, as described above. That is, perform an additional scaling step to ensure equal scales for $(\boldsymbol{A}, \boldsymbol{B})$ for the L_1-method.
5: Proceed with the block giving the maximum improvement first, as measured by the amount of decreasing in the cost function.
6: Circle through all the blocks alternately until convergence.
7: **return** $(\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}})$, $\hat{\boldsymbol{a}}_j$, $j = 1, \ldots U$; $\hat{\boldsymbol{b}}_i$, $i = 1, \ldots, M$. The final solution is $(\hat{\boldsymbol{A}}^{L_1}, \hat{\boldsymbol{B}}^{L_1})$.

Algorithm 1 allows for path-following computation, where λ decreases from a

large λ-value at which the solution becomes zero. Several implementation details need attention. First, the initial value for $(\boldsymbol{A}, \boldsymbol{B})$ seems important for computation efficiency. Based on our limited experience, a better initial value results in faster convergence than a random starting point. In our case, we may use the solution of the L_2-method, as described in Section 3.2, to be an initial value. For the L_2 method, the initial value for $(\boldsymbol{A}, \boldsymbol{B})$ at the first λ-value can be determined as follows. The first U_0 columns of $\boldsymbol{A}$ and the first M_0 columns of $\boldsymbol{B}$ are set to the row averages of observed predictors, and the $U_0 + M_0 + 1$ column of $\boldsymbol{A}$ and $\boldsymbol{B}$ are set to to the row averages of $\boldsymbol{R}$, with other columns to be random. Then they are set to the solution at the adjacent λ-value to utilize the so called "warm-start". This strategy is employed to speed up convergence, and to avoid the solution of being trapped at a stationary point. Second, the order of updating over users or items may be important. In fact, one may choose updating in a random order in the absence of any prior knowledge. However, higher priority should be given to more active users and items, as in on-line prediction. Finally, K is usually set between $30 \sim 50$ with $K = 200$ for the most extreme.

Concerning memory requirement, (2.10) and(2.11) are the jth user-specific and the ith item-specific, which are solved separately for small subproblems.

For the L_0-method, we employ the same strategy as in (2.7), except that we replace (2.10) and (2.11) by (2.13) and (2.14), respectively. In particular, we minimize

$$\sum_{i \in R_{j\cdot}} l(r_{ji}, \boldsymbol{x}_j^T \hat{\boldsymbol{\alpha}} + \hat{\boldsymbol{\beta}}^T \boldsymbol{y}_i + \boldsymbol{a}_j^T \hat{\boldsymbol{b}}_i) + \lambda \sum_{k=1}^{K} J(|a_{jk}|), \tag{2.13}$$

$$\sum_{j \in R_{\cdot i}} l(r_{ji}, \boldsymbol{x}_j^T \hat{\boldsymbol{\alpha}} + \hat{\boldsymbol{\beta}}^T \boldsymbol{y}_i + \hat{\boldsymbol{a}}_j^T \boldsymbol{b}_i) + \lambda \sum_{k=1}^{K} J(|b_{ik}|), \tag{2.14}$$

in $(\boldsymbol{a}_j, \boldsymbol{b}_i)$.

To solve nonconvex problems (2.13) and (2.14), we employ difference convex (DC) programming to decompose (2.13) or (2.14) into a difference of two convex functions,

based on which an iterative scheme is obtained. The reader may consult [24] for details about a DC algorithm. More specifically, for each DC iteration m, given the solution $(\hat{\boldsymbol{a}}_j^{(m-1)}, \hat{\boldsymbol{b}}_i^{(m-1)})$ at DC iteration $m-1$, we solve (2.15) and (2.16) alternatively until convergence, to yield $(\hat{\boldsymbol{a}}_j^{(m)}, \hat{\boldsymbol{b}}_i^{(m)})$. For $\hat{\boldsymbol{a}}_j^{(m)}$, we treat $\boldsymbol{a}_j$ as unknown parameters and minimize

$$\sum_{i \in R_j.} l(r_{ji}, \boldsymbol{x}_j^T \boldsymbol{\alpha} + \hat{\boldsymbol{\beta}}^T \boldsymbol{y}_i + \boldsymbol{a}_j^T \hat{\boldsymbol{b}}_i) + \frac{\lambda}{\tau} \sum_{k=1}^{K} w_{Ajk} |a_{jk}|, \tag{2.15}$$

where $w_{Aj1} = w_{Aj2} = 0$ and $w_{Ajk} = I(\|\hat{a}_{jk}^{(m-1)}\|_2 \leq \tau)$; $U_0 + M_0 \leq k \leq U$. Similarly, we obtain $\hat{\boldsymbol{b}}_i^{(m)}$ by minimizing

$$\sum_{i \in R_j.} l(r_{ji}, \boldsymbol{x}_j^T \hat{\boldsymbol{\alpha}} + \boldsymbol{\beta}^T \boldsymbol{y}_i + \hat{\boldsymbol{a}}_j^T \boldsymbol{b}_i) + \frac{\lambda}{\tau} \sum_{k=1}^{K} w_{Bjk} |b_{ik}|, \tag{2.16}$$

with respect to $\boldsymbol{b}_i$, where $w_{Bi1} = w_{Bi2} = 0$ and $w_{Bik} = I(\|\hat{b}_{ik}^{(m-1)}\|_2 \leq \tau)$; $U_0 + M_0 \leq k \leq M$.

The L_0-method is summarized in Algorithm 2, where sparse matrices are used to store data as in Algorithm 1. In addition, the equal scaling strategy, as described by the L_1 method, is employed similarly with the L_1-norm replaced by the L_2-norm such that $\tilde{\boldsymbol{A}}\tilde{\boldsymbol{B}}^T = \hat{\boldsymbol{A}}\hat{\boldsymbol{B}}^T$, and $\|\hat{\boldsymbol{A}}_i\|_2 = \|\hat{\boldsymbol{B}}_i\|_2$; $i = 1, \cdots, K$, where $(\hat{\boldsymbol{A}}, \hat{\boldsymbol{B}})$ is updated from the present solution $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{B}})$. Now let $(\boldsymbol{W}_A, \boldsymbol{W}_B)$ be weight matrices whose jkth entries are w_{Ajk} and w_{Bjk}.

As indicated by Lemma 2.2, the algorithms converge to a stationary point of the cost function. Before proceeding, let $S_l(\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3)$ be the cost function for the L_l-method, where $\boldsymbol{s}_1 = \{\boldsymbol{\alpha}, \boldsymbol{\beta}\}$, $\boldsymbol{s}_2 = \{\boldsymbol{a}_j\}$ and $\boldsymbol{s}_3 = \{\boldsymbol{b}_i\}$ represent the corresponding three blocks. Then $(\boldsymbol{s}_1^*, \boldsymbol{s}_2^*, \boldsymbol{s}_3^*)$ is a stationary point of $S_j(\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3)$ if

$$s_l^* = \underset{s_l \in D_l, l=1,2,3}{\operatorname{argmin}} S_j(s_1^*, \cdots, s_{l-1}^*, s_l, s_{l+1}^*, \cdots, s_3^*); l = 1, 2, 3,$$

Algorithm 2 Parallel computation: The L_0-method with missing values

Require: Ratings r_{ji}, number of latent factors K, tuning parameters (λ, τ), initial value for $(\boldsymbol{A}, \boldsymbol{B})$, set each entry of initial weight matrix $\boldsymbol{W}_A$ and $\boldsymbol{W}_B$ to 1, and specify a simple bound on each entries of $\boldsymbol{A}$ and $\boldsymbol{B}$.

1: Start outer DC loop: Given $(\boldsymbol{W}_A, \boldsymbol{W}_B)$, start inner loop:
2: Parallel computation: For each user j, update $\boldsymbol{a}_j$ by solving (2.13). For each item i, update $\boldsymbol{b}_i$ by solving (2.14).
3: Parallel computation: Given $\{\hat{\boldsymbol{a}}_i, \hat{\boldsymbol{b}}_i\}$, sequential update over observations for fast computation of regression estimate $(\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}})$. Solve (2.12) to update $(\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}})$.
4: Proceed with the block giving the maximum improvement first, as measured by the amount of decreasing in the cost function.
5: After each alternating updating, we perform an additional scaling step to ensure equal scales for $(\boldsymbol{A}, \boldsymbol{B})$ for the L_0-method, as described above.
6: Circle through blocks alternately until convergence.
7: Update $(\boldsymbol{W}_A, \boldsymbol{W}_B)$ according to new $(\boldsymbol{A}, \boldsymbol{B})$. Go to 1.
8: **return** $(\hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}})$, $\hat{\boldsymbol{a}}_j$; $j = 1, \ldots U$, $\hat{\boldsymbol{b}}_i$, $i = 1, \ldots, M$. The final solution is $(\hat{\boldsymbol{A}}^{L_0}, \hat{\boldsymbol{B}}^{L_0})$

where D_l is a compact domain for s_l. Lemma 2.2 below gives convergence properties of the algorithms as well as properties of the estimates.

Lemma 2.2

(Properties of the estimates) The estimates $(\hat{\boldsymbol{A}}^{L_1}, \hat{\boldsymbol{B}}^{L_1})$ and $(\hat{\boldsymbol{A}}^{L_0}, \hat{\boldsymbol{B}}^{L_0})$, computed from Algorithms 1 and 2, are stationary points of $S_1(\boldsymbol{A}, \boldsymbol{B})$ and $S_0(\boldsymbol{A}, \boldsymbol{B})$, respectively. Moreover, they satisfy the latent factorization property in (2.3). □

The storage cost for our methods is minimal as only user-specific or item-specific information is stored for solving (2.10) and (2.13) or (2.11) and (2.14). The computational complexity is $(2(U+M)Las + 2Reg)I_1I_2$, where Las denotes the complexity of solving single weighted Lasso with K variables and $\max(M, U)$ observations, Reg is that of solving single U_0-dimensional or M_0-dimensional regression, and I_1 and I_2 are the number of blockwise iteration and DC iteration, respectively. Based on our experience, I_1 and I_2 are about $30 \sim 40$ and $3 \sim 4$.

2.3.2 The L_2-method

In the literature, (2.17) without predictors is studied using a maximum margin matrix factorization method is proposed in [22]. The cost function is $\min_{\boldsymbol{A},\boldsymbol{B}} \Big(\sum_{j=1}^{U} \sum_{i=1}^{M} w_{ji}(r_{ji} - \boldsymbol{a}_j^T \boldsymbol{b}_i)^2 + \lambda(\sum_{j=1}^{U} \|\boldsymbol{a}_j\|^2 + \sum_{i=1}^{M} \|\boldsymbol{b}_i\|^2) \Big)$, where a ridge regularizer is employed. The corresponding nonconvex minimization problem is solved by an alternate least squares algorithm [31] or a stochastic gradient descent algorithm [23].

In this section, we generalize our partial latent factor models to this situation to leverage predictors. This L_2 method will be studied further and compared with the L_0- and L_1- methods. The cost function for the L_2-method is written as

$$\min_{\boldsymbol{A},\boldsymbol{B}} \Big(\sum_{j=1}^{U} \sum_{i=1}^{M} w_{ji} l(r_{ji}, \boldsymbol{x}_j^T \boldsymbol{\alpha} + \boldsymbol{\beta}^T \boldsymbol{y}_i + \boldsymbol{a}_j^T \boldsymbol{b}_i) + \lambda(\sum_{j=1}^{U} \|\boldsymbol{a}_j\|^2 + \sum_{i=1}^{M} \|\boldsymbol{b}_i\|^2) \Big), \quad (2.17)$$

where $\lambda > 0$ is a regularization parameter controlling predictive accuracy. Note that no sparse solutions are expected for $(\boldsymbol{A}, \boldsymbol{B})$ in (2.17), as in ridge regression. For (2.17), we employ the same computational strategy as before except (2.10) and (2.11) are replaced by a ridge version.

Lemma 2.3 establishes a connection between (2.17) with complete data and (2.5) with trace-norm minimization. It says that in the case of complete data the L_2-method estimates $r(\boldsymbol{\Theta})$ through the trace-norm that is an upper convex envelope of $r(\boldsymbol{\Theta})$ Note, however, such a result may not be expected for (2.17) with missing observations, suggested by our simulations.

Lemma 2.3

In the case of complete data, when $K = \min(U, M)$, minimizing (2.5) with respect to $(\boldsymbol{A}, \boldsymbol{B})$ reduces to

$$\min_{\boldsymbol{\Theta}} \Big(\|\boldsymbol{\Theta} - \boldsymbol{R}\|_F^2 + 2\lambda \|\boldsymbol{\Theta}\|_* \Big) \quad (2.18)$$

where $\boldsymbol{\Theta} = \boldsymbol{A}\boldsymbol{B}^T$, and $\|\cdot\|_F$ and $\|\boldsymbol{\Theta}\|_* = \sum_{i=1}^{\min(U,M)} |\sigma_i|$ are the Frobenius-norm and trace-norm, respectively, and σ_i^2 is the ith singular value of $\boldsymbol{\Theta}$. Hence the solution of (2.5) is unique with respect to $\boldsymbol{A}\boldsymbol{B}^T$ although it may not be so in $(\boldsymbol{A}, \boldsymbol{B})$. □

2.4 Theory

This section is devoted to theoretical investigation of pursuit of a sparsest factorization in terms of accuracy of reconstructing $\boldsymbol{\Theta}$ with missing values. In particular, we drive recovery error rate for reconstructing $\boldsymbol{\Theta}$, for the L_0-, L_1- and L_2-methods, as a function of the sample size $|\Omega|$, tuning parameter λ or (λ, τ), and a quantity that we call the degree of sparseness. In addition, we will compare these methods to understand the role of a various regularizer plays in pursuing a sparse factorization.

2.4.1 Main results

First we introduce the degree of sparseness given factorizations in (2.3). The degree of sparseness s_q for the L_q-norm is defined to be $s_q = \min_{\{\boldsymbol{\Theta}_0 = \boldsymbol{A}\boldsymbol{B}^T, (\boldsymbol{A},\boldsymbol{B}) \text{ satisfying } (2.3)\}}(\|\boldsymbol{A}\|_q + \|\boldsymbol{B}\|_q)$; $q = 0, 1$, and $s_2 \equiv \min_{\{\boldsymbol{\Theta}_0 = \boldsymbol{A}\boldsymbol{B}^T, (\boldsymbol{A},\boldsymbol{B}) \text{ satisfying } \in(2.3)\}}(\|\boldsymbol{A}\|_2^2 + \|\boldsymbol{B}\|_2^2)$. See Lemma 2.5 for a connection among s_0, s_1 and s_2. In a sense, these methods aim to different aspects of factorization, resulting dramatically different statistical properties of the estimates.

Secondly, we define our parameter space $\mathcal{F}$. Let $L > 0$ be a constant controlling the scale of $\boldsymbol{A}$ and $\boldsymbol{B}$. For personalized information filtering, the support of $\{r_{ji}\}$ such as a preference score is usually finite. It is then sensible to assume that $\|\boldsymbol{A}\|_\infty \leq L$ and $\|\boldsymbol{B}\|_\infty \leq L$, which is equivalent to that $\max_{j,i}(\|\boldsymbol{x}_j\|_\infty, \|\boldsymbol{y}_i\|_\infty, \|\boldsymbol{a}_j\|_\infty, \|\boldsymbol{b}_i\|_\infty) \leq L$ and $\max(\|\boldsymbol{\alpha}\|_\infty, \|\boldsymbol{\beta}\|_\infty) \leq L$. Then $\mathcal{F}$ is defined as $\{\boldsymbol{\Theta} = \boldsymbol{A}\boldsymbol{B}^T : (\boldsymbol{A}, \boldsymbol{B}) \text{ satisfies } (2.3), \|\boldsymbol{A}\|_\infty \leq L, \|\boldsymbol{B}\|_\infty \leq L, \boldsymbol{A} \in \mathbb{M}(M, K), \boldsymbol{B} \in \mathbb{M}(U, K)\}$, where $\mathbb{M}(M, K)$ is a class of $M \times K$ matrices, and $\|\cdot\|_\infty$ denotes the sup-norm of a matrix.

Thirdly, we define a complexity measuring the size of the parameter space $\mathcal{F}$. For any $\boldsymbol{\Theta}_l = (\theta_{ji}^l)$; $l = 1, 2$, define the the Hellinger-distance $h(\boldsymbol{\Theta}_1, \boldsymbol{\Theta}_2)$ to be the averaged Hellinger-distance over their components $(MU)^{-1} \sum_{j=1}^{M} \sum_{i=1}^{U} h(\theta_{ji}^1, \theta_{ji}^2)$, where $h(\theta_{ji}^1, \theta_{ji}^2) \equiv \int (f^{1/2}(r_{ji}, z_{ji}, \theta_{ji}^1) - f^{1/2}(r_{ji}, z_{ji}, \theta_{ji}^2))^2 d\mu(r_{ji}, z_{ji})$ is the Hellinger-distance, where $f(r_{ji}, z_{ji}, \theta_{ji})$ is the probability density of (r_{ji}, z_{ji}) given $(\boldsymbol{x}_j, \boldsymbol{y}_i)$, and $\mu(\cdot)$ is a dominating measure.

The following assumption is made to require that the likelihood function be smooth.

Assumption A: (Smoothness of likelihood) For some constant $d_0 > 0$, any θ_{ji}^1 and θ_{ji}^2,

$$|f^{1/2}(r_{ji}, z_{ji}, \theta_{ji}^1) - f^{1/2}(r_{ji}, z_{ji}, \theta_{ji}^2)| \leq G(r_{ji}, \delta_{ji}) |\theta_{ji}^1 - \theta_{ji}^2|; \quad j = 1, \cdots, M, i = 1, \cdots, U,$$

with $\sup_{1 \leq j \leq U, 1 \leq i \leq M} EG(r_{ji}, \delta_{ji}) \leq d_0$.

Assume that the maximum likelihood estimates $\hat{\boldsymbol{\Theta}}^{L_0}$, $\hat{\boldsymbol{\Theta}}^{L_1}$ and $\hat{\boldsymbol{\Theta}}^{L_2}$, corresponding to L_0-, L_1- and L_2-norm regularization exist, minimizing (2.7), (2.8) and (2.17) over $\mathcal{F}$, respectively. Note that existence of an approximated maximum likelihood estimate is assured if the log-likelihood function is bounded above.

Theorem presents finite-sample error bounds for $\hat{\boldsymbol{\Theta}}^{L_0}$ to reconstruct $\boldsymbol{\Theta}_0$ in terms of the Hellinger-distance. Let P be the probability of (r_{ji}, z_{ji}) under the true $\boldsymbol{\Theta}_0$ given $\{\boldsymbol{x}_j, \boldsymbol{y}_i\}$.

Theorem 2.1

(Error bound for the L_0-method). Under Assumption A, for $\hat{\boldsymbol{\Theta}}^{L_0}$, if $K \geq r_0 \equiv r(\boldsymbol{\Theta}_0)$, then there exists a constant $c_1 > 0$, such that for $(|\Omega|, M, U)$,

$$P(h(\hat{\boldsymbol{\Theta}}^{L_0}, \boldsymbol{\Theta}_0) \geq \varepsilon_{0,|\Omega|}) \leq 4 \exp\big(- c_1 |\Omega| \varepsilon_{0,|\Omega|}^2 \big), \tag{2.19}$$

provided that $\lambda = s_0^{-1} c_3 \varepsilon_{0,|\Omega|}^2$, where $\varepsilon_{0,|\Omega|}^2 = \log(\frac{(M+U)K}{s_0}) \frac{s_0}{|\Omega|}$, which is $\log(\frac{(M+U)r_0}{s_0}) \frac{s_0}{|\Omega|}$

when $K = r_0$ is tuned. As $|\Omega|, M, U \to \infty$,

$$h(\hat{\mathbf{\Theta}}^{L_0}, \mathbf{\Theta}_0) = O_p(\varepsilon_{0,|\Omega|}), \text{ and } Eh^q(\hat{\mathbf{\Theta}}^{L_0}, \mathbf{\Theta}_0) = O(\varepsilon^q_{0,|\Omega|}),$$

for real number $q \geq 1$, where $O_p(\cdot)$ denotes the stochastic order under P. □

Theorem Theorem 2.1 says that $\hat{\mathbf{\Theta}}^{L_0}$ reconstructs $\mathbf{\Theta}_0$ at a rate $\varepsilon^{L_0}_{|\Omega|} \to 0$ in probability P and the risk, provided that (1) the sample size is sufficiently large so that $\varepsilon_{|\Omega|} \to 0$, equivalently, $|\Omega| >> s_0$, and (2) the chance to observe r_{ji} at any location needs to be bounded away from zero, or $\inf_{1\leq j\leq U, 1\leq i\leq M} \delta_{j,i} > 0$. Most importantly, the recovery rate is of order $\varepsilon_{0,|\Omega|} = \sqrt{\log(\frac{(M+U)r_0}{s_0})\frac{s_0}{|\Omega|}}$ when $K = r_0$ is optimized through tuning. In contrast to the recovery rate $\varepsilon^{mat}_{|\Omega|}$ of matrix completion through low-rank approximation (Theorem 9 of [21]), $\varepsilon_{0,|\Omega|} \preceq \varepsilon^{mat}_{|\Omega|}$, when specializing to the case without predictors with $M_0 = U_0 = 0$, as the latter is not applicable to the case with predictors. Here $a_n \preceq b_n$ to mean $a_n \leq cb_n$ for some $c > 0$, for all sufficiently large n. In particular,

$$\begin{aligned}
\varepsilon_{0,|\Omega|} &= \sqrt{\log(\frac{(M+U)r_0}{s_0})\frac{s_0}{|\Omega|}} \leq \sqrt{\log(\frac{(M+U)r_0}{(M+U-r_0+)r_0})\frac{(M+U-r_0+1)r_0}{|\Omega|}} \\
&\sim \sqrt{\frac{(M+U-r_0+1)r_0}{|\Omega|}} \preceq \sqrt{\frac{(M+U)r_0}{|\Omega|}\log\frac{r_0|\Omega|}{(M+U)}} = \varepsilon^{mat}_{|\Omega|}.
\end{aligned}$$

This is because $\varepsilon_{0,|\Omega|}$ decreases in s_0, where $s_0 \leq (M+U-r_0+1)r_0$ by Lemma 2.4. This is expected because additional dimension reduction is achieved after a low rank approximation is identified, whereas a method of matrix completion through low-rank approximation does not share this property.

The next lemma says that the degree of sparseness defined by the L_0-norm s_0 is upper bounded by the effective degree of freedom for rank estimation.

Lemma 2.4

(Connection between sparse factorization and rank estimation) Let $r(\boldsymbol{\Theta}_0) = r_0$. Then $s_0 = \|\boldsymbol{A}_0\|_0 + \|\boldsymbol{B}_0\|_0 \leq (M + U - r_0 + 1)r_0$, where $(\boldsymbol{A}_0, \boldsymbol{B}_0)$ is as defined in (2.3). □

Theorem Theorem 2.2 presents a parallel result of Theorem 1 for the L_1- and L_2-methods.

Theorem 2.2

(Error bound for the L_1- and L_2-methods) Under Assumption A, for $\hat{\boldsymbol{\Theta}}^{L_1}$ and $\hat{\boldsymbol{\Theta}}^{L_2}$, if $K \geq r_0$, there exists a constant $c_2 > 0$, such that

$$P(h(\hat{\boldsymbol{\Theta}}^{L_1}, \boldsymbol{\Theta}_0) \geq \varepsilon_{1,|\Omega|}) \leq 4 \exp\big(-c_2|\Omega|\varepsilon_{1,|\Omega|}^2\big), \tag{2.20}$$

$$P(h(\hat{\boldsymbol{\Theta}}^{L_2}, \boldsymbol{\Theta}_0) \geq \varepsilon_{2,|\Omega|}) \leq 4 \exp\big(-c_2|\Omega|\varepsilon_{2,|\Omega|}^2\big), \tag{2.21}$$

provided that $\lambda = s_1^{-1}c_3\varepsilon_{1,|\Omega|}^2$ and $\lambda = s_2^{-1}c_3\varepsilon_{2,|\Omega|}^2$ in (2.20) and (2.21), respectively, where $\varepsilon_{1,|\Omega|} = \sqrt{\frac{s_1^2 \log((M+U)K)}{|\Omega|}}$ and $\varepsilon_{2,|\Omega|} = \sqrt{\frac{(M+U)K \log s_2}{|\Omega|}}$, which reduce to $\varepsilon_{1,|\Omega|} = \sqrt{\frac{s_1^2 \log((M+U)r_0)}{|\Omega|}}$ and $\varepsilon_{2,|\Omega|} = \sqrt{\frac{(M+U)r_0 \log s_2}{|\Omega|}}$ when $K = r_0$ is tuned. Then the results of Theorem Theorem 2.2 continue to hold in this case with $\hat{\boldsymbol{\Theta}}^{L_0}$, $\varepsilon_{0,|\Omega|}$ replaced by $\hat{\boldsymbol{\Theta}}^{L_q}$, $\varepsilon_{q,|\Omega|}$; $q = 1, 2$. □

To contrast the L_1-method with the L_0-method, we note that $\varepsilon_{0,|\Omega|} = \sqrt{\frac{s_1^2 \log((M+U)K)}{|\Omega|}} \preceq \varepsilon_{1,|\Omega|} = \sqrt{\frac{s_1^2 \log((M+U)r_0)}{|\Omega|}}$ in view of Lemma 2.5. Moreover, the L_0-method enables to recover $r(\boldsymbol{\Theta}_0)$, whereas the L_1-method does not. This aspect is confirmed by our simulation study in Example 1. Moreover, $\varepsilon_{1,|\Omega|} \preceq \varepsilon_{2,|\Omega|}$, which is anticipated due to sparsity for the L_1-norm.

The following connection between the degree of sparseness measured by the L_0-norm s_0 and the L_1-metric s_1 can be established, when $\boldsymbol{\Theta}_0$ is assumed to be bounded away from zero, which is sensible in personalized information filtering with preference scores.

Lemma 2.5

(Connection among the degree of sparseness s_q measured by the L_q-norm) The following results hold: $s_1 \geq s_0 c_{\min}$ and $s_2 \geq s_1 c_{\min}$, where $c_{\min}$ is the minimal of nonzero entries of $\tilde{\boldsymbol{A}}_0$ and $\tilde{\boldsymbol{B}}_0$ in the best L_1-factorization $\boldsymbol{\Theta}_0 = \tilde{\boldsymbol{A}}_0 \tilde{\boldsymbol{B}}_0^T$. If the entries of $\boldsymbol{\Theta}_0$ are bounded away from zero, so is $c_{\min}$. □

In summary, the theory suggests the following recovery rate relation based on the effective degree of freedom determined by each method, that is $\varepsilon_{0,|\Omega|} \preceq \varepsilon_{|\Omega|}^{mat} \preceq \varepsilon_{1,|\Omega|} \preceq \varepsilon_{2,|\Omega|}$. As suggested by simulation studies in Section 5, the L_0-method is expected to deliver higher accuracy as compared with the other methods.

2.4.2 Hellinger distance and the Kullback-Leibier pseudo-distance

We now explore the relation between $h(\boldsymbol{\Theta}_0, \boldsymbol{\Theta})$ and the Kullback-Leibier pseudo-distance $K(\boldsymbol{\Theta}_0, \boldsymbol{\Theta})$ in our context.

It is known that $h^2(\boldsymbol{\Theta}_0, \boldsymbol{\Theta}) \leq K(\boldsymbol{\Theta}_0, \boldsymbol{\Theta})$. Next we give an example to illustrate their connection as well as Assumption A. Consider, in the setting of (2.1), that

$$\theta_{ji} = \boldsymbol{x}_j^T \boldsymbol{\alpha} + \boldsymbol{\beta}^T \boldsymbol{y}_i + \boldsymbol{a}_j^T \boldsymbol{b}_i + \varepsilon_{ji}; \quad \varepsilon_{ji} \quad iid \sim N(0, \sigma^2), \tag{2.22}$$

where $h^2(\boldsymbol{\Theta}_0, \boldsymbol{\Theta}) = 1 - \exp(-\frac{1}{8UM\sigma^2} \sum_{j=1}^{U} \sum_{i=1}^{M} \delta_{ji} (\theta_{ji}^1 - \theta_{ji}^2)^2)$; $-\log(1 - h^2(\boldsymbol{\Theta}_0, \boldsymbol{\Theta})) = \frac{1}{4} K(\boldsymbol{\Theta}_0, \boldsymbol{\Theta})$ with $K(\boldsymbol{\Theta}^0, \boldsymbol{\Theta}) = (2\sigma^2 UM)^{-1} \sum_{j=1}^{U} \sum_{i=1}^{M} \delta_{ji} (\theta_{ji}^0 - \theta_{ji})^2$. Hence,

$$h^2(\boldsymbol{\Theta}, \boldsymbol{\Theta}_0) \leq K(\boldsymbol{\Theta}^0, \boldsymbol{\Theta}) \leq \exp\Big(\frac{L^2}{2\sigma^2}\Big) h^2(\boldsymbol{\Theta}_0, \boldsymbol{\Theta}), \text{ for any } \boldsymbol{\Theta} \in \mathcal{F}.$$

Consequently, $h^2(\boldsymbol{\Theta}, \boldsymbol{\Theta}_0)$ and $K(\boldsymbol{\Theta}_0, \boldsymbol{\Theta})$ are equivalent in this normal case. Moreover, $\inf_{1 \leq j \leq U, 1 \leq i \leq M} \delta_{j,i} \|\boldsymbol{\Theta}^0 - \boldsymbol{\Theta}\|_F^2 \leq 2\sigma^2 K(\boldsymbol{\Theta}^0, \boldsymbol{\Theta})$ for any $\boldsymbol{\Theta} \in \mathcal{F}$.

2.5 Numerical examples

This section investigates numerical aspects of the proposed methods through simulated and real data. In particular, Section 5.1 performs some simulation studies to examine operating characteristics of the proposed methods, and contrast against two scalable methods—a similarity-based neighborhood recommender system and a trace-norm matrix completion method [16]. Note that a trace-norm matrix completion method is not generally equivalent to the L_2-method in the presence of missing values, although some equivalence is established for complete data in Lemma 2. Most critically, neither similarity-based neighborhood recommender systems nor the trace-norm matrix completion method enable to utilize predictors by design. Section 5.2 concerns MovieLens data.

For parallel computation, we code the L_2-, L_1- and L_0 methods in C^{++} through OpenMP, taking the advantage of multiple threads automatically. For mapReduce computation, we code in JAVA to integrate it through Mahout for hadoop implementation. Based on our limited numerical experience, our JAVA version is about 6-7 times slower than our C^{++} version, where slowness may be due to Hadoop implementation of Mahout. In what follows, we shall use our C^{++} version (OpenMP) on an Intel(R) machine (Core(TM) i7 CPU @ 9500 @ 3.07GHz) with eight threads.

For similarity-based neighborhood recommender systems, we use routines in Apache Hadoop using mapReduce paradigm in Mahout, where the following similarity metrics including Cosine, Pearson's correlation, Euclidean, Spearman, LogLikelihood. We shall present only the results for Pearson's correlation similarity measure, because other metrics perform similarly. Here Pearson's correlation similarity measure is $\frac{\langle \boldsymbol{r}_{\cdot i}, \boldsymbol{r}_{\cdot j} \rangle}{\|\boldsymbol{r}_{\cdot i}\| \|\boldsymbol{r}_{\cdot j}\|}$, $\boldsymbol{r}_{\cdot i}$ denotes the ith column vector. For matrix completion, we use the Soft-Impute code in [16].

Our performance metric is predictive accuracy for unseen observations. Here we

use the root mean square error RMSE $= \sqrt{\frac{1}{|\Omega|}\sum_{(u,i)\in\Omega} l(r_{ui}, \hat{r}_{ui})}$, where $l(\cdot,\cdot)$ is a specific loss, r_{ui} and $\hat{r}_{ui}$ are the observed and predicted preferences over each user u and item i. In the numerical examples, the l_2-loss loss is used for estimation and evaluation.

For tuning, we minimize the RMSE with respect to the tuning parameter(s) over a tuning set is minimized over a set of uniform grid points in $\lambda \in \{0, \ldots, \lambda_{max}\}$ for convex regularization methods, and $\tau = 0.01$ additionally for the L_0 method, where λ_{max} is the minimal λ value at which all coefficients become zero. For testing, the RMSE is computed for the models evaluated at the estimated tuning parameters.

2.5.1 Simulated data

The model is generated according to (2.2) with $\theta_{ji} = \alpha_1 x_1 + \beta_1 y_2 + \beta_2 y_2 + \boldsymbol{a}_j \boldsymbol{b}_i^T + \epsilon_{ji}$, where $\alpha_1 = 1$, $\beta_1 = 1$, $\beta_2 = 1$ and $\epsilon_{ji}, x_1, y_1, y_2$ follows standard normal distribution. And $\boldsymbol{a}_j$ and $\boldsymbol{b}_i$ are the jth and ith rows of $\boldsymbol{A}$, a 600×3 matrix, and $\boldsymbol{B}$, a 600×3 matrix. $\boldsymbol{A}$ is constructed as follows

$$\boldsymbol{A} = \Big(\underbrace{\boldsymbol{I}_{3\times3}, \cdots, \boldsymbol{I}_{3\times3}}_{100}, \underbrace{-\boldsymbol{I}_{3\times3}, \cdots, -\boldsymbol{I}_{3\times3}}_{100} \Big)^T,$$

where $\boldsymbol{I}_{3\times3}$ denotes 3×3 identity matrix. And $\boldsymbol{B} = \boldsymbol{A}$. By this design, the columns of $\boldsymbol{A}$ and $\boldsymbol{B}$ are orthogonal to each other and also sparse. Finally, missing occurs at random with missing probability .8, .7, which yields roughly about 20%, 30% of the matrix size $\times$, or $\times$ observations, with $U = 600$, $M = 600$, and $r(\boldsymbol{\Theta}_0) = 5$. The observed matrix

The following situations will be examined, including (1) prediction with versus without predictors, (2) high missing versus moderate missing, and (3) methods with different choices of K, when a true model contains covariates. Moreover, the l_2-loss $l(r,u) = (r-u)^2$ and the logarithm of the logistic loss $l(r,u) = z\log u + (1-z)\log(1-u)$

with $u = \pm 1$ are considered. This former is commonly used due to its computation merits but does not yield a binary prediction, whereas the latter is more suited but is computationally more intensive. In simulations, the RMSE value for each method is reported, in addition to the estimated rank $\hat{r}$ and the degree of sparseness $\hat{s}$ for $(\boldsymbol{a}_j, \boldsymbol{b}_i)$.

First consider a situation with 70% missing values, where the sample size suffices for accurate estimation, exceeding the number of effective degrees of freedom of $\boldsymbol{\Theta}$. With regard to reconstruction of $\boldsymbol{\Theta}$, the L_0-method with predictors performs the best across all the situations. Importantly, inclusion of the three predictors is critical for the L_q-method, as suggested by Table 1, with the amount of improvement with covariates over without predictors about 9.2%–10.2% and 8.4%–10.2% respectively for the L_1- and L_0-methods. Moreover, the L_0-method outperforms its L_1 counterpart, which is consistent with the theoretical results in Theorems 2 and 3. Moreover, with predictors, the L_0- and L_1-methods outperform the trace-norm matrix completion method–SOFT-IMPUTE [16], with the amount of improvement about 13.3% and 8.9% for the L_0-and L_1-methods. However, excluding the predictors, the L_0-method outperforms SOFT-IMPUTE and Pearson's similarity-based method but the L_1-and L_2-methods under-perform them slightly. Interestingly, the L_0-method yields a more sparse representation as compared to its competitors across all the situations, which well estimate the true degree of sparseness of Θ_0. This explains its higher performance against its competitors. With regard to rank estimation, the L_0-method enables to estimate $r_0 = r(\boldsymbol{\Theta}_0)$ precisely, whereas the L_1-method does not well estimate $r(\boldsymbol{\Theta}_0)$. Furthermore, the L_2-method and SOFT-IMPUTE completely misses the mark. In fact, the former gives the largest possible rank and the latter reaches the upper bound value of K.

Next consider the high-missing case with 90% missing values, corresponding to a difficult situation with roughly one observed value per effective parameter. It is ex-

pected that no methods will do well in rank estimation, in addition that an estimated model is more sparse than it should be due to lack of observations. For reconstruction of $\mathbf{\Theta}$, all of these methods are robust in that they continue to perform well but worse than the 70% missing case.

Finally, we observe that the value K plays a role of an upper bound for estimating $r(\Theta_0)$. As a matter of matter, estimation for the L_0- and L_1-methods is not sensitive to the choice of K as long as $K \geq r(\Theta_0)$.

Overall, the L_0-method performs the best across all the situations. The L_2-method without predictors and conventional similarity-based collaborative filtering methods are not competitive. Utilizing informative predictors is essential to predictive performance.

2.5.2 Benchmark: MovieLens data

We now compare our methods with Soft-Impute in terms of predictive performance on two benchmark data sets for personalized prediction, the 1M MovieLens, and 10M MovieLens. The 1M and 10M MovieLens data sets are collected by GroupLens Research Project at the University of Minnesota, during a seven-month period from September 19th, 1997 through April 22nd, 1998. The data are comprised of movie ratings over various periods of time, and are available at http://www.grouplens.org/node/12. The 1M MovieLens data consist of 1,000,209 anonymous ratings on a five-star scale from $6,040$ users on $3,900$ movies, whereas the 10M MovieLens data consist of 10,000,054 ratings together with $95,580$ users applied to $10,681$ movies by $71,567$ users. For the 1M data, there are four categorical and one continuous covariates, including four user-related covariates, gender, age, occupation and zip-code, as well as one content-related covariate, genres, with each user having at least 20 ratings. For simplicity, we treat these predictors as continuous variables in fitting, although we may reparametrize the four categorical covariates. For the 10M data, no demographic

information of users is available but with one content-related covariate, genres.

For each data set, we split the original data into three sets, with 80%, 10%, 10%, that is, the first eight, the ninth and the tenth for every ten observations, for training, tuning and testing. For tuning, the RMSE over the tuning set is minimized in the tuning parameter(s) over a set of grid points in a parallel fashion as in the simulated example, where 50 uniformly grid spaced points over $(0, \lambda_{\max}]$ are used to tune λ, $\tau =, 001, .1$ are used to tuning τ, and $\lambda_{\max}$ is an estimated largest point for λ. For testing, the RMSE is computed for the models evaluated at the estimated tuning parameters.

With regard to reconstruction of $\mathbf{\Theta}$, the L_0 method outperforms the L_1- and L_2-methods, but the amount of improvement is not as large as in the simulated example. Furthermore, the L_q-method with predictors perform better than its counterpart without predictors; $q = 0, 1, 2$. Such results are expected for the 10M data using one less informative predictor—genres, but a bit surprising for the 1M data involving five demographic predictors. A closer examination of the association between the response and predictors indicates that the association is highly nonlinear for both the cases. Moreover, the R^2-values for the linear model with these predictors are only about 3% for both, although these predictors are highly significant under the linear models. In this sense, the linear model assumptions are violated, hence that suitable nonlinear models may be considered for improving predictive accuracy, in addition to construction of informative predictors to ameliorate prediction. This will be a topic of future research.

With regard to runtime, for the 1M and 10M MovieLens datasets, training and tuning for 50 grid λ-points requires about $20 \sim 30$ seconds and $7 \sim 8$ minutes on average for the L_1-method with the l_2-loss. The L_0-method is about 4 times slower, and the L_2-method is about the same speed.

In conclusion, the L_0-method with predictors delivers higher predictive accuracy

for personalized prediction.

2.6 Discussion

This article formulates the problem of personalized information filtering as multi-response regression and classification through partial latent factor models. In addition, a sparse matrix factorization is introduced to reconstruct the user-over-time preference matrix regardless if it is sparse. Several regularization methods are proposed using decomposable cost functions for parallel and mapReduce computation, permitting a treatment of massive data, which a conventional method is incapable of dealing with. A general theory is developed to quantify predictive accuracy of the reconstruction in presence of high percentage of missing observations, where the proposed L_0-method promotes additional sparsity beyond rank estimation, where other competing methods may not share.

For parallel computation, we implement the l_2-loss for least squares regression. Our methods are scalable to a dataset consisting of billions of observations on a single machine with sufficient memory, whose capability can be further expanded with the help of hadoop on storage. Further investigation is necessary with regard to integration of implicit feedback and explicit feedback into latent factor models, in addition to issues relevant to nonignorable missing.

2.7 Appendix

Proof of Lemma 2.2: The proof for convergence of the algorithms follows the same argument as in Theorem 3.1 of [7], thus is omitted. To show that $(\hat{\boldsymbol{A}}^{L_1}, \hat{\boldsymbol{B}}^{L_1})$ satisfies (2.3), note that if $\hat{\boldsymbol{b}}_i = \mathbf{0}$ then it follows from (2.11) that for any $\boldsymbol{a}_j$

$$\sum_{i \in R_{j\cdot}} l(r_{ji}, \boldsymbol{x}_j^T \hat{\boldsymbol{\alpha}} + \hat{\boldsymbol{\beta}}^T \boldsymbol{y}_i + \boldsymbol{a}_j^T \hat{\boldsymbol{b}}_i) + \lambda \|\boldsymbol{a}_j\|_1 \geq \sum_{i \in R_{j\cdot}} l(r_{ji}, \boldsymbol{x}_j^T \hat{\boldsymbol{\alpha}} + \hat{\boldsymbol{\beta}}^T \boldsymbol{y}_i + \mathbf{0} \hat{\boldsymbol{b}}_i) + \lambda \|\mathbf{0}\|_1,$$

implying that the minimizer $\hat{\boldsymbol{a}}_j = \mathbf{0}$. The same holds for $\hat{\boldsymbol{b}}_i$ if $\hat{\boldsymbol{a}}_j = \mathbf{0}$. This completes the proof.

Proof of Lemma 2.3: Consider SVD of $\boldsymbol{\Theta}$ as follows: $\boldsymbol{\Theta} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^T$, where $\boldsymbol{D}$ is a diagonal matrix whose ith diagonal element is the ith singular value of $\boldsymbol{\Theta}$, and $\boldsymbol{U}$ and $\boldsymbol{V}$ are the left and right orthogonal matrices for the SVD. Note that $\|\boldsymbol{A}\|_2$ is rotation-invariant under any orthogonal-transformation. Hence it suffices to consider the constraint that $\boldsymbol{A}\boldsymbol{B}^T = \boldsymbol{D}$, equivalently, $\langle \boldsymbol{a}_j, \boldsymbol{b}_i \rangle = 0$ when $j \neq i$ and $\langle \boldsymbol{a}_i, \boldsymbol{b}_i \rangle = \sigma_i$; $i = 1, \cdots, K$. By the triangular inequality,

$$\|\boldsymbol{A}\|_2^2 + \|\boldsymbol{B}\|_2^2 \geq 2 \sum_{i=1}^{\min(U,M)} |\langle \boldsymbol{a}_i, \boldsymbol{b}_i \rangle| = 2 \sum_{i=1}^{\min(U,M)} |\sigma_i| = 2\|\boldsymbol{\Theta}\|_*.$$

implying that the minimal of (2.5) is no smaller than that of (2.18), where the equality holds when $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}^{1/2}$ and $\boldsymbol{B} = \boldsymbol{V}\boldsymbol{D}^{1/2}$. This establishes the equivalence. Finally, uniqueness of the solution of (2.18) follows from strict convexity of (2.18) in $\boldsymbol{\Theta}$. This completes the proof.

Proof of Theorem Theorem 2.1: The proof uses a large deviation probability inequality of [30] to treat one-sided regularized log-likelihood ratios.

Let $f(\boldsymbol{R}, \{z_{ji}\}, \boldsymbol{\Theta})$ be the probability density of $(\boldsymbol{R}, \{z_{ji}\})$ given $(\boldsymbol{x}_j, \boldsymbol{y}_i)$ at $\boldsymbol{\Theta}$. Let $\mathcal{F}_0(s) = \{(\boldsymbol{A}, \boldsymbol{B}) \in \mathcal{F} : \boldsymbol{\Theta} = \boldsymbol{A}\boldsymbol{B}^T, \|\boldsymbol{A}\|_0 + \|\boldsymbol{B}\|_0 \leq s\}$ be a L_0-constrained parameter space.

First we bound the bracketing u Hellinger metric entropy $H(u, \mathcal{F}_0(s))$ [29]. To define this quantity, for any $u > 0$, call a finite set of pairs of functions $\{(f_j^L, f_j^U), j = 1, \cdots, N\}$ a (Hellinger) u-bracketing of $\mathcal{F}$ if $\|(f_j^L)^{1/2} - (f_j^U)^{1/2}\|_2 \leq u$ for $j = 1, \cdots, N$, and for any $f \in \mathcal{F}$, there is a j such that $f_j^L \leq f \leq f_j^U$. The bracketing Hellinger metric entropy of $\mathcal{F}$, denoted by the function $H(\cdot, \mathcal{F})$, is defined by $H(u, \mathcal{F}) =$ logarithm of the cardinality of the u-bracketing (of $\mathcal{F}$) of the smallest size. For $\boldsymbol{\Theta} = \boldsymbol{A}\boldsymbol{B}^T$ with $(\boldsymbol{A}, \boldsymbol{B}) \in \mathcal{F}_0(s)$, let $B_\delta(\boldsymbol{A}, \boldsymbol{B}) = \{(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{B}}) \in \mathcal{F}_0(s) : \sqrt{\|\tilde{\boldsymbol{A}} - \boldsymbol{A}\|_{F^*}^2 + \|\tilde{\boldsymbol{B}} - \boldsymbol{B}\|_{F^*}^2} \leq \delta\}$

be a ball centered at $(\boldsymbol{A}, \boldsymbol{B})$. Note that $0 \leq \delta_{ji} \leq 1$. For any $\tilde{\boldsymbol{\Theta}} = (\tilde{\theta}_{ji}) = \tilde{\boldsymbol{A}}\tilde{\boldsymbol{B}}^T$ with $(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{B}}) \in B_\delta(\boldsymbol{A}, \boldsymbol{B})$,

$$\begin{aligned}
& (MU)^{-1} \sum_{j=1}^{M} \sum_{i=1}^{U} \int \sup_{\{(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{B}}) \in B_\delta(\boldsymbol{A}, \boldsymbol{B})\}} (f^{1/2}(r_{ji}, z_{ji}, \tilde{\theta}_{ji}) - f^{1/2}(r_{ji}, z_{ji}, \theta_{ji}))^2 d\mu(z_{ji}) \\
\leq \; & \frac{d_0^2}{MU} \sup_{\{(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{B}}) \in B_\delta(\boldsymbol{A}, \boldsymbol{B})\}} 2(\|\tilde{\boldsymbol{A}} - \boldsymbol{A}\|_{F^*}^2 \|\tilde{\boldsymbol{B}}\|_{F^*}^2 + \|\tilde{\boldsymbol{B}} - \boldsymbol{B}\|_{F^*}^2 \|\boldsymbol{A}\|_{F^*}^2) \\
\leq \; & \frac{d_0^2}{MU} \sup_{\{(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{B}}) \in B_\delta(\boldsymbol{A}, \boldsymbol{B})\}} 2sL^2(\|\tilde{\boldsymbol{A}} - \boldsymbol{A}\|_{F^*}^2 + \|\tilde{\boldsymbol{B}} - \boldsymbol{B}\|_{F^*}^2),
\end{aligned}$$

where $\|\cdot\|_{F*}$ is the Frobenius-norm whose jith element is taken over $\sup_{(\tilde{\boldsymbol{A}}, \tilde{\boldsymbol{B}}) \in B_\delta(\boldsymbol{A}, \boldsymbol{B})}$, and the fact that $\|\boldsymbol{A}\boldsymbol{B}^T\|_{F^*}^2 \leq \|\boldsymbol{A}\|_{F^*}^2 \|\boldsymbol{B}\|_{F^*}^2$ has been used. By Lemma 1 of [19], it suffices to bound the entropy of $B_\delta(\boldsymbol{A}, \boldsymbol{B})$. Note that there are s nonzero elements of $(\boldsymbol{A}, \boldsymbol{B})$ with $\binom{(M+U)K}{s}$ possible locations. Then for $u \geq \varepsilon_{0,|\Omega|}^2$,

$$\begin{aligned}
H(u, \mathcal{F}_0(s)) \; & \leq \; \log \binom{(M+U)K}{s} + H_2(\frac{u}{Ld_0\sqrt{2s}(MU)^{-1/2}}, \mathcal{F}_{s,L}) \\
& \leq \; s \log \Big(e \frac{(M+U)K}{s} \Big) + s \log \Big(\min \big(\frac{L^2 d_0 \sqrt{2s}}{u(MU)^{1/2}}, 1 \big) \Big), \quad (2.23)
\end{aligned}$$

where $\mathcal{F}_{s,L} = \big\{ \boldsymbol{x} \in \mathbb{R}^s, \|\boldsymbol{x}\|_\infty \leq L \big\}$, $H_2(\cdot, \mathcal{F}_{s,L})$ is the ℓ_2-metric entropy $\mathcal{F}_{s,L}$ and inequality $\binom{n}{m} \leq \big(e\frac{n}{m}\big)^m$ has been used, c.f., Theorem 2.6 of [25].

To apply Theorem 1 of [30], we verify the required entropy condition there for $\mathcal{F}$:

$$\sup_{\{s \geq s_0\}} \psi_1(\varepsilon, s) \leq c_2 |\Omega|^{1/2}, \quad (2.24)$$

where $\psi_1(x, s) = \int_x^{x^{1/2}} H^{1/2}(u, \mathcal{F}_0(s)) du/x$ with $x = (c_1\varepsilon^2 + \lambda(s-1))$. Using the entropy bound in (2.24), we obtain $\psi_1(x, s) = \frac{\varepsilon_{0,|\Omega|}(1 - x^{1/2})}{x^{1/2}}$ for $0 < x \leq 1$. Here we only focus our attention to the case of $x \leq 1$ because (2.24) is met automatically when $x > 1$. Note that $\psi_1(x, s)$ is nonincreasing in s for any fixed $x > 0$. Then

$\sup_{\{s\geq s_0\}}\psi_1(\varepsilon,s)=\psi_1(\varepsilon,s_0)$. Solving (2.24) yields that $c_1\varepsilon^2+\lambda(s_0-1)=\varepsilon^2_{0,|\Omega|}$, hence that $\varepsilon^2=\frac{1}{2c_1}\varepsilon^2_{0,|\Omega|}$ and $\lambda(s_0-1)\leq\frac{1}{2}\varepsilon^2_{0,|\Omega|}$. The result then follows from Theorem 1 of [30]. This yields (2.19), thus the rate of convergence in P when letting $|\Omega|,M,U\to\infty$. For the corresponding risk result, note that $h(\cdot,\cdot)\leq 1$, and

$$Eh^q(\hat{\boldsymbol{\Theta}}^{L_0},\boldsymbol{\Theta}_0) \;=\; \varepsilon^q_{0,|\Omega|}+\int_{\varepsilon_{0,|\Omega|}}^1 P(h(\hat{\boldsymbol{\Theta}}^{L_0},\boldsymbol{\Theta}_0)\geq x^{1/q})dx\leq \varepsilon^q_{0,|\Omega|}+\exp\big(-c_1|\Omega|\varepsilon^{2/q}_{0,|\Omega|}\big).$$

The desired result then follows. This completes the proof.

Proof of Lemma 2.4: Using the SVD of $\boldsymbol{\Theta}_0$, we obtain $\boldsymbol{\Theta}_0=\bar{\boldsymbol{A}}\bar{\boldsymbol{B}}^T$, where $\bar{\boldsymbol{A}}$ and $\bar{\boldsymbol{B}}$ are $U\times r_0$ and $M\times r_0$. Assume, without loss of generality, that $\bar{\boldsymbol{A}}^T=(\bar{\boldsymbol{A}}_1,\bar{\boldsymbol{A}}_2)$ where $\bar{\boldsymbol{A}}_1$ is a $r_0\times r_0$ nonsingular matrix. Now define $\boldsymbol{A}_0^T=(\boldsymbol{I}_{r_0\times r_0},\bar{\boldsymbol{A}}_1^{-1}\bar{\boldsymbol{A}}_2)$ and $\boldsymbol{B}_0^T=\bar{\boldsymbol{A}}_1^T\bar{\boldsymbol{B}}^T$. By construction, $\boldsymbol{A}_0\boldsymbol{B}_0^T=\bar{\boldsymbol{A}}_1\bar{\boldsymbol{B}}_1^T$ with $\|\boldsymbol{A}_0\|_0+\|\boldsymbol{B}_0\|_0=(M+U-r_0+1)r_0$. This completes the proof.

Proof of Theorem Theorem 2.2: The proof is essentially the same except the entropy calculations.

Let $\mathcal{F}_1(s)=\{(\boldsymbol{A},\boldsymbol{B})\in\mathcal{F}:\boldsymbol{\Theta}=\boldsymbol{A}\boldsymbol{B}^T,\|\boldsymbol{A}\|_1+\|\boldsymbol{B}\|_1\leq s\}$ for the corresponding constrained space for the L_1-method. Similarly, define $\mathcal{F}_2(s)$ as $\{(\boldsymbol{A},\boldsymbol{B})\in\mathcal{F}:\boldsymbol{\Theta}=\boldsymbol{A}\boldsymbol{B}^T,\|\boldsymbol{A}\|_2^2+\|\boldsymbol{B}\|_2^2\leq s\}$ for the corresponding constrained space for the L_2-method.

For the L_1-method, we bound the bracketing Hellinger entropy $H(u,\mathcal{F}_1(s))$. To this end, let $\tilde{B}_\delta(\boldsymbol{A},\boldsymbol{B})=\{(\tilde{\boldsymbol{A}},\tilde{\boldsymbol{B}})\in\mathcal{F}_1(s):\sqrt{\|\tilde{\boldsymbol{A}}-\boldsymbol{A}\|^2_{F^*}+\|\tilde{\boldsymbol{B}}-\boldsymbol{B}\|^2_{F^*}}\leq\delta\}$. Note that $0\leq\delta_{ji}\leq 1$.By Assumption A, the triangular inequality and boundedness of$\|\boldsymbol{A}_j\|_\infty$ and $\|\boldsymbol{B}_j\|_\infty$, For any $\tilde{\boldsymbol{\Theta}}=(\tilde{\theta}_{ji})=\tilde{\boldsymbol{A}}\tilde{\boldsymbol{B}}^T$ with $(\tilde{\boldsymbol{A}},\tilde{\boldsymbol{B}})\in\mathcal{F}_0(s)$,

$$\begin{aligned}
&(MU)^{-1}\sum_{j=1}^M\sum_{i=1}^U\int\sup_{\{(\tilde{\boldsymbol{A}},\tilde{\boldsymbol{B}})\in\tilde{B}_\delta(\boldsymbol{A},\boldsymbol{B})\}}(f^{1/2}(r_{ji},z_{ji},\tilde{\theta}_{ji})-f^{1/2}(r_{ji},z_{ji},\theta_{ji}))^2d\mu(z_{ji})\\
\leq\;&\frac{d_0^2}{MU}\sup_{\{(\tilde{\boldsymbol{A}},\tilde{\boldsymbol{B}})\in\tilde{B}_\delta(\boldsymbol{A},\boldsymbol{B})\}}2(\|\tilde{\boldsymbol{A}}-\boldsymbol{A}\|^2_{F^*}\|\tilde{\boldsymbol{B}}\|^2_{F^*}+\|\tilde{\boldsymbol{B}}-\boldsymbol{B}\|^2_{F^*}\|\boldsymbol{A}\|^2_{F^*})\\
\leq\;&\frac{d_0^2}{MU}\sup_{\{(\tilde{\boldsymbol{A}},\tilde{\boldsymbol{B}})\in\tilde{B}_\delta(\boldsymbol{A},\boldsymbol{B})\}}2(M+U)KL^2(\|\tilde{\boldsymbol{A}}-\boldsymbol{A}\|^2_{F^*}+\|\tilde{\boldsymbol{B}}-\boldsymbol{B}\|^2_{F^*})
\end{aligned}$$

Similarly, it follows from Lemma 3 of [20] with $q = 1$ that

$$H(u, \mathcal{F}_1(s)) \leq cs^2\Big(\min\big(\frac{L\sqrt{2}d_0(K(M+U))^{1/2}}{u(MU)^{1/2}}, 1\big)\Big)^2 \log((M+U)K),$$

for some constant $c > 0$.

For the L_2-method, $H(u, \mathcal{F}_2(s))$ can be bounded similarly. Note that

$$H(u, \mathcal{F}_2(s)) \leq c\log\Big(\min(\frac{\sqrt{s}L\sqrt{2}d_0(K(M+U))^{1/2}}{u(MU)^{1/2}}, 1)\Big)(M+U)K,$$

for some constant $c > 0$. The rest of the proof proceeds as that of Theorem Theorem 2.2. This completes the proof.

Proof of Lemma 2.5: Note that $s_1 \geq \tilde{s}_0 c_{\min}$, where $\tilde{s}_0$ is the number of nonzero element for the best L_1-factorization. The result then follows from the fact that $\tilde{s}_0 \geq s_0$ by definition. Moreover, minimization of $\|\boldsymbol{A}\|_1 + \|\boldsymbol{B}\|_1$ subject to $\boldsymbol{\Theta}_0 = \boldsymbol{AB}$ implies that the elements of its minimizer $\boldsymbol{A}$ and $\boldsymbol{B}$ are bounded away from zero provided that those of $\boldsymbol{\Theta}_0$. Similarly, $s_2 \geq s_1 c_{\min}$ can be proved. This completes the proof.

Table 2.1: Averaged RMSE's in the l_2-loss, estimated rank difference, as well as difference between the estimated degree of sparseness (SD in parentheses), for various methods, over 100 simulation replications under the l_2-loss in Example 1, with $U = 600$, $M = 600$, $r(\mathbf{\Theta}_0) = 3$ and $s_0 = 10\%, 5\%$ denoting the sparsity percentage of $(\boldsymbol{a}_j, \boldsymbol{b}_i)$ for $K = 10$ and $K = 20$ respectively. Here "Soft-Impute", "Pearson", "L_q w predictors", "L_q w/o predictors"; $q = 0, 1, 2$, denotes the trace-norm matrix completion method [16] without predictors, Pearson-similarity based collaborative filtering without predictors, the L_q-methods with and without $\{x_1, x_2, y_1\}$, and "NA" means that it is not available or computationally infeasible for the software. The precision is set to 10^{-4} for our methods and non-zero coefficients, and to the default value for a competing method.

Loss (K, % missing)	Method	RMSE	$\hat{r}$	$\hat{s}$ for $(\boldsymbol{a}_j, \boldsymbol{b}_i)$
	Soft-Impute	1.062(.002)	63.27(7.23)	NA
	Pearson	1.510(.002)	NA	NA
l_2 ($K = 10$, 70% missing)	L_2 w/o predictors	1.040(.002)	10.00(.000)	100%(.00%)
	L_1 w/o predictors	1.026(.005)	7.66(1.52)	38.28%(1.67%)
	L_0 w/o predictors	1.019(.002)	5.0(.000)	29.30%(.12%)
	L_2 w predictors	1.021(.002)	10.00(.000)	100%(.00%)
	L_1 w predictors	1.012(.004)	5.55(1.60)	18.34%(1.39%)
	L_0 w predictors	1.007(.002)	3.0(.000)	10.49%(.06%)
l_2 ($K = 20$, 70% missing)	L_2 w/o predictors	1.041(.002)	20.0(.00)	100%(.00%)
	L_1 w/o predictors	1.025(.004)	9.82(2.97)	19.59%(1.12%)
	L_0 w/o predictors	1.021(.009)	5.00(.000)	14.59%(.30%)
	L_2 w predictors	1.021(.002)	20.00(.000)	100%(.00%)
	L_1 w predictors	1.012(.004)	6.9(2.90)	9.46%(.05%)
	L_0 w predictors	1.007(.002)	3.0(.000)	5.25%(.03%)
	Soft-Impute	1.092(.003)	68.07(6.45)	NA
	Pearson	1.518(.002)	NA	NA
l_2 ($K = 10$, 80% missing)	L_2 w/o predictors	1.048(.002)	10.00(.000)	100%(.00%)
	L_1 w/o predictors	1.038(.003)	7.93(1.42)	37.03%(1.38%)
	L_0 w/o predictors	1.033(.002)	5.00(.000)	29.95%(.16%)
	L_2 w predictors	1.025(.002)	10.00(.000)	100%(.00%)
	L_1 w predictors	1.018(.003)	6.47(1.94)	17.80%(1.31%)
	L_0 w predictors	1.013(.001)	3.00(.00)	11.10%(0.11%)
l_2 ($K = 20$, 80% missing)	L_2 w/o predictors	1.049(.002)	20.00(.00)	100%(.00%)
	L_1 w/o predictors	1.038(.003)	10.7(2.97)	19.02%(.88%)
	L_0 w/o predictors	1.033 (.002)	5.00(.00)	15.00%(.01%)
	L_2 w predictors	1.025(.002)	20.00(.000)	100%(.00%)
	L_1 w predictors	1.018(.002)	7.66(2.49)	9.03%(.63%)
	L_0 w predictors	1.013(.002)	3.00(.000)	5.56%(.05%)

Table 2.2: RMSE's (SD in parentheses) for various methods in benchmark data examples based on 100 random partitions of the original data with 80%, 10% and 10% for training, tuning and testing. The 1M data include four users covariates, age, gender and occupation, and one content covariate, genres, whereas the 10M data has one content covariate, genres. Here "Soft-Impute", "L_q w predictors", "L_q w/o predictors"; $q = 0, 1, 2$, denote the trace-norm matrix completion method [16] without predictors, the L_q-methods with and without $\{x_1, x_2, y_1\}$, and and "NA" means that it is not available or computationally infeasible for the software. The precision is set to 10^{-4} for our methods, and to the default value for a competing method.

Method	1M MovieLens	10M MovieLens
Soft-Imput	0.871(.002)	NA
L_2 w/o predictors	0.866(.002)	.808(.0006)
L_1 w/o predictors	0.863(.003)	.806(.0004)
L_0 w/o predictors	0.860(.004)	.806(.0005)
L_2 w predictors	0.861(.002)	.804(.0004)
L_1 w predictors	0.859(.003)	.801(.0004)
L_0 w predictors	0.858(.004)	.801(.0005)

Chapter 3

Hadoop and Mapreduce Framework

Hadoop created by Doug Cutting, has its origins in Apache Nutch, an open source web search engine, itself a part of the Lucene project. In 2004, Google published the paper that introduced MapReduce to the world. NDFS and the MapReduce implementation in Nutch were applicable beyond the realm of search and in February 2006 Nutch were moved out of Lucene as an independent subproject called Hadoop. At around the same time, DougCutting joined Yahoo!, which provided a dedicated team and the resources to turn Hadoop into a system that run at web scale. This was demonstrated in February 2008 when Yahoo! announced that its production search index was being generated by a 10,000-core Hadoop cluster. In January 2008, Hadoop was made its own top-level project at Apaches, confirming its success. In April 2008, Hadoop broke a world record to become the fastest system to sort a terabyte of data.

3.1 Hadoop and Its Ecosystem

The trend for every individual's data is growing rapidly, but more importantly the amount of data generated by machines will be even larger. Machine logs, sensor networks, vehicle GPS traces, retail transactions–all of these contribute to the growing

of data.

The volume of data being made publicly available increase every year too. Organizations no longer have to merely manage their own data: success in the future will be dictated to a large extent by their ability to extract value from other organizations' data. It has been said that "More data usually beats better alorithms" which is to say that for some problems(such as recommending movies or music based on past preferences), however fiendish the algorithms are, they can often be beaten by having more data and less sophisticated algorithm.

The popularity of Hadoop has grown in the last few years, because it meets the needs of many organizations for flexible data analysis capabilities with an unmatched price-performance curve. The flexible data analysis features apply to data in a variety of formats, from unstructured data, such as raw text, to semi-structured data, such as logs, to structured data with a fixed schema. Hadoop has been particularly useful in environments where massive server farms are used to collect data from a variety of sources. Hadoop is able to process parallel queries as big, background batch jobs on the same server farm. This saves the user from having to acquire additional hardware for a traditional database system to process the data (assume such a system can scale to the required size). Hadoop also reduces the effort and time required to load data into another system; you can process it directly within Hadoop. This overhead becomes impractical in very large data sets. Many of the ideas behind the open source Hadoop project originated from the Internet search community, most notably Google and Yahoo!. Search engines employ massive farms of inexpensive servers that crawl the Internet retrieving Web pages into local clusters where they are analyzes with massive, parallel queries to build search indices and other useful data structures. The Hadoop ecosystem includes other tools to address particular needs, see Figure 3.1. Hive is a SQL dialect and Pig is a dataflow language for that hide the tedium of creating MapReduce jobs behind higher-level abstractions more appropriate for user

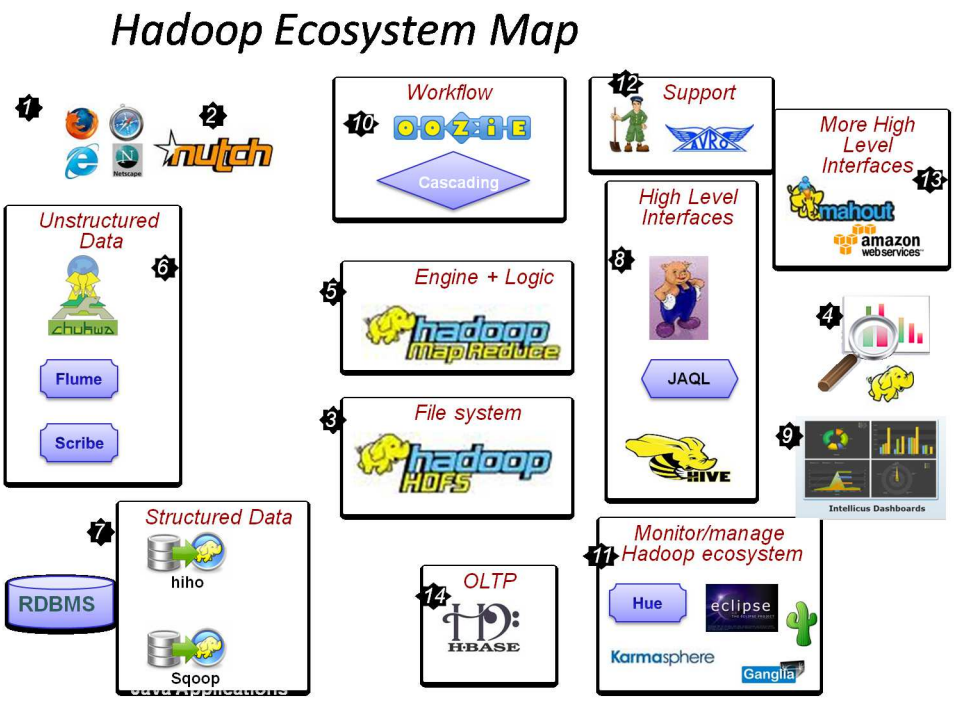


Figure 3.1: Hadoop Ecosystem

goals. Zookeeper is used for federating services and Oozie is a scheduling system. Mahout is a machine learning package implemented in map-reduce framework, see Section 3.2. HBase can provide random, real time read/write access to Big Data.

3.1.1 Data Storage for Big Data

While the storage capabilities of hard drives have increased massively over the years, the rate at which data can be read from drives have not kept up. One typical drive from 1990 could store 1380 MB of data and had transfer speed of 4.4 MB/s, so you could read all the data from a full drive in around five minutes. Almost 20 years later one terabyte drives are the standard, but the transfer speed is around 100MB/s, sot it takes more than two and a half hours to read all data off the disc.

The obvious way to speed up the reading is to read from multiple disks at once.

If we had 100 drives, each holding one hundredth of the data. Working in parallel we could read the data in under two minutes. Only using one hundredth of the disk may seem wasteful. But we can store one hundred data sets, each of which is one terabyte, and provide shared access to them.

The first problem to solve is hardware failure. If one hardware failed, we need to be told immediately and fix the data. One common way of avoiding data loss is through replication: redundant copies of the data are kept by system so that in the event of failure, there is another copies available. This is how Hadoop's file system works.

The second problem is that most analysis tasks need to be able to combine that data in some way. For example data read from one disk may need to be combined with data from any of the other 99 disks. Various distributed systems allow data to be combined from multiple sources, but doing this correctly is notoriously challenging. MapReduce framework provides a programming model that abstracts the problem from dis reads and writes, transforming it into a computation over sets of keys and values.

The hadoop system basically provides: a reliable shared storage and analysis system. The storage is provided by HDFS and the analysis by MapReduce. These two partitions are the kernel of hadoop.

3.1.2 The Hadoop File System

HDFS is a file system designed for storage very large files running on clusters on commodity hardware.

A disk has a block size, which is the minimum amount of data that it can read or write. File system for a single disk build on this by dealing with data in blocks. HDFS has a block with much larger unit– 64 MB by default. Like in a file system for a single disk, files in HDFS are broken into block-sized chunks, which are stored

as independent units. Unlike a file system for a single disk, a file in HDFS that is smaller than a single block does not occupy a full block's worth of underlying storage.

A HDFS cluster has two types of node pertains in a master-worker pattern: a name node(the master) and a number of data nodes(workers). The name node manages the file system name space. It maintains the file system tree and the meta data for all the files and directories in the tree. This information is stored persistently on the local disk in the form of two files: the name space image and the edit log. The name node also knowns the data nodes on which all the blocks for a given file are located, however, it does not store block locations persistently, since this information is reconstructed from data nodes when the system starts.

Data nodes are the work horses of the file system. They store and retrieve blocks when they are told to and they report back to the name node periodically with lists of blocks that they are storing. Without the name node, the file system cannot be used. In fact, if the machine running the name node failed, all the files on the file system would be lost since there is no way of knowing how to reconstruct the files from the blocks on the data nodes. For this reason, it is important to make the name node resilient to failure. A secondary name node is possible to run. Its main role is to periodically merge the name space image with the edit log to prevent the edit log from becoming too large. The secondary name node is usually runs on a separate physical machine, since it requires plenty of CPU and as much as memory as the name node to perform the merge. It keeps a copy of the merged name space image, which can be used in the event of the name node failing. However, the state of the secondary name node lags that of the primary, so in the event of total failure of the primary data, loss is almost guaranteed. The usual course of action in the case is to copy the name-node's meta files that are on NFS to the secondary and run it as the new primary.

```
04/28/2013 09:07:56 AM > jps
29490 DataNode
29578 SecondaryNameNode
29402 NameNode
67833 Jps
29643 JobTracker
29731 TaskTracker
```

Figure 3.2: File system on a Pseudo Distributed System

3.1.3 MapReduce by Small Example

MapREduce is a programming model for data processing. The model is simple, yet not too simple to express useful programs in. Hadoop can run MapReduce programs written in various languages such as Java, Ruby, Python and C++. MapReduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output, the types of which may be chosen by programmer. The programmer also specifies two functions: the map function(mapper) and the reduce function(reducer).

Here we give a simple example. We have some documents, the purpose of this example is to count the number of occurrence of each word in a given input set.

```
Input file 1:
Hello World
Bye World
Input file 2:
Hello Hadoop
Goodbye Hadoop
Output:
Bye 1
Goodbye 1
```

```
Hadoop 2
Hello 2
World 2
```

First the lines are presented to the map function as the key-value pairs:

```
(0, Hello World)
(1, Bye World)
(0, Hello Hadoop)
(1, Goodbye Hadoop)
```

The keys are the line offsets within the file, which is ignored in our map function. The map function merely extracts the lines and break them up into words. With the words as the key, the map function append 1 as the value for all the words.

```
(Hello, 1)
(World, 1)
(Bye, 1)
(World, 1)
(Hello, 1)
(Hadoop, 1)
(Goodbye, 1)
(Hadoop, 1)
```

The output form the map function is not sent to the reduce function directly. The MapReduce framework will process this output by sorts and groups the key-value pairs by key. So for this simple example, the reduce function will see the following input:

```
(Hello, [1, 1])
(World, [1, 1])
```

```
(Bye, 1)
(Hadoop, [1, 1])
(Goodbye, 1)
```

Each word appears with a list of 1's. All the reduce function has to do now is count the length of the appending vectors.

```
(Hello, 2)
(World, 2)
(Bye, 1)
(Hadoop, 2)
(Goodbye, 1)
```

3.1.4 MapReduce Data Flow

We have one example showing that MapReduce works for small inputs, for large input, we need a different data flow. First we introduce some terminology. A MapReduce job is a unit of work that the client wants to be performed: it consist of the input data, the MapReduce program, and the configuration information. Hadoop runs the job by dividing it into tasks, of which there are two types: map tasks and reduce tasks.

There are two types of nodes that control the job execution process: a job tracker and a number of task trackers. The job tracker coordinated all the jobs run on the system by scheduling tasks to run on the task trackers. Task trackers run tasks and send progress reports to the job tracker, which keeps a record of the overall progress of each job. If a tasks fails, the job tracker can reschedule it on a different task tracker. Hadoop divides the input to a MapREduce job into fixed-size chunks call input splits. Hadoop creates one map task for each split, which runs the user-defined map function for each record in the split. Having many splits means the time taken to process each

split is small compared to time to process the whole input. So if we are processing the splits in parallel, the processing is better load-balanced of the splits are small, since a faster machine will be able to process proportionally more splits over the course of the job than a slower machine. Even if the machines are identical, failed processes or other jobs running concurrently make load balancing desirable, and the quality of the load balancing increases as the splits become more find-grained.

On the other hand, if splits are too small, then the overhead of managing the splits and of the map task creation begins to dominate the total job execution time. For most jobs, a good split size tends to be the size of HDFS block, 64 MB by default.

Map tasks write their output to local disk, not HDFS. Because the map output is an intermediate output, once the job is complete, this output can be thrown away. So it is not needed to storing it in HDFS with replication. If the node running the map task fails before the map output has been consumed by the reduce task, then Hadoop will automatically rerun the map task on another node to recreate the map output.

Reduce tasks don't have the advantage of locality. The input to a single reduce task is normally the output from all the mappers. In this case, the sorted map outputs have to be transferred across the network to the node where the reduce task is running. The output of the reduce is normally stored in HDFS for reliability. The whole data flow with a single reduce task in illustrated in Figure 3.5. When there are multiple reducers, the map tasks partition their output, each creating one partition for each reduce task. There can be many keys in each partition, but the records for every key are all in a single partition. The partition can be controlled by a user-defined partition function, but normally the default partitioner which buckets keys using a hash function works very well. The data flow for the general case of multiple reduce tasks is illustrated in Figure 3.4.

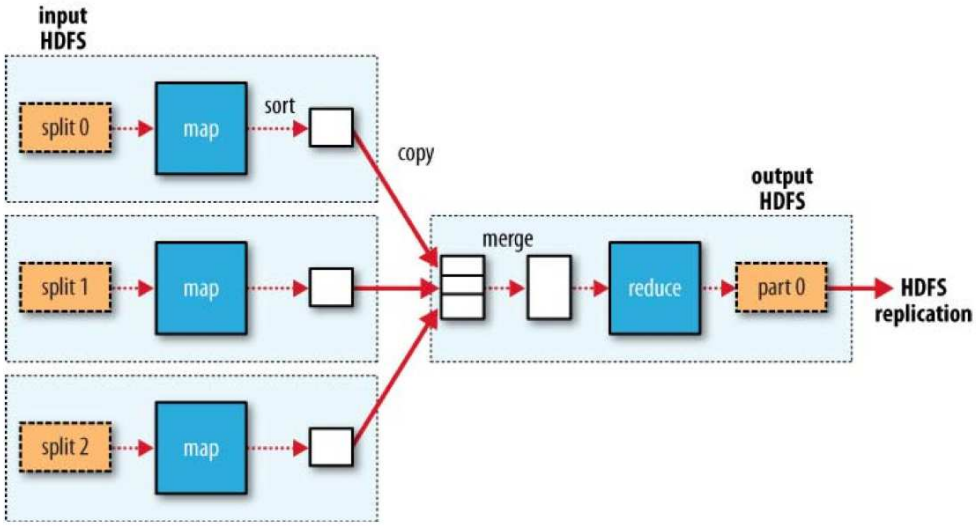


Figure 3.3: MapReduce data flow with a single reduce task

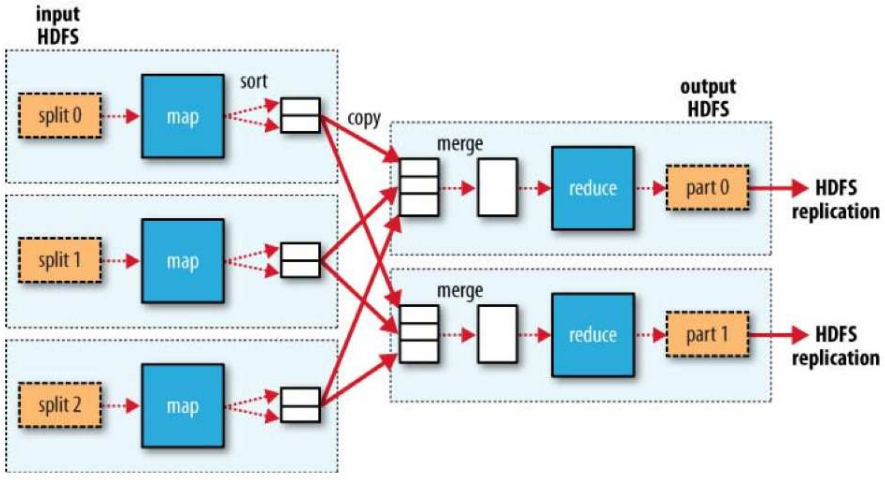


Figure 3.4: MapReduce data flow with multiple reduce tasks

3.2 Mahout ALS algorithm

Mahout has a decent implementation of Map-Reduce alternative least square algorithm. This implementation make good use of sparse vector and distributed system. Start with file with tuples: (userId, movieId, rating), the item rating matrix R' can be obtained by one map-reduce process. A sample input is

```
1,   1, 100, 3
2,   1, 104, 5
3,   2, 5, 3
4,   2, 6, 1
5,   3, 100, 4
6,   4, 5, 5
```

where the key is line number and value is the rating. By using ItemRatingVectorsMapper, we have the following output:

```
100, SparseVector([1, 3])
104, SparseVector([1, 5])
5, SparseVector([2, 3])
6, SparseVector([2, 1])
100, SparseVector([3, 4])
5, SparseVector([4, 5])
```

In current output, the key is the id of the movies and value is a sparse vector with ratings in the index of corresponding user ids. Before the mapper results are passed to the reducer, the result will be sorted by the key value, see Figure 3.4.

```
5, SparseVector([2, 3]) SparseVector([4, 5])
6, SparseVector([2, 1])
100, SparseVector([1, 3]) SparseVector([3, 4])
```

```
104, SparseVector([1, 5])
```

By VectorSumReducer, the ratings for one movies are combined as a new sparse vector.

```
5, SparseVector([2, 3], [4, 5])
6, SparseVector([2, 1])
100, SparseVector([1, 3], [3, 4])
104, SparseVector([1, 5])
```

The user rating matrix R can be obtained from item rating matrix R' by using appropriate map reduce process. With TransposeMapper, we have

```
2, SparseVector([5, 3])
4, SparseVector([5, 5])
2, SparseVector([6, 1])
1, SparseVector([100, 3])
3, SparseVector([100, 4])
1, SparseVector([104, 5])
```

Note that each input record is now spited into two records and again the result will be sorted before passed to another map-reduce process. Here we don't need other mappers and only a reducer left. With input

```
1, SparseVector([100, 3]) SparseVector([104, 5])
2, SparseVector([5, 3]) SparseVector([6, 1])
3, SparseVector([100, 4])
4, SparseVector([5, 5])
```

and MergeVectorsReducer, we obtain R

```
1, SparseVector([100, 3], [104, 5])
2, SparseVector([5, 3], [6, 1])
3, SparseVector([100, 4])
4, SparseVector([5, 5])
```

Now we have R and R' stored in HDFS. With highly distributed HADOOP system, we are ready to use these two matrix to update user preference matrix U and item feature matrix M alternatively. The major loop of alternative least square is defined in class $ParallelALSFactorizationJob$ and the code is

```
/* create an initial M */
   initializeM(averageRatings);
   for (int currentIteration = 0; currentIteration < numIterations;
       currentIteration++) {
     /* broadcast M, read A row-wise, recompute U row-wise */
     log.info("Recomputing U (iteration {}/{})", currentIteration,
                     numIterations);
     runSolver(pathToUserRatings(), pathToU(currentIteration),
                     pathToM(currentIteration - 1), currentIteration, "U",
              numItems);
     /* broadcast U, read A' row-wise, recompute M row-wise */
     log.info("Recomputing M (iteration {}/{})", currentIteration,
                     numIterations);
     runSolver(pathToItemRatings(), pathToM(currentIteration),
                     pathToU(currentIteration), currentIteration, "M",
              numUsers);
   }
```

where we first initial movie feature matrix M by movies' average rating and store the value in HDFS. In each iteration, the *runSolver* function will read movie feature

matrix from $pathToM(currentInteration-1)$ and store updated user preference matrix to $pathToU(currentIteration)$ to HDFS. After that, another $runSolver$ will read user p reference from $pathToU(currentIteration)$ and store updated movie feature to $pathToM(currentIteration)$. In the alternative process, data is never loaded into main memory which is extremely important for big data. T he real Map-Reduce process is handled in $runSolver$ function where we can specify whether we have implicit feedback about the movies. If not, we use the $SolverExplicitFeedbackMapper.class$ as the mapper. The $prepareJob$ function will setup a Map-Reduce process by specifying input and output directory, input and output format class, and the Mapper class we want to use in the process. After that, we can set up necessary parameters for the Mapper class by getting the configuration instance of the Job object. Since alternating updating of user preferences or movie features can be done independently across the users and movies, we can use multiple threads for each mapper task. Table 3.1 shows the performance of $L1$ and $L2$ regularized alternative least square method. Generally a algorithm in map reduce framework is much slower than in-memory implementation. The advantage of map-reduce is it does not have memory constraint and can handle any large data sets as long as it can be stored in HDFS.

3.3 ALS in Spark

Apache Spark is an open-source data analytics cluster computing framework originally developed in the AMPlab at UC Berkeley, see [2]. It is a fast and general-purpose cluster computing system which provides high-level APIs in Scala, Java, and Python that make parallel jobs easy to write, and an optimized engine that supports general computation graphs. It also supports a rich set of higher-level tools including Shark (Hive on Spark), MLlib for machine learning, GraphX for graph processing, and Spark Streaming, see Figure 3.5.

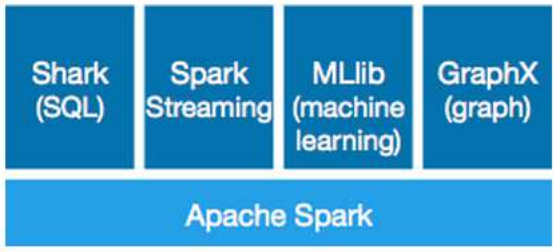


Figure 3.5: Spark Ecosystem

The MLlib package in Spark many popular machine learning algorithms implemented and ready for use on spark cluster. It has a different and more efficient implementation of alternative least square algorithm, see following list. In each iteration, we also get the out-link and in-link information for a user or product block which is quite form mahout. The out-link information includes the original user/product IDs of the elements within this block, and the list of destination blocks that each user or product will need to send its feature vector to. The in-link information includes the original user/product IDs of the elements within this block, as well as an array of indices and ratings that specify which user in the block will be rated by which products from each product block (or vice-versa). Specifically, if this InLinkBlock is for users, ratingsForBlock(b)(i) will contain two arrays, indices and ratings, for the i'th product that will be sent to us by product block b (call this P). These arrays represent the users that product P had ratings for (by their index in this block), as well as the corresponding rating for each one. We can thus use this information when we get product block b's message to update the corresponding users.

```
for (iter <- 1 to iterations) {
    // perform ALS update
    logInfo("Re-computing I given U (Iteration %d/%d)".format(iter,
        iterations))
    // YtY / XtX is an Option[DoubleMatrix] and is only required for the
        implicit feedback model
```

Table 3.1: Run Time of 30 alternative iterations with $\lambda = 5.0$ and 100 features

Method	1M MovieLens	10M MovieLens
L2(mahout)	$14m\ 50s$	$105m\ 31s$
L2(spark)	$1m\ 13s$	$15m\ 40s$
L1(mahout)	$24m\ 34s$	$132m\ 15s$
L1(spark)	$2m\ 12s$	$21m\ 45s$

```
    val YtY = computeYtY(users)
    val YtYb = ratings.context.broadcast(YtY)
    products = updateFeatures(users, userOutLinks, productInLinks,
        partitioner, rank, lambda,
      alpha, YtYb)
    logInfo("Re-computing U given I (Iteration %d/%d)".format(iter,
        iterations))
    val XtX = computeYtY(products)
    val XtXb = ratings.context.broadcast(XtX)
    users = updateFeatures(products, productOutLinks, userInLinks,
        partitioner, rank, lambda,
      alpha, XtXb)
  }
```

We compared mahout and spark ALS algorithm on an Amazon EC2 cluster with 1 master and 4 slaves. All the machines are type m1.large. Table 3.1 includes a comparison of performance on movielens data sets. We can see from the table that spark is much faster than mahout implementation.

Chapter 4

Real world recommender system

Recommender System are software tools and techniques that provide suggestions for items to users. "Item" is the general term used to denote what the system recommends to users. A recommender system normally focus on a specific type of item such as news, movies or CDs. As e-commerce Web began to develop, a pressing need emerged for providing recommendations derived from filtering the whole range of available alternatives. Users were finding it very difficult to arrive at the most appropriate choices from the immense variety of items that these Web sites were offering. The explosive growth and variety of information available on the Web and the rapid introduction of new e-business service frequently overwhelmed users, leading them to make poor decisions. In this section, we will introduce a real world application case of recommender system and discuss how we can make user of Hadoop and other big data tools to build a recommender system.

4.1 AD Network and Tapjoy

An online advertising network or ad network is a company that connects advertisers to web sites that want to host advertisements. The key function of an ad network is aggregation of ad space supply from publishers and matching it with advertiser

Figure 4.1: Top 10 Mobile Ad Company.

demand. There are also many mobile advertising networks such as admob(acquired by Google), InMobi and iAds(acquired by Apple Inc.). Figure 4.1 shows top 10 ad company named by VentureBeats 2014 Mobile Advertising Index. Tapjoy is not a traditional ad network on mobile. It is a mobile performance-based advertising platform that drives deep engagement and monetization opportunities for app publishers, while delivering valuable, engaged consumers to some of the worlds biggest brand advertisers. Tapjoy has a reach of more than 430MM (July, 2013) mobile users each month and its "Mobile Value Exchange" model allows users to receive premium content in exchange for their engagement with advertisements.

The concept of a value exchange is not new. Search advertising is the most successful interactive advertising model ever created (Google) and by definition, a clean value exchange. Consumers raise their hands and receive content that is relevant to the moment, a question, or their lives, and in exchange they reward advertisers that deliver. And unlike display, where advertisers make huge buys and hope it works, we know search advertising works. Of the expected $35 billion in 2011 US interactive marketing spend, about 60% will be spent on search marketing, according to Forrester

Research. Search marketing consistently backs into ROI-positive conversions and transactions. The average click through rate of a sponsored ad is 95% of that of an organic result, according to a Yahoo Research and Cornell Universitys report. The Mobile Value Exchange ad model evolves this transaction to address the most personal device in the world-our phones. The model offers consumers premium content(virtual currency) in exchange for ad engagement. As with search, these consumers raised their hands and requested the ad. And similar to the early days of search, the reach is massive, with 455 million mobile users already engaging.

Suppose one user are playing the game "The Hobbit" and heshe need mithril to purchase materials for better performance. The user can either purchase mithril buy real money(Figure 4.2) or earn free mithril buy click on the "Earn Free Mithril" button and engage in Tapjoy's networks(Figure 4.3). The user can earn one mithril by downloading app "Banjo" or he or she can earn 123 mithril by purchasing McAfee anti-virus software at 50% off the regular price. The offers on offer wall are organized in pages with 24 offers each page. Users are generally picky and very few of them will go to second page for more offers. This make recommendation more important in terms of user engagement and Tapjoy's revenue.

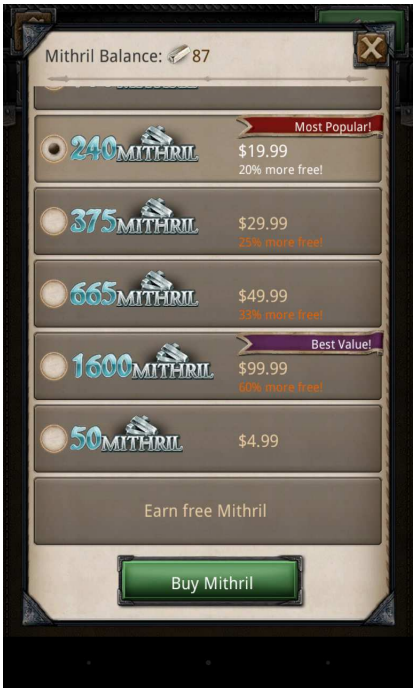


Figure 4.2: Market on one publisher

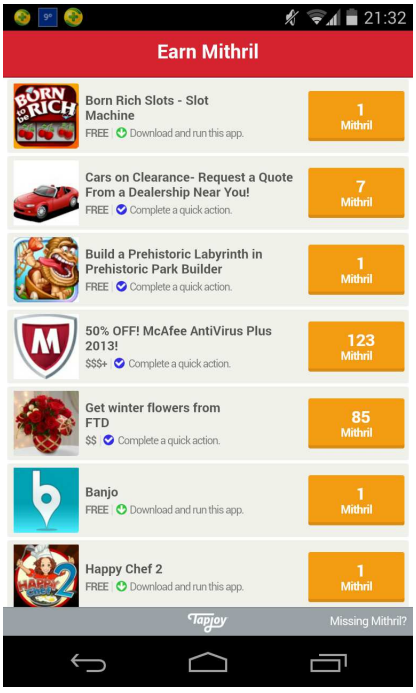


Figure 4.3: Tapjoy Ad offer wall

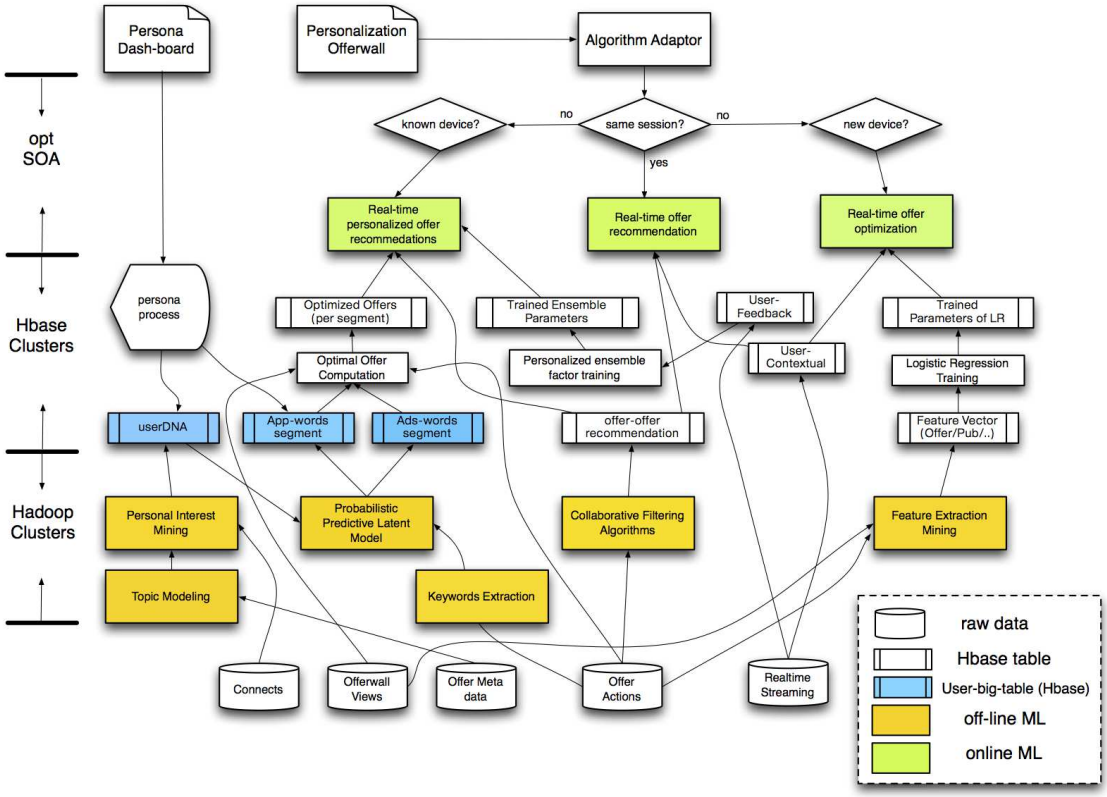


Figure 4.4: Tapjoy Recommender System

4.2 Discussion

A real world recommender system usually involve large amount of off line data processing and storing as well as on line data query, user feedback processing and content serving. The system need a combination of suitable hardware infrastructure and high performance service orientated architect. Fortunately we have good hardware support from Amazon Cloud Service and the board of Tapjoy approved a $3M money on our system. In addition to hardware investment, we have 2 Data Engineer and 3 Data Scientist work on the implementation of optimization SOA, the back end data processing and algorithm implementation. To track the performance of the new algorithm, we also change the original ETL process by adding algorithm label. The A/B testing is performed during the 2013 holiday season from 12/20/2013 - 01/05/2014. The new algorithm is proven to increase revenue per user by 15% which is a big gain. However, this is not the end. Current algorithm does not take account any user life time information at all. As we already see users behavior is quite different when they are in different life stage. Our next step is to utilize life time segmentation on users and do more research on "0" day converters, who are the largest contributor of revenue.

References

[1] Apache Mahout: Scalable machine learning and data mining. *http://mahout.apache.org.*

[2] Spark:Lightning-fast cluster computing. *https://spark.incubator.apache.org/.*

[3] Tom W. Hadoop: The Definitive Guide. *O'Reilly*, Second edition.

[4] Breese, J., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. *Proc. 14th conf on uncertainty in artif. intel.*

[5] Candès, E.J., Romberg, J.K. and Tao, T. (2006). Stable signal recovery from incomplete and inaccurate measurements. *Commun. Pure App. Math.*, **59**, 1207–1223. Wiley.

[6] Chen, S., Donoho, D., and Saunders, M. Automatic decomposition by basis pursuit. *SIAM review*, **43**, 129-159.

[7] Chen, B., He, S., Li, Z., and Zhang, S. (2012). Maximum block improvement and polynomial optimization. *SIAM J. on Optimi.*, 22, 87-107.

[8] Das, A., Dalar, M. and Garg, A. (2007). Google news personalization: scalable online collaborative filtering. *Proc 16th inter. conf. on world wide web.*

[9] Koren, Y.(2010). Collaborative filtering with temporal dynamics. *Communications of the ACM*, **53**, 89–97.

[10] Forbes, P., and Zhu, M. (2011). Content-boosted matrix factorization for recommender systems.: Experiments with recipe recommendation. *Proc the 5th ACM Conf. on Recommender Systems*, 261264.

[11] Little, J. A., and Rubin, D. (2002). *Statistical analysis with missing data*. Second Edition. Wiley.

[12] Lee, T. Q., and Park, Y. (2012). A time-based recommender system using implicit feedback. *The 22nd Intern. WWW Conf.*, Lyon, France.

[13] Liu, J., and Ye, J. (2009). Efficient Euclidean projection in linear time. *ICML*, 2009.

[14] Park, S., Pennock, D., Madani, O., Good, N., and DeCoste, D. (2006). Nave filterbots for robust cold-start recommendations. *Proc. the 12th ACM SIGKDD Intern. Conf. on Knowledge Discovery and Data Mining*, 699705.

[15] McCullagh, P. and Nelder, J.A. (1990). *Generalized linear model*. Second edition. Chapman and Hall.

[16] Mazumder, R., Haste, T. and Tibshirani, R. (2010). Spectral regularization algorithms for learning large incomplete matrices. *J. Mach. Learn. Res.*, 2287-2322.

[17] Nguyen, J., and Zhu, M. (2012). Content-boosted matrix factorization techniques for recommender systems. Tech report, Dept of Stat, Univ of Waterloo.

[18] Mairal, J., Bach, F., Ponce, J, and Sapiro, G. (2009). Online dictionary learning for sparse coding. In *ICML*, 689-696.

[19] Ossiander, M. (1987). A central limit theorem under metric entropy with L_2 bracketing. *Ann. Proba.*, **15**, 897-919.

[20] Raskutii, G., and Wainwright, M. J. (2009). Minimax rates of estimation for high-dimensional linear regression over l_q-balls. Tech report, UC-Berkeley.

[21] Srebro, N., Rennie, J., and Jaakkola, T. (2005). Generalization error bounds for collaborative prediction with low-rank matrices. *Advances in Neural Inform. Processing Sys.*, **17**, 527. MIT Press.

[22] Srebro, N., Rennie, J., and Jaakkola, T. (2005). Maximum-margin matrix factorization. *Advances in Neural Inform. Processing Sys.*, **17**, 13291336. MIT Press.

[23] Takács, G., Pilászy, I., Németh, B., and Tikk, D. (2009). Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learning Res.*, **10**, 623-656.

[24] Shen, X., Pan, W., and Zhu, Y. (2012). Likelihood-based selection and sharp parameter estimation. *J. Ameri. Statist. Assoc.*, **107**, 223-232.

[25] Stanica, P., and Montgomery, A.P. (2001). Good lower and upper bounds on binomial coefficients. *J. Ineq. in Pure. Appl. Math.*, **2**, art 30.

[26] Tibshirani, R. (1996). Regression shrinkage and selection via the LASSO. *JRSS-B*, **58**,267-288.

[27] Tipping, M. E. and Bishop, C. M. (1999). Probabilistic principal component analysis. *JRSS-B*, **61**, 611622.

[28] Van Meteren, R. and Van Someren, M. (2000). Using content-based filtering for recommendation. *Proceedings Mach. Learning in New Info. Age MLnet–ECML2000 Workshop*, p312-321. DOI: 10.1007/11875604_36

[29] Wong, W. H., and Shen, X. (1995). Probability inequalities for likelihood ratios and convergence rates of sieve MLEs. *Ann. Statist.*, **23**, 339-362.

[30] Shen, X. (1998). On the method of penalization. *Statistica Sinica*, **8**, 337-357.

[31] Zhou, Y., Wilkinson, D., Schreiber, R, and Pan, R. (2008). Large-scale collaborative filtering for the Netflix prize. *AAIM*, 337-348.

[32] An, L. and Tao, P. (1997). Solving a class of linearly constrained indefinite quadratic problems by DC algorithms. *Journal of global optimization*, 11(3):253–285.

[33] Chuang, H., Lee, E., Liu, Y., DH., L., and T., I. (2007). Network-based classifica-tion of breast cancer metastasis. *Molecular Systems Biology*, 3:doi : 10.1038/msb4100180.

[34] Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). Least angle regression. *Annals of statistics*, 32(2):407–451.

[35] Fan, J. and Li, R. (2001). Variable selection via nonconvex penalized likelihood and its oracle properties. *J. Amer.Statist.Assoc.*, 96:1348–1360.

[36] Friedman, J., Hastie, T., H
"ofling, H., and Tibshirani, R. (2007). Pathwise coordinate optimization. *Annals*, 1(2):302–332.

[37] Friedman, J., Hastie, T., and Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1.

[38] Li, C. and Li, H. (2008). Network-constrained regularization and variable selection for analysis of genomic data. *Bioinformatics*, 24:1175–1182.

[39] Liu, S., Shen, X., and Wong, W. (2005). Computational developments of ψ-learning. In *Proceedings of The Fifth SIAM International Conference on Data Mining, Newport, California, April*.

[40] Scherzer, C., Eklund, A., Morse, L., Liao, Z., Locascio, J., Fefer, D., Schwarzschild, M., Schlossmacher, M., Hauser, M., Vance, J., Sudarsky, L., Standaert, D., Growdon, J., Jensen, R., and Gullans, S. (2007). Molecular markers of early parkinsons disease based on gene expression in blood. *Proc Natl Acad Sci USA*, 104:955–960.

[41] Shen, X., Pan, W., Zhu, Y., and Zhou, H. (2010). On l_0-regularization in high-dimensinal regression. *manuscript*.

[42] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, B*, 58:267–288.

[43] Tseng, P. (2001). Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, 109(3):475–494.

[44] Tseng, P. and for Intelligent Control Systems (US), C. (1988). *Coordinate ascent for maximizing nondifferentiable concave functions*. Massachusetts Institute of Technology, Laboratory for Information and Decision Systems.

[45] Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag.

[46] Wang, Y., Klijin, J., Zhang, Y., Sieuwerts, A., Look, M., Yang, F., Talantov, D., Timmermans, M., Meijer-van Gelder, M., Yu, J., Jatkoe, T., Berns, E., Atkins, D., and Foekens, J. (2005). Gene-expression profiles to predict distant metastasis of lymph-node-negative primary breast cancer. *Lancet*, 365:671–679.

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

[48] Zhu, Y., Shen, X., and Pan, W. (2009). Support vector machines with disease-gene-centric network penalty for high dimensional micro-array data. Manuscript.

[49] Zhu, Y., Shen, X., and Pan, W. (2008). Network-based support vector machine for classification of microarray samples. *BMC Bioinformatics*.

[50] Zou, H. (2006). The adaptive lasso and its oracle properties. *Journal of the American Statistical Association*, 101(476):1418–1429.

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