Generalized Low Rank Models

Madeleine Udell
Institute for Computational and Mathematical Engineering
Stanford University

Corinne Horn
Department of Electrical Engineering
Stanford University

Reza Zadeh
Institute for Computational and Mathematical Engineering
Stanford University

Stephen Boyd
Department of Electrical Engineering
Stanford University

Abstract

Principal components analysis (PCA) is a well-known technique for approximating a data set represented by a matrix by a low rank matrix. Here, we extend the idea of PCA to handle arbitrary data sets consisting of numerical, Boolean, categorical, ordinal, and other data types. This framework encompasses many well known techniques in data analysis, such as nonnegative matrix factorization, matrix completion, sparse and robust PCA, k-means, and maximum margin matrix factorization. The method handles heterogeneous data sets, and leads to coherent schemes for compressing, denoising, and imputing missing entries across all data types simultaneously. It also admits a number of interesting interpretations of the low rank factors, which allow clustering of examples or of features. We propose a large scale, parallel algorithm for fitting generalized low rank models, and describe implementations and numerical results.

1 Introduction

In application areas like machine learning and data mining, one is often confronted with large collections of high dimensional data. These data sets consist of many heterogeneous data types, with many missing entries. Exploratory data analysis is often difficult in this setting. To better understand a complex data set, one would like to be able to visualize key archetypes, to cluster examples, to find correlated features, to fill in missing entries, and to remove (or identify) spurious or noisy data points. These tasks are difficult because of the heterogeneity of the underlying data.

If the data sets consists only of numerical (real-valued) data, then a simple and well-known technique which supports these tasks is Principal Components Analysis (PCA). PCA finds a low rank matrix that minimizes the approximation error, in the least-squares sense, to the original data set.

We can extend PCA to approximate an arbitrary data set by extending the notion of approximation, replacing least-squares error with an appropriate loss function. We can also add regularization on the low dimensional factors to improve generalization error and avoid overfitting, or to impose or encourage some structure (such as sparsity) in the low dimensional factors. The resulting low rank representation of the data set then admits all the same interpretations familiar from the PCA context. In this paper, we use the term *generalized low rank model* (GLRM) to refer to any low rank approximation of a data set obtained by minimizing a loss function on the approximation error together with regularization of the low dimensional factors.

These low rank approximation problems are not convex, and in general cannot be solved globally and efficiently. However, all of these approximation problems can be heuristically (locally) solved by methods that alternate between updating the two factors in the low rank approximation. Each step involves either a convex problem, or a nonconvex problem that is simple enough that we can solve it exactly. While these alternating methods need not find the globally best low rank approximation, they are often very useful and effective for the original data analysis problem.

Previous work. We are not the first to note that matrix factorization algorithms may be viewed in a unified framework, parametrized by a small number of modeling decisions. The first instance we find in the literature of this unified view appeared in [CDS01], extending PCA to any probabilistic model in the exponential family. Gordon's Generalized^2 Linear^2 models [Gor02] further extended the unified framework to loss functions derived from the generalized Bregman divergence of any convex function, which includes models such as Independent Components Analysis (ICA). Nathan Srebro's PhD thesis [SRJ04] extended the framework to other loss functions (*e.g.*, hinge loss and KL-divergence loss), and added nuclear norm and max-norm regularization. In [SG08], the authors offer a complete view of the state of the literature on matrix factorization as of 2008 in Table 1 of their paper. They note that by changing the loss function and regularization, one may recover algorithms including PCA, weighted PCA, k-means, k-medians, ℓ_1 SVD, probabilistic latent semantic indexing (pLSI), nonnegative matrix factorization with ℓ_2 or KL-divergence loss, exponential family PCA, and MMMF.

Contributions. The present paper differs from previous work in a number of ways. First, we are concerned with the *meaning* of applying these different loss functions and regularizers to approximate a data set. This perspective enables us to extend low rank models to arbitrary data sets, rather than just matrices of real numbers. The resulting models can be used to visualize archetypical examples, to cluster examples, to find correlated features, to fill in missing entries, and to remove (or identify) spurious or noisy data points. The framework allows the user to specify exactly what kinds of factors are desired (*e.g.*, sparse, nonnegative, assignments to clusters, ...) and what kinds of pathologies exist in the data (*e.g.*, small gaussian noise, a few large errors, ...). We describe an algorithm that can solve all problems in our model class, and present two implementations (one serial and one distributed) for modeling and solving GLRMs.

2 Model

Let $A \in \mathbf{R}^{m\times n}$ be a data matrix consisting of m examples each with n features. Thus, A_{ij} is the value of the jth feature in the ith example.

PCA. PCA is one of the oldest and most widely used tools in data analysis [Pea01, Hot33]. PCA seeks a matrix $Z \in \mathbf{R}^{m\times n}$ of rank $k < \min\{m, n\}$ that best approximates A in the least-squares sense. The rank constraint can be encoded implicitly by expressing Z in factored form as $Z = XY$, with $X \in \mathbf{R}^{m\times k}$, $Y \in \mathbf{R}^{k\times n}$. The problem then reduces to choosing the matrices X and Y to minimize $\|A - XY\|_F^2$. The PCA problem can be expressed as

$$\text{minimize} \quad \|A - XY\|_F^2 = \textstyle\sum_{i=1}^m \sum_{j=1}^n (A_{ij} - (XY)_{ij})^2 \tag{1}$$

with variables X and Y.

For convenience, we will define $x_i \in \mathbf{R}^{1\times n}$ to be the ith *row* of X, and $y_j \in \mathbf{R}^m$ to be the jth *column* of Y, and use this notation throughout this paper. Thus $x_i y_j = (XY)_{ij} \in \mathbf{R}$ denotes a dot or inner product.

Generalized low rank models. Suppose now that our data A is a *database* or *table* consisting of m examples (*i.e.*, rows, samples) and n features (*i.e.*, columns, attributes), with entries A_{ij} drawn from a feature set $\mathcal{F}_j$. For example, entries of A can take on real values ($\mathcal{F}_j = \mathbf{R}$), Boolean values ($\mathcal{F}_j = \{T, F\}$), integral values ($\mathcal{F}_j = 1, 2, 3, \ldots$), ordinal values ($\mathcal{F}_j = \{\text{very much, a little, not at all}\}$), or consist of a tuple of these types ($\mathcal{F}_j = \{(a, b) : a \in \mathbf{R}\}$). We observe only entries A_{ij} for $(i, j) \in \Omega \subseteq \{1, \ldots, m\} \times \{1, \ldots, n\}$ from the matrix A, so the other entries are unknown. The user provides a loss function $L_{ij} : \mathbf{R} \times \mathcal{F}_j \to \mathbf{R}$. The loss $L_{ij}(u, a)$ describes the approximation error incurred when we represent a feature value $a \in \mathcal{F}_j$ by the number $u \in \mathbf{R}$. The user also provides regularizers $r_i : \mathbf{R}^k \to \mathbf{R} \cup \{\infty\}$ and $\tilde{r}_j : \mathbf{R}^k \to \mathbf{R} \cup \{\infty\}$ for $i = 1, \ldots n$ and $j = 1, \ldots, m$. These regularizers are used to prevent overfitting to the observations, to encourage the model to have a particularly interpretable form, or to encode side information about the factors.

We now formulate a generalized low rank model on the database A as

$$\text{minimize} \quad \sum_{(i,j)\in\Omega} L_{ij}(x_i y_j, A_{ij}) + \sum_{i=1}^m r_i(x_i) + \sum_{j=1}^n \tilde{r}_j(y_j), \tag{2}$$

with variables $X \in \mathbf{R}^{n\times k}$ and $Y \in \mathbf{R}^{k\times m}$, and with loss L_{ij} and regularizers $r_i(x_i) : \mathbf{R}^{1\times k} \to \mathbf{R}$ and $\tilde{r}_j(y_j) : \mathbf{R}^{k\times 1} \to \mathbf{R}$ as above.

When $L_j(u,a) = (u-a)^2$ for every j, $r = 0$, $\tilde{r} = 0$, and $\Omega = \{1,\dots,m\} \times \{1,\dots,n\}$ (so we have observed every element in A), then the problem reduces to PCA (1). A few examples of GLRMs are given in Table 1.

Model	$\mathbf{L_{ij}(u,a)}$	$\mathbf{r(x)}$	$\mathbf{\tilde{r}(y)}$
PCA	$(u-a)^2$	0	0
Quadratically regularized PCA	$(u-a)^2$	$\|x\|_2^2$	$\|y\|_2^2$
nonnegative matrix factorization	$(u-a)^2$	$I_+(x)$	$I_+(y)$
Sparse PCA	$(u-a)^2$	$\|x\|_1$	$\|y\|_1$
k-means	$(u-a)^2$	$I_1(x)$	0
Robust PCA	$\|u-a\|$	$\|x\|_2^2$	$\|y\|_2^2$
Boolean PCA (MMMF)	$(1-au)_+$	$\|x\|_2^2$	$\|y\|_2^2$
Logistic PCA	$\log(1+\exp(-au))$	$\|x\|_2^2$	$\|y\|_2^2$
Poisson PCA	$\exp(u) - au + a\log a - a$	$\|x\|_2^2$	$\|y\|_2^2$
Ordinal PCA	$\sum_{a'<a}(1-u+a')_+ + \sum_{a'>a}(1+u-a')_+$	$\|x\|_2^2$	$\|y\|_2^2$

Table 1: A few examples of GLRMs. Here I_+ is the indicator of the nonnegative orthant, and I_1 is the indicator of the 1-sparse unit vectors.

3 Interpretations

Archetypes. We can think of each row of Y as an *archetype* which captures the behavior of an idealized example. However, the rows of Y are real numbers. To represent each archetype $l = 1,\dots,k$ in the abstract space as $\mathcal{Y}_l$ with $(Y_l)_j \in \mathcal{F}_j$, we solve

$$(Y_l)_j = \operatorname*{argmin}_{a \in \mathcal{F}_j} L_j(y_{lj}, a).$$

(Here, we assume that the loss $L_{ij} = L_j$ is independent of the example i.)

Archetypical representations. We call x_i the *representation* of example i in terms of the archetypes. The rows of X give an embedding of the examples into $\mathbf{R}^k$, where each coordinate axis corresponds to a different archetype.

In contrast to the initial data, which may consist of arbitrarily complex data types whose entries may be missing or noisy, the representations x_i will be complete denoised low dimensional vectors, and can easily be used in a machine learning algorithm. Using the generalized low rank model, we have converted an abstract feature space into a vector space.

Feature representations. The columns of Y embed the features into $\mathbf{R}^k$. Here, we think of the columns of X as archetypical features, and represent each feature j as a linear combination of the archetypical features. Just as with the examples, we might choose to apply any machine learning algorithm to the feature representations.

This procedure allows us to compare non-numeric features using their representation in $\mathbf{R}^l$. For example, if the features $\mathcal{F}$ are Likert variables giving the extend to which respondents on a questionnaire agree with statements $1,\dots,n$, we might be able to say that questions i and j are similar if $\|y_i - y_j\|$ is small; or that question i is a more polarizing form of question j if $y_i = \alpha y_j$.

Latent variables. Each row of X represents an example by a vector in $\mathbf{R}^k$. The matrix Y maps these representations (nonlinearly) back into the original feature space. We might think of X as discovering the *latent variables* that best explain the observed data, with the added benefit that these latent variables lie in the vector space $\mathbf{R}^k$. If the approximation error $\sum_{(i,j)\in\Omega} L_{ij}(x_i y_j, A_{ij})$ is small, then we view these latent variables as providing a good explanation or summary of the full data set.

Compression. We impose an *information bottleneck* on the data by using a low rank auto-encoder to fit the data. The solution (X,Y) to problem (2) maximizes the information transmitted through this k-dimensional bottleneck, measured according to the loss functions L_{ij}. This X and Y give a compressed and real-valued representation that may be used to more efficiently store or transmit the information present in the data.

Probabilistic intepretation and data imputation. We can give a probabilistic interpretation of X and Y. We suppose that the matrices $\bar{X}$ and $\bar{Y}$ are generated according to a probability distribution with probability proportional to $\exp(-r(\bar{X}))$ and $\exp(-\tilde{r}(\bar{Y}))$, respectively. Our observations A of the entries in the matrix $\bar{Z} = \bar{X}\bar{Y}$ are given by

$$A_{ij} = \psi_{ij}((\bar{X}\bar{Y})_{ij}),$$

where the random variable $\psi_{ij}(u)$ takes value a with probability proportional to

$$\exp\left(-L_{ij}(u,a)\right).$$

We observe each entry $(i,j) \in \Omega$. Then to find the maximum a posteriori (MAP) estimator (X,Y) of $(\bar{X},\bar{Y})$, we solve

$$\text{maximize} \qquad \exp\left(-\sum_{(i,j)\in\Omega} L_{ij}(x_i y_j, A_{ij})\right) \exp(-r(X)) \exp(-\tilde{r}(Y)),$$

which is equivalent, by taking logs, to the generalized low rank model (2).

Missing data and data imputation We can use the solution (X,Y) to a low rank model to impute values corresponding to missing data $(i,j) \notin \Omega$. This process is sometimes also called *inference*.

To fill in missing entries in the original matrix A, we compute the value $\hat{A}_{ij}$ that minimizes the loss for $x_i y_j$, given the MAP estimators (X,Y):

$$\hat{A}_{ij} = \operatorname*{argmin}_{a} L_{ij}(x_i y_j, a).$$

This equation implicitly constrains $\hat{A}_{ij}$ to lie in the domain $\mathcal{F}_j$ of L_{ij}. When $L_{ij} : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$, then $\hat{A}_{ij} = x_i y_j$. But when the data is of an abstract type, the minimum $\operatorname{argmin}_a L_{ij}(u,a)$ will not in general be equal to u. For example, when the data is Boolean, $L_{ij} : \{0,1\} \times \mathbf{R} \to \mathbf{R}$, we compute the Boolean matrix $\hat{A}$ implied by our low rank model by solving

$$\hat{A}_{ij} = \operatorname*{argmin}_{a\in\{0,1\}} L_{ij}(u,a).$$

When $\mathcal{F}_j$ is finite, inference *partitions* the real numbers into regions

$$\mathcal{R}_a = \{x \in \mathbf{R} : L_{ij}(u,x) = \min_{a} L_{ij}(u,a)\}$$

corresponding to different values $a \in \mathcal{F}_j$. When L_{ij} is convex, these regions are intervals.

We can use the estimate $\hat{A}_{ij}$ even when $(i,j) \in \Omega$ *was* observed. If the original observations have been corrupted by noise, we can view $\hat{A}_{ij}$ as a denoised version of the original data. This is an unusual kind of denoising: both the noisy (A_{ij}) and denoised $(\hat{A}_{ij})$ versions of the data lie in the *abstract* space $\mathcal{F}_j$.

4 Numerical example

In this experiment, we fit a GLRM to a data table with numerical, Boolean, and ordinal columns generated as follows. Let $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$ partition the column indices $1,\ldots,n$. Choose $X^{\text{true}} \in \mathbf{R}^{m\times k_{\text{true}}}$, $Y^{\text{true}} \in \mathbf{R}^{k_{\text{true}}\times n}$ to have independent, standard normal entries. Assign entries of A as follows:

$$A_{ij} = \begin{cases} x_i y_j & j \in \mathcal{N}_1 \\ \mathbf{sign}\,(x_i y_j) & j \in \mathcal{N}_2 \\ \mathbf{round}(3x_i y_j + 1) & j \in \mathcal{N}_3, \end{cases}$$

where the function **round** maps a to the nearest integer in the set $\{1,\ldots,7\}$. Thus, $\mathcal{N}_1$ corresponds to real-valued data; $\mathcal{N}_2$ corresponds to Boolean data; and $\mathcal{N}_3$ corresponds to ordinal data. We then censor one large block of entries in the table (constituting 3.75% of numerical, 50% of Boolean, and 50% of ordinal data), removing them from the observed set Ω. We consider a problem instance in which $m = 100$, $n_1 = 40$, $n_2 = 30$, $n_3 = 30$, and $k_{\text{true}} = k = 10$.

We fit a heterogeneous loss GLRM to this data with loss function

$$L_{ij}(u,a) = \begin{cases} L_{\text{real}}(u,a) & j \in \mathcal{N}_1 \\ L_{\text{bool}}(u,a) & j \in \mathcal{N}_2 \\ L_{\text{ord}}(u,a) & j \in \mathcal{N}_3, \end{cases}$$

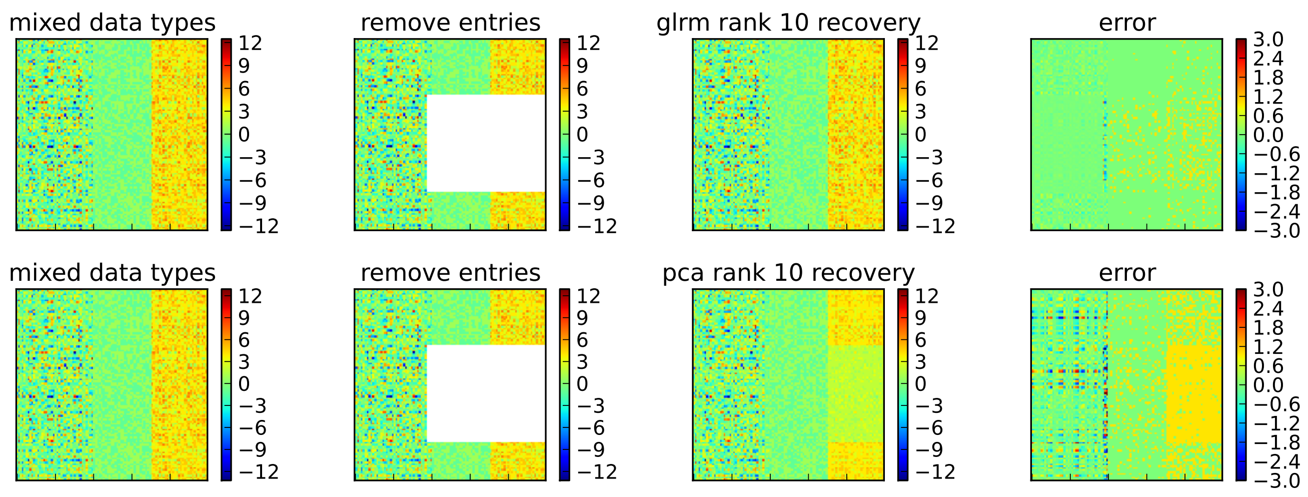

Figure 1: GLRMs on missing data. The top row uses a heterogeneous loss, while the bottom row uses quadratic loss.

where $L_{\text{real}}(u,a) = (u-a)^2$, $L_{\text{bool}}(u,a) = \max(0, 1-au)$, and

$$L_{\text{ord}}(u,a) = \sum_{a'=1}^{d-1} (1 - I_{a>a'} u_{a'})_+,$$

and with quadratic regularization $r(u) = \tilde{r}(u) = .1\|u\|_2^2$. We fit the GLRM to produce the model $(X^{\text{mix}}, Y^{\text{mix}})$. For comparison, we also fit quadratically regularized PCA to the same data, using $L_{ij}(u,a) = (u-a)^2$ for all j and quadratic regularization $r(u) = \tilde{r}(u) = .1\|u\|_2^2$, to produce the model $(X^{\text{real}}, Y^{\text{real}})$.

Figure 1 shows the results of fitting the GLRMs described above to the censored data. The first column shows the original ground-truth data A; the second shows the block of data that has been removed from the observation set Ω; the third shows the imputed data given the model, $\hat{A}$, generated by rounding the entries of XY to the closest number in $\{0,1\}$; the fourth shows the error $A - \hat{A}$. The heterogeneous loss GLRM performs much better than quadradically regularized PCA.

5 Algorithms

The matrix factorization literature presents a wide variety of algorithms to solve special cases of our problem. For example, there are variants on alternating Newton methods [Gor02, SG08], (stochastic or incremental) gradient descent [NRRW11, RR11, LRS$^+$10, BRRT12], conjugate gradients [RS05, SJ03], expectation minimization (EM) [SJ03], multiplicative updates [LS99] and semidefinite programming [SRJ04]. Generally, expectation minimization has been found to underperform relative to other methods [SG08], while semidefinite programming becomes computationally intractable for very large (or even just large) scale problems [RS05].

Here, we discuss a class of methods to fit GLRMs based on alternating updates of X given Y and of Y given X. Algorithm 1 presents this templated algorithmic scheme. Here, the loops over $i = 1, \ldots, N$ and over $j = 1, \ldots, M$ may both be executed in parallel.

Algorithm 1 implements *alternating minimization* if

$$\underset{L,r}{\textbf{update}}(x_i^{k-1}, Y^{k-1}, A) = \underset{x}{\operatorname{argmin}} \left(\sum_{j:(i,j)\in\Omega} L_{ij}(x y_j^{k-1}, A_{ij}) + r(x) \right).$$

However, it is not very useful to spend a lot of effort optimizing over X before we have a good estimate for Y. In general, we may consider any update rule that moves towards the minimum. Empirically, we find that this approach

Algorithm 1

```
given $X^0, Y^0$
for $k = 1, 2, \ldots$ do
    for $i = 1, \ldots, M$ do
        $x_i^k = \mathbf{update}_{L,r}(x_i^{k-1}, Y^{k-1}, A)$
    end for
    for $j = 1, \ldots, N$ do
        $y_j^k = \mathbf{update}_{L,\tilde{r}}(y_j^{(k-1)T}, X^{(k)T}, A^T)$
    end for
end for
```

often finds a better local minimum than performing a full optimization over each factor in every iteration, in addition to saving computational effort on each iteration.

For example, in the implementations described below in §6, we use a proximal gradient update $\mathbf{update}_{L,r}$ that takes a single proximal gradient step on the objective. Recall that the proximal operator of a function f [PB13] is

$$\mathbf{prox}_f(z) = \operatorname*{argmin}_x (f(x) + \frac{1}{2}\|x - z\|_2^2).$$

For example, if f is the indicator function of a set $\mathcal{C}$, the proximal operator of f is just (Euclidean) projection onto $\mathcal{C}$. A proximal gradient update is implemented as follows. Let

$$g_i = \sum_{j:(i,j)\in\Omega} \nabla L_{ij}(x_i y_j, A_{ij}) y_j.$$

(If any of these functions L_{ij} are not differentiable, replace ∇L_{ij} by any subgradient of L_{ij} [BXM03].) Then set

$$x_i^k = \mathbf{prox}_{\alpha_k r}(x_i^{k-1} - \alpha_k g_i),$$

for some step size α_k. In numerical experiments, we find that using a fixed step size α on the order of $1/\|g\|_2$ gives fast convergence in practice. Notice that the updates for each row of X are executed independently, so different step sizes may be used for each.

5.1 Offsets and scaling

It is standard practice to *standardize* the data before appplying PCA: the column means are subtracted from each column, and the columns are normalized by their variances. On the other hand, in other GLRMs such as nonnegative matrix factorization, it can be disastrous to standardize the data, since subtracting column means may introduce negative entries into the matrix or otherwise worsen the fit of the model.

In PCA, standardization rescales the data in order to compensate for unequal scaling in different features. It is possible to instead rescale the *loss functions* in order to compensate for unequal scaling. The scaling proposed here generalizes the idea of standardization to a setting with heterogeneous loss funtions.

Given initial loss functions L_{ij}, for each feature j let

$$\mu_j = \operatorname*{argmin}_\mu \sum_{i:(i,j)\in\Omega} L_{ij}(\mu, A_{ij}), \qquad \sigma_j = \frac{1}{n_j - 1} \sum_{i:(i,j)\in\Omega} L_{ij}(\mu_j, A_{ij}).$$

It is easy to see that μ_j generalizes the mean of column j, while σ_j generalizes the variance. For example, if $L_{ij}(u, a) = (u - a)^2$ for every $i = 1, \ldots, m$, $j = 1, \ldots, n$, then μ_j will be the mean of the jth column of A; but if $L_{ij}(u, a) = |u - a|$ for every $i = 1, \ldots, m$, $j = 1, \ldots, n$, then μ_j will be the median of the jth column of A.

We rescale the loss functions by σ_j and solve

$$\text{minimize} \quad \textstyle\sum_{(i,j)\in\Omega} L_{ij}(A_{ij}, x_i y_j + \mu_j)/\sigma_j + \sum_{i=1}^m r_i(x_i) + \sum_{j=1}^n \tilde{r}_j(y_j) \qquad (3)$$

with variables X, Y, and μ.

Note that this problem can be recast in the standard form for a generalized low rank model (2) by rescaling the loss functions and modifying the regularizers to encode the offset in the $k + 1$th row of Y.

6 Implementations

The authors have developed and released two codes for modelling and fitting generalized low rank models: a serial implementation written in Julia, and a distributed implementation written in Scala using the Spark framework. Both use the proximal gradient method described in §5 to fit GLRMs. In this section we briefly discuss these implementations, and report some timing results. For a full description and up-to-date information about available functionality, we encourage the reader to consult the on-line documentation. In both implementations, the user may specify a loss function (regularizer) for every entry $(i, j) \in \Omega$ by choosing one from a list of pre-existing losses (regularizers), or may specify a new loss (regularizer) by implementing the gradient (prox) of the loss (regularizer).

6.1 Julia implementation

`LowRankModels` is a code written in Julia [BKSE12] for modelling and fitting GLRMs. The implementation is available on-line at

`https://github.com/madeleineudell/LowRankModels.jl`.

Usage. To form a GLRM using `LowRankModels`, the user specifies

- the data `A` (A), which can be any array or array-like data structure (*e.g.*, a Julia `DataFrame`);
- the observed entries `obs` (Ω), a list of tuples of the indices of the observed entries in the matrix, which may be omitted if all the entries in the matrix have been observed;
- the list of loss functions `losses` (L_j, $j = 1, \ldots, n$), one for each column of `A`;
- the regularizers `rx` (r) and `ry` ($\tilde{r}$); and
- the rank `k` (k).

For example, the following code forms and fits a k-means model with $k = 5$ on the matrix $A \in \mathbf{R}^{m \times n}$.

```
losses = fill(quadratic(),n)    # quadratic loss
rx = onesparse()                # x is 1-sparse
ry = zeroreg()                  # y is not regularized
glrm = GLRM(A,losses,rx,ry,k)   # form GLRM
X,Y,ch = fit(glrm)              # fit GLRM
```

The optimal model is returned in the factors `X` and `Y`, while `ch` gives the convergence history.

Automatic modeling. `LowRankModels` is capable of adding offsets to a GLRM, and of automatically scaling the loss functions, as described in §5.1. It can also automatically detect the types of different columns of a data frame and select an appropriate loss. Using these features, `LowRankModels` implements a method

```
glrm(dataframe, k)
```

that forms a rank k model on a data frame, automatically selecting loss functions and regularization that suit the data well, and ignoring any missing (`NA`) element in the data frame. This GLRM can then be fit with the function `fit`.

6.2 Spark implementation

`SparkGLRM` is a code written in Scala, built on the Spark cluster programming framework [ZCF$^+$10], for modelling and fitting GLRMs. The implementation is available on-line at

`http://git.io/glrmspark`.

Design. The data matrix A is split entry-wise across many machines, just as in [HMLZ14]. The model (factors X and Y) is replicated and stored in memory on every machine. Thus the total computation time required to fit the model is proportional to the number of nonzeros divided by the number of cores, with the restriction that the model should fit in memory. (The authors leave to future work an extension to models that do not fit in memory, *e.g.*, by using a parameter server [SSZ14].) Where possible, hardware acceleration (via breeze and BLAS) is used for local linear algebraic operations.

At every iteration, the current model is broadcast to all machines, so there is only one copy of the model on each machine. This particularly important in machines with many cores, because it avoids duplicating the model those machines. Each core on a machine will process a partition of the input matrix, using the local copy of the model.

Experiments. We ran experiments on several large matrices. For size comparison, a very popular matrix in the recommender systems community is the Netflix Prize Matrix, which has 17770 rows, 480189 columns, and 100480507 nonzeros. Below we report results on several larger matrices, up to 10 times larger. The matrices are generated by fixing the dimensions and number of nonzeros per row, then uniformly sampling the locations for the nonzeros, and finally filling in those locations with a uniform random number in $[0, 1]$.

We report iteration times using an Amazon EC2 cluster with 10 slaves and one master, of instance type "c3.4xlarge". Each machine has 16 CPU cores and 30 GB of RAM. We ran `SparkGLRM` to fit two GLRMs on matrices of varying sizes. Table 2 gives results for quadratically regularized PCA (*i.e.*, quadratic loss and quadratic regularization) with $k = 5$. To illustrate the capability to write and fit more specialized loss functions, we also fit a GLRM using a custom loss function

$$L_{ij}(u, a) = \begin{cases} |u - a| & i + j \text{ is even} \\ (u - a)^2 & i + j \text{ is odd,} \end{cases}$$

with $r(x) = \|x\|_1$ and $\tilde{r}(y) = \|y\|_2^2$, setting $k = 10$. We report results for this custom GLRM in Table 3.

Matrix size	**# nonzeros**	**Time per iteration (s)**
$10^6 \times 10^6$	10^6	7
$10^6 \times 10^6$	10^9	11
$10^7 \times 10^7$	10^9	227

Table 2: `SparkGLRM` for quadratically regularized PCA, $k = 5$.

Matrix size	**# nonzeros**	**Time per iteration (s)**
$10^6 \times 10^6$	10^6	9
$10^6 \times 10^6$	10^9	13
$10^7 \times 10^7$	10^9	294

Table 3: `SparkGLRM` for custom GLRM, $k = 10$.

The table gives the time per iteration. The number of iterations required for convergence depends on the size of the ambient dimension. On the matrices with the dimensions shown in Tables 2 and 3, convergence typically requires about 100 iterations, but we note that useful GLRMs often emerge after only a few tens of iterations.

Acknowledgements

The authors are grateful to Lester Mackey, Ben Recht, Chris Ré, Ashok Srivastava, Haesun Park, and Nicolas Gillis for comments on early drafts of this paper, and to Matei Zaharia and Debasish Das for their insights into creating a successful Spark implementation. This work was developed with support from the National Science Foundation Graduate Research Fellowship program (under Grant No. DGE-1147470), the Gabilan Stanford Graduate Fellowship, the Gerald J. Lieberman Fellowship, and the DARPA X-DATA program.

References

[BKSE12] J. Bezanson, S. Karpinski, V. B. Shah, and A. Edelman. Julia: A fast dynamic language for technical computing. *arXiv preprint arXiv:1209.5145*, 2012.

[BRRT12] V. Bittorf, B. Recht, C. Ré, and J. A. Tropp. Factoring nonnegative matrices with linear programs. *Advances in Neural Information Processing Systems*, 25:1223–1231, 2012.

[BXM03] S. Boyd, L. Xiao, and A. Mutapcic. Subgradient methods. *Lecture notes for EE364b, Stanford University*, 2003.

[CDS01] M. Collins, S. Dasgupta, and R. E. Schapire. A generalization of principal component analysis to the exponential family. In *Advances in Neural Information Processing Systems*, volume 13, page 23, 2001.

[Gor02] G. J. Gordon. Generalized^2 linear^2 models. In *Advances in Neural Information Processing Systems*, pages 577–584, 2002.

[HMLZ14] T. Hastie, R. Mazumder, J. Lee, and R. Zadeh. Matrix completion and low-rank svd via fast alternating least squares. *arXiv*, 2014.

[Hot33] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24(6):417, 1933.

[LRS$^+$10] J. D. Lee, B. Recht, R. Salakhutdinov, N. Srebro, and J. A. Tropp. Practical large-scale optimization for max-norm regularization. In *Advances in Neural Information Processing Systems*, pages 1297–1305, 2010.

[LS99] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[NRRW11] F. Niu, B. Recht, C. Ré, and S. J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, 2011.

[PB13] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):123–231, 2013.

[Pea01] K. Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[RR11] B. Recht and C. Ré. Parallel stochastic gradient algorithms for large-scale matrix completion. *Optimization Online*, 2011.

[RS05] J. D. M. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 713–719. ACM, 2005.

[SG08] A. P. Singh and G. J. Gordon. A unified view of matrix factorization models. In *Machine Learning and Knowledge Discovery in Databases*, pages 358–373. Springer, 2008.

[SJ03] N. Srebro and T. Jaakkola. Weighted low-rank approximations. In *ICML*, volume 3, pages 720–727, 2003.

[SRJ04] N. Srebro, J. D. M. Rennie, and T. Jaakkola. Maximum-margin matrix factorization. In *Advances in Neural Information Processing Systems*, volume 17, pages 1329–1336, 2004.

[SSZ14] S. Schelter, V. Satuluri, and R. Zadeh. Factorbird — a parameter server approach to distributed matrix factorization. *NIPS 2014 Workshop on Distributed Machine Learning and Matrix Computations*, 2014.

[ZCF$^+$10] M. Zaharia, M. Chowdhury, M. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on hot topics in cloud computing*, page 10, 2010.