Relational Learning via Collective Matrix Factorization

Ajit P. Singh
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
ajit@cs.cmu.edu

Geoffrey J. Gordon
Machine Learning Department
Carnegie Mellon University
Pittsburgh, PA 15213
ggordon@cs.cmu.edu

ABSTRACT

Relational learning is concerned with predicting unknown values of a relation, given a database of entities and observed relations among entities. An example of relational learning is movie rating prediction, where entities could include users, movies, genres, and actors. Relations encode users' ratings of movies, movies' genres, and actors' roles in movies. A common prediction technique given one pairwise relation, for example a #users $\times$ #movies ratings matrix, is low-rank matrix factorization. In domains with multiple relations, represented as multiple matrices, we may improve predictive accuracy by exploiting information from one relation while predicting another. To this end, we propose a collective matrix factorization model: we simultaneously factor several matrices, sharing parameters among factors when an entity participates in multiple relations. Each relation can have a different value type and error distribution; so, we allow nonlinear relationships between the parameters and outputs, using Bregman divergences to measure error. We extend standard alternating projection algorithms to our model, and derive an efficient Newton update for the projection. Furthermore, we propose stochastic optimization methods to deal with large, sparse matrices. Our model generalizes several existing matrix factorization methods, and therefore yields new large-scale optimization algorithms for these problems. Our model can handle any pairwise relational schema and a wide variety of error models. We demonstrate its efficiency, as well as the benefit of sharing parameters among relations.

Categories and Subject Descriptors

H.1.1 [**Information Systems**]: Models and Principles; G.1.6 [**Optimization**]: Nonlinear programming, Stochastic programming

General Terms

Algorithms, Theory, Experimentation

KDD'08, August 24–27, 2008, Las Vegas, Nevada, USA.

1. INTRODUCTION

Relational data consists of entities and relations between them. In many cases, such as relational databases, the number of entity types and relation types are fixed. Two important tasks in such domains are *link prediction*, determining whether a relation exists between two entities, and *link regression*, determining the value of a relation between two entities given that the relation exists.

Many relational domains involve only one or two entity types: documents and words; users and items; or academic papers where links between entities represent counts, ratings, or citations. In such domains, we can represent the links as an $m \times n$ matrix X: rows of X correspond to entities of one type, columns of X correspond to entities of the other type, and the element X_{ij} indicates either whether a relation exists between entities i and j. A low-rank factorization of X has the form $X \approx f(UV^T)$, with factors $U \in \mathbb{R}^{m \times k}$ and $V \in \mathbb{R}^{n \times k}$. Here $k > 0$ is the rank, and f is a possibly-nonlinear *link function*. Different choices of f and different definitions of $\approx$ lead to different models: minimizing squared error with an identity link yields the singular value decomposition (corresponding to a Gaussian error model), while other choices extend generalized linear models [26] to matrices [14, 17] and lead to error models such as Poisson, Gamma, or Bernoulli distributions.

In domains with more than one relation matrix, one could fit each relation separately; however, this approach would not take advantage of any correlations between relations. For example, a domain with users, movies, and genres might have two relations: an integer matrix representing users' ratings of movies on a scale of 1–5, and a binary matrix representing the genres each movie belongs to. If users tend to rate dramas higher than comedies, we would like to exploit this correlation to improve prediction.

To do so, we extend generalized linear models to arbitrary relational domains. We factor each relation matrix with a generalized-linear link function, but whenever an entity type is involved in more than one relationship, we tie factors of different models together. We refer to this approach as *collective matrix factorization*.

We demonstrate that a general approach to collective matrix factorization can work efficiently on large, sparse data sets with relational schemas and nonlinear link functions. Moreover, we show that, when relations are correlated, collective matrix factorization can achieve higher prediction accuracy than factoring each matrix separately. Our code is available under an open license.[1]

[1]Source code is available at `http://www.cs.cmu.edu/`

2. A UNIFIED VIEW OF FACTORIZATION

The building block of collective factorization is single-matrix factorization, which models a single relation between two entity types $\mathcal{E}_1$ and $\mathcal{E}_2$. If there are m entities of type $\mathcal{E}_1$ and n of type $\mathcal{E}_2$, we write $X \in \mathbb{R}^{m\times n}$ for our matrix of observations, and $U \in \mathbb{R}^{m\times k}$ and $V \in \mathbb{R}^{n\times k}$ for the low-rank factors. A factorization algorithm can be defined by the following choices, which are sufficient to include most existing approaches (see Sec. 2.2 for examples):

1. Prediction link $f : \mathbb{R}^{m\times n} \to \mathbb{R}^{m\times n}$
2. Loss function $\mathcal{D}(X, f(UV^T)) \geq 0$, a measure of the error in predicting $f(UV^T)$ when the answer is X.
3. Optional data weights $W \in \mathbb{R}_+^{m\times n}$, which if used must be an argument of the loss.
4. Hard constraints on factors, $(U, V) \in \mathcal{C}$
5. Regularization penalty, $\mathcal{R}(U, V) \geq 0$.

For the model $X \approx f(UV^T)$, we solve:

$$\operatorname*{argmin}_{(U,V)\in\mathcal{C}} [\mathcal{D}(X, f(UV^T)) + \mathcal{R}(U, V)]. \tag{1}$$

The loss $\mathcal{D}(\cdot,\cdot)$ quantifies $\approx$ in the model. It is typically convex in its second argument, and often decomposes into a weighted sum over the elements of X. For example, the loss for weighted SVD [32] is

$$\mathcal{D}_W(X, UV^T) = ||W \odot (X - UV^T)||^2_{Fro},$$

where $\odot$ denotes the element-wise product of matrices.

Prediction links f allow nonlinear relationships between UV^T and the data X. The choices of f and $\mathcal{D}$ are closely related to distributional assumptions on X; see Section 2.1. Common regularizers for linear models, such as ℓ_p-norms, are easily adapted to matrix factorization. Other regularizers have been proposed specifically for factorization; for example, the trace norm of UV^T, the sum of its singular values, has been proposed as a continuous proxy for rank [33]. For clarity, we treat hard constraints $\mathcal{C}$ separately from regularizers. Examples of hard constraints include orthogonality; stochasticity of rows, columns, or blocks (for example, in matrix co-clustering each row of U and V sums to 1); non-negativity; and sparsity or cardinality.

2.1 Bregman Divergences

A large class of matrix factorization algorithms restrict $\mathcal{D}$ to generalized Bregman divergences: *e.g.*, singular value decomposition [16] and non-negative matrix factorization [21].

DEFINITION 1 ([17]). *For a closed, proper, convex function $F : \mathbb{R}^{m\times n} \to \mathbb{R}$, the generalized Bregman divergence between matrices Z and Y is*

$$\mathbb{D}_F(Z \,||\, Y) = F(Z) + F^*(Y) - Y \circ Z$$

where $A \circ B$ is the matrix dot product $\operatorname{tr}(A^T B) = \sum_{ij} A_{ij} B_{ij}$ and F^* is the convex dual $F^*(\mu) = \sup_{\theta\in\operatorname{dom} F} [\langle\theta, \mu\rangle - F(\theta)]$.

If F^* is differentiable, this is equivalent to the standard definition [10, 11], except that the standard definition uses arguments Z and $\nabla F^*(Y)$ instead of Z and Y. If F decomposes into a sum over components of Z, we can define a *weighted* divergence, overloading F to denote a single component of the sum,

$$\mathbb{D}_F(Z \,||\, Y, W) = \sum_{ij} W_{ij} \left(F(Z_{ij}) + F^*(Y_{ij}) - Y_{ij} Z_{ij}\right).$$

Examples include weighted versions of squared loss, $F(x) = x^2$, and I-divergence, $F(x) = x \log x - x$. Our primary focus is on decomposable regular Bregman divergences [6], which correspond to maximum likelihood in exponential families:

DEFINITION 2. *A parametric family of distributions $\psi_F = \{p_F(x|\theta) : \theta\}$ is a regular exponential family if each density has the form*

$$\log p_F(x|\theta) = \log p_0(x) + \theta^T x - F(\theta)$$

where θ is the vector of natural parameters for the distribution, x is the vector of minimal sufficient statistics, and $F(\theta)$ is the log-partition function

$$F(\theta) = \log \int p_0(x) \cdot \exp(\theta^T x)\, dx.$$

A distribution in ψ_F is uniquely identified by its natural parameters. For regular exponential families

$$\log p_F(x|\theta) = \log p_0(x) + F^*(x) - \mathbb{D}_{F^*}(x \,||\, f(\theta))$$

where the *matching* prediction link is $f(\theta) = \nabla F(\theta)$ [15, 4, 14, 6]. Minimizing a Bregman divergence under a matching link is equivalent to maximum likelihood for the corresponding exponential family distribution.

The relationship between matrix factorization and exponential families is seen by treating the data matrix X as a collection of samples, $\mathcal{X} = \{X_{11}, \ldots, X_{mn}\}$. Modeling $X = f(UV^T)$, we have that X_{ij} is drawn from the distribution in ψ_F with natural parameter $(UV^T)_{ij}$.

Decomposable losses, which can be expressed as the sum of losses over elements, follows from matrix exchangeability [2, 3]. A matrix X is row-and-column exchangeable if permuting the rows and columns of X does not change the distribution of $\mathcal{X}$. For example, if X is a document-word matrix of counts, the relative position of two documents in the matrix is unimportant, the rows are exchangeable; likewise for words. A surprising consequence of matrix exchangeability is that the distribution of $\mathcal{X}$ can be described by a function of a global matrix mean, row and column effects (*e.g.*, row biases, column biases), and a per-element effect (*e.g.*, the natural parameters UV^T above). The per-element effect leads naturally to decomposable losses. An example where decomposability is not a legitimate assumption is when one dimension indexes a time-varying quantity.

2.2 Examples

The simplest case of matrix factorization is the singular value decomposition: the data weights are constant, the prediction link is the identity function, the divergence is the sum of squared errors, and the factors are unregularized. A hard constraint that one factor is orthogonal and the other orthonormal ensures uniqueness of the global optimum (up to permutations and sign changes), which can be found using Gaussian elimination or the Power method [16].

Variations of matrix factorization change one or more of the above choices. Non-negative matrix factorization [21] maximizes the objective

$$X \circ \log(UV^T) - \mathbf{1} \circ UV^T \tag{2}$$

`~ajit/cmf`. A longer version of the paper is available as a technical report [31]

where $\mathbf{1}$ is a matrix with all elements equal to 1. Maximizing Equation 2 is equivalent to minimizing the I-divergence $\mathbb{D}_H(X \,||\, \log(UV^T))$ under the constraints $U, V \geq 0$. Here $H(x) = x\log(x) - x$. The prediction link is $f(\theta) = \log(\theta)$.

The scope of matrix factorizations we consider is broader than [17], but the same alternating Newton-projections approach (see Sections 4-5) can be generalized to all the following scenarios, as well as to collective matrix factorization: (i) constraints on the factors, which are not typically considered in Bregman matrix factorization as the resulting loss is no longer a regular Bregman divergence. Constraints allow us to place methods like non-negative matrix factorization [21] or matrix co-clustering into our framework. (ii) non-Bregman matrix factorizations, such as max-margin matrix factorization [30], which can immediately take advantage of the large scale optimization techniques in Sections 4-5; (iii) row and column biases, where a column of U is paired with a fixed, constant column in V (and vice-versa). If the prediction link and loss correspond to a Bernoulli distribution, then margin losses are special cases of biases; (iv) methods based on plate models, such as pLSI [19], can be placed in our framework just as well as methods that factor data matrices. While these features can be added to collective matrix factorization, we focus primarily on relational issues herein.

3. RELATIONAL SCHEMAS

A relational schema contains t entity types, $\mathcal{E}_1 \ldots \mathcal{E}_t$. There are n_i entities of type i, denoted $\{x_e^{(i)}\}_{e=1}^{n_i}$. A relation between two types is $\mathcal{E}_i \sim_u \mathcal{E}_j$; index $u \in \mathbb{N}$ allows us to distinguish multiple relations between the same types, and is omitted when no ambiguity results. In this paper, we only consider binary relations. The matrix for $\mathcal{E}_i \sim_u \mathcal{E}_j$ has n_i rows, n_j columns, and is denoted $X^{(ij,u)}$. If we have not observed the values of all possible relations, we fill in unobserved entries with 0 (so that $X^{(ij,u)}$ is a sparse matrix), and assign them zero weight when learning parameters. By convention, we assume $i \leq j$. Without loss of generality, we assume that it is possible to traverse links from any entity type to any other; if not, we can fit each connected component in the schema separately. This corresponds to a fully connected entity-relationship model [12].

We fit each relation matrix as the product of latent factors, $X^{(ij)} \approx f^{(ij)}(U^{(i)}(U^{(j)})^T)$, where $U^{(i)} \in \mathbb{R}^{n_i \times k_{ij}}$ and $U^{(j)} \in \mathbb{R}^{n_j \times k_{ij}}$ for $k_{ij} \in \{1, 2, \ldots\}$. Unless otherwise noted, the prediction link $f^{(ij)}$ is an element-wise function on matrices. If $\mathcal{E}_j$ participates in more than one relation, we allow our model to use only a subset of the columns of $U^{(j)}$ for each one. This flexibility allows us, for example, to have relations with different latent dimensions, or to have more than one relation between $\mathcal{E}_i$ and $\mathcal{E}_j$ without forcing ourselves to predict the same value for each one. In an implementation, we would store a list of participating column indices from each factor for each relation; but to avoid clutter, we ignore this possibility in our notation.

4. COLLECTIVE FACTORIZATION

For concision, we introduce collective matrix factorization on the three-entity-type schema $\mathcal{E}_1 \sim \mathcal{E}_2 \sim \mathcal{E}_3$, and use simplified notation: the two data matrices are $X = X^{(12)}$ and $Y = X^{(23)}$, of dimensions $m = n_1$, $n = n_2$, and $r = n_3$. The factors are $U = U^{(1)}$, $V = U^{(2)}$, and $Z = U^{(3)}$. The latent dimension is $k = k_{12} = k_{23}$. The weight matrix for X is W, and the weight matrix for Y is $\tilde{W}$. Since $\mathcal{E}_2$ participates in both relations, we use the factor V in both reconstructions: $X \approx f_1(UV^T)$ and $Y \approx f_2(VZ^T)$.

An example of this schema is collaborative filtering: $\mathcal{E}_1$ are users, $\mathcal{E}_2$ are movies, and $\mathcal{E}_3$ are genres. X is a matrix of observed ratings, and Y indicates which genres a movie belongs to (each column corresponds to a genre, and movies can belong to multiple genres).

One model of Bregman matrix factorization [17] proposes the following decomposable loss function for $X \approx f_1(UV^T)$:

$$L_1(U, V|W) = \mathbb{D}_{F_1}(UV^T \,||\, X, W) + \mathbb{D}_G(0 \,||\, U) + \mathbb{D}_H(0 \,||\, V),$$

where $G(u) = \lambda u^2/2$ and $H(v) = \gamma v^2/2$ for $\lambda, \gamma > 0$ corresponds to ℓ_2 regularization. Ignoring terms that do not vary with the factors the loss is

$$L_1(U, V|W) = W \odot \Big(F(UV^T) - X \circ UV^T\Big) + G^*(U) + H^*(V).$$

Similarly, if Y were factored alone, the loss would be

$$L_2(V, Z|\tilde{W}) = \mathbb{D}_{F_2}(VZ^T \,||\, Y, \tilde{W}) + \mathbb{D}_H(0 \,||\, V) + \mathbb{D}_I(0 \,||\, Z).$$

Since V is a shared factor we average the losses:

$$L(U, V, Z|W, \tilde{W}) = \alpha L_1(U, V|W) + (1-\alpha) L_2(V, Z|\tilde{W}), \tag{3}$$

where $\alpha \in [0, 1]$ weights the relative importance of relations.

Each term in the loss, L_1 and L_2, is decomposable and twice-differentiable, which is all that is required for the alternating projections technique described in Section 4.1. Despite the simplicity of Equation 3, it has some interesting implications. The distribution of X_{ij} given $x_i^{(1)}$ and $x_j^{(2)}$, and the distribution of Y_{jk} given $x_j^{(2)}$ and $x_k^{(3)}$, need not agree on the marginal distribution of $x_j^{(2)}$. Extending the notion of row-column exchangeability, each entity $x_j^{(2)}$ corresponds to a record whose features are the possible relations with entities of types $\mathcal{E}_1$ and $\mathcal{E}_3$. Let $\mathcal{F}_{2,1}$ denote the features corresponding to relations involving entities of $\mathcal{E}_1$, and $\mathcal{F}_{2,3}$ the features corresponding to relations involving entities of $\mathcal{E}_3$. If the features are binary, they indicate whether or not an entity participates in a relation with $x_j^{(2)}$. The latent representation of $x_j^{(2)}$ is $V_{j\cdot}$, where $UV_{j\cdot}^T$ and $V_{j\cdot}Z^T$ determines the distribution over $\mathcal{F}_{2,1}$ and $\mathcal{F}_{2,3}$ respectively.

4.1 Parameter Estimation

Equation 3 is convex in any one of its arguments. We extend the alternating projection algorithm for matrix factorization, fixing all but one argument of $\mathcal{L} = L(U, V, Z|W, \tilde{W})$ and updating the free factor using a Newton-Raphson step. Differentiating the loss with respect to each factor:

$$\nabla_U \mathcal{L} = \alpha \left(W \odot \Big(f_1(UV^T) - X\Big)\right) V + \nabla G^*(U), \tag{4}$$

$$\nabla_V \mathcal{L} = \alpha \left(W \odot \Big(f_1(UV^T) - X\Big)\right)^T U + (1-\alpha) \left(\tilde{W} \odot \Big(f_2(VZ^T) - Y\Big)\right) Z + \nabla H^*(V), \tag{5}$$

$$\nabla_Z \mathcal{L} = (1-\alpha) \left(\tilde{W} \odot \Big(f_2(VZ^T) - Y\Big)\right)^T V + \nabla I^*(Z). \tag{6}$$

Setting the gradients equal to zero yields update equations for U, V, and Z. Note that the gradient step does not require the divergence to be decomposable, nor does it require that that the matching losses be differentiable; simply replace gradients with subgradients in the prequel. For ℓ_2 regularization on U, $G(U) = \lambda||U||^2/2$, $\nabla G^*(U) = U/\lambda$. The gradient for a factor is a linear combination of the gradients with respect to the individual matrix reconstructions the factor participates in.

A cursory inspection of Equations 4-6 suggests that an Newton step is infeasible. The Hessian with respect to U would involve nk parameters. However, if L_1 and L_2 are each decomposable functions, then we can show that almost all the second derivatives of $\mathcal{L}$ with respect to a single factor U are zero. Moreover, the Newton update for the factors reduces to row-wise optimization of U, V, and Z. For the subclass of models where Equations 4-6 are differentiable and the loss is decomposable, define

$$q(U_{i\cdot}) = \alpha \left(W_{i\cdot} \odot \left(f_1(U_{i\cdot}V^T) - X_{i\cdot} \right) \right) V + \nabla G^*(U_{i\cdot}),$$

$$q(V_{i\cdot}) = \alpha \left(W_{\cdot i} \odot \left(f_1(UV_{i\cdot}^T) - X_{\cdot i} \right) \right)^T U + (1-\alpha) \left(\tilde{W}_{i\cdot} \odot \left(f_2(V_{i\cdot}Z^T) - Y_{i\cdot} \right) \right) Z + \nabla H^*(V_{i\cdot}),$$

$$q(Z_{i\cdot}) = (1-\alpha) \left(\tilde{W}_{\cdot i} \odot \left(f_2(VZ_{i\cdot}^T) - Y_{\cdot i} \right) \right)^T V + \nabla I^*(Z_{i\cdot}).$$

Since all but one factor is fixed, consider the derivatives of $q(U_{i\cdot})$ with respect to any scalar parameter in U: $\nabla_{U_{js}} q(U_{i\cdot})$. Because U_{js} only appears in $q(U_{i\cdot})$ when $j = i$, the derivative equals zero when $j \neq i$. Therefore the Hessian $\nabla^2_U \mathcal{L}$ is block-diagonal, where each non-zero block corresponds to a row of U. The inverse of a block-diagonal matrix is the inverse of each block, and so the Newton direction for U, $[\nabla_U \mathcal{L}][\nabla^2_U \mathcal{L}]^{-1}$, can be reduced to updating each row $U_{i\cdot}$ using the direction $[q(U_{i\cdot})][q'(U_{i\cdot})]^{-1}$. The above argument applies to V and Z as well, since the loss is a sum of per-matrix losses and the derivative is a linear operator.

Any (local) optima of the loss $\mathcal{L}$ corresponds to roots of the equations $\{q(U_{i\cdot})\}_{i=1}^m$, $\{q(V_{i\cdot})\}_{i=1}^n$, and $\{q(Z_{i\cdot})\}_{i=1}^r$. We derive the Newton step for $U_{i\cdot}$,

$$U_{i\cdot}^{\text{new}} = U_{i\cdot} - \eta \cdot q(U_{i\cdot})[q'(U_{i\cdot})]^{-1}, \tag{7}$$

where we suggest using the Armijo criterion [28] to set η. To concisely describe the Hessian we introduce terms for the contribution of the regularizer,

$$G_i \equiv \text{diag}(\nabla^2 G^*(U_{i\cdot})),$$
$$H_i \equiv \text{diag}(\nabla^2 H^*(V_{i\cdot})),$$
$$I_i \equiv \text{diag}(\nabla^2 I^*(Z_{i\cdot})),$$

and terms for the contribution of the reconstruction error,

$$D_{1,i} \equiv \text{diag}(W_{i\cdot} \odot f_1'(U_{i\cdot}V^T)), \quad D_{2,i} \equiv \text{diag}(W_{\cdot i} \odot f_1'(UV_{i\cdot}^T)),$$
$$D_{3,i} \equiv \text{diag}(\tilde{W}_{i\cdot} \odot f_2'(V_{i\cdot}^T Z)), \quad D_{4,i} \equiv \text{diag}(\tilde{W}_{\cdot i} \odot f_2'(VZ_{i\cdot}^T)).$$

The Hessians with respect to the loss $\mathcal{L}$ are

$$q'(U_{i\cdot}) \equiv \nabla q(U_{i\cdot}) = \alpha V^T D_{1,i} V + G_i$$
$$q'(Z_{i\cdot}) \equiv \nabla q(Z_{i\cdot}) = (1-\alpha) V^T D_{4,i} V + I_i$$
$$q'(V_{i\cdot}) \equiv \nabla q(V_{i\cdot}) = \alpha U^T D_{2,i} U + (1-\alpha) Z^T D_{3,i} Z + H_i$$

Each update of U, V, and Z reduces at least one term in Equation 3. Iteratively cycling through the update leads to a local optima. In practice, we simplify the update by taking one Newton step instead of running to convergence.

4.2 Weights

In addition to weighing the importance of reconstructing different parts of a matrix, W and $\tilde{W}$ serve other purposes. First, the data weights can be used to turn the objective into a per-element loss by scaling each element of X by $(nm)^{-1}$ and each element of Y by $(nr)^{-1}$. This ensures that larger matrices do not dominate the model simply because they are larger. Second, weights can be used to correct for differences in the scale of $L_1(U, V)$ and $L_2(V, Z)$. If the Bregman divergences are regular, we can use the corresponding log-likelihoods as a consistent scale. If the Bregman divergences are not regular, computing

$$D_{F_1}(UV^T \,||\, X, W) / D_{F_2}(VZ^T \,||\, Y, \tilde{W}),$$

averaged over uniform random parameters U, V, and Z, provides an adequate estimate of the relative scale of the two losses. A third use of data weights is missing values. If the value of a relation is unobserved, the corresponding weight is set to zero.

4.3 Generalizing to Arbitrary Schemas

The three-factor model generalizes to any pairwise relational schema, where binary relations are represented as a set of edges: $E = \{(i,j) : \mathcal{E}_i \sim \mathcal{E}_j \ \wedge \ i < j\}$. Let $[U]$ denote the set of latent factors and $[W]$ the weight matrices. The loss of the model is

$$L([U] \,|\, [W]) = \sum_{(i,j) \in E} \alpha^{(ij)} \left(\mathbb{D}_{F^{(ij)}}(U^{(i)}(U^{(j)})^T \,||\, X^{(ij)}, W^{(ij)}) \right) + \sum_{i=1}^{t} \left(\sum_{j:(i,j) \in E} \alpha^{(ij)} \right) \mathbb{D}_{G^{(i)}}(0 \,||\, U^{(i)}),$$

where $F^{(ij)}$ defines the loss for a particular reconstruction, and $G^{(i)}$ defines the loss for a regularizer. The relative weights $\alpha^{(ij)} \geq 0$ measure the importance of each matrix in the reconstruction. Since the loss is a linear function of individual losses, and the differential operator is linear, both gradient and Newton updates can be derived in a manner analogous to Section 4.1, taking care to distinguish when $U^{(i)}$ acts as a column factor as opposed to a row factor.

5. STOCHASTIC APPROXIMATION

In optimizing a collective factorization model, we are in the unusual situation that our primary concern is not the cost of computing the Hessian, but rather the cost of computing the gradient itself: if k is the largest embedding dimension, then the cost of a gradient update for a row $U_r^{(i)}$ is $O(k \sum_{j:\mathcal{E}_i \sim \mathcal{E}_j} n_j)$, while the cost of a Newton update for the same row is $O(k^3 + k^2 \sum_{j:\mathcal{E}_i \sim \mathcal{E}_j} n_j)$. Typically k is much smaller than the number of entities, and so the Newton update costs only a factor of k more. (The above calculations assume dense matrices; for sparsely-observed relations, we can replace n_j by the number of entities of type $\mathcal{E}_j$ which are related to entity $x_r^{(i)}$, but the conclusion remains the same.)

The expensive part of the gradient calculation for $U_r^{(i)}$ is to compute the predicted value for each observed relation that entity $x_r^{(i)}$ participates in, so that we can sum

all of the weighted prediction errors. One approach to reducing this cost is to compute errors only on a subset of observed relations, picked randomly at each iteration. This technique is known as stochastic approximation [7]. The best-known stochastic approximation algorithm is stochastic gradient descent; but, since inverting the Hessian is not a significant part of our computational cost, we will recommend a stochastic Newton's method instead.

Consider the update for $U_{i\cdot}$ in the three factor model. This update can be viewed as a regression where the data are $X_{i\cdot}$ and the features are the columns of V. If we denote a sample of the data as $s \subseteq \{1, \ldots, n\}$, then the sample gradient at iteration τ is

$$\hat{q}_\tau(U_{i\cdot}) = \alpha \left(W_{is} \odot \left(f(U_{i\cdot} V_{s\cdot}^T) - X_{is} \right) \right) V_{s\cdot} + \nabla G^*(U_{i\cdot}),$$

Similarly, given subsets $p \subseteq \{1, \ldots, n\}$ and $q \subseteq \{1, \ldots, r\}$, the sample gradients for the other factors are

$$\hat{q}_\tau(V_{i\cdot}) = \alpha \left(W_{pi} \odot \left(f(U_{p\cdot} V_{i\cdot}^T) - X_{pi} \right) \right)^T U_{p\cdot} + (1-\alpha) \left(\tilde{W}_{iq} \odot \left(f(V_{i\cdot} Z_{q\cdot}^T) - Y_{iq} \right) \right) Z_{q\cdot} + \nabla H^*(V_{i\cdot}),$$

$$\hat{q}_\tau(Z_{i\cdot}) = (1-\alpha) \left(\tilde{W}_{si} \odot \left(f(V_{s\cdot} Z_{i\cdot}^T) - Y_{si} \right) \right)^T V_{s\cdot} + \nabla I^*(Z_{i\cdot}).$$

The stochastic gradient update for U at iteration τ is

$$U_{i\cdot}^{\tau+1} = U_{i\cdot}^{\tau} - \tau^{-1} \hat{q}_\tau(U_{i\cdot}).$$

and similarly for the other factors. Note that we use a fixed, decaying sequence of learning rates instead of a line search: sample estimates of the gradient are not always descent directions. An added advantage of the fixed schedule over line search is that the latter is computationally expensive.

We sample data non-uniformly, without replacement, from the distribution induced by the data weights. That is, for a row $U_{i\cdot}$, the probability of drawing X_{ij} is $W_{ij} / \sum_j W_{ij}$. This sampling distribution provides a compelling relational interpretation: to update the latent factors of $x_r^{(i)}$, we sample only observed relations involving $x_r^{(i)}$. For example, to update a user's latent factors, we sample only movies that the user rated. We use a separate sample for each row of U: this way, errors are independent from row to row, and their effects tend to cancel. In practice, this means that our actual training loss decreases at almost every iteration.

With sampling, the cost of the gradient update no longer grows linearly in the number of entities related to $x_r^{(i)}$, but only in the number of entities sampled. Another advantage of this approach is that when we sample one entity at a time, $|s| = |p| = |q| = 1$, stochastic gradient yields an online algorithm, which need not store all the data in memory.

As mentioned above, we can often improve the rate of convergence by moving from stochastic gradient descent to stochastic Newton-Raphson updates [7, 8]. For the three-factor model the stochastic Hessians are

$$\hat{q}'_\tau(U_{i\cdot}) = \alpha V_{s\cdot}^T \hat{D}_{1,i} V_{s\cdot} + G_i,$$
$$\hat{q}'_\tau(Z_{i\cdot}) = (1-\alpha) V_{s\cdot}^T \hat{D}_{4,i} V_{s\cdot} + I_i,$$
$$\hat{q}'_\tau(V_{i\cdot}) = \alpha U_{p\cdot}^T \hat{D}_{2,i} U_{p\cdot} + (1-\alpha) Z_{q\cdot}^T \hat{D}_{3,i} Z_{q\cdot} + H_i.$$

where

$$\hat{D}_{1,i} \equiv \mathrm{diag}(W_{is} \odot f_1'(U_{i\cdot} V_{s\cdot}^T)), \hat{D}_{2,i} \equiv \mathrm{diag}(W_{pi} \odot f_1'(U_{p\cdot} V_{i\cdot}^T)),$$
$$\hat{D}_{3,i} \equiv \mathrm{diag}(\tilde{W}_{iq} \odot f_2'(V_{i\cdot}^T Z_{q\cdot})), \hat{D}_{4,i} \equiv \mathrm{diag}(\tilde{W}_{si} \odot f_2'(V_{s\cdot} Z_{i\cdot}^T)).$$

To satisfy convergence conditions, which will be discussed in Section 5.1, we use an exponentially weighted moving average of the Hessian:

$$\bar{q}_{\tau+1}(\cdot) = \left(1 - \frac{2}{\tau+1}\right) \bar{q}_\tau(\cdot) + \frac{2}{\tau+1} \hat{q}'_{\tau+1}(\cdot) \qquad (8)$$

When the sample at each step is small compared to the embedding dimension, the Sherman-Morrison-Woodbury lemma (*e.g.*, [7]) can be used for efficiency. The stochastic Newton update is analogous to Equation 7, except that $\eta = 1/\tau$, the gradient is replaced by its sample estimate $\hat{q}$, and the Hessian is replaced by its sample estimate $\bar{q}$.

5.1 Convergence

We consider three properties of stochastic Newton, which together are sufficient conditions for convergence to a local optimum of the empirical loss $\mathcal{L}$ [8]. These conditions are also satisfied by setting the Hessian to the identity, $\bar{q}(\cdot) = I_k$ — *i.e.*, stochastic gradient.

Local Convexity: The loss must be locally convex around its minimum, which must be contained in its domain. In alternating projections the loss is convex for any Bregman divergence; and, for regular divergences, has $\mathbb{R}$ as its domain. The non-regular divergences we consider, such as Hinge loss, also satisfy this property.

Uniformly Bounded Hessian: The eigenvalues of the sample Hessians are bounded in some interval $[-c, c]$ with probability 1. This condition is satisfied by testing whether the condition number of the sample Hessian is below a large fixed value, *i.e.*, the Hessian is invertible. Using the ℓ_2 regularizer always yields an instantaneous Hessian $\hat{q}$ that is full rank. The eigenvalue condition implies that the elements of $\bar{q}$ and its inverse are uniformly bounded.

Convergence of the Hessian: There are two choices of convergence criteria for the Hessian. Either one suffices for proving convergence of stochastic Newton. (i) The sequence of inverses of the sample Hessian converges in probability to the true Hessian: $\lim_{\tau\to\infty} (\bar{q}_\tau)^{-1} = (q')^{-1}$. Alternately, (ii) the perturbation of the sample Hessian from its mean is bounded. Let $\mathcal{P}_{\tau-1}$ consist of the history of the stochastic Newton iterations: the data samples and the parameters for the first $\tau - 1$ iterations. Let $g_\tau = o_s(f_\tau)$ denote an almost uniformly bounded stochastic order of magnitude. The stochastic o-notation is similar to regular o-notation, except that we are allowed to ignore measure-zero events and $E[o_s(f_\tau)] = f_\tau$. The alternate convergence criteria is a concentration of measure statement:

$$E[\bar{q}_\tau | \mathcal{P}_{\tau-1}] = \bar{q}_\tau + o_s(1/\tau).$$

For Equation 8 this condition is easy to verify:

$$E[\bar{q}_\tau | \mathcal{P}_{\tau-1}] = \left(1 - \frac{2}{\tau}\right) \bar{q}_{\tau-1} + \frac{2}{\tau} E[\hat{q}'_\tau | \mathcal{P}_{\tau-1}]$$

since $\mathcal{P}_{\tau-1}$ contains $\bar{q}_{\tau-1}$. Any perturbation from the mean is due to the second term. If $\hat{q}$ is invertible then its elements are uniformly bounded, and so are the elements of $E[\hat{q}_\tau | \mathcal{P}_{\tau-1}]$; since this term has bounded elements and is scaled by $2/\tau$, the perturbation is $o_s(1/\tau)$. One may fold in an instantaneous Hessian that is not invertible, so long as the moving average $\bar{q}$ remains invertible. The above proves the convergence of a factor to the value which minimizes the expected loss, assuming the other factors are fixed. With

respect to the alternating projection, we only have convergence to a local optima of the empirical loss $\mathcal{L}$.

6. RELATED WORK

Collective matrix factorization provides a unified view of matrix factorization for relational data: different methods correspond to different distributional assumptions on individual matrices, different schemas tying factors together, and different optimization procedures. We distinguish our work from prior methods on three points: (i) competing methods often impose a clustering constraint, whereas we cover both cluster and factor analysis (although our experiments focus on factor analysis); (ii) our stochastic Newton method lets us handle large, sparsely observed relations by taking advantage of decomposability of the loss; and (iii) our presentation is more general, covering a wider variety of models, schemas, and losses. In particular, for (iii), our model emphasizes that there is little difference between factoring two matrices versus three or more; and, our optimization procedure can use any twice differentiable decomposable loss, including the important class of Bregman divergences. For example, if we restrict our model to a single relation $\mathcal{E}_1 \sim \mathcal{E}_2$, we can recover all of the single-matrix models mentioned in Sec. 2.2. While our alternating projections approach is conceptually simple, and allows one to take advantage of decomposability, there is a panoply of alternatives for factoring a single matrix. The more popular ones includes majorization [22], which iteratively minimize a sequence of convex upper bounding functions tangent to the objective, including the multiplicative update for NMF [21] and the EM algorithm, which is used both for pLSI [19] and weighted SVD [32]. Direct optimization solves the non-convex problem with respect to (U, V) using gradient or second-order methods, such as the fast variant of max-margin matrix factorization [30].

The next level of generality is a three-entity-type model $\mathcal{E}_1 \sim \mathcal{E}_2 \sim \mathcal{E}_3$. A well-known example of such a schema is pLSI-pHITS [13], which models document-word counts and document-document citations: $\mathcal{E}_1$ = words and $\mathcal{E}_2 = \mathcal{E}_3$ = documents, but it is trivial to allow $\mathcal{E}_2 \neq \mathcal{E}_3$. Given relations $\mathcal{E}_1 \sim \mathcal{E}_2$ and $\mathcal{E}_2 \sim \mathcal{E}_3$, with corresponding integer relationship matrices $X^{(12)}$ and $X^{(23)}$, the likelihood is

$$\mathcal{L} = \alpha X^{(12)} \circ \log\left(UV^T\right) + (1-\alpha) X^{(23)} \circ \log\left(VZ^T\right), \quad (9)$$

where the parameters U, V, and Z correspond to probabilities $u_{ik} = p(x_i^{(1)} \mid h_k)$, $v_{ik} = p(h_k \mid x_i^{(2)})$, and $z_{ik} = p(x_i^{(3)} \mid h_k)$ for clusters $\{h_1, \ldots, h_K\}$. Probability constraints require that each column of U, V^T, and Z must sum to one, which induces a clustering of entities. Since different entities can participate in different numbers of relations (*e.g.,* some words are more common than others) the data matrices $X^{(12)}$ and $X^{(23)}$ are usually normalized; we can encode this normalization using weight matrices. The objective, Equation 9, is the weighted average of two probabilistic LSI [19] models with shared latent factors h_k. Since each pLSI model is a one-matrix example of our general model, the two-matrix version can be placed within our framework.

Matrix co-clustering techniques have a stochastic constraint: if an entity increases its membership in one cluster, it must decrease its membership in others clusters. Examples of matrix and relational co-clustering include pLSI, pLSI-pHITS, the symmetric block models of Long et. al. [23, 24, 25], and Bregman tensor clustering [5] (which can handle higher arity relations). Matrix analogues of factor analysis place no stochastic constraint on the parameters. Collective matrix factorization has been presented using matrix factor analyzers, but the stochastic constraint, that each row of $U^{(r)}$ sums to 1, distributes over the alternating projection to an equality constraint on each update of $U_{i\cdot}^{(r)}$. This additional equality constraint can be folded into the Newton step using a Lagrange multiplier, yielding an unconstrained optimization (*c.f.*, ch. 10 [9]). Comparing the extension of collective matrix factorization to the alternatives above is a topic for future work. It should be noted that our choice of $X = UV^T$ is not the only one for matrix factorization. Long et. al. [23] proposes a symmetric block model $X \approx C_1 A C_2^T$, where $C_1 \in \{0,1\}^{n_1 \times k}$ and $C_2 \in \{0,1\}^{n_2 \times k}$ are cluster indicator matrices, and $A \in \mathbb{R}^{k \times k}$ contains the predicted output for each combination of row and column clusters. Early work on this model uses a spectral relaxation specific to squared loss [23], while later generalizations to regular exponential families [25] use EM. An equivalent formulation in terms of regular Bregman divergences [24] uses iterative majorization [22, 34] as the inner loop of alternating projection. An improvement on Bregman co-clustering accounts for systematic biases, block effects, in the matrix [1].

The three-factor schema $\mathcal{E}_1 \sim \mathcal{E}_2 \sim \mathcal{E}_3$ also includes supervised matrix factorization. In this problem, the goal is to classify entities of type $\mathcal{E}_2$: matrix $X^{(12)}$ contains class labels according to one or more related concepts (one concept per row), while $X^{(23)}$ lists the features of each entity. An example of a supervised matrix factorization algorithm is the support vector decomposition machine [29]: in SVDMs, the features $X^{(23)}$ are factored under squared loss, while the labels $X^{(12)}$ are factored under Hinge loss. A similar model was proposed by Zhu et al. [37], using a once-differentiable variant of the Hinge loss. Another example is supervised LSI [35], which factors both the data and label matrices under squared loss, with an orthogonality constraint on the shared factors. Principal components analysis, which factors a doubly centered matrix under squared loss, has also been extended to the three-factor schema [36].

Another interesting type of schema contains multiple parallel relations between two entity types. An example of this sort of schema is max-margin matrix factorization (MMMF) [30]. In MMMF, the goal is to predict ordinal values, such as a user's rating of movies on a scale of $\{1, \ldots, R\}$. We can reduce this prediction task to a set of binary threshold problems, namely, predicting $r \geq 1, r \geq 2, \ldots, r \geq R$. If we use a Hinge loss for each of these binary predictions and add the losses together, the result is equivalent to a collective matrix factorization where $\mathcal{E}_1$ are users, $\mathcal{E}_2$ are movies, and $\mathcal{E}_1 \sim_u \mathcal{E}_2$ for $u = 1 \ldots R$ are the binary rating prediction tasks. In order to predict different values for the R different relations, we need to allow the latent factors $U^{(1)}$ and $U^{(2)}$ to contain some untied columns, *i.e.*, columns which are not shared among relations. For example, the MMMF authors have suggested adding a bias term for each rating level or for each (user, rating level) pair. To get a bias for each (user, rating level) pair, we can append R untied columns to $U^{(1)}$, and have each of these columns multiply a fixed column of ones in $U^{(2)}$. To get a shared bias for each rating level, we can do the same, but constrain each of the untied columns in $U^{(1)}$ to be a multiple of the all-ones vector.

7. EXPERIMENTS

7.1 Movie Rating Prediction

Our experiments focus on two tasks: (i) predicting whether a user rated a particular movie: `israted`; and (ii) predicting the value of a rating for a particular movie: `rating`. User ratings are sampled from the Netflix Prize data [27]: a rating can be viewed as a relation taking on five ordinal values (1-5 stars), *i.e.*, Rating(user, movie). We augment these ratings with two additional sources of movie information, from the Internet Movie Database [20]: genres for each movie, encoded as a binary relation, *i.e.*, HasGenre(movie, genre); and a list of actors in each movie, encoded as a binary relation, *i.e.*, HasRole(actor, movie). In schema notation $\mathcal{E}_1$ corresponds to users, $\mathcal{E}_2$ corresponds to movies, $\mathcal{E}_3$ corresponds to genres, and $\mathcal{E}_4$ corresponds to actors. Ordinal ratings are denoted $\mathcal{E}_1 \sim_1 \mathcal{E}_2$; for the `israted` task the binarized version of the ratings is denoted $\mathcal{E}_1 \sim_2 \mathcal{E}_2$. Genre membership is denoted $\mathcal{E}_2 \sim \mathcal{E}_3$. The role relation is $\mathcal{E}_2 \sim \mathcal{E}_4$.

There is a significant difference in the amount of data for the two tasks. In the `israted` problem we know whether or not a user rated a movie for all combinations of users and movies, so the ratings matrix has no missing values. In the `rating` problem we observe the relation only when a user rated a movie—unobserved combinations of users and movies have their data weight set to zero.

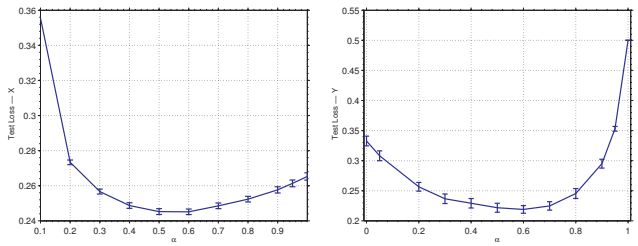
7.1.1 Model and Optimization Parameters

For consistency, we control many of the model and optimization parameters across the experiments. In the `israted` task all the relations are binary, so we use a logistic model: sigmoid link with the matching log-loss. To evaluate test error we use mean absolute error (MAE) for both tasks, which is the average zero-one loss for binary predictions. Since the data for `israted` is highly imbalanced in favour of movies not being rated, we scale the weight of those entries down by the fraction of observed relations where the relation is true. We use ℓ_2 regularization throughout. Unless otherwise stated the regularizers are all $G(U) = 10^5||U||_F^2/2$. In Newton steps, we use an Armijo line search, rejecting updates with step length smaller than $\eta = 2^{-4}$. In Newton steps, we run till the change in training loss falls below 5% of the objective. Using stochastic Newton, we run for a fixed number of iterations.

7.2 Relations Improve Predictions

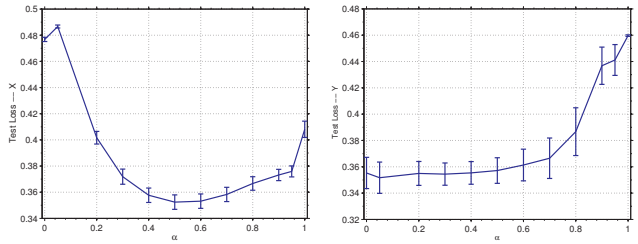
Our claim regarding relational data is that collective factorization yields better predictions than using a single matrix. We consider the `israted` task on two relatively small data sets, to allow for repeated trials. Since this task involves a three factor model there is a single mixing factor, α in Equation 3. We learn a model for several values of α, starting from the same initial random parameters, using full Newton steps. The performance on a test set, entries sampled from the matrices according to the test weights, is measured at each α. Each trial is repeated ten times to provide 1-standard deviation error bars.

Two scenarios are considered. First, where the users and movies were sampled uniformly at random; all genres that occur in more than 1% of the movies are retained. We only use the users' ratings on the sampled movies. Second, where we only sample users that rated at most 40 movies, which greatly reduces the number of ratings for each user and each movie. In the first case, the median number of ratings per user is 60 (the mean, 127); in the second case, the median number of ratings per user is 9 (the mean, 10). In the first case, the median number of ratings per movie is 9 (the mean, 21); in the second case, the median number of ratings per movie is 2 (the mean, 8). In the first case we have $n_1 = 500$ users and $n_2 = 3000$ movies and in the second case we have $n_1 = 750$ users and $n_2 = 1000$ movies. We use a $k = 20$ embedding dimension for both matrices.



(a) Ratings (b) Genres

Figure 1: Test errors (MAE) for predicting whether a movie was rated, and the genre, on the dense rating example.



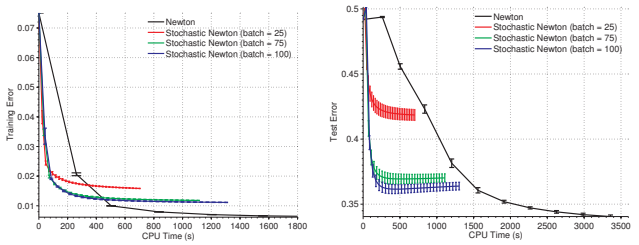
(a) Ratings (b) Genres

Figure 2: Test errors (MAE) for predicting whether a movie was rated, and the genre, on sparse rating example.

The dense rating scenario, Figure 1, shows that collective matrix factorization improves both prediction tasks: whether a user rated a movie, and which genres a movie belongs to. When $\alpha = 1$ the model uses only rating information; when $\alpha = 0$ it uses only genre information.

In the sparse rating scenario, Figure 2, there is far less information in the ratings matrix. Half the movies are rated by only one or two users. Because there is so little information between users, the extra genre information is more valuable. However, since few users rate the same movies there is no significant improvement in genre prediction.

We hypothesized that adding in the roles of popular actors, in addition to genres, would further improve performance. By symmetry the update equation for the actor factor is analogous to the update for the genre factor. Since there are over 100,000 actors in our data, most of which appear in only one or two movies, we selected 500 popular actors (those that appeared in more than ten movies). Under a wide variety of settings for the mixing parameters $\{\alpha^{(12)}, \alpha^{(23)}, \alpha^{(24)}\}$ there was no statistically significant improvement on either the `israted` or `rating` task.



(a) Training Loss (Log-loss) (b) Test Error (MAE)

Figure 3: Behaviour of Newton vs. Stochastic Newton on a three-factor model.

7.3 Stochastic Approximation

Our claim regarding stochastic optimization is that it provides an efficient alternative to Newton updates in the alternating projections algorithm. Since our interest is in the case with a large number of observed relations we use the `israted` task with genres. There are $n_1 = 10000$ users, $n_2 = 2000$ movies, and $n_3 = 22$ of the most common genres in the data set. The mixing coefficient is $\alpha = 0.5$. We set the embedding dimension of both factorizations to $k = 30$.

On this three factor problem we learn a collective matrix factorization using both Newton and stochastic Newton methods with batch sizes of 25, 75, and 100 samples per row. The batch size is larger than the number of genres, and so they are all used. Our primary concern is sampling the larger user-movie matrix. Using Newton steps ten cycles of alternating projection are used; using stochastic Newton steps thirty cycles are used. After each cycle, we measure the training loss (log-loss) and the test error (mean absolute error), which are plotted against the CPU time required to reach the given cycle in Figure 3. This experiment was repeated five times, yielding 2-standard deviation error bars.

Using only a small fraction of the data we achieve results comparable to full Newton after five iterations. At batch size 100, we are sampling 1% of the users and 5% of the movies; yet its performance on test data is the same as a full Newton step given 8x longer to run. Diminishing returns with respect to batch size suggests that using very large batches is unnecessary. Even if the batch size were equal to $\max\{n_1, n_2, n_3\}$ stochastic Newton would not return the same result as full Newton due to the $1/\tau$ damping factor on the sample Hessian.

It should be noted that `rating` is a computationally simpler problem. On a three factor problem with $n_1 = 100000$ users, $n_2 = 5000$ movies, and $n_3 = 21$ genres, with over 1.3M observed ratings, alternating projection with full Newton steps runs to convergence in 32 minutes on a single 1.6 GHz CPU. We use a small embedding dimension, $k = 20$, but one can exploit common tricks for large Hessians. We used the Poisson link for ratings, and the logistic for genres; convergence is typically faster under the identity link.

7.4 Comparison to pLSI-pHITS

In this section we provide an example where the additional flexibility of collective matrix factorization leads to better results; and another where a co-clustering model, pLSI-pHITS, has the advantage.

We sample two instances of `israted`, controlling for the number of ratings each movie has. In the dense data set, the median number of ratings per movie (user) is 11 (76); in the sparse data set, the median number of ratings per movie (user) is 2 (4). In both cases there are 1000 randomly selected users, and 4975 randomly selected movies, all the movies in the dense data set.

Since pLSI-pHITS is a co-clustering method, and our collective matrix factorization model is a link prediction method, we choose a measure that favours neither inherently: ranking. We induce a ranking of movies for each user, measuring the quality of the ranking using mean average precision (MAP) [18]: queries correspond to user's requests for ratings, "relevant" items are the movies of the held-out links, we use only the top 200 movies in each ranking[2], and the averaging is over users. Most movies are unrated by any given user, and so relevance is available only for a fraction of the items: the absolute MAP values will be small, but relative differences are meaningful. We compare four different models for generating rankings of movies for users:

CMF-Identity: Collective matrix factorization using identity prediction links, $f_1(\theta) = f_2(\theta) = \theta$ and squared loss. Full Newton steps are used. The regularization and optimization parameters are the same as those described in Section 7.1.1, except that the smallest step length is $\eta = 2^{-5}$. The ranking of movies for user i is induced by $f(U_{i\cdot}V^T)$.

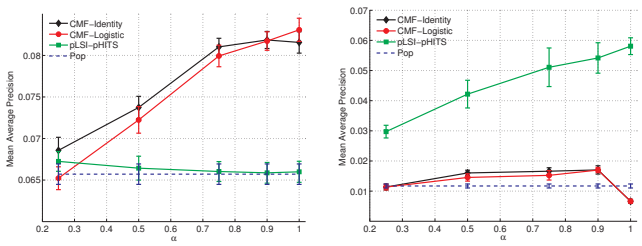
CMF-Logistic: Like CMF-Identity, except that the matching link and loss correspond to a Bernoulli distribution, as in logistic regression: $f_1(\theta) = f_2(\theta) = 1/(1 + \exp^{-\theta})$.

pLSI-pHITS: Makes a multinomial assumption on each matrix, which is somewhat unnatural for the `rating` task—a rating of 5 stars does not mean that a user and movie participated in the rating relation five times. Hence our use of `israted`. We give the regularization advantage to pLSI-pHITS. The amount of regularization $\beta \in [0, 1]$ is chosen at each iteration using tempered EM. The smaller β is, the stronger the parameter smoothing towards the uniform distribution. We are also more careful about setting β than Cohn et. al. [13], using a decay rate of 0.95 and minimum β of 0.7. To have a consistent interpretation of iterations between this method and CMF, we use tempering to choose the amount of regularization, and then fit the parameters from a random starting point with the best choice of β. Movie rankings are generated using $p(movie|user)$.

Pop: A baseline method that ignores the genre information. It generates a single ranking of movies, in order of how frequently they are rated, for all users.

In each case the models, save popularity ranking, have embedding dimension $k = 30$ and run for at most 10 iterations. We compare on a variety of values of α, but we make no claim that mixing information improves the quality of rankings. Since α is a free parameter we want to confirm the relative performance of these methods at several values. In Figure 4, collective matrix factorization significantly outperforms pLSI-pHITS on the dense data set; the converse is true on the sparse data set. Ratings do not benefit from mixing information in any of the approaches, on either data set. While the flexibility of collective matrix factorization has its advantages, especially computational ones, we do not claim unequivocal superiority over relational models based on matrix co-clustering.

[2]The relations between the curves in Figure 4 are the same if the rankings are not truncated.



(a) Dense (b) Sparse

Figure 4: Ranking movies for users on a data set where each movie has many ratings (dense) or only a handful (sparse). The methods are described in Section 7.4. Errors bars are 1-standard deviation.

8. CONTRIBUTIONS

We present a unified view of matrix factorization, building on it to provide collective matrix factorization as a model of pairwise relational data. Experimental evidence suggests that mixing information from multiple relations leads to better predictions in our approach, which complements the same observation made in relational co-clustering [23]. Under the common assumption of a decomposable, twice differentiable loss, we derive a full Newton step in an alternating projection framework. This is practical on relational domains with hundreds of thousands of entities and millions of observations. We present a novel application of stochastic approximation to collective matrix factorization, which allows one handle even larger matrices using a sampled approximation to the gradient and Hessian, with provable convergence and a fast rate of convergence in practice.

Acknowledgements

The authors thank Jon Ostlund for his assistance in merging the Netflix and IMDB data. This research was funded in part by a grant from DARPA's RADAR program. The opinions and conclusions are the authors' alone.

9. REFERENCES

[1] D. Agarwal and S. Merugu. Predictive discrete latent factor models for large scale dyadic data. In *KDD*, pages 26–35, 2007.

[2] D. J. Aldous. Representations for partially exchangeable arrays of random variables. *J. Multi. Anal.*, 11(4):581–598, 1981.

[3] D. J. Aldous. *Exchangeability and related topics*, chapter 1. Springer, 1985.

[4] K. S. Azoury and M. Warmuth. Relative loss bounds for on-line density estimation with the exponential family of distributions. *Mach. Learn.*, 43:211–246, 2001.

[5] A. Banerjee, S. Basu, and S. Merugu. Multi-way clustering on relation graphs. In *SDM*. SIAM, 2007.

[6] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with Bregman divergences. *J. Mach. Learn. Res.*, 6:1705–1749, 2005.

[7] L. Bottou. Online algorithms and stochastic approximations. In *Online Learning and Neural Networks*. Cambridge UP, 1998.

[8] L. Bottou and Y. LeCun. Large scale online learning. In *NIPS*, 2003.

[9] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge UP, 2004.

[10] L. Bregman. The relaxation method of finding the common points of convex sets and its application to the solution of problems in convex programming. *USSR Comp. Math and Math. Phys.*, 7:200–217, 1967.

[11] Y. Censor and S. A. Zenios. *Parallel Optimization: Theory, Algorithms, and Applications*. Oxford UP, 1997.

[12] P. P. Chen. The entity-relationship model: Toward a unified view of data. *ACM Trans. Data. Sys.*, 1(1):9–36, 1976.

[13] D. Cohn and T. Hofmann. The missing link–a probabilistic model of document content and hypertext connectivity. In *NIPS*, 2000.

[14] M. Collins, S. Dasgupta, and R. E. Schapire. A generalization of principal component analysis to the exponential family. In *NIPS*, 2001.

[15] J. Forster and M. K. Warmuth. Relative expected instantaneous loss bounds. In *COLT*, pages 90–99, 2000.

[16] G. H. Golub and C. F. V. Loan. *Matrix Computions*. John Hopkins UP, 3rd edition, 1996.

[17] G. J. Gordon. Generalized^2 linear^2 models. In *NIPS*, 2002.

[18] D. Harman. Overview of the 2nd text retrieval conference (TREC-2). *Inf. Process. Manag.*, 31(3):271–289, 1995.

[19] T. Hofmann. Probabilistic latent semantic indexing. In *SIGIR*, pages 50–57, 1999.

[20] Internet Movie Database Inc. IMDB interfaces. `http://www.imdb.com/interfaces`, Jan. 2007.

[21] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, 2001.

[22] J. D. Leeuw. Block relaxation algorithms in statistics, 1994.

[23] B. Long, Z. M. Zhang, X. Wú;, and P. S. Yu. Spectral clustering for multi-type relational data. In *ICML*, pages 585–592, 2006.

[24] B. Long, Z. M. Zhang, X. Wu, and P. S. Yu. Relational clustering by symmetric convex coding. In *ICML*, pages 569–576, 2007.

[25] B. Long, Z. M. Zhang, and P. S. Yu. A probabilistic framework for relational clustering. In *KDD*, pages 470–479, 2007.

[26] P. McCullagh and J. Nelder. *Generalized Linear Models*. Chapman and Hall: London., 1989.

[27] Netflix. Netflix prize dataset. `http://www.netflixprize.com`, Jan. 2007.

[28] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 1999.

[29] F. Pereira and G. Gordon. The support vector decomposition machine. In *ICML*, pages 689–696, 2006.

[30] J. D. M. Rennie and N. Srebro. Fast maximum margin matrix factorization for collaborative prediction. In *ICML*, pages 713–719, 2005.

[31] A. P. Singh and G. J. Gordon. Relational learning via collective matrix factorization. Technical Report CMU-ML-08-109, Machine Learning Department, Carnegie Mellon University, 2008.

[32] N. Srebro and T. Jaakola. Weighted low-rank approximations. In *ICML*, 2003.

[33] N. Srebro, J. D. Rennie, and T. S. Jaakkola. Maximum-margin matrix factorization. In *NIPS*, 2004.

[34] P. Stoica and Y. Selen. Cyclic minimizers, majorization techniques, and the expectation-maximization algorithm: a refresher. *Sig. Process. Mag., IEEE*, 21(1):112–114, 2004.

[35] K. Yu, S. Yu, and V. Tresp. Multi-label informed latent semantic indexing. In *SIGIR*, pages 258–265, 2005.

[36] S. Yu, K. Yu, V. Tresp, H.-P. Kriegel, and M. Wu. Supervised probabilistic principal component analysis. In *KDD*, pages 464–473, 2006.

[37] S. Zhu, K. Yu, Y. Chi, and Y. Gong. Combining content and link for classification using matrix factorization. In *SIGIR*, pages 487–494, 2007.