PCA on a Data Frame

Madeleine Udell
Computational and Mathematical Engineering
Stanford, CA 94305
udell@stanford.edu

Stephen Boyd
Information Systems Laboratory
Stanford, CA 94305
boyd@stanford.edu

ABSTRACT

Principal components analysis (PCA) is a well-known technique for constructing a low rank model of a data matrix. Here, we propose a new method which performs an analogue of PCA on a *data frame*, *i.e.*, an arbitrary data set consisting of numerical, Boolean, categorical, ordinal, and other data types. We illustrate our method by fitting a low rank model to the 2013 American Community Survey (ACS), a demographic survey covering 1% of the population of the United States.

1. INTRODUCTION

In applications of machine learning and data mining, one frequently encounters large collections of high dimensional data organized into a *data frame*, an arbitrary data set consisting of numerical, Boolean, categorical, ordinal, and other data types. Each row in the data frame represents an example, and each column a feature or attribute. These tables may have columns of different (sometimes, non-numeric) types, and often have many missing entries.

For example, in social science, the data frame might record survey responses: each row of the table records survey results for a particular respondent, and each column corresponds to a distinct survey question. The values in the table might be numerical (income), Boolean (unemployed), ordinal (level of education), or categorical (state of residence). Tests not administered or questions left blank result in missing entries in the data set. Other examples abound: in finance, the table might record known characteristics of companies or asset classes; in medical settings, it might record test results and vital signs; in marketing, it might record known customer characteristics and purchase history.

Exploratory data analysis can be difficult in this setting. To better understand a complex data set, one would like to be able to visualize the features and examples; to find clusters of similar or correlated features or examples; to identify archetypical or representative features or examples; to fill in (impute) missing entries; and to remove (or identify) spurious or noisy data points. This paper introduces a method designed to enable these analyses even on large data sets with heterogeneous values and with many missing entries.

The method works by embedding both the rows (examples) and columns (features) of the data frame into the same k dimensional vector space, returning one short vector corresponding to every example and to every feature.

When a data set consists only of numerical (real-valued) data and has no missing entries, it is easy to find such an embedding using *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. A factorization of this low rank matrix embeds the original high dimensional features into a low dimensional space. Here, we extend PCA to approximate an arbitrary data set by replacing the least-squares error used in PCA with a loss function that is appropriate for the given data type, and adding regularization in order to correctly impute missing entries. Moreover, we provide automatic rules for scaling the loss functions in order to sensibly trade off errors of different types: say, approximating "California" by "Nevada", or "rarely" by "sometimes". Minimizing the sum of the loss function and regularization produces a low dimensional embedding of the data set, as in PCA.

This low rank approximation problem is not convex, and in general cannot be solved globally and efficiently. However, the problem can be heuristically (locally) solved by *alternating minimization* and its variants: methods that alternate between updating the two factors in the low rank approximation. Each step involves solving a convex problem, and can be efficiently solved in parallel. 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.

1.1 Previous work

Heterogeneous data. Many authors have proposed the use of low rank models as a tool for integrating heterogeneous data. The earliest example of this approach is canonical correlation analysis, developed by Hotelling [20] in 1936 to understand the relations between two sets of variates in terms of the eigenvectors of their covariance matrix. In the 1970s, De Leeuw et al. proposed the use of low rank models to fit data measured in nominal, ordinal and cardinal levels [10]. More recently, Goldberg et al. [14] used a low rank model to perform transduction (*i.e.*, multi-label learning) in the presence of missing data by fitting a low rank model to the features and the labels simultaneously. Low rank models have also been used to embed image, text and video data into a common low dimensional space [16], and have recently come into vogue in the natural language processing community as a means to embed words and documents into a low dimensional vector space [30; 31; 34; 44].

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 least squares [10; 49; 45; 8; 9], alternating Newton methods [15; 41], (stochastic or incremental) gradient descent [23; 26; 32; 38; 2; 50; 37], conjugate gradients [39; 42], expectation minimization (EM) (or "soft-impute") methods [42; 29; 18], multiplicative updates [25], and convex relaxations to semidefinite programs [43; 12; 36; 13].

Generally, expectation minimization, which proceeds by iteratively imputing missing entries in the matrix and solving the fully observed problem, has been found to underperform relative to other methods [41]. However, when used in conjunction with computational tricks exploiting a particular problem structure, such as Gram matrix caching, these methods can still work extremely well [18].

Semidefinite programming is usually considered computationally intractable for very large (or even just large) scale problems [39].

(Stochastic) gradient descent methods are often preferred for extremely large scale problems since these methods parallelize naturally in both shared memory and distributed memory architectures. See [37; 50] and references therein for some recent innovative approaches to speeding up stochastic gradient descent for matrix factorization by eliminating locking and reducing interprocess communication. These methods can also easily be applied in our setting.

1.2 Contributions

In this paper, we describe a method to automatically fit a PCA-style low rank model on a data frame. This method extends previous work in several respects.

- We describe new loss functions for categorical and ordinal data that have not previously been considered in a matrix factorization setting.
- We propose a rule of thumb for scaling these losses in order to trade off approximation errors on incomparable data types. For example, we show how to compare the loss due to a misclassification of a categorical value with an error of 0.1 (say) in predicting a real value.
- We propose a new large-scale, parallel algorithm for fitting these low rank models on heterogeneous data sets with real, Boolean, ordinal, and categorical entries, and describe concretely how to choose parameters of the algorithm to achieve fast convergence in practice.
- We describe a new initialization procedure for this algorithm, building off theoretical work in the statistics literature [24; 6] and generalizing it to operate on a data frame, that results in substantially better final solutions compared to random initialization.

Finally, we demonstrate our algorithm by fitting a low rank model to the 2013 American Community Survey (ACS), a demographic survey covering 1% of the population of the United States. This application demonstrates how low rank models for data frames can be used for representation learning, to produce "small data" from "big data": the low rank model produces short, vector summaries of the categorical features (states) in the survey that correspond to an intuitive notion of distance between states in "demography space".

1.3 Organization

We begin by reviewing a few properties of PCA in §2. We then introduce the generalized low rank model framework in §3, and discuss a few loss functions appropriate for each kind of data. We move on to describe in §4 convenient rules for introducing offsets and scalings into the model without performing arithmetic on the data, which preserves data sparsity and prevents us from subtracting, say, the number 4 from the state "California". In §5, we discuss how to use the model to impute missing data, which enables us to cross-validate these low rank models, to understand how well they fit the data and to choose model parameters. We discuss our algorithm and initialization rule in §6, and conclude with an application of the method to fit a model on the 2013 ACS.

2. PCA

Data matrix. In this section, we let $A \in \mathbf{R}^{m\times n}$ be a data matrix consisting of m examples each with n numerical features. Thus $A_{ij} \in \mathbf{R}$ is the value of the jth feature in the ith example, the ith row of A is the vector of n feature values for the ith example, and the jth column of A is the vector of the jth feature across our set of m examples.

It is common to represent other data types in a numerical matrix using certain canonical encoding tricks. For example, Boolean data is often encoded as 1 for true and 0 (or -1) for false, ordinal data is often encoded using consecutive integers to represent the different levels of the variable, and categorical data is often encoded by creating a column for each possible value of the categorical variable, and representing the data using a 1 (true) in the column corresponding to the observed value, and 0 (false) in all other columns. We will see more systematic and principled ways to deal with these data types, and others, in §3. In this section, we assume the entries in the data matrix consist of real numbers.

2.1 PCA

PCA is one of the oldest and most widely used tools in data analysis [33; 19]. We review some of its well-known properties here in order to set notation and as a warm-up to the variants presented later. 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 factorization of Z is of course not unique.) The problem then reduces to choosing the matrices X and Y to minimize $\|A - XY\|_F^2$, where $\|\cdot\|_F$ is the Frobenius norm of a matrix, *i.e.*, the square root of the sum of the squares of the entries. We 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.

The PCA problem can be expressed as

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

with variables X and Y.

2.2 Missing data and matrix completion

Suppose we observe only entries A_{ij} for $(i, j) \in \Omega \subset \{1, \ldots, m\}\times\{1, \ldots, n\}$ from the matrix A, so the other entries are unknown. Then to find a low rank matrix that fits the data

well, we solve the problem

$$\text{minimize} \quad \sum_{(i,j)\in\Omega} (A_{ij} - x_i y_j)^2 + \gamma \sum_{i=1}^{m} \|x_i\|_2^2 + \gamma \sum_{j=1}^{n} \|y_j\|_2^2, \quad (2)$$

with variables X and Y, with $\gamma > 0$. A solution of this problem gives an estimate $\hat{A}_{ij} = x_i y_j$ for the value of those entries $(i,j) \notin \Omega$ that were not observed. In some applications, this data imputation (*i.e.*, guessing entries of a matrix that are not known) is the main point.

There are two very different regimes in which solving the problem (2) may be useful.

Imputing missing entries to borrow strength. Consider a matrix A in which very few entries are missing. The typical approach in data analysis is to simply remove any rows with missing entries from the matrix and exclude them from subsequent analysis. If instead we solve the problem above *without* removing these affected rows, we "borrow strength" from the entries that are *not* missing to improve our global understanding of the data matrix A. In this regime we are imputing the (few) missing entries of A, using the examples that ordinarily we would discard.

Low rank matrix completion. Now consider a matrix A in which most entries are missing, *i.e.*, we only observe relatively few of the mn elements of A, so that by discarding every example with a missing feature or every feature with a missing example, we would discard the *entire* matrix. Then the solution to (2) becomes even more interesting: we are guessing all the entries of a (presumed low rank) matrix, given just a few of them. It is a surprising recent theoretical result that this is possible: if the original matrix A has rank $k \ll n$, and least $|\Omega| = O(nk \log n)$ entries are observed, then the solution to (2) exactly recovers the matrix A with high probability (subject to a few technical conditions on the structure of the matrix A) [24].

3. GENERALIZED LOW RANK MODELS

In this section, we introduce the low rank modeling framework we will use to fit a PCA-style model to a data frame. This framework is a special case of the general framework described in detail in [46]. We then introduce a few loss functions appropriate to each kind of data found in a data frame: real, Boolean, ordinal, and categorical.

Low rank models. Suppose now that our data A is a data frame consisting of m examples (*i.e.*, rows, samples, observations) and n features (*i.e.*, columns, attributes, variables), 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 ($\mathcal{F}_j = \{\text{very much}, \text{a little}, \text{not at all}\}$), 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.

For each data type, we provide a loss function $L_j : \mathbf{R} \times \mathcal{F}_j \to \mathbf{R}$ which describes the approximation error incurred when we represent a feature value $a \in \mathcal{F}_j$ by the number $u \in \mathbf{R}$.

We now formulate PCA on the data frame A as

$$\text{minimize} \quad \sum_{(i,j)\in\Omega} L_j(x_i y_j, A_{ij}) + \gamma \sum_{i=1}^{m} \|x_i\|_2^2 + \gamma \sum_{j=1}^{n} \|y_j\|_2^2, \quad (3)$$

with variables $X \in \mathbf{R}^{n\times k}$ and $Y \in \mathbf{R}^{k\times m}$, and with losses L_j. The regularization parameter γ is chosen to prevent overfitting to the observations and to make the model easier to fit, and can be chosen via cross validation (see §5).

If $L_j(u,a) = (u-a)^2$ for every j, $\gamma = 0$, and $\Omega = \{1,\ldots,m\} \times \{1,\ldots,n\}$ (so we have observed every element in A), then the problem reduces to PCA (1). However, the loss function L_j can now depend on the data in a more interesting way.

Problem (3) is not convex, even when the losses L_j are all convex. However, we can exploit the similarity to PCA to find very good local solutions to Problem 3, using a fast, parallel algorithm. We find an initial guess for the solution using the top k elements of the SVD of a matrix closely related to the data table, which can be computed efficiently using randomized linear algebra. We then use an alternating proximal gradient algorithm to refine our initial guess. We defer a detailed discussion of this algorithm to §6.

3.1 Loss functions for real-valued data

Quadratic loss. The simplest and most commonly used loss function is the quadratic loss,

$$L(u,a) = (u-a)^2.$$

However, the quadratic loss is quite sensitive to large outliers in the data. To fit data sets in which large (non-Gaussian) errors are expected, a loss function that grows less quickly with its argument is preferred.

ℓ_1 loss. Many authors have proposed a robust version of PCA obtained by replacing least-squares loss with ℓ_1 loss,

$$L(u,a) = |u-a|,$$

in order to obtain a *robust* version of PCA which is less sensitive to large outliers [5; 47; 48].

Huber loss. The Huber function is defined as

$$\mathbf{huber}(x) = \begin{cases} (1/2)x^2 & |x| \le 1 \\ |x| - (1/2) & |x| > 1. \end{cases}$$

Using Huber loss,

$$L(u,a) = \mathbf{huber}(u-a),$$

in place of ℓ_1 loss also yields an estimator robust to occasionaly large outliers [21]. The Huber function is less sensitive to small errors $|u-a|$ than the ℓ_1 norm, but becomes linear in the error for large errors; so it is robust both to large outliers and to small Gaussian perturbations in the data. In fact, one can see Huber fitting as finding the most likely explanation of the error as the sum of a Gaussian random variable n (with negative log likelihood $(1/2)n^2$) and a Laplacian random variable s (with negative log likelihood $|s|$):

$$\mathbf{huber}(x) = \inf\{|s| + (1/2)n^2 : x = n + s\}.$$

3.2 Loss functions for Boolean data

To define loss functions for Boolean data, we will suppose the data $A_{ij} \in \{-1, 1\}$; that is, we will represent the value "true" with $+1$ and "false" with -1. The model is insensitive to our choice of encoding, which merely lightens the notation.

Hinge loss. We can use a hinge loss to measure the approximation quality [43; 39]. The hinge loss is defined as

$$L(u,a) = (1-au)_+$$

(see Figure 1). With this loss, fixing X and minimizing over y_j is equivalent to fitting a support vector machine (SVM) to predict the labels A_{ij}.

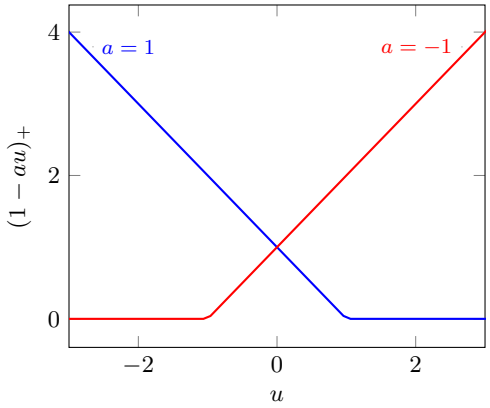

Figure 1: Hinge loss.

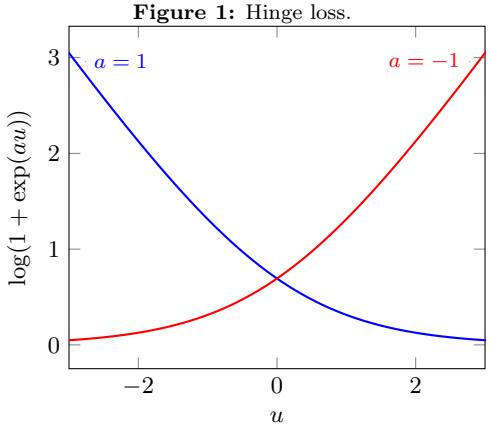

Figure 2: Logistic loss.

Logistic PCA. We can also use a logistic loss to measure the approximation quality [40]. Let

$$L(u, a) = \log(1 + \exp(-au))$$

(see Figure 2). With this loss, fixing X and minimizing over y_j is equivalent to using logistic regression to predict the labels A_{ij}.

3.3 Loss functions for ordinal data

Suppose the data A_{ij} denote the levels of some ordinal variable, encoded as $\{1, 2, \ldots, d\}$. (Here, the encoding chosen *does* matter; we will see an ordinal loss insensitive to the encoding below.) We wish to penalize the entries of the low rank matrix XY which deviate by many levels from the encoded ordinal value.

Ordinal hinge loss. A convex version of this penalty is given by the ordinal hinge loss,

$$L(u, a) = \sum_{a'=1}^{a-1} (1 - u + a')_+ + \sum_{a'=a+1}^{d} (1 + u - a')_+, \quad (4)$$

which generalizes the hinge loss to ordinal data (see Figure 3). The slope of the loss function increases whenever the ordinal value is misclassified by one more level.

This loss function may be useful for encoding Likert-scale data indicating degrees of agreement with a question [28].

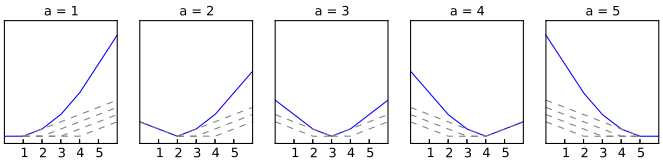

Figure 3: Ordinal hinge loss.

For example, we might have

$$\begin{aligned} \mathcal{F}_j \quad = \quad & \{\text{strongly disagree}, \text{disagree}, \\ & \text{neither agree nor disagree}, \text{agree}, \text{strongly agree}\}. \end{aligned}$$

We can encode these levels as the integers $1, \ldots, 5$ and use the above loss to fit a model to ordinal data. Note that this approach implies that we view every increment of error as equally bad: for example, that approximating "agree" by "strongly disagree" is just as bad as aproximating "neither agree nor disagree" by "agree".

3.4 Multi-dimensional loss functions

In this section, we generalize the procedure to allow the loss functions to depend on *blocks* of the matrix XY, which allows us to represent ordinal and categorical data types more naturally. This generalization corresponds to the standard method for coping with categorical data: namely, expanding a categorical feature with l levels into l Boolean features. Viewing the loss function as a loss function on categorical data, rather than a collection of loss functions for Boolean data, provides a convenient abstraction. For example, it allows us to maintain consistent semantics for imputing missing data regardless of the data type.

We suppose now that our loss functions are $L_j : \mathbf{R}^{1 \times d_j} \times \mathcal{F}_j \to \mathbf{R}$, where d_j is the *embedding dimension* of feature j, and $d = \sum_j d_j$ is the total dimension of the embedded features. The loss $L_j(u, a)$ describes the approximation error incurred when we represent a feature value $a \in \mathcal{F}_j$ by the vector $u \in \mathbf{R}^{d_j}$.

Let $x_i \in \mathbf{R}^{1 \times k}$ be the ith row of X (as before), and let $Y_j \in \mathbf{R}^{k \times d_j}$ be the jth block matrix of Y so the columns of Y_j correspond to the columns of embedded feature j. We now formulate a (multi-dimensional) low rank model on the data frame A,

$$\begin{array}{ll} \text{minimize} & \sum_{(i,j) \in \Omega} L_j(x_i Y_j, A_{ij}) \\ & +\gamma \sum_{i=1}^m \|x_i\|_2^2 + \gamma \sum_{j=1}^n \|Y_j\|_F^2, \end{array} \quad (5)$$

with variables $X \in \mathbf{R}^{n \times k}$ and $Y \in \mathbf{R}^{k \times d}$, and with loss L_j. Note that the first argument of L_j is a *row* vector with d_j entries, and the first argument of r_j is a matrix with d_j columns. When $d_j = 1$ for every j, then we recover the low rank model (3) seen in the previous section.

One-vs-all categorical loss. Suppose that $a \in \mathcal{F}$ is a categorical variable, taking on one of d values or labels. Identify the labels with the integers $\{1, \ldots, d\}$. Define the one-vs-all categorical loss function

$$L(u, a) = (1 - u_a)_+ + \sum_{a' \in \mathcal{F},\ a' \neq a} (1 + u_{a'})_+.$$

Suppose that every data type j is categorical, and we use this loss function in (5). Fixing X and optimizing over Y is equivalent to training one SVM per label to separate that

label from all the others: the jth column of Y gives the weight vector corresponding the jth SVM. Optimizing over X identifies the low-dimensional feature vectors for each example that allow these SVMs to most accurately predict the labels.

The difference between categorical PCA and Boolean PCA is in how missing labels are imputed. To impute a label for entry (i, j) with feature vector x_i according to the procedure described below in 5, we project the representation Y_j onto the line spanned by x_i to form $u = x_i Y_j$. Given u, the imputed label is simply $\text{argmax}_l\, u_l$. This model has the interesting property that if column l' of Y_j lies in the interior of the convex hull of the columns of Y_j, then $u_{l'}$ will lie in the interior of the interval $[\min_l u_l, \max_l u_l]$ [3]. Hence the model will never impute label l' for any example.

Many other categorical loss functions can be used in place of the one-vs-all categorical loss. In fact, any loss function that can be used to train a classifier for categorical variables (also called a multi-class classifier) can be used to fit a categorical PCA model, so long as the loss function depends only on the inner products between the parameters of the model and the features corresponding to each example. The loss function becomes the loss function L used in (5); the optimal parameters of the model give the optimal matrix Y, while the implied features will populate the optimal matrix X. For example, it is possible to use loss functions derived from error-correcting output codes [11]; the Directed Acyclic Graph SVM [35]; the Crammer-Singer multi-class loss [7]; or the multi-category SVM [27].

Multidimensional ordinal loss. We saw in §3.3 one way loss function for ordinal data. Here, we use a larger embedding dimension for ordinal features. The multi-dimensional embedding will be particularly useful when the best mapping of the ordinal variable onto a linear scale is not uniform; *e.g.*, if level 1 of the ordinal variable is much more similar to level 2 than level 2 is to level 3. Using a larger embedding dimension allows us to infer the relations between the levels from the data itself. Here we again identify the labels $a \in \mathcal{F}$ with the integers $\{1, \ldots, d\}$.

One multi-dimensional loss function suitable for embedding ordinal values

$$L(u, a) = \sum_{a'=1}^{d-1} (1 - I_{a>a'} u_{a'})_+. \tag{6}$$

Fixing X and optimizing over Y is equivalent to training an SVM to separate labels $a \leq l$ from $a > l$ for each $l \in \mathcal{F}$. This approach produces a set of hyperplanes (given by the columns of Y) separating each level l from the next. The hyperplanes need not be parallel to each other. Fixing Y and optimizing over X finds the low dimensional features vector for each example that places the example between the appropriate hyperplanes. (See Figure 4 for an illustration of an optimal fit of this loss function, with $k = 2$, to a simple synthetic data set.)

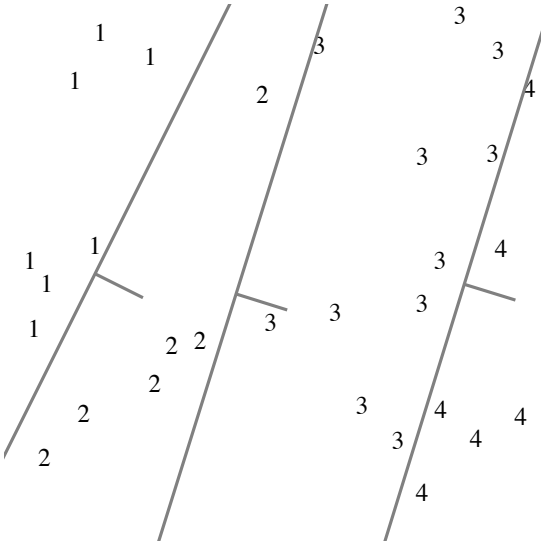

Figure 4: Multi-dimensional ordinal loss

4. OFFSETS AND SCALING

PCA is the maximum likelihood estimator for an observation model in which elements of a low rank matrix are observed with $\mathcal{N}(0, 1)$ noise. If instead each column j of XY is observed with $\mathcal{N}(\mu_j, \sigma_j^2)$ noise, the model is no longer unbiased, and may fit very poorly, particularly if some of the column means μ_j are large.

For this reason 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. (This can be done approximately; there is no need to get the scaling and offset exactly right.) Formally, define $m_j = |\{i : (i, j) \in \Omega\}|$, and let

$$\begin{aligned} \mu_j &= \frac{1}{m_j} \sum_{(i,j)\in\Omega} A_{ij} \\ \sigma_j^2 &= \frac{1}{m_j - 1} \sum_{(i,j)\in\Omega} (A_{ij} - \mu_j)^2 \end{aligned}$$

estimate the mean and variance of each column of the data matrix. PCA is then applied to the matrix whose (i, j) entry is $(A_{ij} - \mu_j)/\sigma_j$.

In our setting, we prefer to rescale the *loss functions* and modify the *model*, rather than perform arithmetic on the data itself, in order to compensate for unequal scaling or offsets. This approach has two advantages. First, when the data is categorical or ordinal, it obviates the need to make the data numeric; we need not decide how to take the average of "California" and "Utah" and subtract it from "Nevada". Second, when the data is numeric, it preserves the sparsity of the underlying data matrix; we need never subtract the mean (which is, in general, nonzero) from the entries of the matrix. (For another interesting approach to preserving sparsity under standardization, see [18].)

Offsets. To allow an offset in the model, we solve

$$\begin{array}{ll} \text{minimize} & \sum_{(i,j)\in\Omega} (A_{ij} - x_i y_j - \mu_j)^2 \\ & +\gamma \sum_{i=1}^m \|x_i\|_2^2 + \gamma \sum_{j=1}^n \|y_j\|_2^2, \end{array} \tag{7}$$

with variables x_i, y_j, and μ_j. Here, μ_j takes the role of the column mean, and in fact will be equal to the column mean in the trivial case $k = 0$.

An offset may be included in (nearly) the standard form problem (3) by augmenting the problem slightly. We increase the rank k of the model by 1, and require the last element of x_i to be 1 for every $i = 1, \ldots, m$. Fitting this model is equivalent to Problem 7

Of course, it is also possible to introduce row offsets in the same way.

Scaling. Given initial loss functions L_j, which we assume are nonnegative, for each feature j let

$$\mu_j = \operatorname*{argmin}_{\mu} \sum_{i:(i,j)\in\Omega} L_j(\mu, A_{ij}) \quad (8)$$

$$\sigma_j^2 = \frac{1}{n_j - 1} \sum_{i:(i,j)\in\Omega} L_j(\mu_j, A_{ij}). \quad (9)$$

It is easy to see that μ_j generalizes the mean of column j, while σ_j^2 generalizes the column variance. When $L_j(u,a) = (u-a)^2$ for every $i = 1,\dots,m$, $j = 1,\dots,n$, μ_j is the mean and σ_j^2 is the sample variance of the jth column of A. When $L_j(u,a) = |u-a|$ for every $i = 1,\dots,m$, $j = 1,\dots,n$, μ_j is the median of the jth column of A, and σ_j^2 is the sum of the absolute values of the deviations of the entries of the jth column from the median value.

To fit a standardized GLRM, we rescale the loss functions and column regularizers by σ_j^2 and solve

$$\begin{array}{ll}\text{minimize} & \sum_{(i,j)\in\Omega} L_j(A_{ij}, x_i y_j + \mu_j)/\sigma_j^2 \\ & +\gamma \sum_{i=1}^m \|x_i\|_2^2 + \sum_{j=1}^n \gamma_j \|y_j\|_2^2,\end{array} \quad (10)$$

where $\gamma_j = \gamma/\sigma_j^2$. Note that this problem can also be (nearly) recast in the standard form of Problem (3), with the one difference that now the regularization parameter for the column regularization depends on the column index. For the offset, we may use the same trick described above to encode the offset in the regularization; and for the scaling, we simply replace the original loss function L_j by L_j/σ_j^2.

5. 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*. Above, we saw that the MAP estimator for the missing entry A_{ij} was equal to $x_i y_j$. This is still true for many of the loss functions above, such as the Huber function or ℓ_1 loss, for which it makes sense for the data to take on any real value.

However, to approximate abstract data types we must consider a more nuanced view. While we can still think of the solution (X, Y) to the generalized low rank model (3) in Boolean PCA as approximating the Boolean matrix A, the solution is not a Boolean matrix. Instead we say that we have *encoded* the original Boolean matrix as a real-valued low rank matrix XY, or that we have *embedded* the original Boolean matrix into the space of real-valued matrices.

To fill in missing entries in the original matrix A, we compute the value $\hat{A}_{ij}$ that minimizes the loss for $x_i y_j$:

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

This implicitly constrains $\hat{A}_{ij}$ to lie in the domain $\mathcal{F}_j$ of L_j. When $L_j : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$, as is the case for the losses in §3 above (including ℓ_2, ℓ_1, and Huber loss), then $\hat{A}_{ij} = x_i y_j$. But when the data is of an abstract type, the minimum $\operatorname{argmin}_a L_j(u,a)$ will not in general be equal to u.

For example, when the data is Boolean, $L_j : \{0,1\} \times \mathbf{R} \to \mathbf{R}$, we compute the Boolean matrix $\hat{A}$ implied by our low rank model by solving

$$\hat{A}_{ij} = \operatorname*{argmin}_{a\in\{0,1\}} (a(XY)_{ij} - 1)_+$$

for MMMF, or

$$\hat{A}_{ij} = \operatorname*{argmin}_{a\in\{0,1\}} \log(1 + \exp(-a(XY)_{ij}))$$

for logistic PCA. These problems both have the simple solution

$$\hat{A}_{ij} = \mathbf{sign}(x_i y_j).$$

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

$$\mathcal{R}_a = \{x \in \mathbf{R} : L_j(u,x) = \min_a L_j(u,a)\}$$

corresponding to different values $a \in \mathcal{F}_j$. When L_j 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 $\mathcal{F}_j$.

Using this imputation method allows us to cross-validate a low rank model in order to choose a value for the rank k and regularization parameter γ. Partition Ω into $\tilde{\Omega}$ and $\tilde{\Omega}^{\mathrm{C}}$ at random, fit a low rank model on the observations in $\tilde{\Omega}$, impute the values of the entries in $\tilde{\Omega}^{\mathrm{C}}$ according to the model, and compare the imputed values with the observed, held-out data in $\tilde{\Omega}^{\mathrm{C}}$.

6. ALGORITHM

In this section, we discuss the algorithm we use to solve Problem 3. Through this section, we use the notation of Problem 3, with the understanding that offsets and scalings can be included as discussed in §4 with minimal modifications to the structure of the algorithm.

Our algorithm is based on the idea of alternating minimization: alternately minimizing the objective function over the factors X and Y. 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 replacing the minimization over x and y above by any update rule that moves towards the minimum. Here, we use a single (sub)gradient step, presented as Algorithm 1, to perform a step towards the minimum. Empirically, we find that this approach 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.

Algorithm 1

given X^0, Y^0
for $t = 1, 2, \dots$ **do**
  for $i = 1, \dots, m$ **do**
    $g_i^t = \sum_{j:(i,j)\in\Omega} \nabla L_j(x_i y_j, A_{ij}) y_j + 2\gamma x_i^t$
    $x_i^t = x_i^{t-1} - \alpha_i^t g_i^t$
  end for
  for $j = 1, \dots, n$ **do**
    $g_j^t = \sum_{i:(i,j)\in\Omega} \nabla L_j(x_i^t y_j^t, A_{ij}) y_j^t + 2\gamma y_j^t$
    $y_j^t = x_i^{k-1} - \alpha_j^t g_j^t$
  end for
end for

Here, $\nabla L(u,a)$ selects an element arbitrarily from the subgradient of the function L at (u,a), and α_i^t and α_j^t are step

sizes, which we discuss further below.

The loops over $i = 1, \ldots, m$ and $j = 1, \ldots, n$ are trivially parallelizable: each row update is independent of all other rows, and each column update is independent of all other columns. Hence the updates for different rows and for different columns can be performed simultaneously by different processors, so long as each processor waits until all row updates have finished before updating a column, and until all column updates have finished before updating a row. Some researchers [32; 50] have proposed relaxing this constraint in order to reduce the time spent waiting for the last, lagging processor to finish all of its updates; this is particularly important for large problems, or for problems in which it is difficult to partition the rows and columns onto processors so that the loads on different processors are balanced.

The last ingredient for a successful algorithm is to specify a step size rule. The step size rule $\alpha_i^t = \alpha_j^t = 1/t$ for every $i = 1, \ldots, m$, $j = 1, \ldots, n$ guarantees convergence to the globally optimal X if Y is fixed [4], while using a fixed, but sufficiently small, step size α guarantees convergence to a small $O(\alpha)$ neighborhood around the optimum [1]. The technical condition required is that $\alpha < 1/L$, where L is the Lipshitz constant of the gradient of the objective function. In numerical experiments, we find that using a slightly more nuanced rule allowing *different* step sizes for different rows and columns can allow fast progress towards convergence while ensuring that the value of the objective never increases. The safeguards on step sizes we propose are quite important in practice: without these checks, we observe *divergence* when the initial step sizes are chosen too large.

Motivated by the convergence proof in [1], for each *row*, we seek a step size on the order of $1/\|g_i\|_2$, where g_i is the gradient of the objective function with respect to x_i. We start by choosing an initial step size scale α_i for each row of the same order as the average gradient of the loss functions for that row. In the numerical experiments reported here, we choose $\alpha_i = 1$ for $i = 1, \ldots, m$. Since g_i grows with the number of observations $n_i = |\{j : (i,j) \in \Omega\}|$ in row i, we achieve the desired scaling by setting $\alpha_i^0 = \alpha_i/n_i$. We take a gradient step on each row x_i using the step size α_i. Our procedure for choosing α_j^0 is the same, except that we set the scale $\alpha_j = 1/\sigma_j^2$ to be proportional to the scale of the loss function for that column.

We then check whether the objective value for the row,

$$\sum_{j:(i,j)\in\Omega} L_j(x_i y_j, A_{ij}) + \gamma \|x_i\|_2^2,$$

has increased or decreased. If it has increased, then we trust our first order approximation to the objective function less far, and reduce the step size; if it has decreased, we gain confidence, and increase the step size. In the numerical experiments reported below, we decrease the step size by 30% when the objective increases, and increase the step size by 5% when the objective decreases. This check stabilizes the algorithm and prevents divergence even when the initial scale has been chosen poorly.

We then do the same with respect to each column y_j: we take a gradient step, check if the objective value for the column has increased or decreased, and adjust the step size.

The time per iteration is thus $O(k(m+n+|\Omega|))$: computing the gradient of the ith loss function with respect to x_i takes time $O(kn_i)$; computing the prox operator of the square loss takes time $O(k)$; summing these over all the rows $i = 1, \ldots, m$ gives time $O(k(m+|\Omega|))$; and adding the same costs for the column updates gives time $O(k(m+n+|\Omega|))$. The checks on the objective value take time $O(k)$ per observation (to compute the inner product $x_i y_j$ and value of the loss function for each observation) and time $O(1)$ per row and column to compute the value of the regularizer. Hence the total time per iteration is $O(k(m+n+|\Omega|))$.

By partitioning the job of updating different rows and different columns onto different processors, we can achieve an iteration time of $O(k(m+n+|\Omega|)/p)$ using p processors.

6.1 Initalization

Alternating minimization need not converge to the same solution (or the same objective value) when initialized at different starting points. In practice, we observe that alternating minimization can converge to models with optimal values that differ significantly. It is critical to choose a good initialization for the algorithm in order to converge to a model which fits well.

If one desires a low rank model for the data, initializing with the SVD of the data (even if the data is incomplete, and the loss function is not quadratic) can sometimes help alternating minimization find a good local optimum. In separate lines of research, [24] and [22] show that alternating minimization converges to the global optimum when initialized with the SVD of the data matrix (or of a slightly modified, or *trimmed*, matrix), so long as the number of entries in the matrix is sufficiently large, and the entries have been chosen uniformly at random. Indeed, the method even converges quickly: [22] show that this approach achieves a quadratic convergence rate.

But we will need a matrix on which to perform the SVD. What matrix corresponds to our dataframe? Here, we give a simple proposal for how to construct such a matrix, motivated by [24; 22; 6]. Our key insight is that the SVD is the solution to our problem when the entries in the table have mean zero and variance one (and all the loss functions are quadratic). Our initialization will construct a matrix from our data frame with mean zero and variance one, take its SVD, and invert the construction to produce the correct initialization.

Our first step is to expand the categorical columns taking on d values into d Boolean columns, and to reinterpret ordinal and Boolean columns as numbers. The scaling we propose below is insensitive to the values of the numbers in the expansion of the Booleans; for example, using (false, true)$= (0,1)$ or (false, true)$= (-1,1)$ produces the same initialization. The scaling *is* sensitive to the differences between ordinal values; while encoding (never, sometimes, always) as $(1,2,3)$ or as $(-5,0,5)$ will make no difference, encoding these ordinals as $(0,1,10)$ *will* result in a different initialization.

We here make use of the assumption, implicit throughout this paper, that the *rows* of the data frame are independent and identically distributed, so we assume they each have equal means and variances. Our mission is to standardize the *columns*. The observed entries in column j have mean

μ_j and variance σ_j^2,

$$\mu_j = \operatorname*{argmin}_{\mu} \sum_{i:(i,j)\in\Omega} L_j(\mu, A_{ij})$$
$$\sigma_j^2 = \frac{1}{n_j - 1} \sum_{i:(i,j)\in\Omega} L_j(\mu_j, A_{ij}),$$

so the observed entries in $(A_j - \mu_j)/\sigma_j$ have mean 0 and variance 1.

Now we address the missing entries. Each missing entry can be safely replaced with 0 in the scaled version of the data without changing the column mean. But the column *variance* will decrease to m_j/m. If instead we define

$$\tilde{A}_{ij} = \begin{cases} \frac{m}{\sigma_j m_j}(A_{ij} - \mu_j) & (i,j) \in \Omega \\ 0 & \text{otherwise,} \end{cases}$$

then the column will have mean 0 and variance 1.

Now we take the SVD $U\Sigma V^T$ of $\tilde{A}$, and let $\tilde{U} \in \mathbf{R}^{m\times k}$, $\tilde{\Sigma} \in \mathbf{R}^{k\times k}$, and $\tilde{V} \in \mathbf{R}^{n\times k}$ denote these matrices truncated to the top k singular vectors and values. We initialize $X = \tilde{U}\tilde{\Sigma}^{1/2}$, and $Y = \tilde{\Sigma}^{1/2}\tilde{V}^T\,\mathbf{diag}(\sigma)$. The offset row in the model is initialized with the means, *i.e.*, the kth column of X is filled with 1's, and the kth row of Y is filled with the means, so $Y_{kj} = \mu_j$.

Note that the scaling we choose here minimizes the scaled regularization penalty $\gamma \sum_{i=1}^m \|x_i\|_2^2 + \sum_{j=1}^n \gamma_j \|y_j\|_2^2$ (where γ_j is defined as in §4) over all matrices X and Y with the same product $\tilde{U}\tilde{\Sigma}\tilde{V}^T\,\mathbf{diag}(\sigma)$.

Finally, we mention that we need not compute the full SVD of $\tilde{A}$, but instead can simply compute the top k singular triples. Using, for example, the randomized top k SVD algorithm proposed in [17], we are able to compute the top k singular triples of $\tilde{A}$ in time linear in $|\Omega|$, m, and n (and quadratic in k).

Figure 5 compares the convergence of this algorithm on a low rank model for census data described in detail below in §7. We initialize the algorithm at six different points: from five different random normal initializations (entries of X^0 and Y^0 drawn iid from $\mathcal{N}(0,1)$), and from the SVD of $\tilde{A}$. The SVD initialization produces a better initial value for the objective function, and also allows the algorithm to converge to a substantially lower final objective value than can be found from *any* of the five random starting points. This behaviour indicates that the "good" local minimum discovered by the SVD initialization is located in a basin of attraction that has low probability with respect to the measure induced by random normal initialization.

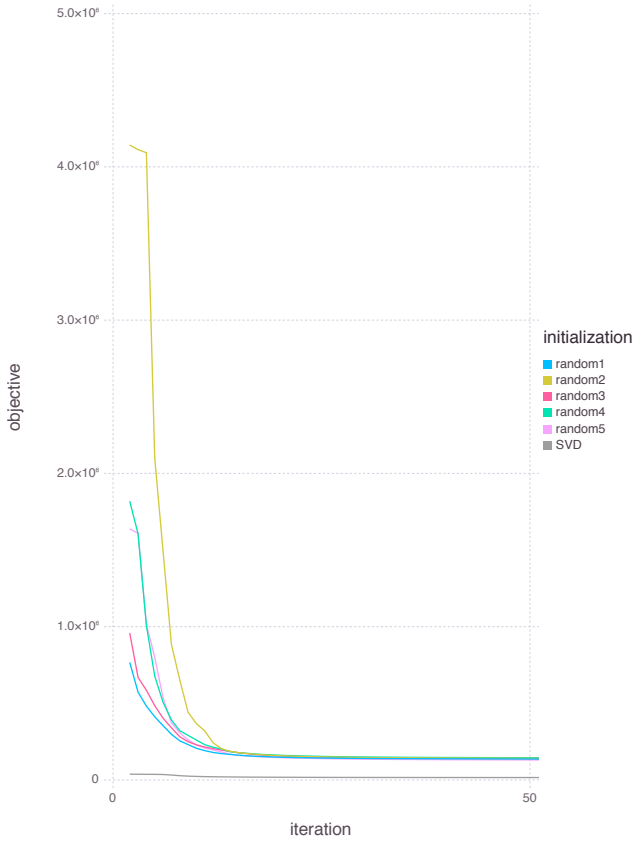

Figure 5: Convergence from random and SVD initializations

7. PCA ON THE CENSUS

We fit a low rank model to the 2013 American Community Survey (ACS) to illustrate the method. The ACS is a survey administered to 1% of the population of the United States each year to gather their responses to a variety of demographic and economic questions. Our data sample consists of $m = 3132796$ responses gathered from residents of the US, excluding Puerto Rico, in the year 2013, on the 23 questions listed in Table 1.

We fit a rank 10 model to this data using the huber loss for real valued data, the hinge loss for Boolean data, the ordinal hinge loss for ordinal data, the one-vs-all categorical loss for categorical data, and regularization parameter $\gamma = .1$. We

Variable	Description	Type
HHTYPE	household type	categorical
STATEICP	state	categorical
OWNERSHP	own home	Boolean
COMMUSE	commercial use	Boolean
ACREHOUS	house on $\geq$ 10 acres	Boolean
HHINCOME	household income	real
COSTELEC	monthly electricity bill	real
COSTWATR	monthly water bill	real
COSTGAS	monthly gas bill	real
FOODSTMP	on food stamps	Boolean
HCOVANY	have health insurance	Boolean
SCHOOL	currently in school	Boolean
EDUC	highest level of education	ordinal
GRADEATT	highest grade level attained	ordinal
EMPSTAT	employment status	categorical
LABFORCE	in labor force	Boolean
CLASSWKR	class of worker	Boolean
WKSWORK2	weeks worked per year	ordinal
UHRSWORK	usual hours worked per week	real
LOOKING	looking for work	Boolean
MIGRATE1	migration status	categorical

Table 1: ACS variables

Feature	Most similar features
Alaska	Montana, North Dakota
California	Illinois, cost of water
Colorado	Oregon, Idaho
Ohio	Indiana, Michigan
Pennsylvania	Massachusetts, New Jersey
Virginia	Maryland, Connecticut
Hours worked	weeks worked, education

Table 2: Most similar features in demography space

allow an offset in the model and scale the loss functions and regularization as described in §4.

In Table 2, we select a few features j from the model, along with their associated vectors y_j, and find the two features most similar to them by finding the two features j' which minimize $\cos(y_j, y_{j'})$. The model automatically groups states which intuitively share demographic features: for example, three wealthy states adjoining (but excluding) a major metropolitan area — Virginia, Maryland, and Connecticut — are grouped together. The low rank structure also identifies the results (high water prices) of the prolonged drought afflicting California, and corroborates the intuition that work leads only to more work: hours worked per week, weeks worked per year, and education level are highly correlated.

Acknowledgements

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.

8. REFERENCES

[1] D. P. Bertsekas. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning*, 2010:1–38, 2011.

[2] 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.

[3] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

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

[5] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):11, 2011.

[6] S. Chatterjee. Matrix estimation by universal singular value thresholding. *The Annals of Statistics*, 43(1):177–214, 2014.

[7] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *The Journal of Machine Learning Research*, 2:265–292, 2002.

[8] J. De Leeuw. The Gifi system of nonlinear multivariate analysis. *Data analysis and informatics*, 3:415–424, 1984.

[9] J. De Leeuw and P. Mair. Gifi methods for optimal scaling in R: The package homals. *Journal of Statistical Software*, pages 1–30, 2009.

[10] J. De Leeuw, F. Young, and Y. Takane. Additive structure in qualitative data: An alternating least squares method with optimal scaling features. *Psychometrika*, 41(4):471–503, 1976.

[11] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *CoRR*, cs.AI/9501101, 1995.

[12] M. Fazel, H. Hindi, and S. Boyd. Rank minimization and applications in system theory. In *Proceedings of the 2004 American Control Conference (ACC)*, volume 4, pages 3273–3278. IEEE, 2004.

[13] W. Fithian and R. Mazumder. Scalable convex methods for flexible low-rank matrix modeling. *arXiv preprint arXiv:1308.4211*, 2013.

[14] A. Goldberg, B. Recht, J. Xu, R. Nowak, and X. Zhu. Transduction with matrix completion: Three birds with one stone. In *Advances in neural information processing systems*, pages 757–765, 2010.

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

[16] A. Gress and I. Davidson. A flexible framework for projecting heterogeneous data. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 1169–1178, New York, NY, USA, 2014. ACM.

[17] N. Halko, P.-G. Martinsson, and J. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

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

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

[20] H. Hotelling. Relations between two sets of variates. *Biometrika*, 28(3-4):321–377, 1936.

[21] P. Huber. *Robust statistics*. Wiley, New York, 1981.

[22] P. Jain, P. Netrapalli, and S. Sanghavi. Low-rank matrix completion using alternating minimization. In *Proceedings of the 45th annual ACM Symposium on the Theory of Computing*, pages 665–674. ACM, 2013.

[23] R. Keshavan and S. Oh. A gradient descent algorithm on the Grassman manifold for matrix completion. *arXiv preprint arXiv:0910.5260*, 2009.

[24] R. H. Keshavan, A. Montanari, and S. Oh. Matrix completion from a few entries. *IEEE Transactions on Information Theory*, 56(6):2980–2998, 2010.

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

[26] 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.

[27] Y. Lee, Y. Lin, and G. Wahba. Multicategory support vector machines: Theory and application to the classification of microarray data and satellite radiance data. *Journal of the American Statistical Association*, 99(465):67–81, 2004.

[28] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 1932.

[29] R. Mazumder, T. Hastie, and R. Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *The Journal of Machine Learning Research*, 11:2287–2322, 2010.

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[31] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.

[32] 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.

[33] 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.

[34] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12, 2014.

[35] J. C. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin DAGs for multiclass classification. In *Advances in Neural Information Processing Systems*, pages 547–553, 1999.

[36] B. Recht, M. Fazel, and P. A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Review*, 52(3):471–501, Aug. 2010.

[37] B. Recht and C. Ré. Parallel stochastic gradient algorithms for large-scale matrix completion. *Mathematical Programming Computation*, 5(2):201–226, 2013.

[38] B. Recht, C. Ré, S. Wright, and F. Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 693–701, 2011.

[39] 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.

[40] A. I. Schein, L. K. Saul, and L. H. Ungar. A generalized linear model for principal component analysis of binary data. In *Proceedings of the Ninth International Workshop on Artificial Intelligence and Statistics*, volume 38, page 46, 2003.

[41] 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.

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

[43] 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.

[44] V. Srikumar and C. Manning. Learning distributed representations for structured output prediction. In *Advances in Neural Information Processing Systems*, pages 3266–3274, 2014.

[45] Y. Takane, F. Young, and J. De Leeuw. Nonmetric individual differences multidimensional scaling: an alternating least squares method with optimal scaling features. *Psychometrika*, 42(1):7–67, 1977.

[46] M. Udell, C. Horn, R. Zadeh, and S. Boyd. Generalized low rank models. *arXiv preprint arXiv:1410.0342*, 2014.

[47] J. Wright, A. Ganesh, S. Rao, Y. Peng, and Y. Ma. Robust principal component analysis: Exact recovery of corrupted low-rank matrices by convex optimization. In *Advances in Neural Information Processing Systems*, volume 3, 2009.

[48] H. Xu, C. Caramanis, and S. Sanghavi. Robust PCA via outlier pursuit. *IEEE Transactions on Information Theory*, 58(5):3047–3064, 2012.

[49] F. Young, J. De Leeuw, and Y. Takane. Regression with qualitative and quantitative variables: An alternating least squares method with optimal scaling features. *Psychometrika*, 41(4):505–529, 1976.

[50] H. Yun, H.-F. Yu, C.-J. Hsieh, S. V. N. Vishwanathan, and I. Dhillon. NOMAD: Non-locking, stOchastic Multi-machine algorithm for Asynchronous and Decentralized matrix completion. *arXiv preprint arXiv:1312.0193*, 2013.