Foundations and Trends® in Machine Learning
Vol. 9, No. 1 (2016) 1–118
© 2016 M. Udell, C. Horn, R. Zadeh and S. Boyd
DOI: 10.1561/2200000055

Generalized Low Rank Models

Madeleine Udell
Operations Research and Information Engineering
Cornell University
`udell@cornell.edu`

Corinne Horn
Electrical Engineering
Stanford University
`cehorn@stanford.edu`

Reza Zadeh
Computational and Mathematical Engineering
Stanford University
`rezab@stanford.edu`

Stephen Boyd
Electrical Engineering
Stanford University
`boyd@stanford.edu`

Contents

Abstract

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

M. Udell, C. Horn, R. Zadeh and S. Boyd. *Generalized Low Rank Models*.
Foundations and Trends® in Machine Learning, vol. 9, no. 1, pp. 1–118, 2016.
DOI: 10.1561/2200000055.

1 Introduction

In applications of machine learning and data mining, one frequently encounters large collections of high dimensional data organized into a table. Each row in the table 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 medicine, the table might record patient attributes or lab tests: each row of the table lists test or survey results for a particular patient, and each column corresponds to a distinct test or survey question. The values in the table might be numerical (3.14), Boolean (yes, no), ordinal (never, sometimes, always), or categorical (A, B, O). 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 social science settings, it might record survey responses; 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 archetypical examples, to cluster examples, to find correlated features, to fill in (impute) missing entries, and to remove (or simply identify) spurious, anomalous, or noisy data points. This paper introduces a templated method to enable these analyses even on large data sets with heterogeneous values and with many missing entries. Our approach will be to embed both the rows (examples) and columns (features) of the table into the same low dimensional vector space. These low dimensional vectors can then be plotted, clustered, and used to impute missing entries or identify anomalous ones.

If the data set consists only of numerical (real-valued) data, then a simple and well-known technique to find this embedding is Principal Components Analysis (PCA). PCA finds a low rank matrix that minimizes the approximation error, in the least-squares sense, to the original data set. A factorization of this low rank matrix embeds the original high dimensional features into a low dimensional space. Extensions of PCA can handle missing data values, and can be used to impute missing entries.

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. Another extension beyond PCA is to add regularization on the low dimensional factors to impose or encourage some structure, such as sparsity or nonnegativity, in the low dimensional factors. In this paper we use the term *generalized low rank model* (GLRM) to refer to the problem of approximating a data set as a product of two low dimensional factors by minimizing an objective function. The objective will consist of a loss function on the approximation error together with regularization of the low dimensional factors. With these extensions of PCA, the resulting low rank representation of the data set still produces a low dimensional embedding of the data set, as in PCA.

Many of the low rank modeling problems we must solve will be familiar. We recover an optimization formulation of nonnegative matrix factorization, matrix completion, sparse and robust PCA, k-means, k-SVD, and maximum margin matrix factorization, to name just a few.The scope of the problems we consider, however, is more broad, encompassing many different combinations of loss function and regularizer. A few of the choices we consider are shown in Tables A.1 and

A.2 of Appendix A for reference; all of these are discussed in detail later in the paper.

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

1.1 Previous work

Unified views of matrix factorization. We are certainly not the first to note that matrix factorization algorithms may be viewed in a unified framework, parametrized by a small number of modeling decisions. The first instance we find in the literature of this unified view appeared in a paper by Collins, Dasgupta, and Schapire, [29], extending PCA to use loss functions derived from any probabilistic model in the exponential family. Gordon's Generalized^2 Linear^2 models [53] extended the framework to loss functions derived from the generalized Bregman divergence of any convex function, which includes models such as Independent Components Analysis (ICA). Srebro's 2004 PhD thesis [133] extended the framework to other loss functions, including hinge loss and KL-divergence loss, and to other regularizers, including the nuclear norm and max-norm. Similarly, Chapter 8 in Tropp's 2004 PhD thesis [144] explored a number of new regularizers, presenting a range of clustering problems as matrix factorization problems with constraints, and anticipated the k-SVD algorithm [4]. Singh and Gordon [129] offered a complete view of the state of the literature on matrix factorization in Table 1 of their 2008 paper, and noted that by changing the loss function and regularizer, one may recover algorithms including PCA, weighted PCA, k-means, k-medians, ℓ_1 SVD, probabilistic latent semantic indexing (pLSI), nonnegative matrix factorization with ℓ_2 or KL-divergence loss, exponential family PCA, and MMMF. Witten et al. introduced the statistics community to sparsity-inducing matrix factorization in a 2009 paper on penalized matrix decomposition, with applications to sparse PCA and canonical correlation analysis [155]. Recently, Markovsky's monograph on low rank approximation [97] reviewed some of this literature, with a focus on applications in system, control, and signal processing. The GLRMs discussed in this paper include all of these models, and many more.

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 [63] in 1936 to understand the relations between two sets of variates in terms of the eigenvectors of their covariance matrix. This approach was extended by Witten et al. [155] to encourage structured (*e.g.*, sparse) factors. In the 1970s, De Leeuw et al. proposed the use of low rank models to fit data measured in nominal, ordinal and cardinal levels [37]. More recently, Goldberg et al. [52] 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 [54], 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 [99, 100, 112, 136].

Algorithms. In general, it can be computationally hard to find the global optimum of a generalized low rank model. For example, it is NP-hard to compute an exact solution to k-means [43], nonnegative matrix factorization [149], and weighted PCA and matrix completion [50], all of which are special cases of low rank models.

However, there are many (efficient) ways to go about *fitting* a low rank model, by which we mean finding a good model with a small objective value. The resulting model may or may not be the global solution of the low rank optimization problem. We distinguish a model fit in this way from the *solution* to an optimization problem, which always refers to the global solution.

The matrix factorization literature presents a wide variety of methods to fit low rank models in a variety of special cases. For example, there are variants on alternating minimization (with alternating least squares as a special case) [37, 158, 141, 35, 36], alternating Newton methods [53, 129], (stochastic or incremental) gradient descent [75, 88, 104, 119, 10, 159, 118], conjugate gradients [120, 134], expectation minimization (EM) (or "soft-impute") methods [142, 134, 98, 60], multiplicative updates [85], and convex relaxations to semidefinite programs [135, 46, 117, 48].

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 [129]. 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 [60].

Semidefinite programming becomes computationally intractable for very large (or even just large) scale problems [120]. However, a theoretical analysis of optimality conditions for rank-constrained semidefinite programs [20] has led to a few algorithms for semidefinite programming based on matrix factorization [19, 1, 70] which guarantee global optimality and converge quickly if the global solution to the problem is exactly low rank. Fast approximation algorithms for rank-constrained semidefinite programs have also been developed [127].

Recently, there has been a resurgence of interest in methods based on alternating minimization, as numerous authors have shown that alternating minimization (suitably initialized, and under a few technical assumptions) provably converges to the global minimum for a range of problems including matrix completion [72, 66, 58], robust PCA [103], and dictionary learning [2].

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 [118, 159] 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 stochastic non-locking methods often run faster than their deterministic counterparts; and for the matrix completion problem in particular, these methods can be shown to provably converge to the global minimum under the same conditions required for alternating minimization [38].

Contributions. The present paper differs from previous work in a number of ways. We are consistently concerned with the *meaning* of applying these different loss functions and regularizers to approximate a data set. The generality of our view allows us to introduce a number of loss functions and regularizers that have not previously been considered. Moreover, our perspective enables us to extend these ideas to arbitrary data sets, rather than just matrices of real numbers.

A number of new considerations emerge when considering the problem so broadly. First, we must face the problem of comparing approximation errors across data of different types. For example, we must choose a scaling to trade off the loss due to a misclassification of a categorical value with an error of 0.1 (say) in predicting a real value.

Second, we require algorithms that can handle the full gamut of losses and regularizers, which may be smooth or nonsmooth, finite or infinite valued, with arbitrary domain. This work is the first to consider these problems in such generality, and therefore also the first to wrestle with the algorithmic consequences. Below, we give a number of algorithms appropriate for this setting, including many that have not been previously proposed in the literature. Our algorithms are all based on alternating minimization and variations on alternating minimization that are more suitable for large scale data and can take advantage of parallel computing resources.

These algorithms for fitting *any* GLRM are particularly useful for interactive data analysis: a practitioner can mix and match different loss functions and regularizers, and test which combinations provide the best fit to the data, without having to identify a different method to fit each particular model. We present a few software packages designed for this purpose, with interfaces in Julia, R, Java, Python, and Scala, in §9.

Finally, we present some new results on some old problems. For example, in Appendix A.1, we derive a formula for the solution to quadratically regularized PCA, and show that quadratically regularized PCA has no local nonglobal minima; and in §7.6 we show how to certify (in some special cases) that a model is a global solution of a GLRM.

1.2 Organization

The organization of this paper is as follows. In §2 we first recall some properties of PCA and its common variations to familiarize the reader with our notation. We then generalize the regularization on the low dimensional factors in §3, and the loss function on the approximation error in §4. Returning to the setting of heterogeneous data, we extend these dimensionality reduction techniques to abstract data types in §5 and to multi-dimensional loss functions in §6. Finally, we address algorithms for fitting GLRMs in §7, discuss a few practical considerations in choosing a GLRM for a particular problem in §8, and describe some implementations of the algorithms that we have developed in §9.

2

PCA and quadratically regularized 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 -1 (for false), ordinal data is often encoded using consecutive integers to represent the consecutive 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 in the column corresponding to the observed value, and -1 or 0 in all other columns. We will see more systematic and principled ways to deal with these data types, and others, in §4–6. For now, we assume the entries in the data matrix consist of real numbers.

2.1 PCA

Principal components analysis (PCA) is one of the oldest and most widely used tools in data analysis [111, 62, 67]. 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 the best rank-k approximation to the matrix A in the least-squares sense, by solving

$$\begin{array}{ll} \text{minimize} & \|A - Z\|_F^2 \\ \text{subject to} & \text{Rank}(Z) \le k, \end{array} \tag{2.1}$$

with variable $Z \in \mathbf{R}^{m\times n}$. Here, $\|\cdot\|_F$ is the Frobenius norm of a matrix, *i.e.*, the square root of the sum of the squares of the entries.

The rank constraint can be encoded implicitly by expressing Z in factored form as $Z = XY$, with $X \in \mathbf{R}^{m\times k}$, $Y \in \mathbf{R}^{k\times n}$. Then the PCA problem can be expressed as

$$\text{minimize} \quad \|A - XY\|_F^2 \tag{2.2}$$

with variables $X \in \mathbf{R}^{m\times k}$ and $Y \in \mathbf{R}^{k\times n}$. (The factorization of Z is of course not unique.)

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. Thus $x_i y_j = (XY)_{ij} \in \mathbf{R}$ denotes a dot or inner product. (We will use this notation throughout the paper.) Using this definition, we can rewrite the objective in problem (2.2) as

$$\sum_{i=1}^{m}\sum_{j=1}^{n}(A_{ij} - x_i y_j)^2.$$

We will give several interpretations of the low rank factorization (X, Y) solving (2.2) in §2.5. But for now, we note that (2.2) can interpreted as a method for compressing the n features in the original data set to $k < n$ new features. The row vector x_i is associated with example i; we can think of it as a feature vector for the example using the compressed set of $k < n$ features. The column vector y_j is associated with the original feature j; it can be interpreted as mapping the k new features onto the original feature j.

2.2 Quadratically regularized PCA

We can add quadratic regularization on X and Y to the objective. The quadratically regularized PCA problem is

$$\text{minimize} \quad \textstyle\sum_{i=1}^m \sum_{j=1}^n (A_{ij} - x_i y_j)^2 + \gamma \sum_{i=1}^m \|x_i\|_2^2 + \gamma \sum_{j=1}^n \|y_j\|_2^2, \tag{2.3}$$

with variables $X \in \mathbf{R}^{m \times k}$ and $Y \in \mathbf{R}^{k \times n}$, and regularization parameter $\gamma \geq 0$. Problem (2.3) can be written more concisely in matrix form as

$$\text{minimize} \quad \|A - XY\|_F^2 + \gamma \|X\|_F^2 + \gamma \|Y\|_F^2. \tag{2.4}$$

When $\gamma = 0$, the problem reduces to the PCA problem (2.2).

2.3 Solution methods

Singular value decomposition. It is well known that a solution to (2.2) can be obtained by truncating the *singular value decomposition* (SVD) of A [44]. The (compact) SVD of A is given by $A = U \Sigma V^T$, where $U \in \mathbf{R}^{m \times r}$ and $V \in \mathbf{R}^{n \times r}$ have orthonormal columns, and $\Sigma = \mathbf{diag}(\sigma_1, \ldots, \sigma_r) \in \mathbf{R}^{r \times r}$, with $\sigma_1 \geq \cdots \geq \sigma_r > 0$ and $r = \text{Rank}(A)$. The columns of $U = [u_1 \cdots u_r]$ and $V = [v_1 \cdots v_r]$ are called the left and right singular vectors of A, respectively, and $\sigma_1, \ldots, \sigma_r$ are called the singular values of A.

Using the orthogonal invariance of the Frobenius norm, we can rewrite the objective in problem (2.1) as

$$\|A - XY\|_F^2 = \|\Sigma - U^T XYV\|_F^2.$$

That is, we would like to find a matrix $U^T XYV$ of rank no more than k approximating the diagonal matrix Σ. It is easy to see that there is no better rank k approximation for Σ than $\Sigma_k = \mathbf{diag}(\sigma_1, \ldots, \sigma_k, 0, \ldots, 0) \in \mathbf{R}^{r \times r}$. Here we have *truncated* the SVD to keep only the top k singular values. We can achieve this approximation by choosing $U^T XYV = \Sigma_k$, or (using the orthogonality of U and V) $XY = U \Sigma_k V^T$. For example, define

$$U_k = [u_1 \cdots u_k], \quad V_k = [v_1 \cdots v_k], \tag{2.5}$$

and let
$$X = U_k \Sigma_k^{1/2}, \qquad Y = \Sigma_k^{1/2} V_k^T. \tag{2.6}$$
The solution to (2.3) is clearly not unique: if X, Y is a solution, then so is XG, $G^{-1}Y$ for any invertible matrix $G \in \mathbf{R}^{k \times k}$. When $\sigma_k > \sigma_{k+1}$, all solutions to the PCA problem have this form. In particular, letting $G = tI$ and taking $t \to \infty$, we see that the solution set of the PCA problem is unbounded.

It is less well known that a solution to the quadratically regularized PCA problem can be obtained in the same way. (Proofs for the statements below can be found in Appendix A.1.) Define U_k and V_k as above, and let $\tilde{\Sigma}_k = \mathbf{diag}((\sigma_1 - \gamma)_+, \ldots, (\sigma_k - \gamma)_+)$, where $(a)_+ = \max(a, 0)$. Here we have both *truncated* the SVD to keep only the top k singular values, and performed *soft-thresholding* on the singular values to reduce their values by γ. A solution to the quadratically regularized PCA problem (2.3) is then given by
$$X = U_k \tilde{\Sigma}_k^{1/2}, \qquad Y = \tilde{\Sigma}_k^{1/2} V_k^T. \tag{2.7}$$
For $\gamma = 0$, the solution reduces to the familiar solution to PCA (2.2) obtained by truncating the SVD to the top k singular values.

The set of solutions to problem (2.3) is significantly smaller than that of problem (2.2), although solutions are still not unique: if X, Y is a solution, then so is XT, $T^{-1}Y$ for any orthogonal matrix $T \in \mathbf{R}^{k \times k}$. When $\sigma_k > \sigma_{k+1}$, all solutions to (2.3) have this form. In particular, adding quadratic regularization results in a solution set that is bounded.

The quadratically regularized PCA problem (2.3) (including the PCA problem as a special case) is the only problem we will encounter for which an analytical solution exists. The analytical tractability of PCA explains its popularity as a technique for data analysis in the era before computers were machines. For example, in his 1933 paper on PCA [62], Hotelling computes the solution to his problem using power iteration to find the eigenvalue decomposition of the matrix $A^T A = V \Sigma^2 V^T$, and records in the appendix to his paper the intermediate results at each of the (three) iterations required for the method to converge.

Alternating minimization. Here we mention a second method for solving (2.3), which extends more readily to the extensions of PCA that we discuss below. The *alternating minimization* algorithm simply alternates between minimizing the objective over the variable X, holding Y fixed, and then minimizing over Y, holding X fixed. With an initial guess for the factors Y^0, we repeat the iteration

$$\begin{aligned} X^l &= \operatorname*{argmin}_X \left(\sum_{i=1}^m \sum_{j=1}^n (A_{ij} - x_i y_j^{l-1})^2 + \gamma \sum_{i=1}^m \|x_i\|_2^2 \right) \\ Y^l &= \operatorname*{argmin}_Y \left(\sum_{i=1}^m \sum_{j=1}^n (A_{ij} - x_i^l y_j)_{ij})^2 + \gamma \sum_{j=1}^n \|y_j\|_2^2 \right) \end{aligned}$$

for $l = 1, \ldots$ until a stopping condition is satisfied. (If X and Y are full rank, or $\gamma > 0$, the minimizers above are unique; when they are not, we can take any minimizer.) The objective function is nonincreasing at each iteration, and therefore bounded. This implies, for $\gamma > 0$, that the iterates X^l and Y^l are bounded.

This algorithm does not always work. In particular, it has stationary points that are not solutions of problem (2.3). In particular, if the rows of Y^l lie in a subspace spanned by a subset of the (right) singular vectors of A, then the columns of X^{l+1} will lie in a subspace spanned by the corresponding left singular vectors of A, and vice versa. Thus, if the algorithm is initialized with Y^0 orthogonal to any of the top k (right) singular vectors, then the algorithm (implemented in exact arithmetic) will not converge to the global solution to the problem.

But all *stable* stationary points of the iteration are solutions (see Appendix A.1). So as a practical matter, the alternating minimization method always works, *i.e.*, the objective converges to the optimal value.

Parallelizing alternating minimization. Alternating minimization parallelizes easily over examples and features. The problem of minimizing over X splits into m independent minimization problems. We can solve the simple quadratic problems

$$\text{minimize} \quad \textstyle\sum_{j=1}^n (A_{ij} - x_i y_j)^2 + \gamma \|x_i\|_2^2 \tag{2.8}$$

with variable x_i, in parallel, for $i = 1, \dots, m$. Similarly, the problem of minimizing over Y splits into n independent quadratic problems,

$$\text{minimize} \quad \textstyle\sum_{i=1}^{m} (A_{ij} - x_i y_j)^2 + \gamma \|y_j\|_2^2 \tag{2.9}$$

with variable y_j, which can be solved in parallel for $j = 1, \dots, n$.

Caching factorizations. We can speed up the solution of the quadratic problems using a simple factorization caching technique.

For ease of exposition, we assume here that X and Y have full rank k. The updates (2.8) and (2.9) can be expressed as

$$X = AY^T (YY^T + \gamma I)^{-1}, \qquad Y = (X^T X + \gamma I)^{-1} X^T A.$$

We show below how to efficiently compute $X = AY^T (YY^T + \gamma I)^{-1}$; the Y update admits a similar speedup using the same ideas. We assume here that k is modest, say, not more than a few hundred or a few thousand. (Typical values used in applications are often far smaller, on the order of tens.) The dimensions m and n, however, can be very large.

First compute the Gram matrix $G = YY^T$ using an outer product expansion

$$G = \sum_{j=1}^{n} y_j y_j^T.$$

This sum can be computed on-line by streaming over the index j, or in parallel, split over the index j. This property allows us to scale up to extremely large problems even if we cannot store the entire matrix Y in memory. The computation of the Gram matrix requires $2k^2 n$ floating point operations (flops), but is trivially parallelizable: with r workers, we can expect a speedup on the order of r. We next add the diagonal matrix γI to G in k flops, and form the Cholesky factorization of $G + \gamma I$ in $k^3/3$ flops and cache the factorization.

In parallel over the rows of A, we compute $D = AY^T$ ($2kn$ flops per row), and use the factorization of $G + \gamma I$ to compute $D(G + \gamma I)^{-1}$ with two triangular solves ($2k^2$ flops per row). These computations are also trivially parallelizable: with r workers, we can expect a speedup on the order of r.

Hence the total time required for each update with r workers scales as $\mathcal{O}(\frac{k^2(m+n)+kmn}{r})$. For k small compared to m and n, the time is dominated by the computation of AY^T.

2.4 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

$$\text{minimize} \quad \textstyle\sum_{(i,j)\in\Omega}(A_{ij} - x_i y_j)^2 + \gamma\|X\|_F^2 + \gamma\|Y\|_F^2, \qquad (2.10)$$

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 (2.10) 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 (2.10) 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 fact that this is possible: typical results from the matrix completion literature show that one can recover an unknown $m \times n$ matrix A of low rank r from just about $nr \log_2 n$ noisy samples Ω with an error that is proportional to the noise level [23, 24, 117, 22], so long as the matrix A satisfies a certain incoherence condition and the samples Ω are chosen uniformly at random. These works use an estimator that minimizes a nuclear norm penalty along with a data fitting term to encourage low rank structure in the solution.

The argument in §7.6 shows that problem (2.10) is equivalent to the rank-constrained nuclear-norm regularized convex problem

$$\begin{array}{ll} \text{minimize} & \sum_{(i,j)\in\Omega}(A_{ij} - Z_{ij})^2 + 2\gamma\|Z\|_* \\ \text{subject to} & \mathrm{Rank}(Z) \leq k, \end{array}$$

where the *nuclear norm* $\|Z\|_*$ (also known as the trace norm) is defined to be the sum of the singular values of Z. Thus, the solutions to problem (2.10) correspond exactly to the solutions of these proposed estimators so long as the rank k of the model is chosen to be larger than the true rank r of the matrix A. Nuclear norm regularization is often used to encourage solutions of rank less than k, and has applications ranging from graph embedding to linear system identification [46, 92, 102, 130, 107].

Low rank matrix completion problems arise in applications like predicting customer ratings or customer (potential) purchases. Here the matrix consists of the ratings or numbers of purchases that m customers give (or make) for each of n products. The vast majority of the entries in this matrix are missing, since a customer will rate (or purchase) only a small fraction of the total number of products available. In this application, imputing a missing entry of the matrix as $x_i y_j$, for $(i,j) \notin \Omega$, is guessing what rating a customer would give a product, if she were to rate it. This can used as the basis for a recommendation system, or a marketing plan.

Alternating minimization. When $\Omega \neq \{1,\ldots,m\} \times \{1,\ldots,n\}$, the problem (2.10) has no known analytical solution, but it is still easy to fit a model using alternating minimization. Algorithms based on alternating minimization have been shown to converge quickly (even geometrically [66]) to a global solution satisfying a recovery guarantee when the initial values of X and Y are chosen carefully [74, 76, 73, 66, 58, 55, 59, 140].

However, none of these results applies to the algorithm — alternating minimization on problem (2.10) — most often used in practice. For example, none uses the quadratic regularizer above that corresponds to the nuclear norm penalized estimator; and all of these analytical results, until the recent paper [140], rely on using a *fresh* batch of samples Ω for each iteration of alternating minimization. Interestingly, Hardt [58] notes that none of these alternating methods achieves the same sample complexity guarantees found in the convex matrix completion literature which, unlike the alternating minimization guarantees, match the information theoretic lower bound [24] up to logarithmic factors. We expect that these shortcomings — weaker error bounds for more complex algorithms — are an artifact of current proof techniques, rather than a fundamental limitation of alternating approaches. But for now, alternating minimization applied to Problem (2.10) should still be considered a (very good) heuristic optimization method.

2.5 Interpretations and applications

The recovered matrices X and Y in the quadratically regularized PCA problems (2.3) and (2.10) admit a number of interesting interpretations. We introduce some of these interpretations now; the terminology we use here will recur throughout the paper. Of course these interpretations are related to each other, and not distinct.

Feature compression. Quadratically regularized PCA (2.3) can be interpreted as a method for compressing the n features in the original data set to $k < n$ new features. The row vector x_i is associated with example i; we can think of it as a feature vector for the example using the compressed set of $k < n$ features. The column vector y_j is associated with the original feature j; it can be interpreted as the mapping from the original feature j into the k new features.

Low-dimensional geometric embedding. We can think of each y_j as associating feature j with a point in a low (k-) dimensional space. Similarly, each x_i associates example i with a point in the low dimensional space. We can use these low dimensional vectors to judge which features (or examples) are similar. For example, we can run a clustering algorithm on the low dimensional vectors y_j (or x_i) to find groups of similar features (or examples).

Archetypes. We can think of each row of Y as an *archetype* which captures the behavior of one of k idealized and maximally informative examples. These archetypes might also be called profiles, factors, or atoms. Every example $i = 1, \ldots, m$ is then represented (approximately) as a linear combination of these archetypes, with the row vector x_i giving the coefficients. The coefficient x_{il} gives the resemblance or *loading* of example i to the lth archetype.

Archetypical representations. We call x_i the *representation* of example i in terms of the archetypes. The rows of X give an embedding of the examples into $\mathbf{R}^k$, where each coordinate axis corresponds to a different archetype. If the archetypes are simple to understand or interpret, then the representation of an example can provide better intuition about that example.

The examples can be clustered according to their representations in order to determine a group of similar examples. Indeed, one might choose to apply any machine learning algorithm to the representations x_i rather than to the initial data matrix: in contrast to the initial data, which may consist of high dimensional vectors with noisy or missing entries, the representations x_i will be low dimensional, less noisy, and complete.

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

Latent variables. Each row of X represents an example by a vector in $\mathbf{R}^k$. The matrix Y maps these representations back into $\mathbf{R}^m$. We might think of X as discovering the *latent variables* that best explain the observed data. If the approximation error $\sum_{(i,j)\in\Omega}(A_{ij}-x_iy_j)^2$ is small, then we view these latent variables as providing a good explanation or summary of the full data set.

Probabilistic interpretation. We can give a probabilistic interpretation of X and Y, building on the probabilistic model of PCA developed by Tipping and Bishop [142]. We suppose that the matrices $\bar{X}$ and $\bar{Y}$ have entries which are generated by taking independent samples from a normal distribution with mean 0 and variance γ^{-1} for $\gamma > 0$. The entries in the matrix $\bar{X}\bar{Y}$ are observed with noise $\eta_{ij} \in \mathbf{R}$,

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

where the noise η in the (i,j)th entry is sampled independently from a standard normal distribution. We observe each entry $(i,j) \in \Omega$. Then to find the maximum a posteriori (MAP) estimator (X,Y) of $(\bar{X},\bar{Y})$, we solve

$$\begin{array}{ll} \text{maximize} & \exp\left(-\frac{\gamma}{2}\|\bar{X}\|_F^2\right)\exp\left(-\frac{\gamma}{2}\|\bar{Y}\|_F^2\right) \\ & \times \prod_{(i,j)\in\Omega}\exp\left(-(A_{ij}-x_iy_j)^2\right), \end{array}$$

which is equivalent, by taking logs, to (2.3).

This interpretation explains the recommendation we gave above for imputing missing observations $(i,j) \notin \Omega$. We simply use the MAP estimator x_iy_j to estimate the missing entry $(\bar{X}\bar{Y})_{ij}$. Similarly, we can interpret $(XY)_{ij}$ for $(i,j) \in \Omega$ as a denoised version of the observation A_{ij}.

Auto-encoder. The matrix X encodes the data; the matrix Y decodes it back into the full space. We can view PCA as providing the best linear auto-encoder for the data; among all (bi-linear) low rank encodings (X) and decodings (Y) of the data, PCA minimizes the squared reconstruction error.

Compression. We impose an *information bottleneck* [143] on the data by using a low rank auto-encoder to fit the data. PCA finds X and Y to maximize the information transmitted through this k-dimensional information bottleneck. We can interpret the solution as a compressed representation of the data, and use it to efficiently store or transmit the information present in the original data.

2.6 Offsets and scaling

For good practical performance of a generalized low rank model, it is critical to ensure that model assumptions match the data. We saw above in §2.5 that quadratically regularized PCA corresponds to a model in which features are observed with $\mathcal{N}(0,1)$ errors. If instead each column j of XY is observed with $\mathcal{N}(\mu_j, \sigma_j^2)$ errors, our 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 applying PCA or quadratically regularized 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 $n_j = |\{i : (i,j) \in \Omega\}|$, and let

$$\mu_j = \frac{1}{n_j} \sum_{i\colon (i,j)\in\Omega} A_{ij}, \qquad \sigma_j^2 = \frac{1}{n_j - 1} \sum_{i\colon (i,j)\in\Omega} (A_{ij} - \mu_j)^2$$

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

3

Generalized regularization

It is easy to see how to extend PCA to allow arbitrary regularization on the rows of X and columns of Y. We form the *regularized PCA problem*

$$\text{minimize} \quad \textstyle\sum_{(i,j)\in\Omega}(A_{ij} - x_i y_j)^2 + \sum_{i=1}^m r_i(x_i) + \sum_{j=1}^n \tilde{r}_j(y_j), \quad (3.1)$$

with variables x_i and y_j, with given regularizers $r_i : \mathbf{R}^k \to \mathbf{R} \cup \{\infty\}$ and $\tilde{r}_j : \mathbf{R}^k \to \mathbf{R} \cup \{\infty\}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$. Regularized PCA (3.1) reduces to quadratically regularized PCA (2.3) when $r_i = \gamma\|\cdot\|_2^2$, $\tilde{r}_j = \gamma\|\cdot\|_2^2$. We do not restrict the regularizers to be convex.

The objective in problem (3.1) can be expressed compactly in matrix notation as

$$\|A - XY\|_F^2 + r(X) + \tilde{r}(Y),$$

where $r(X) = \sum_{i=1}^n r_i(x_i)$ and $\tilde{r}(Y) = \sum_{j=1}^n \tilde{r}_j(y_j)$. The regularization functions r and $\tilde{r}$ are separable across the rows of X, and the columns of Y, respectively.

Infinite values of r_i and $\tilde{r}_j$ are used to enforce constraints on the values of X and Y. For example, the regularizer

$$r_i(x) = \begin{cases} 0 & x \geq 0 \\ \infty & \text{otherwise,} \end{cases}$$

the indicator function of the nonnegative orthant, imposes the constraint that x_i be nonnegative.

Solutions to (3.1) need not be unique, depending on the choice of regularizers. If X and Y are a solution, then so are XT and $T^{-1}Y$, where T is any nonsingular matrix that satisfies $r(UT) = r(U)$ for all U and $\tilde{r}(T^{-1}V) = r(V)$ for all V.

By varying our choice of regularizers r and $\tilde{r}$, we are able to represent a wide range of known models, as well as many new ones. We will discuss a number of choices for regularizers below, but turn now to methods for solving the regularized PCA problem (3.1).

3.1 Solution methods

In general, there is no analytical solution for (3.1). The problem is not convex, even when r and $\tilde{r}$ are convex. However, when r and $\tilde{r}$ are convex, the problem is bi-convex: it is convex in X when Y is fixed, and convex in Y when X is fixed.

Alternating minimization. There is no reason to believe that alternating minimization will always converge to the global minimum of the regularized PCA problem (3.1). Indeed, we will see many cases below in which the problem is known to have many local minima. However, alternating minimization can still be applied in this setting, and it still parallelizes over the rows of X and columns of Y. To minimize over X, we solve, in parallel,

$$\text{minimize} \quad \textstyle\sum_{j:(i,j)\in\Omega}(A_{ij} - x_i y_j)^2 + r_i(x_i) \tag{3.2}$$

with variable x_i, for $i = 1, \ldots, m$. Similarly, to minimize over Y, we solve, in parallel,

$$\text{minimize} \quad \textstyle\sum_{i:(i,j)\in\Omega}(A_{ij} - x_i y_j)^2 + \tilde{r}_j(y_j) \tag{3.3}$$

with variable y_j, for $j = 1, \ldots, n$.

When the regularizers are convex, these problems are convex. When the regularizers are not convex, there are still many cases in which we can find analytical solutions to the nonconvex subproblems (3.2) and

(3.3), as we will see below. A number of concrete algorithms, in which these subproblems are solved explicitly, are given in §7.

Caching factorizations. Often, the X and Y updates (3.2) and (3.3) reduce to convex quadratic programs. For example, this is the case for nonnegative matrix factorization, sparse PCA, and quadratic mixtures (which we define and discuss below in §3.2). The same factorization caching of the Gram matrix that was described above in the case of PCA can be used here to speed up the solution of these updates. Variations on this idea are described in detail in §7.3.

3.2 Examples

Here and throughout the paper, we present a set of examples chosen for pedagogical clarity, not for completeness. In all of the examples below, $\gamma > 0$ is a parameter that controls the strength of the regularization, and we drop the subscripts from r (or $\tilde{r}$) to lighten the notation. Of course, it is possible to mix and match these regularizers, *i.e.*, to choose different r_i for different i, and choose different $\tilde{r}_j$ for different j.

Nonnegative matrix factorization (NNMF). Consider the regularized PCA problem (3.1) with $r = \mathbf{I}_+$ and $\tilde{r} = \mathbf{I}_+$, where $\mathbf{I}_+$ is the indicator function of the nonnegative reals. (Here, and throughout the paper, we define the indicator function of a set C, to be 0 when its argument is in C and ∞ otherwise.) Then problem (3.1) is NNMF: a solution gives the matrix best approximating A that has a nonnegative factorization (*i.e.*, a factorization into elementwise nonnegative matrices) [85]. It is NP-hard to solve NNMF problems exactly [149]. However, these problems have a rich analytical structure which can sometimes be exploited [49, 10, 31], and a wide range of uses in practice [85, 126, 7, 151, 77, 47]. Hence a number of specialized algorithms and codes for fitting NNMF models are available [86, 91, 78, 80, 13, 79, 81].

We can also replace the nonnegativity constraint with any interval constraint. For example, r and $\tilde{r}$ can be 0 if all entries of X and Y, respectively, are between 0 and 1, and infinite otherwise.

Sparse PCA. If very few of the coefficients of X and Y are nonzero, it can be easier to interpret the archetypes and representations. We can understand each archetype using only a small number of features, and can understand each example as a combination of only a small number of archetypes. To get a sparse version of PCA, we use a sparsifying penalty as the regularization. Many variants on this basic idea have been proposed, together with a wide variety of algorithms [33, 161, 128, 94, 155, 122, 152].

For example, we could enforce that no entry A_{ij} depend on more than s columns of X or of Y by setting r to be the indicator function of a s-sparse vector, *i.e.*,

$$r(x) = \begin{cases} 0 & \mathbf{card}(x) \leq s \\ \infty & \text{otherwise,} \end{cases}$$

and defining $\tilde{r}(y)$ similarly, where $\mathbf{card}(x)$ denotes the cardinality (number of nonzero entries) in the vector x. The updates (3.2) and (3.3) are not convex using this regularizer, but one can find approximate solutions using a pursuit algorithm (see, *e.g.*, [28, 145]), or exact solutions (for small s) using the branch and bound method [84, 15].

As a simple example, consider $s = 1$. Here we insist that each x_i have at most one nonzero entry, which means that each example is a multiple of *one* of the rows of Y. The X-update is easy to carry out, by evaluating the best quadratic fit of x_i with each of the k rows of Y. This reduces to choosing the row of Y that has the smallest angle to the ith row of A.

The s-sparse regularization can be relaxed to a convex, but still sparsifying, regularization using $r(x) = \|x\|_1$, $\tilde{r}(y) = \|y\|_1$ [161]. In this case, the X-update reduces to solving a (small) ℓ_1-regularized least-squares problem.

Orthogonal nonnegative matrix factorization. One well known property of PCA is that the principal components obtained (*i.e.*, the columns of X and rows of Y) can be chosen to be orthogonal, so X^TX and YY^T are both diagonal. We can impose the same condition on a nonnegative matrix factorization. Due to nonnegativity of the matrix, two columns of X cannot be orthogonal if they both have a nonzero in the same row. Conversely, if X has only one nonzero per row, then its columns are mutually orthogonal. So an orthogonal nonnegative matrix factorization is identical to a nonnegativity condition in addition to the 1-sparse condition described above. Orthogonal nonnegative matrix factorization can be achieved by using the regularizer

$$r(x) = \begin{cases} 0 & \mathbf{card}(x) = 1, \quad x \geq 0 \\ \infty & \text{otherwise,} \end{cases}$$

and letting $\tilde{r}(y)$ be the indicator of the nonnegative orthant, as in NNMF.

Geometrically, we can interpret this problem as modeling the data A as a union of rays. Each row of Y, interpreted as a point in $\mathbf{R}^n$, defines a ray from the origin passing through that point. Orthogonal nonnegative matrix factorization models each row of X as a point along one of these rays.

Some authors [41] have also considered how to obtain a *bi-orthogonal* nonnegative matrix factorization, in which both X and Y^T have orthogonal columns. By the same argument as above, we see this is equivalent to requiring both X and Y^T to have only one positive entry per row, with the other entries equal to 0.

Max-norm matrix factorization. We take $r = \tilde{r} = \phi$ with

$$\phi(x) = \begin{cases} 0 & \|x\|_2^2 \leq \mu \\ \infty & \text{otherwise.} \end{cases}$$

This penalty enforces that

$$\|X\|_{2,\infty}^2 \leq \mu, \qquad \|Y^T\|_{2,\infty}^2 \leq \mu,$$

where the $(2,\infty)$ norm of a matrix X with rows x_i is defined as $\max_i \|x_i\|_2$. This is equivalent to requiring the *max-norm* (sometimes called the γ_2-*norm*) of $Z = XY$, which is defined as

$$\|Z\|_{\max} = \inf\{\|X\|_{2,\infty}\|Y^T\|_{2,\infty} : XY = Z\},$$

to be bounded by μ. This penalty has been proposed by [88] as a heuristic for low rank matrix completion, which can perform better than Frobenius norm regularization when the low rank factors are known to have bounded entries.

Quadratic clustering. Consider (3.1) with $\tilde{r} = 0$. Let r be the indicator function of a selection, *i.e.*,

$$r(x) = \begin{cases} 0 & x = e_l \text{ for some } l \in \{1, \ldots, k\} \\ \infty & \text{otherwise,} \end{cases}$$

where e_l is the lth standard basis vector. Thus x_i encodes the cluster (one of k) to which the data vector $(A_{i1}, \ldots, A_{im})$ is assigned.

Alternating minimization on this problem reproduces the well-known k-means algorithm (also known as Lloyd's algorithm) [93]. The y update (3.3) is a least squares problem with the simple solution

$$Y_{lj} = \frac{\sum_{i:(i,j)\in\Omega} A_{ij} X_{il}}{\sum_{i:(i,j)\in\Omega} X_{il}},$$

i.e., each row of Y is updated to be the mean of the rows of A assigned to that archetype. The x update (3.2) is not a convex problem, but is easily solved. The solution is given by assigning x_i to the closest archetype (often called a *cluster centroid* in the context of k-means): $x_i = e_{l^\star}$ for $l^\star = \operatorname{argmin}_l \left(\sum_{j=1}^n (A_{ij} - Y_{lj})^2 \right)$.

Quadratic mixtures. We can also implement partial assignment of data vectors to clusters. Take $\tilde{r} = 0$, and let r be the indicator function of the set of probability vectors, *i.e.*,

$$r(x) = \begin{cases} 0 & \sum_{l=1}^k x_l = 1, \quad x_l \geq 0 \\ \infty & \text{otherwise.} \end{cases}$$

Subspace clustering. PCA approximates a data set by a single low dimensional subspace. We may also be interested in approximating a data set as a *union* of low dimensional subspaces. This problem is known as *subspace clustering* (see [150] and references therein). Subspace clustering may also be thought of as generalizing quadratic clustering to assign each data vector to a low dimensional subspace rather than to a single cluster centroid.

To frame subspace clustering as a regularized PCA problem (3.1), partition the columns of X into k blocks. Then let r be the indicator function of block sparsity (*i.e.*, $r(x) = 0$ if only one block of x has nonzero entries, and otherwise $r(x) = \infty$).

It is easy to perform alternating minimization on this objective function. This method is sometimes called the *k-planes* algorithm [150, 146, 3], which alternates over assigning examples to subspaces, and fitting the subspaces to the examples. Once again, the X update (3.2) is not a convex problem, but can be easily solved. Each block of the columns of X defines a subspace spanned by the corresponding rows of Y. We compute the distance from example i (the ith row of A) to each subspace (by solving a least squares problem), and assign example i to the subspace that minimizes the least squares error by setting x_i to be the solution to the corresponding least squares problem.

Many other algorithms for this problem have also been proposed, such as the k-SVD [144, 4] and sparse subspace clustering [45], some with provable guarantees on the quality of the recovered solution [131].

Supervised learning. Sometimes we want to understand the variation that a certain set of features can explain, and the variance that remains unexplainable. To this end, one natural strategy would be to regress the labels in the dataset on the features; to subtract the predicted values from the data; and to use PCA to understand the remaining variance. This procedure gives the same answer as the solution to a single regularized PCA problem. Here we present the case in which the features we wish to use in the regression are present in the data as the first column of A. To construct the regularizers, we make sure the first column of A appears as a feature in the supervised learning problem by setting

$$r_i(x) = \begin{cases} r_0(x_2, \ldots, x_{k+1}) & x_1 = A_{i1} \\ \infty & \text{otherwise,} \end{cases}$$

where $r_0 = 0$ can be chosen as in any regularized PCA model. The regularization on the first row of Y is the regularization used in the supervised regression, and the regularization on the other rows will be that used in regularized PCA.

Thus we see that regularized PCA can naturally combine supervised and unsupervised learning into a single problem.

Feature selection. We can use regularized PCA to perform feature selection. Consider (3.1) with $r(x) = \|x\|_2^2$ and $\tilde{r}(y) = \|y\|_2$. (Notice that we are *not* using $\|y\|_2^2$.) The regularizer $\tilde{r}$ encourages the matrix $\tilde{Y}$ to be column-sparse, so many columns are all zero. If $\tilde{y}_j = 0$, it means that feature j was uninformative, in the sense that its values do not help much in predicting any feature in the matrix A (including feature j itself). In this case we say that feature j was not selected. For this approach to make sense, it is important that the columns of the matrix A should have mean zero. Alternatively, one can use the debiasing regularizers r' and $\tilde{r}'$ introduced in §3.3 along with the feature selection regularizer introduced here.

Dictionary learning. Dictionary learning (also sometimes called *sparse coding*) has become a popular method to design concise representations for very high dimensional data [106, 87, 95, 96]. These representations have been shown to perform well when used as features in subsequent (supervised) machine learning tasks [116]. In dictionary learning, each row of A is modeled as a linear combination of dictionary atoms, represented by rows of Y. The total size of the dictionary used is often very large ($k \gg \max(m, n)$), but each example is represented using a very small number of atoms. To fit the model, one solves the regularized PCA problem (3.1) with $r(x) = \|x\|_1$, to induce sparsity in the number of atoms used to represent any given example, and with $\tilde{r}(y) = \|y\|_2^2$ or $\tilde{r}(y) = \mathbf{I}_+(c - \|y\|_2)$ for some $c > 0 \in \mathbf{R}$, in order to ensure the problem is well posed. (Note that our notation transposes the usual notation in the literature on dictionary learning.)

Mix and match. It is possible to combine these regularizers to obtain a factorization with any combination of the above properties. As an example, one may require that both X and Y be simultaneously sparse and nonnegative by choosing

$$r(x) = \|x\|_1 + \mathbf{I}_+(x) = \mathbf{1}^T x + \mathbf{I}_+(x),$$

and similarly for $\tilde{r}(y)$. Similarly, [77] show how to obtain a nonnegative matrix factorization in which one factor is sparse by using $r(x) = \|x\|_1^2 +$

$\mathbf{I}_+(x)$ and $\tilde{r}(y) = \|y\|_2^2 + \mathbf{I}_+(y)$; they go on to use this factorization as a clustering technique.

3.3 Offsets and scaling

In our discussion of the quadratically regularized PCA problem (2.3), we saw that it can often be quite important to standardize the data before applying PCA. Conversely, in regularized PCA problems such as nonnegative matrix factorization, it makes no sense to standardize the data, since subtracting column means introduces negative entries into the matrix.

A flexible approach is to allow an offset in the model: we solve

$$\text{minimize} \quad \textstyle\sum_{(i,j)\in\Omega}(A_{ij} - x_i y_j - \mu_j)^2 + \sum_{i=1}^m r_i(x_i) + \sum_{j=1}^n \tilde{r}_j(y_j), \tag{3.4}$$

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 the standard form regularized PCA problem (3.1) by augmenting the problem slightly. Suppose we are given an instance of the problem (3.1), *i.e.*, we are given k, r, and $\tilde{r}$. We can fit an offset term μ_j by letting $k' = k + 1$ and modifying the regularizers. Extend the regularization $r : \mathbf{R}^k \to \mathbf{R}$ and $\tilde{r} : \mathbf{R}^k \to \mathbf{R}$ to new regularizers $r' : \mathbf{R}^{k+1} \to \mathbf{R}$ and $\tilde{r}' : \mathbf{R}^{k+1} \to \mathbf{R}$ which enforce that the first column of X is constant and the first row of Y is not penalized. Using this scheme, the first row of the optimal Y will be equal to the optimal μ in (3.4).

Explicitly, let

$$r'(x) = \begin{cases} r(x_2, \ldots, x_{k+1}) & x_1 = 1 \\ \infty & \text{otherwise,} \end{cases}$$

and $\tilde{r}'(y) = \tilde{r}(y_2, \ldots, y_{k+1})$. (Here, we identify $r(x) = r(x_1, \ldots, x_k)$ to explicitly show the dependence on each coordinate of the vector x, and similarly for $\tilde{r}$.)

It is also possible to introduce row offsets in the same way.

4

Generalized loss functions

We may also generalize the *loss* function in PCA to form a *generalized low rank model*,

$$\text{minimize} \quad \textstyle\sum_{(i,j)\in\Omega} L_{ij}(x_i y_j, A_{ij}) + \sum_{i=1}^m r_i(x_i) + \sum_{j=1}^n \tilde{r}_j(y_j), \tag{4.1}$$

where $L_{ij} : \mathbf{R} \times \mathbf{R} \to \mathbf{R}_+$ are given loss functions for $i = 1, \ldots, m$ and $j = 1, \ldots, n$. Problem (4.1) reduces to PCA with generalized regularization when $L_{ij}(u, a) = (a - u)^2$. However, the loss function L_{ij} can now depend on the data A_{ij} in a more complex way.

4.1 Solution methods

As before, problem (4.1) is not convex, even when L_{ij}, r_i and $\tilde{r}_j$ are convex; but if all these functions are convex, then the problem is biconvex.

Alternating minimization. Alternating minimization can still be used to find a local minimum, and it is still often possible to use factorization caching to speed up the solution of the subproblems that arise in alternating minimization. We defer a discussion of how to solve these subproblems explicitly to §7.

Stochastic proximal gradient method. For use with extremely large scale problems, we discuss fast variants of the basic alternating minimization algorithm in §7. For example, we present an alternating directions stochastic proximal gradient method. This algorithm accesses the functions L_{ij}, r_i, and $\tilde{r}_j$ only through a subgradient or proximal interface, allowing it to generalize trivially to nearly any loss function and regularizer. We defer a more detailed discussion of this method to §7.

4.2 Examples

Weighted PCA. A simple modification of the PCA objective is to weight the importance of fitting each element in the matrix A. In the generalized low rank model, we let $L_{ij}(u-a) = w_{ij}(a-u)^2$, where w_{ij} is a weight, and take $r = \tilde{r} = 0$. Unlike PCA, the weighted PCA problem has no known analytical solution [134]. In fact, it is NP-hard to find an exact solution to weighted PCA [50], although it is not known whether it is always possible to find approximate solutions of moderate accuracy efficiently.

Robust PCA. Despite its widespread use, PCA is very sensitive to outliers. Many authors have proposed a robust version of PCA obtained by replacing least-squares loss with ℓ_1 loss, which is less sensitive to large outliers [21, 156, 157]. They propose to solve the problem

$$\begin{array}{ll} \text{minimize} & \|S\|_1 + \|Z\|_* \\ \text{subject to} & S + Z = A. \end{array} \tag{4.2}$$

The authors interpret Z as a robust version of the principal components of the data matrix A, and S as the sparse, possibly large noise corrupting the observations.

We can frame robust PCA as a GLRM in the following way. If $L_{ij}(u,a) = |a-u|$, and $r(x) = \frac{\gamma}{2}\|x\|_2^2$, $\tilde{r}(y) = \frac{\gamma}{2}\|y\|_2^2$, then (4.1) becomes

$$\text{minimize} \quad \|A - XY\|_1 + \frac{\gamma}{2}\|X\|_F^2 + \frac{\gamma}{2}\|Y\|_F^2.$$

Using the arguments in §7.6, we can rewrite the problem by introducing a new variable $Z = XY$ as

$$\begin{array}{ll} \text{minimize} & \|A - Z\|_1 + \gamma\|Z\|_* \\ \text{subject to} & \operatorname{Rank}(Z) \leq k. \end{array}$$

This results in a rank-constrained version of the estimator proposed in the literature on robust PCA [156, 21, 157]:

$$\begin{array}{ll} \text{minimize} & \|S\|_1 + \gamma\|Z\|_* \\ \text{subject to} & S + Z = A \\ & \operatorname{Rank}(Z) \leq k, \end{array}$$

where we have introduced the new variable $S = A - Z$.

Huber PCA. The Huber function is defined as

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

Using Huber loss,

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

in place of ℓ_1 loss also yields an estimator robust to occasionally large outliers [65]. 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. This choice of loss function results in a generalized low rank model formulation that is robust both to large outliers and to small Gaussian perturbations in the data.

Previously, the problem of Gaussian noise in robust PCA has been treated by decomposing the matrix $A = L + S + N$ into a low rank matrix L, a sparse matrix S, and a matrix with small Gaussian entries N by minimizing the loss

$$\|L\|_* + \|S\|_1 + (1/2)\|N\|_F^2$$

over all decompositions $A = L + S + N$ of A [157].

In fact, this formulation is equivalent to Huber PCA with quadratic regularization on the factors X and Y. The argument showing this is very similar to the one we made above for robust PCA. The only added ingredient is the observation that

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

In other words, the Huber function is the infimal convolution of the negative log likelihood of a Gaussian random variable and a Laplacian random variable: it represents the most likely assignment of (additive) blame for the error x to a Gaussian error n and a Laplacian error s.

Robust regularized PCA. We can design robust versions of all the regularized PCA problems above by the same transformation we used to design robust PCA. Simply replace the quadratic loss function with an ℓ_1 or Huber loss function. For example, k-mediods [71, 110] is obtained by using ℓ_1 loss in place of quadratic loss in the quadratic clustering problem. Similarly, robust subspace clustering [132] can be obtained by using an ℓ_1 or Huber penalty in the subspace clustering problem.

Quantile PCA. For some applications, it can be much worse to *overestimate* the entries of A than to *underestimate* them, or vice versa. One can capture this asymmetry by using the loss function

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

and choosing $\alpha \in (0, 1)$ appropriately. This loss function is sometimes called a *scalene* loss, and can be interpreted as performing *quantile regression*, *e.g.*, fitting the 20th percentile [83, 82].

Fractional PCA. For other applications, we may be interested in finding an approximation of the matrix A whose entries are close to the original matrix on a relative, rather than an absolute, scale. Here, we assume the entries A_{ij} are all positive. The loss function

$$L(u, a) = \max\left(\frac{a - u}{u}, \frac{u - a}{a}\right)$$

can capture this objective. A model (X, Y) with objective value less than $0.10mn$ gives a low rank matrix XY that is on average within 10% of the original matrix.

Logarithmic PCA. Logarithmic loss functions may also be useful for finding an approximation of A that is close on a relative, rather than absolute, scale. Once again, we assume all entries of A are positive. Define the logarithmic loss

$$L(u, a) = \log^2(u/a).$$

This loss is *not* convex, but has the nice property that it fits the *geometric mean* of the data:

$$\operatorname*{argmin}_u \sum_i L(u, a_i) = (\prod_i a_i)^{1/n}.$$

To see this, note that we are solving a least squares problem in log space. At the solution, $\log(u)$ will be the mean of $\log(a_i)$, *i.e.*,

$$\log(u) = 1/n \sum_i \log(a_i) = \log\left((\prod_i a_i)^{1/n}\right).$$

Exponential family PCA. It is easy to formulate a version of PCA corresponding to any loss in the exponential family. Here we give some interesting loss functions generated by exponential families when all the entries A_{ij} are positive. (See [29] for a general treatment of exponential family PCA.) One popular loss function in the exponential family is the KL-divergence loss,

$$L(u, a) = a \log\left(\frac{a}{u}\right) - a + u,$$

which corresponds to a Poisson generative model [29].

Another interesting loss function is the Itakura-Saito (IS) loss,

$$L(u, a) = \log\left(\frac{a}{u}\right) - 1 + \frac{a}{u},$$

which has the property that it is scale invariant, so scaling a and u by the same factor produces the same loss [139]. The IS loss corresponds to *Tweedie* distributions (*i.e.*, distributions for which the variance is some power of the mean) [147]. This makes it interesting in applications, such as audio processing, where fractional errors in recovery are perceived.

The β-divergence,

$$L(u,a) = \frac{a^\beta}{\beta(\beta-1)} + \frac{u^\beta}{\beta} - \frac{au^{\beta-1}}{\beta-1},$$

generalizes both of these losses. With $\beta = 2$, we recover quadratic loss; in the limit as $\beta \to 1$, we recover the KL-divergence loss; and in the limit as $\beta \to 0$, we recover the IS loss [139].

4.3 Offsets and scaling

In §2.6, we saw how to use standardization to rescale the data in order to compensate for unequal scaling in different features. In general, standardization destroys sparsity in the data by subtracting the (column) means (which are in general non-zero) from each element of the data matrix A. It is possible to instead rescale the *loss functions* in order to compensate for unequal scaling. Scaling the loss functions instead has the advantage that no arithmetic is performed directly on the data A, so sparsity in A is preserved.

A savvy user may be able to select loss functions L_{ij} that are scaled to reflect the importance of fitting different columns. However, it is useful to have a default automatic scaling for times when no savvy user can be found. The scaling proposed here generalizes the idea of standardization to a setting with heterogeneous loss functions.

Given initial loss functions L_{ij}, which we assume are nonnegative, for each feature j let

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

It is easy to see that μ_j generalizes the mean of column j, while σ_j^2 generalizes the column variance. For example, when $L_{ij}(u,a) = (u-a)^2$ for every $i = 1,\ldots,m$, $j = 1,\ldots,n$, μ_j is the mean and σ_j^2 is the sample variance of the jth column of A. When $L_{ij}(u,a) = |u-a|$ for every $i = 1,\ldots,m$, $j = 1,\ldots,n$, 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 by σ_j^2 and solve

$$\text{minimize} \quad \textstyle\sum_{(i,j)\in\Omega} L_{ij}(A_{ij}, x_i y_j + \mu_j)/\sigma_j^2 + \sum_{i=1}^m r_i(x_i) + \sum_{j=1}^n \tilde{r}_j(y_j). \tag{4.3}$$

Note that this problem can be recast in the standard form for a generalized low rank model (4.1). For the offset, we may use the same trick described in §3.3 to encode the offset in the regularization; and for the scaling, we simply replace the original loss function L_{ij} by L_{ij}/σ_j^2.

5

Loss functions for abstract data types

We began our study of generalized low rank modeling by considering the best way to approximate a matrix by another matrix of lower rank. In this section, we apply the same procedure to approximate a data table that may not consist of real numbers, by choosing a loss function that respects the data type.

We now consider A to be a *table* consisting of m examples (*i.e.*, rows, samples) and n features (*i.e.*, columns, attributes), with each entry A_{ij} drawn from a feature set $\mathcal{F}_j$. The feature set $\mathcal{F}_j$ may be discrete or continuous. So far, we have only considered numerical data ($\mathcal{F}_j = \mathbf{R}$ for $j = 1, \ldots, n$), but now $\mathcal{F}_j$ can represent more abstract data types. For example, entries of A can take on 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 are given a loss function $L_{ij} : \mathbf{R} \times \mathcal{F}_j \to \mathbf{R}$. The loss $L_{ij}(u, a)$ describes the approximation error incurred when we represent a feature value $a \in \mathcal{F}_j$ by the number $u \in \mathbf{R}$. We give a number of examples of these loss functions below.

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

$$\text{minimize} \quad \textstyle\sum_{(i,j)\in\Omega} L_{ij}(x_i y_j, A_{ij}) + \sum_{i=1}^m r_i(x_i) + \sum_{j=1}^n \tilde{r}_j(y_j), \quad (5.1)$$

with variables $X \in \mathbf{R}^{n\times k}$ and $Y \in \mathbf{R}^{k\times m}$, and with loss L_{ij} as above and regularizers $r_i(x_i) : \mathbf{R}^{1\times k} \to \mathbf{R}$ and $\tilde{r}_j(y_j) : \mathbf{R}^{k\times 1} \to \mathbf{R}$ (as before). When the domain of each loss function is $\mathbf{R} \times \mathbf{R}$, we recover the generalized low rank model on a matrix (4.1).

5.1 Solution methods

As before, this problem is not convex, but it is bi-convex if r_i, and $\tilde{r}_j$ are convex, and L_{ij} is convex in its first argument. The problem is also separable across samples $i = 1, \ldots, m$ and features $j = 1, \ldots, m$. These properties makes it easy to perform alternating minimization on this objective. Once again, we defer a discussion of how to solve these subproblems explicitly to §7.

5.2 Examples

Boolean PCA. Suppose $A_{ij} \in \{-1, 1\}^{m\times n}$, and we wish to approximate this Boolean matrix. For example, we might suppose that the entries of A are generated as noisy, 1-bit observations from an underlying low rank matrix XY. Surprisingly, it is possible to accurately estimate the underlying matrix with only a few observations $|\Omega|$ from the matrix by solving problem (5.1) (under a few mild technical conditions) with an appropriate loss function [34].

We may take the loss to be

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

which is the hinge loss (see Figure 5.1), and solve the problem (5.1) with or without regularization. When the regularization is sum of squares ($r(x) = \lambda\|x\|_2^2$, $\tilde{r}(y) = \lambda\|y\|_2^2$), fixing X and minimizing over y_j is equivalent to training a support vector machine (SVM) on a data set

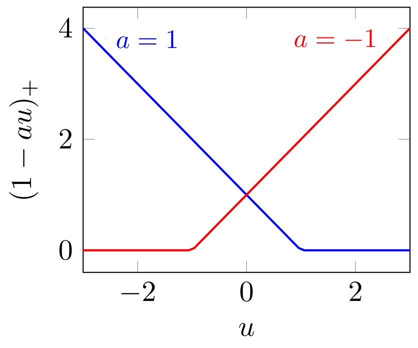


Figure 5.1: Hinge loss.

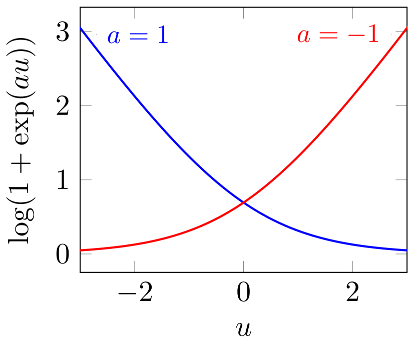


Figure 5.2: Logistic loss.

consisting of m examples with features x_i and labels A_{ij}. Hence alternating minimization for the problem (4.1) reduces to repeatedly training an SVM. This model has been previously considered under the name Maximum Margin Matrix Factorization (MMMF) [135, 120].

Logistic PCA. Again supposing $A_{ij} \in \{-1,1\}^{m\times n}$, we can also use a logistic loss to measure the approximation quality. Let

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

(see Figure 5.2). With this loss, fixing X and minimizing over y_j is equivalent to using logistic regression to predict the labels A_{ij}. This model has been previously considered under the name logistic PCA [124].

Poisson PCA. Now suppose the data A_{ij} are nonnegative integers. We can use any loss function that might be used in a regression framework to predict integral data to construct a generalized low rank model for Poisson PCA. For example, we can take

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

This is the exponential family loss corresponding to Poisson data. (It differs from the KL-divergence loss from §4.2 only in that u has been replaced by $\exp(u)$, which allows u to take negative values.)

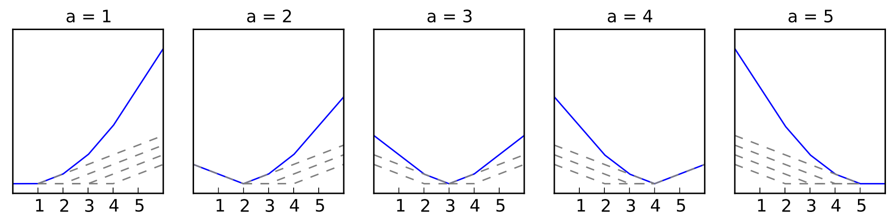


Figure 5.3: Ordinal hinge loss.

Ordinal PCA. Suppose the data A_{ij} records the levels of some ordinal variable, encoded as $\{1, 2, \ldots, d\}$. We wish to penalize the entries of the low rank matrix XY which deviate by many levels from the encoded ordinal value. A convex version of this penalty is given by the ordinal hinge loss,

$$L(u, a) = \sum_{a'=1}^{a-1} (1 - u + a')_+ + \sum_{a'=a+1}^{d} (1 + u - a')_+, \qquad (5.2)$$

which generalizes the hinge loss to ordinal data (see Figure 5.3).

This loss function may be useful for encoding Likert-scale data indicating degrees of agreement with a question [90]. For example, we might have

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

We can encode these levels as the integers $1, \ldots, 5$ and use the above loss to fit a model to ordinal data.

This approach assumes that every increment of error is equally bad: for example, that approximating "agree" by "strongly disagree" is just as bad as approximating "neither agree nor disagree" by "agree". In §6.1 we introduce a more flexible ordinal loss function that can learn a more flexible relationship between ordinal labels. For example, it could determine that the difference between "agree" and "strongly disagree" is smaller than the difference between "neither agree nor disagree" and "agree".

Interval PCA. Suppose that the data $A_{ij} \in \mathbf{R}^2$ are tuples denoting the endpoints of an interval, and we wish to find a low rank matrix whose entries lie inside these intervals. We can capture this objective using, for example, the deadzone-linear loss

$$L(u, a) = \max((a_1 - u)_+, (u - a_2)_+).$$

5.3 Missing data and data imputation

We can use the solution (X, Y) to a low rank model to impute values corresponding to missing data $(i, j) \notin \Omega$. This process is sometimes also called *inference*. Above, we saw that for quadratically regularized PCA, the MAP estimator for the missing entry A_{ij} is 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 (4.1) 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 $x_i y_j$:

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

This implicitly constrains $\hat{A}_{ij}$ to lie in the domain $\mathcal{F}_j$ of L_{ij}. When $L_{ij} : \mathbf{R} \times \mathbf{R} \to \mathbf{R}$, as is the case for the losses in §4 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 $\operatorname{argmin}_a L_{ij}(u, a)$ will not in general be equal to u.

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

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

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

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

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

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

5.4 Interpretations and applications

We have already discussed some interpretations of X and Y in the PCA setting. Now we reconsider those interpretations in the context of approximating these abstract data types.

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

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

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

Archetypical representations. As before, we call x_i the *representation* of example i in terms of the archetypes. The rows of X give an embedding of the examples into $\mathbf{R}^k$, where each coordinate axis corresponds to a different archetype. If the archetypes are simple to understand or interpret, then the representation of an example can provide better intuition about that example.

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

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

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

Even more interesting, it allows us to compare features of different types. We could say that the real-valued feature i is similar to Likert-valued question j if $\|y_i - y_j\|$ is small.

Latent variables. Each row of X represents an example by a vector in $\mathbf{R}^k$. The matrix Y maps these representations back into the original feature space (now nonlinearly) as described in the discussion on data imputation in §5.3. We might think of X as discovering the *latent variables* that best explain the observed data, with the added benefit that these latent variables lie in the vector space $\mathbf{R}^k$. If the approximation error $\sum_{(i,j)\in\Omega} L_{ij}(x_i y_j, A_{ij})$ is small, then we view these latent variables as providing a good explanation or summary of the full data set.

Probabilistic interpretation. We can give a probabilistic interpretation of X and Y, generalizing the hierarchical Bayesian model presented by Fithian and Mazumder in [48]. We suppose that the matrices $\bar{X}$ and $\bar{Y}$ are generated according to a probability distribution with probability proportional to $\exp(-r(\bar{X}))$ and $\exp(-\tilde{r}(\bar{Y}))$, respectively. Our observations A of the entries in the matrix $\bar{Z} = \bar{X}\bar{Y}$ are given by

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

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

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

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

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

which is equivalent, by taking logs, to problem (5.1).

This interpretation gives us a simple way to interpret our procedure for imputing missing observations $(i, j) \notin \Omega$. We are simply computing the MAP estimator $\hat{A}_{ij}$.

Auto-encoder. The matrix X encodes the data; the matrix Y decodes it back into the full space. We can view (5.1) as providing the best linear auto-encoder for the data. Among all linear encodings (X) and decodings (Y) of the data, the abstract generalized low rank model (5.1) minimizes the reconstruction error measured according to the loss functions L_{ij}.

Compression. We impose an information bottleneck by using a low rank auto-encoder to fit the data. The bottleneck is imposed by both the dimensionality reduction and the regularization, giving both soft and hard constraints on the information content allowed. The solution (X, Y) to problem (5.1) maximizes the information transmitted through this k-dimensional bottleneck, measured according to the loss functions L_{ij}. This X and Y give a compressed and real-valued representation that may be used to more efficiently store or transmit the information present in the data.

5.5 Offsets and scaling

Just as in the previous section, better practical performance can often be achieved by allowing an offset in the model as described in §3.3, and automatic scaling of loss functions as described in §4.3. As we noted in §4.3, scaling the loss functions (instead of standardizing the data) has the advantage that no arithmetic is performed directly on the data A. When the data A consists of abstract types, it is quite important that no arithmetic is performed on the data, so that we need not take the average of, say, "very much" and "a little", or subtract it from "not at all".

5.6 Numerical examples

In this section we give results of some small experiments illustrating the use of different loss functions adapted to abstract data types, and comparing their performance to quadratically regularized PCA. To fit these GLRMs, we use alternating minimization and solve the subproblems with subgradient descent. This approach is explained more fully in §7. Running the alternating subgradient method multiple times on the same GLRM from different initial conditions yields different models, all with very similar (but not identical) objective values.

Boolean PCA. For this experiment, we generate Boolean data $A \in \{-1,+1\}^{n\times m}$ as

$$A = \mathbf{sign}\left(X^{\text{true}} Y^{\text{true}}\right),$$

where $X^{\text{true}} \in \mathbf{R}^{n\times k_{\text{true}}}$ and $Y^{\text{true}} \in \mathbf{R}^{k_{\text{true}}\times m}$ have independent, standard normal entries. We consider a problem instance with $m = 50$, $n = 50$, and $k_{\text{true}} = k = 10$.

We fit two GLRMs to this data to compare their performance. Boolean PCA uses hinge loss $L(u,a) = \max(1 - au, 0)$ and quadratic regularization $r(u) = \tilde{r}(u) = .1\|u\|_2^2$, and produces the model $(X^{\text{bool}}, Y^{\text{bool}})$. Quadratically regularized PCA uses squared loss $L(u,a) = (u-a)^2$ and the same quadratic regularization, and produces the model $(X^{\text{real}}, Y^{\text{real}})$.

Figure 5.4 shows the results of fitting Boolean PCA to this data. The first column shows the original ground-truth data A; the second shows the imputed data given the model, $\hat{A}^{\text{bool}}$, generated by rounding the entries of $X^{\text{bool}}Y^{\text{bool}}$ to the closest number in $0, 1$ (as explained in §5.3); the third shows the error $A - \hat{A}^{\text{bool}}$. Figure 5.4 shows the results of running quadratically regularized PCA on the same data, and shows A, $\hat{A}^{\text{real}}$, and $A - \hat{A}^{\text{real}}$.

As expected, Boolean PCA performs substantially better than quadratically regularized PCA on this data set. Define the misclassification error (percentage of misclassified entries)

$$\epsilon(X, Y; A) = \frac{\#\{(i,j) \mid A_{ij} \neq \textbf{sign}\,(XY)_{ij}\}}{mn}. \tag{5.3}$$

On average over 100 draws from the ground truth data distribution, the misclassification error is much lower using hinge loss ($\epsilon(X^{\text{bool}}, Y^{\text{bool}}; A) = 0.0016$) than squared loss ($\epsilon(X^{\text{real}}, Y^{\text{real}}; A) = 0.0051$). The average RMS errors

$$\text{RMS}(X, Y; A) = \left(\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n}(A_{ij} - (XY)_{ij})^2\right)^{1/2}$$

using hinge loss ($\text{RMS}(X^{\text{bool}}, Y^{\text{bool}}; A) = 0.0816$) and squared loss ($\text{RMS}(X^{\text{real}}, Y^{\text{real}}; A) = 0.159$) also indicate an advantage for Boolean PCA.

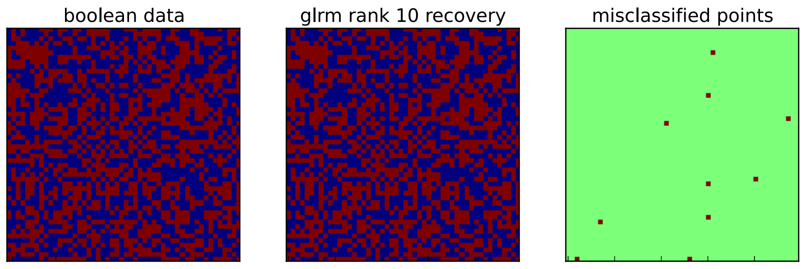


Figure 5.4: Boolean PCA on Boolean data.

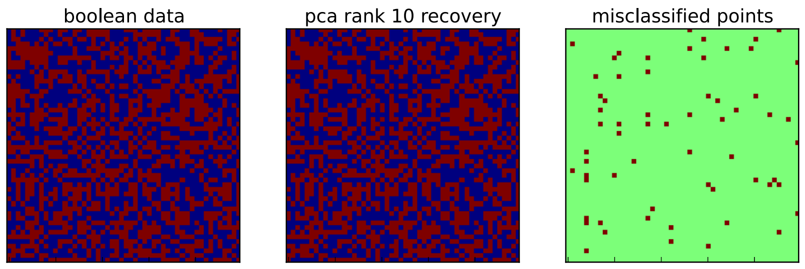


Figure 5.5: Quadratically regularized PCA on Boolean data.

Censored PCA. In this example, we consider the performance of Boolean PCA when only a subset of positive entries in the Boolean matrix $A \in \{-1, 1\}^{m \times n}$ have been observed, *i.e.*, the data has been *censored*. For example, a retailer might know only a subset of the products each customer purchased; or a medical clinic might know only a subset of the diseases a patient has contracted, or of the drugs the patient has taken. Imputation can be used in this setting to (attempt to) distinguish true negatives $A_{ij} = -1$ from unobserved positives $A_{ij} = +1$, $(i, j) \notin \Omega$.

We generate a low rank matrix $B = XY \in [0, 1]^{m \times n}$ with $X \in \mathbf{R}^{m \times k}$, $Y \in \mathbf{R}^{k \times n}$, where the entries of X and Y are drawn from a uniform distribution on $[0, 1]$, $m = n = 300$ and $k = 3$. Our data matrix A is chosen by letting $A_{ij} = 1$ with probability proportional to B_{ij}, and -1 otherwise; the constant of proportionality is chosen so that half of the entries in A are positive. We fit a rank 5 GLRM to an observation set Ω consisting of 10% of the positive entries in the matrix, drawn uniformly at random, using hinge loss and quadratic regularization. (Note that the rank of the model is higher than the (unobserved) true rank of the data; we will see below in §8.2 how to choose the right rank for a model.) That is, we fit the low rank model

$$\text{minimize} \quad \sum_{(i,j)\in\Omega} \max(1 - x_i y_j A_{ij}, 0) + \gamma \sum_{i=1}^m \|x_i\|_2^2 + \gamma \sum_{j=1}^n \|y_j\|_2^2$$

and vary the regularization parameter γ.

We consider three error metrics to measure the performance of the fitted model (X, Y): normalized training error,

$$\frac{1}{|\Omega|} \sum_{(i,j)\in\Omega} \max(1 - A_{ij} x_i y_j, 0),$$

normalized test error,

$$\frac{1}{|\Omega^C|} \sum_{(i,j)\in\Omega^C} \max(1 - A_{ij} x_i y_j, 0),$$

and *precision at 10* (p@10), which is computed as the fraction of the top ten predicted values not in the observation set, $\{x_i y_j : (i,j) \in \Omega^C\}$, for which $A_{ij} = 1$. (Here, $\Omega^C = \{1, \ldots, m\} \times \{1, \ldots, n\} \setminus \Omega$.) Precision at 10 measures the usefulness of the model: if we predicted that the top 10 unseen elements (i,j) had values $+1$, how many would we get right?

Figure 5.6 shows the regularization path as γ ranges from 0 to 40, averaged over 50 samples from the distribution generating the data. Here, we see that while the training error decreases as γ decreases, the test error reaches a minimum around $\gamma = 5$. Interestingly, the precision at 10 improves as the regularization increases; since precision at 10 is computed using only relative rather than absolute values of the model, it is insensitive to the shrinkage of the parameters introduced by the regularization. The grey line shows the probability of identifying a positive entry by guessing randomly; precision at 10, which exceeds 80% when $\gamma \gtrsim 30$, is significantly higher. This performance is particularly impressive given that the observations Ω are generated by sampling from rather than rounding the auxiliary matrix B.

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

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

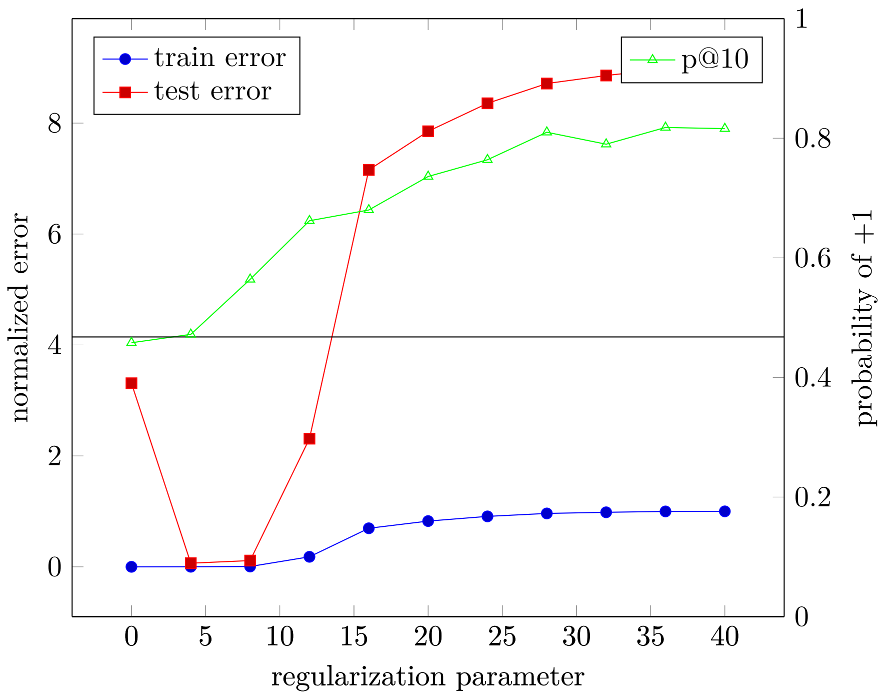


Figure 5.6: Error metrics for Boolean GLRM on censored data. The grey line shows the probability that a random guess identifies a positive entry.

where the function **round** maps a to the nearest integer in the set $\{1, \ldots, 7\}$. Thus, $\mathcal{N}_1$ corresponds to real-valued data; $\mathcal{N}_2$ corresponds to Boolean data; and $\mathcal{N}_3$ corresponds to ordinal data. We consider a problem instance in which $m = 100$, $n_1 = 40$, $n_2 = 30$, $n_3 = 30$, and $k_{\text{true}} = k = 10$.

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

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

where $L_{\text{real}}(u,a) = (u-a)^2$, $L_{\text{bool}}(u,a) = (1-au)_+$, and $L_{\text{ord}}(u,a)$ is defined in (5.2), and with quadratic regularization $r(u) = \tilde{r}(u) = .1\|u\|_2^2$. We fit the GLRM to produce the model $(X^{\text{mix}}, Y^{\text{mix}})$. For comparison, we also fit quadratically regularized PCA to the same data, using $L_{ij}(u,a) = (u-a)^2$ for all j and quadratic regularization $r(u) = \tilde{r}(u) = .1\|u\|_2^2$, to produce the model $(X^{\text{real}}, Y^{\text{real}})$.

Figure 5.7 compares the performance of the heterogeneous loss GLRM to quadratically regularized PCA fit to the same data. Panel 5.7a shows the results of fitting the heterogeneous loss GLRM above. Panel 5.7b shows the results of fitting quadratically regularized PCA. The first column shows the original ground-truth data A; the second shows the imputed data given the model, $\hat{A}^{\text{mix}}$, generated by rounding the entries of $X^{\text{mix}}Y^{\text{mix}}$ to the closest number in $0,1$ (as explained in §5.3); the third shows the error $A - \hat{A}^{\text{mix}}$.

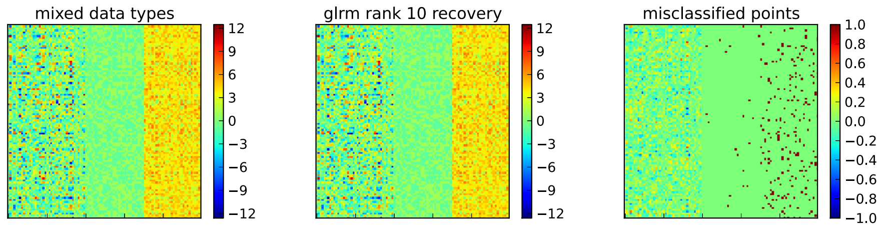
To evaluate error for Boolean and ordinal data, we use the misclassification error ϵ (5.3) defined above. For notational convenience, we let $Y_{\mathcal{N}_l}$ $(A_{\mathcal{N}_l})$ denote Y (A) restricted to the columns $\mathcal{N}_l$ in order to pick out real-valued columns $(l=1)$, Boolean columns $(l=2)$, and ordinal columns $(l=3)$.

Table 5.1 compare the average error (difference between imputed entries and ground truth) over 100 draws from the ground truth distribution for models using heterogeneous loss (X^{mix}, Y^{mix}) and quadratically regularized loss (X^{real}, Y^{real}). Columns are labeled by error metric. We use misclassification error ϵ (defined in (5.3)) for Boolean and ordinal data and MSE for numerical data.

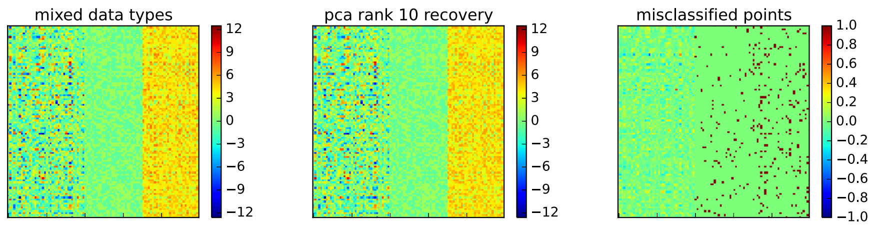
	$\text{MSE}(X, Y_{\mathcal{N}_1}; A_{\mathcal{N}_1})$	$\epsilon(X, Y_{\mathcal{N}_2}; A_{\mathcal{N}_2})$	$\epsilon(X, Y_{\mathcal{N}_3}; A_{\mathcal{N}_3})$
X^{mix}, Y^{mix}	0.0224	0.0074	0.0531
X^{real} , Y^{real}	0.0076	0.0213	0.0618

Table 5.1: Average error for numerical, Boolean, and ordinal features using GLRM with heterogeneous loss and quadratically regularized loss.

Missing data. Here, we explore the effect of missing entries on the accuracy of the recovered model. We generate data A as detailed above, but then censor one large block of entries in the table (constituting 3.75% of numerical, 50% of Boolean, and 50% of ordinal data), removing them from the observed set Ω.



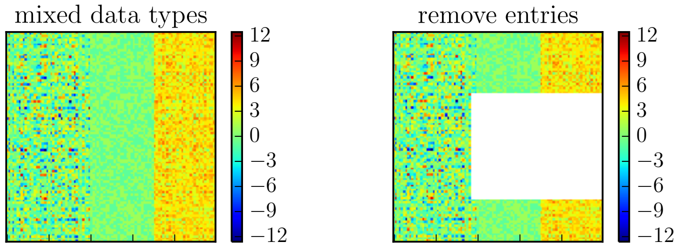
(a) Heterogeneous loss GLRM on mixed data.



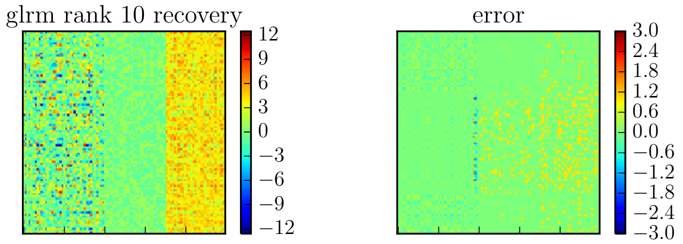
(b) Quadratically regularized PCA on mixed data.

Figure 5.7: Models for mixed data.

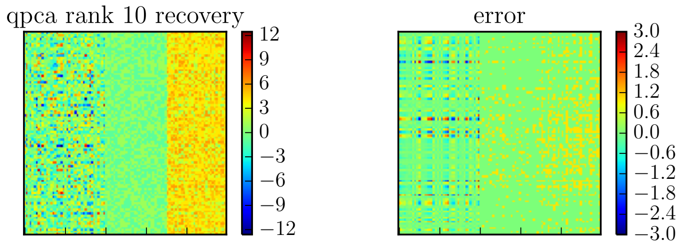
Figure 5.8 shows the results of fitting three different models with rank 10 to the censored data. Panel 5.8a shows the original ground-truth data A and the block of data that has been removed from the observation set Ω. Panel 5.8b shows the results of fitting the heterogeneous loss GLRM described above to the block-censored data: the first column shows the imputed data given the model, $\hat{A}^{\text{mix}}$, generated by rounding the entries of $X^{\text{mix}}Y^{\text{mix}}$ to the closest number in $\{0,1\}$ (as explained in §5.3), while the second column shows the error $A-\hat{A}^{\text{mix}}$. Two other models are provided for comparison. Panel 5.8c shows the imputed values $\hat{A}^{\text{real}}$ and error $A-\hat{A}^{\text{real}}$ obtained by running quadratically regularized PCA on the same data and with the same held-out block. Panel 5.8d shows the imputed values $\hat{A}^{\text{real}}$ and error $A-\hat{A}^{\text{real}}$ obtained by running (unregularized) PCA on the same data after replacing each missing entry with the column mean. While quadradically regularized PCA and the heterogeneous loss GLRM performed similarly when no data was missing, the heterogeneous loss GLRM performs better than quadradically regularized PCA when a large block of



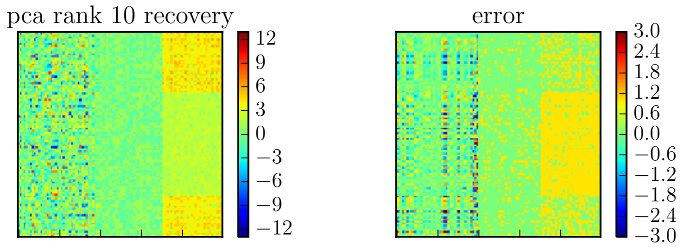
(a) Original data.



(b) Heterogeneous loss GLRM on missing data.



(c) Quadratically regularized PCA on missing data.



(d) PCA on missing data fit by replacing missing entries with column mean.

Figure 5.8: Three methods for imputing a block of heterogeneous missing data.

data is censored; interestingly, using maladapted (quadratic) loss functions for the Boolean and ordinal data results in a model that fits even the real valued data more poorly. The third (and all too common in practice) approach, which fills in missing data with the column mean and runs PCA, performs disastrously.

We compare the average error (difference between imputed entries and ground truth) over 100 draws from the ground truth distribution in Table 5.2. As above, we use misclassification error ϵ (defined in (5.3)) for Boolean and ordinal data and MSE for numerical data.

	$\mathrm{MSE}(X, Y_{\mathcal{N}_1}; A_{\mathcal{N}_1})$	$\epsilon(X, Y_{\mathcal{N}_2}; A_{\mathcal{N}_2})$	$\epsilon(X, Y_{\mathcal{N}_3}; A_{\mathcal{N}_3})$
X^{mix}, Y^{mix}	0.392	0.2968	0.3396
X^{real} , Y^{real}	0.561	0.4029	0.9418

Table 5.2: Average error over imputed data comparing two GLRMs: one using heterogeneous loss and one using regularized quadratic loss.

6

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 abstract data types more naturally. For example, we can now represent categorical values , permutations, distributions, and rankings.

We are given a loss function $L_{ij} : \mathbf{R}^{1\times d_j} \times \mathcal{F}_j \to \mathbf{R}$, where d_j is the *embedding dimension* of feature j, and $d = \sum_j d_j$ is the *embedding dimension* of the model. The loss $L_{ij}(u, a)$ 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 generalized low rank model on the database A,

$$\text{minimize} \quad \textstyle\sum_{(i,j)\in\Omega} L_{ij}(x_i Y_j, A_{ij}) + \sum_{i=1}^m r_i(x_i) + \sum_{j=1}^n \tilde{r}_j(Y_j), \tag{6.1}$$

with variables $X \in \mathbf{R}^{n\times k}$ and $Y \in \mathbf{R}^{k\times d}$, and with loss L_{ij} as above and regularizers $r_i(x_i) : \mathbf{R}^{1\times k} \to \mathbf{R}$ (as before) and $\tilde{r}_j(Y_j) : \mathbf{R}^{k\times d_j} \to \mathbf{R}$. Note that the first argument of L_{ij} is a *row* vector with d_j entries, and the first argument of r_j is a matrix with d_j columns. When every entry

A_{ij} is real-valued (*i.e.*, $d_j = 1$), then we recover the generalized low rank model (4.1) seen in the previous section.

6.1 Examples

Categorical PCA. Suppose that $a \in \mathcal{F}$ is a categorical variable, taking on one of d values or labels. Identify the labels with the integers $\{1, \ldots, d\}$. In (6.1), set

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

and use the quadratic regularizer $r_i = \gamma \| \cdot \|_2^2$, $\tilde{r} = \gamma \| \cdot \|_2^2$.

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 to the jth SVM. (This is sometimes called one-vs-all multiclass classification [123].) 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 above in 5.3, 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 $\operatorname{argmax}_l u_l$. 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 $[\min_l u_l, \max_l u_l]$ [17]. Hence the model will never impute label l' for any example.

We need not restrict ourselves to the loss function given above. 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 (6.1); 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 [40]; the Directed Acyclic Graph SVM [114]; the Crammer-Singer multi-class loss [30]; or the multi-category SVM [89].

Of these loss functions, only the one-vs-all loss is separable across the classes $a \in \mathcal{F}$. (By separable, we mean that the objective value can be written as a sum over the classes.) Hence fitting a categorical features with any other loss functions is *not* the same as fitting d Boolean features. For example, in the Crammer-Singer loss

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

the classes are combined according to their maximum, rather than their sum. While one-vs-all classification performs about as well as more sophisticated loss functions on small data sets [123], these more sophisticated nonseparable loss tend to perform much better as the number of classes (and examples) increases [56].

Some interesting nonconvex loss functions have also been suggested for this problem. For example, consider a generalization of Hamming distance to this setting,

$$L(u, a) = \delta(u_a \neq 1) + \sum_{a' \neq a} \delta(u_{a'} \neq -1),$$

where δ is a function that returns 1 if its argument is true and 0 otherwise. In this case, alternating minimization with regularization that enforces a clustered structure in the low rank model (see the discussion of quadratic clustering in §3.2) reproduces the k-modes algorithm [64].

Ordinal PCA. We saw in §5 one way to fit a GLRM to ordinal data. 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 $\{1, \ldots, d\}$.

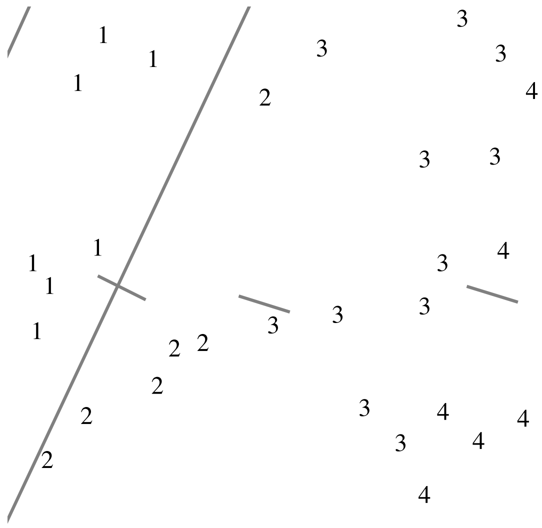


Figure 6.1: Multi-dimensional ordinal loss. Fitting a GLRM with this loss function simultaneously finds the best locations x_i for each ordinal observation (here shown as the numbers 1–4), and the best hyperplanes (here shown as grey lines) to separate each level from the next. The perpendicular segment on each line shows (as a vector) the column of Y corresponding to that hyperplane.

One approach we can use for (multi-dimensional) ordinal PCA is to solve (6.1) with the loss function

$$L(u, a) = \sum_{a'=1}^{d-1} (1 - \mathbf{I}_{a>a'} u_{a'})_+, \tag{6.2}$$

and with quadratic regularization. Here, the embedding dimension is $d-1$, so $u \in \mathbf{R}^{d-1}$. This approach fits 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 X and optimizing over Y is equivalent to training an SVM to separate labels $a \leq l$ from $a > l$ for each $l \in \mathcal{F}$. 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 6.1 for an illustration of an optimal fit of this loss function, with $k = 2$, to a simple synthetic data set.)

Permutation PCA. Suppose that a is a permutation of the numbers $1, \ldots, d$. Define the permutation loss

$$L(u, a) = \sum_{i=1}^{d-1} (1 - u_{a_i} + u_{a_{i+1}})_+.$$

This loss is zero if $u_{a_i} > u_{a_{i+1}} + 1$ for $i = 1, \ldots, d-1$, and increases linearly when these inequalities are violated. Define $\mathbf{sort}(u)$ to return a permutation $\hat{a}$ of the indices $1, \ldots, d$ so that $u_{\hat{a}_i} \geq u_{\hat{a}_{i+1}}$ for $i = 1, \ldots, d-1$. It is easy to check that $\operatorname{argmin}_a L(u, a) = \mathbf{sort}(u)$. Hence using the permutation loss function in generalized PCA (6.1) finds a low rank approximation of a given table of permutations.

Ranking PCA. Many variants on the permutation PCA problem are possible. For example, in ranking PCA, we interpret the permutation as a ranking of the choices $1, \ldots, d$, and penalize deviations of many levels more strongly than deviations of only one level by choosing the loss

$$L(u, a) = \sum_{i=1}^{d-1} \sum_{j=i+1}^{d} (1 - u_{a_i} + u_{a_j})_+.$$

From here, it is easy to generalize to a setting in which the rankings are only partially observed. Suppose that we observe pairwise comparisons $a \subseteq \{1, \ldots, d\} \times \{1, \ldots, d\}$, where $(i, j) \in a$ means that choice i was ranked above choice j. Then a loss function penalizing deviations from these observed rankings is

$$L(u, a) = \sum_{(i,j) \in a} (1 - u_{a_i} + u_{a_j})_+.$$

Many other modifications to ranking loss functions have been proposed in the literature that interpolate between the the two first loss functions proposed above, or which prioritize correctly predicting the top ranked choices. These losses include the area under the curve loss [138], ordered weighted average of pairwise classification losses [148], the weighted approximate-rank pairwise loss [153], the k-order statistic loss [154], and the accuracy at the top loss [14].

6.2 Offsets and scaling

Just as in the previous section, better practical performance can often be achieved by allowing an offset in the model as described in §3.3, and scaling loss functions as described in §4.3.

6.3 Numerical examples

We fit a low rank model to the 2013 American Community Survey (ACS) to illustrate how to fit a low rank model to heterogeneous data.

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 6.1.

We fit a rank 10 model to this data using Huber loss for real valued data, hinge loss for Boolean data, ordinal hinge loss for ordinal data, one-vs-all categorical loss for categorical data, and regularization parameter $\gamma = .1$. We allow an offset in the model and scale the loss functions and regularization as described in §4.3.

In Table 6.2, 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.

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 6.1: 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 6.2: Most similar features in demography space.

7

Fitting low rank models

In this section, we discuss a number of algorithms that may be used to fit generalized low rank models. As noted earlier, it can be computationally hard to find the global optimum of a generalized low rank model. For example, it is NP-hard to compute an exact solution to k-means [43], nonnegative matrix factorization [149], and weighted PCA and matrix completion [50] all of which are special cases of low rank models.

In §7.1, we will examine a number of local optimization methods based on alternating minimization. Algorithms implementing lazy variants of alternating minimization, such as the alternating gradient, proximal gradient, or stochastic gradient algorithms, are faster per iteration than alternating minimization, although they may require more iterations for convergence. In numerical experiments, we notice that lazy variants often converge to points with a lower objective value: it seems that these lazy variants are less likely to be trapped at a saddle point than is alternating minimization. §7.4 explores the convergence of these algorithms in practice.

We then consider a few special cases in which we can show that alternating minimization converges to the global optimum in some sense:

for example, we will see convergence with high probability, approximately, and in retrospect. §7.5 discusses a few strategies for initializing these local optimization methods, with provable guarantees in special cases. §7.6 shows that for problems with convex loss functions and quadratic regularization, it is sometimes possible to certify global optimality of the resulting model.

7.1 Alternating minimization

We showed earlier how to use alternating minimization to find an (approximate) solution to a generalized low rank model. Algorithm (1) shows how to explicitly extend alternating minimization to a generalized low rank model (4.1) with observations Ω.

Algorithm 1

```
given $X^0$, $Y^0$
for $k = 1, 2, \ldots$ do
    for $i = 1, \ldots, M$ do
        $x_i^k = \mathrm{argmin}_x \left( \sum_{j:(i,j)\in\Omega} L_{ij}(x y_j^{k-1}, A_{ij}) + r_i(x) \right)$
    end for
    for $j = 1, \ldots, N$ do
        $y_j^k = \mathrm{argmin}_y \left( \sum_{i:(i,j)\in\Omega} L_{ij}(x_i^k y, A_{ij}) + \tilde{r}_j(y) \right)$
    end for
end for
```

Parallelization. Alternating minimization parallelizes naturally over examples and features. In Algorithm 1, the loops over $i = 1, \ldots, N$ and over $j = 1, \ldots, M$ may both be executed in parallel.

7.2 Early stopping

It is not very useful to spend a lot of effort optimizing over X before we have a good estimate for Y. If an iterative algorithm is used to compute the minimum over X, it may make sense to stop the optimization over X early before going on to update Y. In general, we may consider replacing the minimization over x and y above by any update rule that moves towards the minimum. This templated algorithm is presented as Algorithm 2. 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 2

```
given $X^0$, $Y^0$
for $t = 1, 2, \ldots$ do
    for $i = 1, \ldots, m$ do
        $x_i^t = \mathbf{update}_{L_{ij}, r_i}(x_i^{t-1}, Y^{t-1}, A)$
    end for
    for $j = 1, \ldots, n$ do
        $y_j^t = \mathbf{update}_{L_{ij}, \tilde{r}_j}(y_j^{(t-1)T}, X^{(t)T}, A^T)$
    end for
end for
```

We describe below a number of different update rules $\mathbf{update}_{L,r}$ by writing the X update. The Y update can be implemented similarly. (In fact, it can be implemented by substituting $\tilde{r}$ for r, switching the roles of X and Y, and transposing all matrix arguments.) All of the approaches outlined below can still be executed in parallel over examples (for the X update) and features (for the Y update).

Gradient method. For example, we might take just one gradient step on the objective. This method can be used as long as L, r, and $\tilde{r}$ do not take infinite values. (If any of these functions f is not differentiable, replace ∇f below by any subgradient of f [12, 18].)

We implement $\mathbf{update}_{L,r}$ as follows. Let

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

Then set

$$x_i^t = x_i^{t-1} - \alpha_t g,$$

for some step size α_t. For example, the step size rule $\alpha_t = 1/t$, which guarantees convergence to the globally optimal X if Y is fixed [12, 18]. A faster approach in practice might be to use a backtracking line search [105].

Proximal gradient method. If a function takes on the value ∞, it need not have a subgradient at that point, which limits the gradient update to cases where the regularizer and loss are (finite) real-valued. When the regularizer (but not the loss) takes on infinite values (say, to represent a hard constraint), we can use a proximal gradient method instead.

The proximal operator of a function f [109] is

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

If f is the indicator function of a set $\mathcal{C}$, the proximal operator of f is just (Euclidean) projection onto $\mathcal{C}$.

A proximal gradient update $\mathbf{update}_{L,r}$ is implemented as follows. Let

$$g = \sum_{j:(i,j)\in\Omega} \nabla L_{ij}(x_i^{t-1} y_j^{t-1}, A_{ij}) y_j^{t-1}.$$

Then set

$$x_i^t = \mathbf{prox}_{\alpha_t r_i}(x_i^{t-1} - \alpha_t g),$$

for some step size α_t. The step size rule $\alpha_t = 1/t$ guarantees convergence to the globally optimal X if Y is fixed, while using a fixed, but sufficiently small, step size α guarantees convergence to a small $O(\alpha)$ neighborhood around the optimum [8]. The technical condition required on the step size is that $\alpha_t < 1/L$, where L is the Lipshitz constant of the gradient of the objective function. Bolte et al. have shown that the iterates x_i^t and y_j^t produced by the proximal gradient update rule (which they call proximal alternating linearized minimization, or PALM) globally converge to a critical point of the objective function under very mild conditions on the loss functions and regularizers [11].

Prox-prox method. Letting $f_t(X) = \sum_{(i,j)\in\Omega} L_{ij}(x_i y_j^t, A_{ij})$, define the proximal-proximal (prox-prox) update

$$X^{t+1} = \mathbf{prox}_{\alpha_t r_i}(\mathbf{prox}_{\alpha_t f_t}(X^t)).$$

The prox-prox update is simply a proximal gradient step on the objective when f_t is replaced by its *Moreau envelope*,

$$M_{f_t}(X) = \inf_{X'} \left(f_t(X') + \|X - X'\|_F^2 \right).$$

(See [109] for details.) The Moreau envelope has the same minimizers as the original objective. Thus, just as the proximal gradient method repeatedly applied to X converges to global minimum of the objective if Y is fixed, the prox-prox method repeatedly applied to X also converges to global minimum of the objective if Y is fixed under the same conditions on the step size α_t, for any constant stepsize $\alpha \leq \|G\|_2^2$. (Here, $\|G\|_2 = \sup_{\|x\|_2 \leq 1} \|Gx\|_2$ is the operator norm of G.)

This update can also be seen as a single iteration of ADMM when the dual variable in ADMM is initialized to 0; see [16]. In the case of quadratic objectives, we will see below that the prox-prox update can be applied very efficiently, making iterated prox-prox, or ADMM, an effective means of computing the solution to the subproblems arising in alternating minimization.

Choosing a step size. 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 [8], for each row i, we seek a step size α_i 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 of the same order as the average gradient of the loss functions. In the numerical experiments reported here, we choose $\alpha = 1$.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 $\alpha_i = \alpha/n_i$. We take a gradient step on each row x_i using the step size α_i. Our procedure for choosing α_j is the same.

We then check whether the objective value for the row,

$$\sum_{j:(i,j)\in\Omega} L_j(x_i y_j, A_{ij}) + r_i(x_i),$$

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 proximal 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 observed entry (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.

Stochastic gradients. Instead of computing the full gradient of L with respect to x_i above, we can replace the gradient g in either the gradient or proximal gradient method by any *stochastic gradient* g, which is a vector that satisfies

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

A stochastic gradient can be computed by sampling j uniformly at random from among observed features of i, and setting $g = |\{j : (i,j) \in \Omega\}| \nabla L_{ij}(x_i y_j, A_{ij}) y_j$. More samples from $\{j : (i,j) \in \Omega\}$ can be used to compute a less noisy stochastic gradient.

7.3 Quadratic objectives

Here we describe how to efficiently implement the prox-prox update rule for quadratic objectives and arbitrary regularizers, extending the factorization caching technique introduced in §2.3. We assume here that the objective is given by

$$\|A - XY\|_F^2 + r(X) + \tilde{r}(Y).$$

We will concentrate here on the X update; as always, the Y update is exactly analogous.

As in the case of quadratic regularization, we first form the Gram matrix $G = YY^T$. Then the proximal gradient update for X is fast to evaluate:

$$\mathbf{prox}_{\alpha_k r}(X - \alpha_k(XG - 2AY^T)).$$

But we can also take advantage of the ease of inverting the Gram matrix G to design a faster algorithm using the prox-prox update than is possible with general loss functions. For a quadratic objective with Gram matrix $G = Y^T Y$, the prox-prox update takes the simple form

$$\mathbf{prox}_{\alpha_k r}((G + \frac{1}{\alpha_k} I)^{-1}(AY^T + \frac{1}{\alpha_k} X)).$$

As in §2.3, we can compute $(G + \frac{1}{\alpha_k} I)^{-1}(AY^T + \frac{1}{\alpha_k} X)$ in parallel by first caching the factorization of $(G + \frac{1}{\alpha_k} I)^{-1}$. Hence it is advantageous to repeat this update many times before updating Y, since most of the computational effort is in forming G and AY^T.

For example, in the case of nonnegative least squares, this update is just

$$\Pi_+((G + \frac{1}{\alpha_k} I)^{-1}(AY^T + \frac{1}{\alpha_k} X)),$$

where Π_+ projects its argument onto the nonnegative orthant.

7.4 Convergence

Alternating minimization need not converge to the same model (or the same objective value) when initialized at different starting points. Through examples, we explore this idea here. These examples are fit using the Julia implementation (presented in §9) of the alternating proximal gradient updates method. The timing results were obtained using a single core of a standard laptop computer.

Global convergence for quadratically regularized PCA. Figure 7.1 shows the convergence of the alternating proximal gradient update method on a quadratically regularized PCA problem with randomly generated, fully observed data $A = X^{\text{true}}Y^{\text{true}}$, where entries of X^{true} and Y^{true} are drawn from a standard normal distribution. We pick five different random initializations of X and Y with standard normal entries to generate five different convergence trajectories. Quadratically regularized PCA is a simple problem with an analytical solution

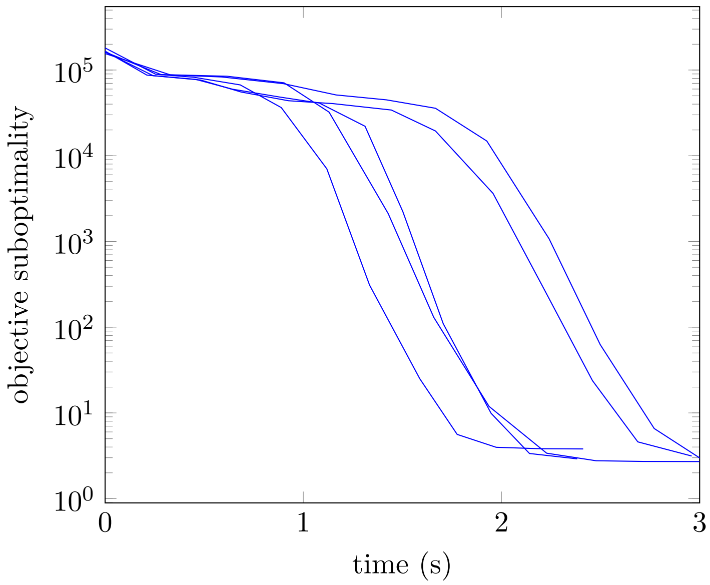


Figure 7.1: Convergence of alternating proximal gradient updates on quadratically regularized PCA for $n = m = 200$, $k = 2$.

(see §2), and with no local minimum that is not global (see Appendix A.1). Hence it should come as no surprise that the trajectories all converge to the same, globally optimal value.

Local convergence for nonnegative matrix factorization. Figure 7.2 shows convergence of the same algorithm on a nonnegative matrix factorization model, with data generated in the same way as in Figure 7.1. (Note that A has negative entries as well as positive entries, so the minimal objective value is strictly greater than zero.) Here, we plot the convergence of the objective value, rather than the suboptimality, since we cannot (efficiently, provably) compute the global minimum of the objective function even in the rank 1 case [51].

We see that the algorithm converges to a different optimal value (and point) depending on the initialization of X and Y. Three trajectories converge to the same optimal value (though one does so much

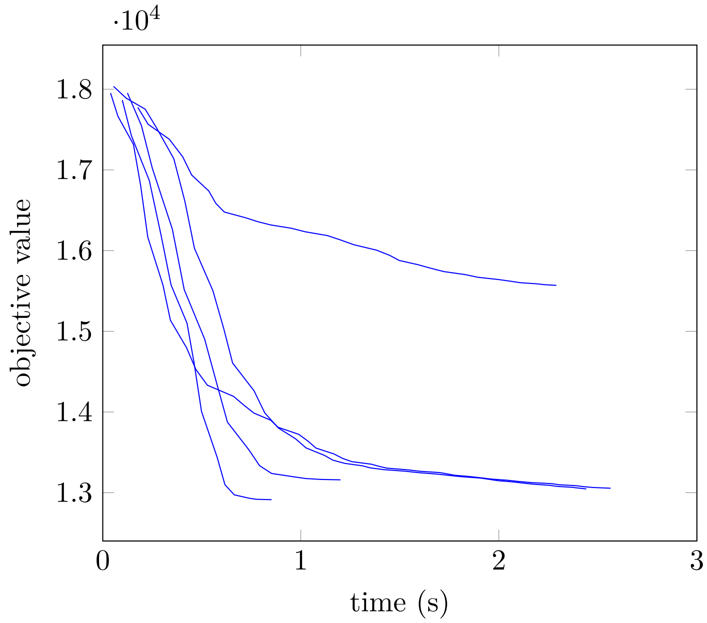


Figure 7.2: Convergence of alternating proximal gradient updates on NNMF for $n = m = 200$, $k = 2$.

faster than the others), one to a value that is somewhat better, and one to a value that is substantially worse.

7.5 Initialization

Above, we saw that alternating minimization can converge to models with optimal values that differ significantly. Here, we discuss two approaches to initialization that result in provably good solutions, for special cases of the generalized low rank problem. We then discuss how to apply these initialization schemes to more general models.

SVD. A literature that is by now extensive shows that the SVD provides a provably good initialization for the quadratic matrix completion problem (2.10) [74, 76, 73, 66, 58, 55]. Algorithms based on alternating minimization have been shown to converge quickly (even geometrically [66]) to a global solution satisfying a recovery guarantee when the initial values of X and Y are chosen carefully; see §2.4 for more details.

Here, we extend the SVD initialization previously proposed for matrix completion to one that works well for all PCA-like problems: problems with convex loss functions that have been scaled as in §4.3; with data A that consists of real values, Booleans, categoricals, and ordinals; and with quadratic (or no) regularization.

But we will need a matrix on which to perform the SVD. What matrix corresponds to our data table? Here, we give a simple proposal for how to construct such a matrix, motivated by [76, 66, 26]. 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 with mean zero and variance one from the data table, 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 re-interpret 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.

Now we assume that the *rows* of the data table are independent and identically distributed; our mission is to standardize the *columns*. The observed entries in column j have mean μ_j and variance σ_j^2,

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

so the matrix whose (i, j)th entry is $(A_{ij} - \mu_j)/\sigma_j$ for $(i, j) \in \Omega$ has columns whose observed entries have mean 0 and variance 1.

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.

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 $Y_{kj} = \mu_j$.

Finally, note that we need not compute the full SVD of $\tilde{A}$, but instead can simply compute the top k singular triples. For example, the randomized top k SVD algorithm proposed in [57] computes the top k singular triples of $\tilde{A}$ in time linear in $|\Omega|$, m, and n (and quadratic in k).

Figure 7.3 compares the convergence of this SVD-based initialization with random initialization on a low rank model for census data described in detail in §6.3. We initialize the algorithm at six different

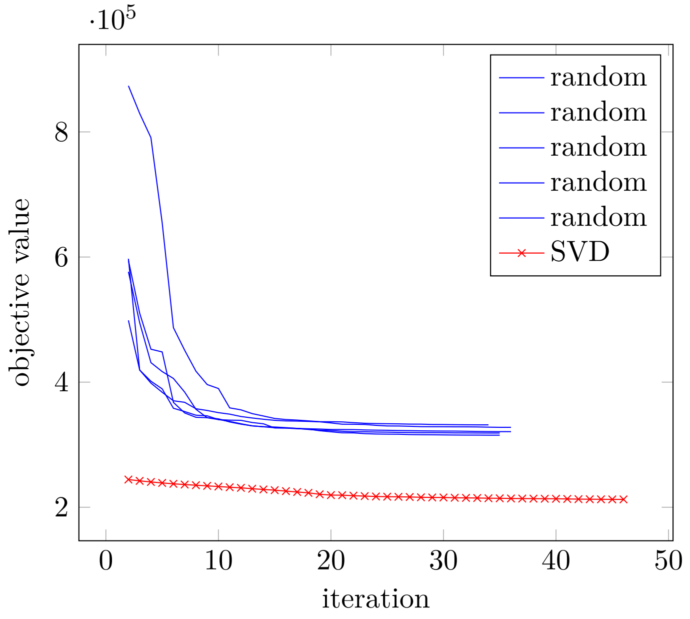


Figure 7.3: Convergence from five different random initializations, and from the SVD initialization.

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 behavior 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.

***k*-means++.** The k-means++ algorithm is an initialization scheme designed for quadratic clustering problems [5]. It consists of choosing an initial cluster centroid at random from the points, and then choosing the remaining $k-1$ centroids from the points x that have not yet been chosen with probability proportional to $D(x)^2$, where $D(x)$ is the minimum distance of x to any previously chosen centroid.

Quadratic clustering is known to be NP-hard, even with only two clusters ($k = 2$) [43]. However, k-means++ followed by alternating minimization gives a solution with expected approximation ratio within $O(\log k)$ of the optimal value [5]. (Here, the expectation is over the randomization in the initialization algorithm.) In contrast, an arbitrary initialization of the cluster centers for k-means can result in a solution whose value is arbitrarily worse than the true optimum.

A similar idea can be used for other low rank models. If the model rewards a solution that is spread out, as is the case in quadratic clustering or subspace clustering, it may be better to initialize the algorithm by choosing elements with probability proportional to a distance measure, as in k-means++. In the k-means++ procedure, one can use the loss function $L(u)$ as the distance metric D.

7.6 Global optimality

All generalized low rank models are non-convex, but some are more non-convex than others. In particular, for some problems, the only important source of non-convexity is the low rank constraint. For these problems, it is sometimes possible to certify global optimality of a model by considering an equivalent rank-constrained convex problem.

The arguments in this section are similar to ones found in [117], in which Recht et al. propose using a factored (nonconvex) formulation of the (convex) nuclear norm regularized estimator in order to efficiently solve the large-scale SDP arising in a matrix completion problem. However, the algorithm in [117] relies on a subroutine for finding a local minimum of an augmented Lagrangian which has the same biconvex form as problem (2.10). Finding a local minimum of this problem (rather than a saddle point) may be hard. In this section, we avoid the issue of finding a local minimum of the nonconvex problem; we consider instead whether it is possible to verify global optimality when presented with some putative solution.

The factored problem is equivalent to the rank constrained problem. Consider the *factored* problem

$$\text{minimize} \quad L(XY) + \tfrac{\gamma}{2}\|X\|_F^2 + \tfrac{\gamma}{2}\|Y\|_F^2, \tag{7.1}$$

with variables $X \in \mathbf{R}^{m\times k}$, $Y \in \mathbf{R}^{k\times n}$, where $L : \mathbf{R}^{m\times n} \to \mathbf{R}$ is any convex loss function. Compare this to the *rank-constrained* problem

$$\begin{array}{ll} \text{minimize} & L(Z) + \gamma\|Z\|_* \\ \text{subject to} & \operatorname{Rank}(Z) \le k, \end{array} \tag{7.2}$$

with variable $Z \in \mathbf{R}^{m\times n}$. Here, we use $\|\cdot\|_*$ to denote the nuclear norm, the sum of the singular values of a matrix.

Theorem 7.1. $(X^\star, Y^\star)$ is a solution to the factored problem (7.1) if and only if $Z^\star = X^\star Y^\star$ is a solution to the rank-constrained problem (7.2), and $\|X^\star\|_F^2 = \|Y^\star\|_F^2 = \frac{1}{2}\|Z^\star\|_*$.

We will need the following lemmas to understand the relation between the rank-constrained problem and the factored problem.

Lemma 7.2. Let $XY = U\Sigma V^T$ be the SVD of XY, where $\Sigma = \mathbf{diag}(\sigma)$. Then

$$\|\sigma\|_1 \le \frac{1}{2}(||X||_F^2 + ||Y||_F^2). \tag{7.3}$$

Proof. We may derive this fact as follows:

$$\begin{array}{rcl} \|\sigma\|_1 & = & \mathbf{tr}(U^T XYV) \\ & \le & \|U^T X\|_F \|YV\|_F \\ & \le & \|X\|_F \|Y\|_F \\ & \le & \dfrac{1}{2}(||X||_F^2 + ||Y||_F^2), \end{array}$$

where the first inequality above uses the Cauchy-Schwartz inequality, the second relies on the orthogonal invariance of the Frobenius norm, and the third follows from the basic inequality $ab \le \frac{1}{2}(a^2+b^2)$ for any real numbers a and b. □

The following result is well known.

Lemma 7.3. For any matrix Z, $\|Z\|_* = \inf_{XY=Z} \frac{1}{2}(||X||_F^2 + ||Y||_F^2)$.

Proof. Writing $Z = UDV^T$ and recalling the definition of the nuclear norm $\|Z\|_* = \|\sigma\|_1$, we see that Lemma 7.2 implies

$$\|Z\|_* \leq \inf_{XY=Z} \frac{1}{2}(||X||_F^2 + ||Y||_F^2).$$

But taking $X = U\Sigma^{1/2}$ and $Y = \Sigma^{1/2}V^T$, we have

$$\frac{1}{2}(||X||_F^2 + ||Y||_F^2) = \frac{1}{2}(\|\Sigma^{1/2}\|_F^2 + \|\Sigma^{1/2}\|_F^2) = \|\sigma\|_1,$$

(using once again the orthogonal invariance of the Frobenius norm), so the bound is satisfied with equality. □

Note that the infimum is achieved by $X = U\Sigma^{1/2}T$ and $Y = T^T\Sigma^{1/2}V^T$ for any orthonormal matrix T.

Theorem 7.1 follows as a corollary, since $L(Z) = L(XY)$ so long as $Z = XY$.

The rank constrained problem is sometimes equivalent to an unconstrained problem. Note that problem (7.2) is still a hard problem to solve: it is a rank-constrained semidefinite program. On the other hand, the same problem without the rank constraint is convex and tractable (though not easy to solve at scale). In particular, it is possible to write down an optimality condition for the problem

$$\text{minimize} \quad L(Z) + \gamma\|Z\|_* \tag{7.4}$$

that certifies that a matrix Z is globally optimal. This problem is a relaxation of problem (7.2), and so has an optimal value that is at least as small. Furthermore, if any solution to problem (7.4) has rank no more than k, then it is feasible for problem (7.2), so the optimal values of problem (7.4) and problem (7.2) must be the same. Hence any solution of problem (7.4) with rank no more than k also solves problem (7.2).

Recall that the matrix Z is a solution to problem (7.4) if and only if

$$0 \in \partial(L(Z) + \gamma\|Z\|_*),$$

where $\partial f(Z)$ is the *subgradient* of the function f at Z. The subgradient is a *set-valued* function.

The subgradient of the nuclear norm for $Z = U\Sigma V^T$ is easily seen to be

$$\partial\|Z\|_* = \{UV^T + W : U^T W = 0, WV = 0, \|W\|_2 \le 1\}.$$

Define the objective $\operatorname{obj}(Z) = L(Z) + \gamma\|Z\|_*$. Then, if $G \in \partial L(Z)$ and $UV^T + W \in \partial\|Z\|_*$, we can use the convexity of the objective to see that

$$\begin{array}{rcl} \operatorname{obj}(Z) & \ge & \operatorname{obj}(Z^\star) \ge \operatorname{obj}(Z) + \langle G + \gamma UV^T + \gamma W, Z^\star - Z\rangle \\ & \ge & \operatorname{obj}(Z) - \|G + \gamma UV^T + \gamma W\|_F \|Z^\star - Z\|_F, \end{array}$$

using the Cauchy–Schwarz inequality to obtain the last relation. Hence we might say that $\|G + \gamma UV^T + \gamma W\|_F$ bounds the (relative) suboptimality of the estimate Z. Furthermore, Z is optimal for problem (7.4) if and only if $0 \in \partial\operatorname{obj}(Z)$, which means that $G + \gamma UV^T + \gamma W = 0$ for some $G \in \partial L(Z)$ and $UV^T + W \in \partial\|Z\|_*$.

To find the tightest bound on the suboptimality of Z, we can minimize the bound over valid subgradients G and $UV^T + W$:

$$\begin{array}{ll} \text{minimize} & \|G + \gamma UV^T + \gamma W\|_F^2 \\ \text{subject to} & \|W\|_2 \le 1 \\ & U^T W = 0 \\ & WV = 0 \\ & G \in \partial L(Z). \end{array} \tag{7.5}$$

If the optimal value of this program is 0, then Z is optimal for problem (7.4).

This result allows us to (sometimes) certify global optimality of a particular model. Given a model (X, Y), we compute the SVD of the product $XY = U\Sigma V^T$. Solve (7.5). If the optimal value is 0, then (X, Y) is globally optimal.

If G is fixed, we can rewrite problem 7.5 by decomposing G into a sum of four parts: $G^{\|} = UU^T G V V^T$, $G^\perp = (I - UU^T)G(I - VV^T)$, and two parts that do not require names, $(I - UU^T)GVV^T$ and $UU^T G(I - VV^T)$. Noticing that the objective decomposes additively over the components of G, the optimal W is given by

$$W^\star = \frac{G^\perp}{\max(\gamma, \|G^\perp\|_2)}. \tag{7.6}$$

If $\|G + \gamma UV^T + \gamma W^\star\|_F^2 = 0$, then (X, Y) is globally optimal.

8

Choosing low rank models

8.1 Regularization paths

Suppose that we wish to understand the entire *regularization path* for a GLRM; that is, we would like to know the solution $(X(\gamma), Y(\gamma))$ to the problem

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

as a function of γ. Frequently, the regularization path may be computed almost as quickly as the solution for a single value of γ. We can achieve this by initially fitting the model with a very high value for γ, which is often a very easy problem. (For example, when r and $\tilde{r}$ are norms, the solution is $(X, Y) = (0, 0)$ for sufficiently large γ.) Then we may fit models corresponding to smaller and smaller values of γ by initializing the alternating minimization algorithm from our previous solution. This procedure is sometimes called a *homotopy method*.

For example, Figure 8.1 shows the regularization path for quadratically regularized Huber PCA on a synthetic data set. We generate a dataset $A = XY + S$ with $X \in \mathbf{R}^{m\times k}$, $Y \in \mathbf{R}^{k\times n}$, and $S \in \mathbf{R}^{m\times n}$, with $m = n = 300$ and $k = 3$. The entries of X and Y are drawn from a standard normal distribution, while the entries of the sparse noise matrix S are drawn from a uniform distribution on $[0, 1]$ with probability 0.05, and are 0 otherwise. We choose a rank for the model that is higher than the (unobserved) true rank of the data; we will see below in §8.2 how to choose the right rank for a model.

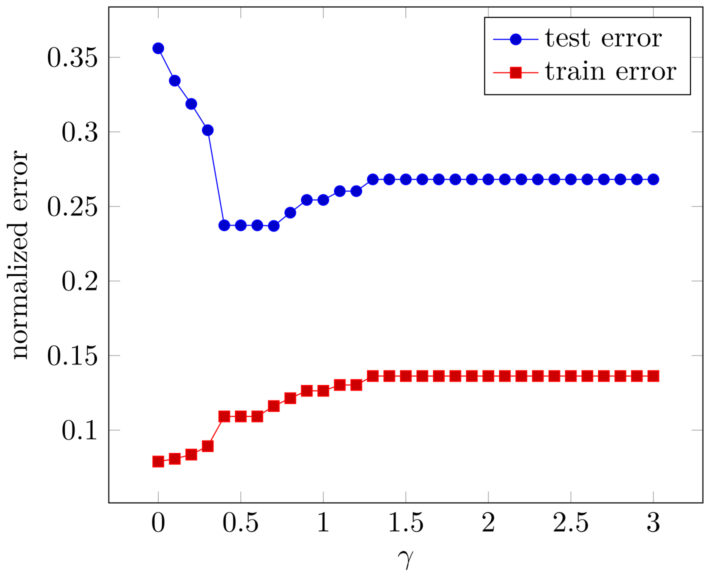


Figure 8.1: Regularization path.

We fit a rank 5 GLRM to an observation set Ω consisting of 10% of the entries in the matrix, drawn uniformly at random from $\{1, \dots, m\} \times \{1, \dots, n\}$, using Huber loss and quadratic regularization, and vary the regularization parameter. That is, we fit the model

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

and vary the regularization parameter γ. The figure plots both the normalized training error,

$$\frac{1}{|\Omega|} \sum_{(i,j)\in\Omega} \mathbf{huber}(x_i y_j, A_{ij}),$$

and the normalized test error,

$$\frac{1}{nm - |\Omega|} \sum_{(i,j)\notin\Omega} \mathbf{huber}(x_i y_j, A_{ij}),$$

of the fitted model (X, Y), for γ ranging from 0 to 3. Here, we see that while the training error decreases and γ decreases, the test error reaches a minimum around $\gamma = .5$. Interestingly, it takes only three times longer (about 3 seconds) to generate the entire regularization path than it does to fit the model for a single value of the regularization parameter (about 1 second).

8.2 Choosing model parameters

To form a generalized low rank model, one needs to specify the loss functions L_j, regularizers r and $\tilde{r}$, and a rank k. The loss function should usually be chosen by a domain expert to reflect the intuitive notion of what it means to "fit the data well". On the other hand, the regularizers and rank are often chosen based on statistical considerations, so that the model generalizes well to unseen (missing) data.

There are three major considerations to balance in choosing the regularization and rank of the model. In the following discussion, we suppose that the regularizers $r = \gamma r^0$ and $\tilde{r} = \gamma \tilde{r}^0$ have been chosen up to a scaling γ.

Compression. A low rank model (X, Y) with rank k and no sparsity represents the data table A with only $(m+n)k$ nonzeros, achieving a compression ratio of $(m+n)k/(mn)$. If the factors X or Y are sparse, then we have used fewer than $(m+n)k$ numbers to represent the data A, achieving a higher compression ratio. We may want to pick parameters of the model (k and γ) in order to achieve a good error $\sum_{(i,j)\in\Omega} L_j(A_{ij} - x_i y_j)$ for a given compression ratio. For each possible combination of model parameters, we can fit a low rank model with those parameters, observing both the error and the compression ratio. We can then choose the best model parameters (highest compression rate) achieving the error we require, or the best model parameters (lowest error rate) achieving the compression we require.

More formally, one can construct an information criterion for low rank models by analogy with the Aikake Information Criterion (AIC) or the Bayesian Information Criterion (BIC). For use in the AIC, the number of degrees of freedom in a low rank model can be computed as the difference between the number of nonzeros in the model and the dimensionality of the symmetry group of the problem. For example, if the model (X, Y) is dense, and the regularizer is invariant under orthogonal transformations (*e.g.*, $r(x) = \|x\|_2^2$), then the number of degrees of freedom is $(m + n)k - k^2$ [142]. Minka [101] proposes a method based on the BIC to automatically choose the dimensionality in PCA, and observes that it performs better than cross validation in identifying the true rank of the model when the number of observations is small (m, $n \lesssim 100$).

Denoising. Suppose we observe every entry in a true data matrix contaminated by noise, *e.g.*, $A_{ij} = A_{ij}^{\text{true}} + \epsilon_{ij}$, with ϵ_{ij} some random variable. We may wish to choose model parameters to identify the truth and remove the noise: we would like to find k and γ to minimize $\sum_{(i,j)\in\Omega} L_j(A_{ij}^{\text{true}} - x_i y_j)$.

A number of commonly used rules-of-thumb have been proposed in the case of PCA to distinguish the signal (the true rank k of the data) from the noise, some of which can be generalized to other low rank models. These include using scree plots, often known as the "elbow method" [25]; the eigenvalue method; Horn's parallel analysis [61, 42]; and other related methods [162, 115]. A recent, more sophisticated method adapts the idea of dropout training [137] to regularize low-rank matrix estimation [68].

Some of these methods can easily be adapted to the GLRM context. The "elbow method" increases k until the objective value decreases less than linearly; the eigenvalue method increases k until the objective value decreases by less than some threshold; Horn's parallel analysis increases k until the objective value compares unfavorably to one generated by fitting a model to data drawn from a synthetic noise distribution.

Cross validation is also simple to apply, and is discussed further below as a means of predicting missing entries. However, applying cross validation to the denoising problem is somewhat tricky, since leaving out too few entries results in overfitting to the noise, while leaving out too many results in underfitting to the signal. The optimal number of entries to leave out may depend on the aspect ratio of the data, as well as on the type of noise present in the data [113], and is not well understood except in the case of Gaussian noise [108]. We explore the problem of choosing a holdout size numerically below.

Predicting missing entries. Suppose we observe some entries in the matrix and wish to predict the others. A GLRM with a higher rank will always be able to fit the (noisy) data better than one of lower rank. However, a model with many parameters may also overfit to the noise. Similarly, a GLRM with no regularization ($\gamma = 0$) will always produce a model with a lower empirical loss $\sum_{(i,j)\in\Omega} L_j(x_i y_j, A_{ij})$. Hence, we cannot pick a rank k or regularization γ simply by considering the objective value obtained by fitting the low rank model.

But by resampling from the data, we can simulate the performance of the model on out of sample (missing) data to identify GLRMs that neither over nor underfit. Here, we discuss a few methods for choosing model parameters by cross-validation; that is, by resampling from the data to evaluate the model's performance. Cross validation is commonly used in regression models to choose parameters such as the regularization parameter γ, as in Figure 8.1. In GLRMs, cross validation can also be used to choose the rank k. Indeed, using a lower rank k can be considered another form of model regularization.

We can distinguish between three sources of noise or variability in the data, which give rise to three different resampling procedures.

- The *rows* or *columns* of the data are chosen at random, *i.e.*, drawn iid from some population. In this case it makes sense to resample the *rows* or *columns*.
- The rows or columns may be fixed, but the indices of the observed entries in the matrix are chosen at random. In this case, it makes sense to resample from the observed *entries* in the matrix.
- The indices of the observed entries are fixed, but the values are observed with some measurement error. In this case, it makes sense to resample the *errors* in the model.

Each of these leads to a different reasonable kind of resampling scheme. The first two give rise to resampling schemes based on cross validation (*i.e.*, resampling the rows, columns, or individual entries of the matrix) which we discuss further below. The third gives rise to resampling schemes based on the bootstrap or jackknife procedures, which resample from the errors or residuals after fitting the model. A number of methods using the third kind of resampling have been proposed in order to perform inference (*i.e.*, generate confidence intervals) for PCA; see Josse et al. [69] and references therein.

As an example, let us explore the effect of varying $|\Omega|/mn$, γ, and k. We generate random data as follows. Let $X \in \mathbf{R}^{m\times k^{\text{true}}}$, $Y \in \mathbf{R}^{k^{\text{true}}\times n}$, and $S \in \mathbf{R}^{m\times n}$, with $m = n = 300$ and $k^{\text{true}} = 3$. Draw the entries of X and Y from a standard normal distribution, and draw the entries of the sparse outlier matrix S are drawn from a uniform distribution on $[0, 3]$ with probability 0.05, and are 0 otherwise. Form $A = XY + S$. Select an observation set Ω by picking entries in the matrix uniformly at random from $\{1, \dots, n\} \times \{1, \dots, m\}$. We fit a rank k GLRM with Huber loss and quadratic regularization $\gamma\|\cdot\|_2^2$, varying $|\Omega|/mn$, γ, and k, and compute the test error. We average our results over 5 draws from the distribution generating the data.

In Figure 8.2, we see that the true rank $k = 3$ performs best on cross-validated error for any number of observations $|\Omega|$. (Here, we show performance for $\gamma = 0$. The plot for other values of the regularization parameter is qualitatively the same.) Interestingly, it is easiest to identify the true rank with a small number of observations: higher numbers of observations make it more difficult to overfit to the data even when allowing higher ranks.

In Figure 8.3, we consider the interdependence of our choice of γ and k. Regularization is most important when few matrix elements have been observed: the curve for each k is nearly flat when more than about 10% of the entries have been observed, so we show here a plot for $|\Omega| = .1mn$. Here, we see that the true rank $k = 3$ performs best on cross-validated error for any value of the regularization parameter. Ranks that are too high ($k > 3$) benefit from increased regularization γ, whereas higher regularization hurts the performance of models with

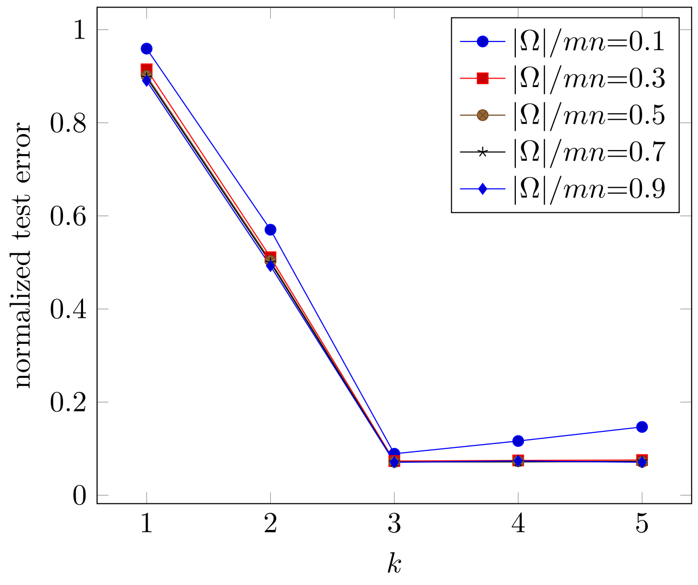


Figure 8.2: Test error as a function of k, for $\gamma = 0$.

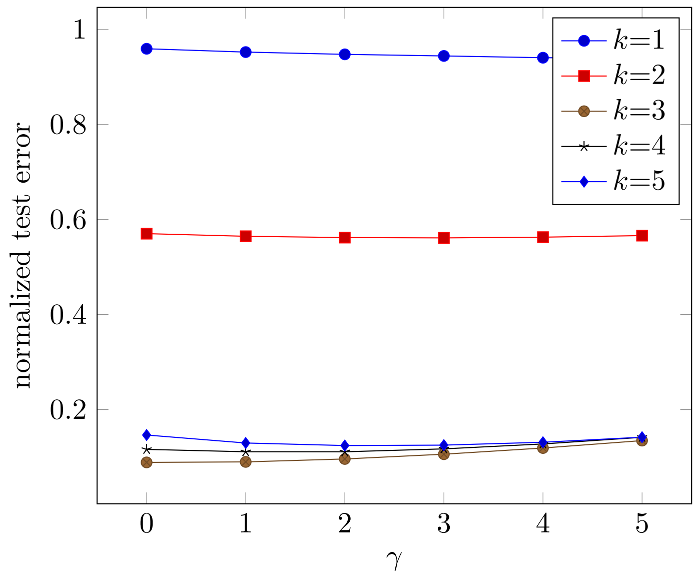


Figure 8.3: Test error as a function of γ when 10% of entries are observed.

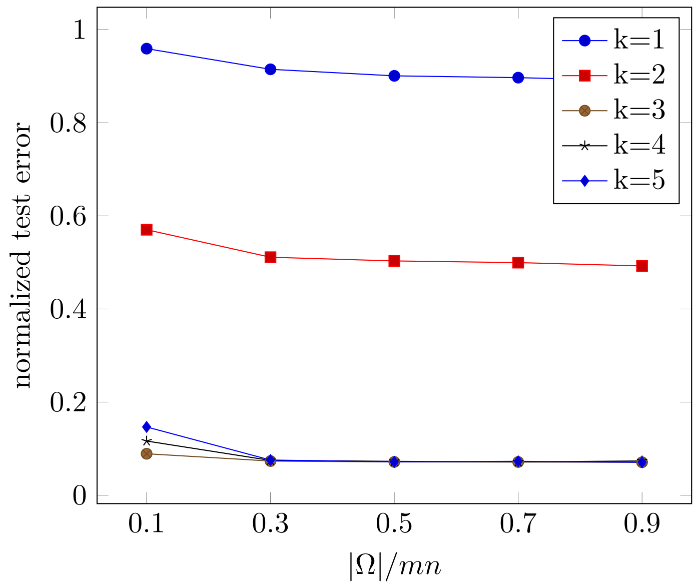


Figure 8.4: Test error as a function of observations $|\Omega|/mn$, for $\gamma = 0$.

k lower than the true rank. That is, regularizing the rank (small k) can substitute for explicit regularization of the factors (large γ).

Finally, in Figure 8.4 we consider how the fit of the model depends on the number of observations. If we correctly guess the rank $k = 3$, we find that the fit is insensitive to the number of observations. If our rank is either too high or too low, the fit improves with more observations.

8.3 On-line optimization

Suppose that new examples or features are being added to our data set continuously, and we wish to perform *on-line optimization*, which means that we should have a good estimate at any time for the representations of those examples x_i or features y_j which we have seen. This model is equivalent to adding new rows or columns to the data table A as the algorithm continues. In this setting, alternating minimization performs quite well, and has a very natural interpretation. Given an estimate for Y, when a new example is observed in row i, we may solve

$$\text{minimize} \quad \sum_{j:(i,j)\in\Omega} L_{ij}(A_{ij}, xy_j) + r_i(x)$$

with variable x to compute a representation for row i. This computation is exactly the same as one step of alternating minimization. Here, we are finding the best feature representation for the new example in terms of the (already well understood) archetypes Y. If the number of other examples previously seen is large, the addition of a single new example should not change the optimal Y by very much; hence if (X, Y) was previously the global minimum of (4.1), this estimate of the feature representation for the new example will be very close to its optimal representation (*i.e.*, the one that minimizes problem (4.1)). A similar interpretation holds when new columns are added to A.

9

Implementations

The authors and collaborators have developed and released four open source codes for modeling and fitting generalized low rank models:

- a serial implementation written in Python;
- a fully featured serial and shared-memory parallel implementation written in Julia;
- a basic distributed implementation written in Scala using the Spark framework; and
- a distributed implementation written in Java using the H2O framework, with Java and R interfaces.

The Julia, Spark and H2O implementations use the alternating proximal gradient method described in §7 to fit GLRMs, while the Python implementation uses alternating minimization and a `cvxpy` [39] backend for each subproblem. The Python implementation is suitable for problems with no more than a few hundred rows and columns. The Julia implementation is suitable for problems that fit in memory on a single computer, including those with thousands of columns and millions of rows. The H2O and Spark implementations must be used for larger problem sizes. For most uses, we recommend the Julia implementation or the H2O implementation. As of March 2016, the Julia implementation is the most fully featured, with an ample library of losses and regularizers, as well as routines to cross validate, impute, and test goodness-of-fit. For a full description and up-to-date information about available functionality of each of these implementations, we encourage the reader to consult the on-line documentation for each of these packages.

There are also many implementations available for fitting special cases of GLRMs. For example, an implementation capable of fitting any GLRM for which the subproblems in an alternating minimization method are quadratic programs was recently developed in Spark by Debasish Das and Santanu Das [32].

In this section we briefly discuss the Python, Julia, and Spark implementations, and report some timing results. The H2O implementation will not be discussed below; documentation and tutorials are available at `http://learn.h2o.ai/content/tutorials/glrm/glrm-tutorial.html`.

9.1 Python implementation

`GLRM.py` is a Python implementation for fitting GLRMs that can be found, together with documentation, at `https://github.com/cehorn/glrm`.

Usage. The user initializes a GLRM by specifying

- the data table `A` (A), stored as a Python list of 2-D arrays, where each 2-D array in `A` contains all data associated with a particular loss function,
- the list of loss functions `L` (L_j, $j = 1, \dots, n$), that correspond to the data as specified by `A`,
- regularizers `regX` (r) and `regY` ($\tilde{r}$),
- the rank `k` (k),

- an optional list `missing_list` with the same length as `A` so that each entry of `missing_list` is a list of missing entries corresponding to the data from `A`, and
- an optional convergence object `converge` that characterizes the stopping criterion for the alternating minimization procedure.

The following example illustrates how to use `GLRM.py` to fit a GLRM with Boolean (`A_bool`) and numerical (`A_real`) data, with quadratic regularization and a few missing entries.

```
from glrm import GLRM                              # import model
from glrm.loss import QuadraticLoss, HingeLoss    # and losses
from glrm.reg import QuadraticReg                  # and regularizer

A = [A_bool, A_real]                               # data (as list)
L = [Hinge_Loss, QuadraticLoss]                    # losses (as list)
regX = QuadraticReg(0.1)                           # penalty scale 0.1
regY = QuadraticReg(0.1)
missing_list = [[], [(0,0), (0,1)]]                # missing entries

model = GLRM(A, L, regX, regY, k, missing_list)   # initialize GLRM
model.fit()                                        # fit GLRM
```

The `fit()` method automatically adds an offset to the GLRM and scales the loss functions as described in §4.3.

`GLRM.py` fits GLRMs by alternating minimization. The code instantiates `cvxpy` problems [39] corresponding to the X- and Y-update steps, then iterates by alternately solving each problem until convergence criteria are met.

The following loss functions and regularizers are supported by `GLRM.py`:

- quadratic loss `QuadraticLoss`,
- Huber loss `HuberLoss`,
- hinge loss `HingeLoss`,
- ordinal loss `OrdinalLoss`,
- no regularization `ZeroReg`,

- ℓ_1 regularization `LinearReg`,
- quadratic regularization `QuadraticReg`, and
- nonnegative constraint `NonnegativeReg`.

Users may implement their own loss functions (regularizers) using the abstract class `Loss` (`Reg`).

9.2 Julia implementation

LowRankModels is a code written in Julia [9] for modeling and fitting GLRMs. The implementation is available on-line at `https://github.com/madeleineudell/LowRankModels.jl`. We discuss some aspects of the usage and features of the code here.

Usage. The LowRankModels package transposes some of the notation from this paper for computational speed. It approximates the $m \times n$ data table `A` by a model $X^T Y$, where $X \in \mathbf{R}^{k \times m}$ and $Y \in \mathbf{R}^{k \times n}$. For most GLRMs, $d = n$, but for multidimensional loss functions, $d = \sum_{j=1}^{n} d_j$ is the embedding dimension of the model (see §6).

To form a GLRM using LowRankModels, the user specifies, in order:

- `A`: the data (A), which can be any array or array-like data structure (*e.g.*, a sparse matrix, or a Julia `DataFrame`);
- `losses`: either one loss function to be applied to every entry of `A`; or a list of loss functions (L_j, $j = 1, \ldots, n$), one for each column of `A`;
- `rx`: a regularizer (r) on the rows of X
- `ry`: a regularizer ($\tilde{r}$) on the columns of Y; or a list of regularizers ($\tilde{r}_j$, $j = 1, \ldots, n$), one for each column of `A`, and
- `k`: the rank (k),

and optional named arguments:

- the observed entries `obs` (Ω), a list of tuples of the indices of the observed entries in the matrix, which may be omitted if all the entries in the matrix have been observed;
- initial values `X` (X) and `Y` (Y)
- if `offset` is `true`, an offset will be added to the model for each column; it is false by default
- if `scale` is `true`, the losses for each column are scaled as in §4.3; it is false by default.
- if `sparse_na` is `true`, the data matrix `A` is given as a sparse matrix, and the keyword argument `obs` is omitted, implicit zeros of `A` will be interpreted as missing entries; `sparse_na` is true by default.

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

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

LowRankModels uses the proximal gradient method described in §7.2 to fit GLRMs. The optimal model is returned in the factors `X` and `Y`, while `ch` gives the convergence history. The exclamation mark suffix is a naming convention in Julia, and denotes that the function mutates at least one of its arguments. In this case, it caches the best fit X and Y as `glrm.X` and `glrm.Y` [27].

Losses and regularizers must be of type `Loss` and `Regularizer`, respectively, and may be chosen from a list of supported losses and regularizers, shown in Table 9.1 and Table 9.2 respectively. In the tables, w, c, and d are parameters: w is the weight of the loss function, and takes default value 1; c is the relative importance of false positive examples compared to false negative examples, and has default value 1; and d is the number of levels of an ordinal or categorical variable. Users may also implement their own losses and regularizers.

loss	**code**	$L(u,a)$
quadratic	`QuadLoss(w)`	$w(u-a)^2$
ℓ_1	`L1Loss(w)`	$w\|u-a\|$
huber	`HuberLoss(w)`	$w\,\mathbf{huber}(u-a)$
Poisson	`PoissonLoss(w)`	$w(\exp(u)-au)$
logistic	`LogisticLoss(w)`	$w\log(1+\exp(-au))$
hinge	`HingeLoss(w)`	$w\max(1-au,0)$
weighted hinge	`WeightedHingeLoss(w,c)`	$(\delta(a=-1)+c\delta(a=1))$ $\times w(1-au)_+$
ordinal hinge	`OrdinalHingeLoss(w,d)`	$w\sum_{a'=1}^{a-1}(1-u+a')_+$ $+w\sum_{a'=a+1}^{d}(1+u-a')_+$
multinomial ordinal	`MultinomialOrdinalLoss(w,d)`	$w\left(\sum_{i=1}^{a-1}u_i-\sum_{i=a}^{d}u_i\right.$ $+\log(\sum_{a'=1}^{d}\exp(\sum_{i=1}^{a'-1}u_i$ $\left.-\sum_{i=a'}^{d-1}u_i))\right)$
multinomial	`MultinomialLoss(w,d)`	$-\log\left(\frac{\exp(u_a)}{\sum_{a'=1}^{d}\exp(u_{a'})}\right)$

Table 9.1: Loss functions available in the LowRankModels Julia package. Here δ is a function that returns 1 if its argument is true and 0 otherwise.

regularizer	**code**	$r(x)$
nothing	`ZeroReg()`	0
quadratic	`QuadReg(w)`	$w\|x\|_2^2$
ℓ_1	`OneReg(w)`	$w\|x\|_1$
nonnegative	`NonNegConstraint()`	$\mathbf{I}_+(x)$
1-sparse	`OneSparseConstraint()`	$\mathbf{I}_1(x)$
clustered	`UnitOneSparseConstraint()`	$\mathbf{I}_1(x)+\mathbf{I}(\sum_{l=1}^{k}x_l=1)$
mixture	`SimplexConstraint()`	$\mathbf{I}_+(x)+\mathbf{I}(\sum_{l=1}^{k}x_l=1)$

Table 9.2: Regularizers available in the LowRankModels Julia package. Here $\mathbf{I}_+$ is the indicator of the nonnegative orthant, $\mathbf{I}_1$ is the indicator of the 1-sparse vectors, and $\mathbf{I}$ is a function that returns 0 if its argument is true and ∞ otherwise.

Shared memory parallelism. LowRankModels takes advantage of Julia's `SharedArray` data structure to implement a shared-memory parallel fitting procedure. While Julia does not yet support threading, `SharedArray`s in Julia allow separate processes on the same computer to access the same block of memory at the same time. To fit a model using multiple processes, LowRankModels loads the data A and the initial model X and Y into shared memory, broadcasts other problem data (*e.g.*, the losses and regularizers) to each process, and assigns to each process a partition of the rows of X and columns of Y. At every iteration, each process updates its rows of X, its columns of Y, and computes its portion of the objective function, synchronizing after each of these steps to ensure that, *e.g.*, the X update is completed before the Y update begins; then the master process checks a convergence criterion and adjusts the step length.

Automatic modeling. LowRankModels is capable of adding offsets to a GLRM, and of automatically scaling the loss functions, as described in §4.3. It can also pick appropriate loss functions for columns whose types are specified in an array `datatypes` whose elements take the values `:real`, `:bool`, `:ord`, or `:cat`. Using these features, LowRankModels implements a method

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

that forms a rank k model on a data frame with datatypes specified in the array `datatypes`. This function automatically selects loss functions and regularization that suit the data well, and ignores any missing (`NA`) element in the data frame. This GLRM can then be fit with the function `fit!`. By default, the four data types are fit with quadratic loss, logistic loss, multinomial ordinal loss, and ordinal loss, respectively, but other mappings can be specified by setting the keyword argument `loss_map` to a dictionary mapping datatypes to loss functions.

Example. As an example, we fit a GLRM to the Motivational States Questionnaire (MSQ) data set [121]. This data set measures 3896 subjects on 92 aspects of mood and personality type, as well as recording the time of day the data were collected. The data include real-valued, Boolean, and ordinal measurements, and approximately 6% of the measurements are missing (`NA`).

The following code loads the MSQ data set and encodes it in two dimensions:

```
using RDatasets
using LowRankModels
# pick a data set
df = RDatasets.dataset("psych","msq")
# encode it!
X,Y,labels,ch = fit(glrm(df,2))
```

Figure 9.1 uses the rows of Y as a coordinate system to plot some of the features of the data set. Here we see the automatic embedding separates positive from negative emotions along the y axis. This embedding is notable for being interpretable despite having been generated completely automatically. Of course, better embeddings may be obtained by a more careful choice of loss functions, regularizers, scaling, and rank k.

9.3 Spark implementation

`SparkGLRM` is a code written in Scala, built on the Spark cluster programming framework [160], for modelling and fitting GLRMs. The implementation is available on-line at `http://git.io/glrmspark`.

Design. In `SparkGLRM`, the data matrix A is split entry-wise across many machines, just as in [60]. The model (X, Y) is replicated and stored in memory on every machine. Thus the total computation time required to fit the model is proportional to the number of nonzeros divided by the number of cores, with the restriction that the model should fit in memory. (The authors leave to future work an extension to models that do not fit in memory, *e.g.*, by using a parameter server [125].) Where possible, hardware acceleration (via breeze and BLAS) is used for local linear algebraic operations.

At every iteration, the current model is broadcast to all machines, so there is only one copy of the model on each machine. This particularly important in machines with many cores, because it avoids duplicating

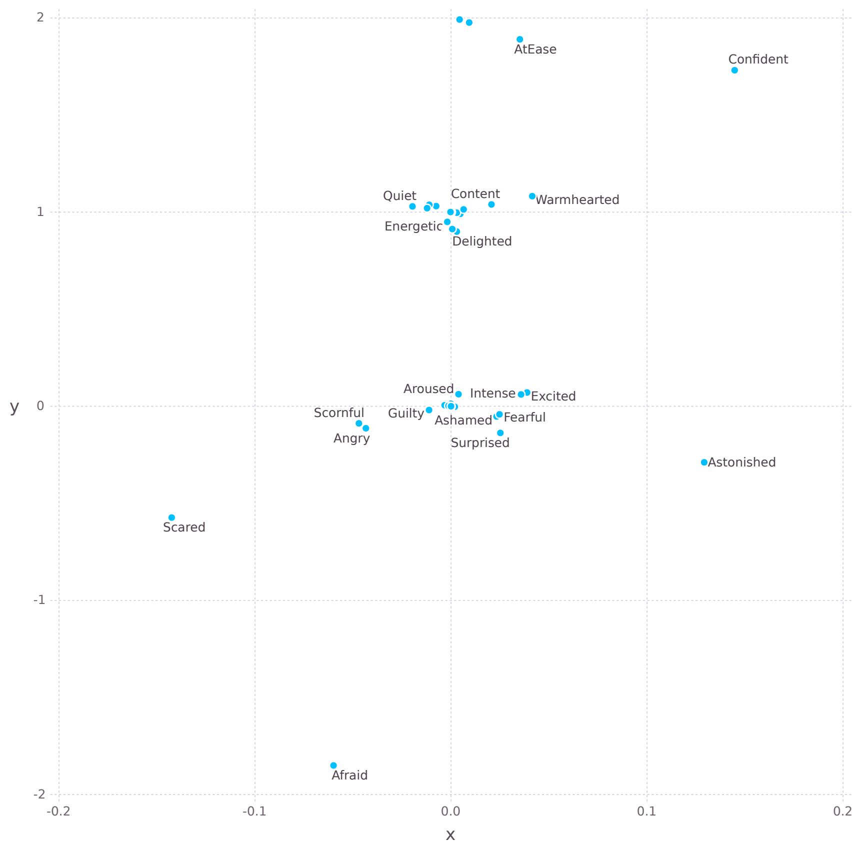


Figure 9.1: An automatic embedding of the MSQ [121] data set into two dimensions.

the model on those machines. Each core on a machine will process a partition of the input matrix, using the local copy of the model.

Usage. The user provides loss functions $L_{ij}(u,a)$ indexed by $i = 0,\ldots,m-1$ and $j = 0,\ldots,n-1$, so a different loss function can be defined for each column, or even for each entry. Each loss function is defined by its gradient (or a subgradient). The method signature is

```
loss_grad(i: Int, j: Int, u: Double, a: Double)
```

whose implementation can be customized by particular i and j. As an example, the following line implements squared error loss ($L(u,a) = 1/2(u-a)^2$) for all entries:

```
u - a
```

Similarly, the user provides functions implementing the proximal operator of the regularizers r and $\tilde{r}$, which take a dense vector and perform the appropriate proximal operation.

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

We report iteration times using an Amazon EC2 cluster with 10 slaves and one master, of instance type "c3.4xlarge". Each machine has 16 CPU cores and 30 GB of RAM. We ran `SparkGLRM` to fit two GLRMs on matrices of varying sizes. Table 9.3 gives results for quadratically regularized PCA (*i.e.*, quadratic loss and quadratic regularization) with $k = 5$. To illustrate the capability to write and fit custom loss functions, we also fit a GLRM using a loss function that depends on the parity of $i + j$:

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

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

Table 9.3: `SparkGLRM` for quadratically regularized PCA, $k = 5$.

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

Table 9.4: `SparkGLRM` for custom GLRM, $k = 10$.

with $r(x) = \|x\|_1$ and $\tilde{r}(y) = \|y\|_2^2$, setting $k = 10$. (This loss function was chosen merely to illustrate the generality of the implementation. Usually losses will be the same for each entry in the same column.) The results for this custom GLRM are given in Table 9.4.

The table gives the time per iteration. The number of iterations required for convergence depends on the size of the ambient dimension. On the matrices with the dimensions shown in Tables 9.3 and 9.4, convergence typically requires about 100 iterations, but we note that useful GLRMs often emerge after only a few tens of iterations.

Acknowledgments

The authors are grateful to Chris De Sa, Yash Deshpande, Nicolas Gillis, Maya Gupta, Trevor Hastie, Irene Kaplow, Tamara Kolda, Lester Mackey, Andrea Montanari, Art Owen, Haesun Park, David Price, Chris Ré, Ben Recht, Yoram Singer, Nati Srebro, Ashok Srivastava, Peter Stoica, Sze-chuan Suen, Stephen Taylor, Joel Tropp, Ben Van Roy, and Stefan Wager for a number of illuminating discussions and comments on early drafts of this paper; to Debasish Das and Matei Zaharia for their insights into creating a successful Spark implementation; and to Anqi Fu and H2O.ai for their work on the H2O implementation. This work was developed with support from the National Science Foundation Graduate Research Fellowship program (under Grant No. DGE-1147470), the Gabilan Stanford Graduate Fellowship, the Gerald J. Lieberman Fellowship, and the DARPA X-DATA program.

Appendices

A

Examples, loss functions, and regularizers

A.1 Quadratically regularized PCA

In this appendix we describe some properties of the quadratically regularized PCA problem (2.3),

$$\text{minimize} \quad \|A - XY\|_F^2 + \gamma\|X\|_F^2 + \gamma\|Y\|_F^2. \tag{A.1}$$

In the sequel, we let $U\Sigma V^T = A$ be the SVD of A and let r be the rank of A. We assume for convenience that all the nonzero singular values $\sigma_1 > \sigma_2 > \cdots > \sigma_r > 0$ of A are distinct.

A.1.1 Solution

A solution is given by

$$X = \tilde{U}\tilde{\Sigma}^{1/2}, \qquad Y = \tilde{\Sigma}^{1/2}\tilde{V}^T, \tag{A.2}$$

where $\tilde{U}$ and $\tilde{V}$ are defined as in (2.5), and $\tilde{\Sigma} = \mathbf{diag}((\sigma_1 - \gamma)_+, \ldots, (\sigma_k - \gamma)_+)$.

To prove this, let us consider the optimality conditions of (2.3). The optimality conditions are

$$-(A - XY)Y^T + \gamma X = 0, \qquad -(A - XY)^T X + \gamma Y^T = 0.$$

Multiplying the first optimality condition on the left by X^T and the second on the left by Y and rearranging, we find

$$X^T(A - XY)Y^T = \gamma X^T X, \qquad Y(A - XY)^T X = \gamma YY^T,$$

which shows, by taking a transpose, that $X^T X = YY^T$ at any stationary point.

We may rewrite the optimality conditions together as

$$\begin{aligned}\begin{bmatrix} -\gamma I & A \\ A^T & -\gamma I \end{bmatrix}\begin{bmatrix} X \\ Y^T \end{bmatrix} &= \begin{bmatrix} 0 & XY \\ (XY)^T & 0 \end{bmatrix}\begin{bmatrix} X \\ Y^T \end{bmatrix} \\ &= \begin{bmatrix} X(YY^T) \\ Y^T(X^T X) \end{bmatrix} \\ &= \begin{bmatrix} X \\ Y^T \end{bmatrix}(X^T X),\end{aligned}$$

where we have used the fact that $X^T X = YY^T$.

Now we see that (X, Y^T) lies in an *invariant subspace* of the matrix $\begin{bmatrix} -\gamma I & A \\ A^T & -\gamma I \end{bmatrix}$. Recall that V is an invariant subspace of a matrix A if $AV = VM$ for some matrix M. If $\operatorname{Rank}(M) \leq \operatorname{Rank}(A)$, we know that the eigenvalues of M are eigenvalues of A, and that the corresponding eigenvectors lie in the span of V.

Thus the eigenvalues of $X^T X$ must be eigenvalues of $\begin{bmatrix} -\gamma I & A \\ A^T & -\gamma I \end{bmatrix}$, and (X, Y^T) must span the corresponding eigenspace. More concretely, notice that $\begin{bmatrix} -\gamma I & A \\ A^T & -\gamma I \end{bmatrix}$ is (symmetric, and therefore) diagonalizable, with eigenvalues $-\gamma \pm \sigma_i$. The larger eigenvalues $-\gamma + \sigma_i$ correspond to the eigenvectors (u_i, v_i), and the smaller ones $-\gamma - \sigma_i$ to $(u_i, -v_i)$.

Now, $X^T X$ is positive semidefinite, so the eigenvalues shared by $X^T X$ and $\begin{bmatrix} -\gamma I & A \\ A^T & -\gamma I \end{bmatrix}$ must be positive. Hence there is some set $|\Omega| \leq k$ with $\sigma_i \geq \gamma$ for $i \in \Omega$ such that X has singular values $\sqrt{-\gamma + \sigma_i}$ for $i \in \Omega$. (Recall that $X^T X = YY^T$, so Y has the same singular values as X.) Then (X, Y^T) spans the subspace generated by the vectors $(u_i, v_i$ for $i \in \Omega$. We say the stationary point (X, Y) has active subspace Ω. It is easy to verify that $XY = \sum_{i \in \Omega} u_i(\sigma_i - \gamma)v_i^T$.

Each active subspace gives rise to an orbit of stationary points. If (X, Y) is a stationary point, then $(XT, T^{-1}Y)$ is also a stationary point so long as

$$-(A - XY)Y^T T^{-T} + \gamma XT = 0, \qquad -(A - XY)^T XT + \gamma Y^T T^{-T} = 0,$$

which is always true if $T^{-T} = T$, *i.e.*, T is orthogonal. This shows that the set of stationary points is invariant under orthogonal transformations.

To simplify what follows, we choose a representative element for each orbit. Represent any stationary point with active subspace Ω by

$$X = U_\Omega(\Sigma_\Omega - \gamma I)^{1/2}, \quad Y = (\Sigma_\Omega - \gamma I)^{1/2} V_\Omega^T,$$

where by U_Ω we denote the submatrix of U with columns indexed by Ω, and similarly for Σ and V. At any value of γ, let $k'(\gamma) = \max\{i : \sigma_i \geq \gamma\}$. Then we have $\sum_{i=0}^{k} \binom{k'(\gamma)}{i}$ (representative) stationary points, one for each choice of Ω The number of (representative) stationary points is decreasing in γ; when $\gamma > \sigma_1$, the only stationary point is $X = 0$, $Y = 0$.

These stationary points can have quite different values. If (X, Y) has active subspace Ω, then

$$||A - XY||_F^2 + \gamma(||X||_F^2 + ||Y||_F^2) = \sum_{i \notin \Omega} \sigma_i^2 + \sum_{i \in \Omega} \left(\gamma^2 + 2\gamma|\sigma_i - \gamma|\right).$$

From this form, it is clear that we should choose Ω to include the top singular values $i = 1, \ldots, k'(\gamma)$. Choosing any other subset Ω will result in a higher (worse) objective value: that is, the other stationary points are *not* global minima.

A.1.2 Fixed points of alternating minimization

Theorem A.1. The quadratically regularized PCA problem (2.3) has only one local minimum, which is the global minimum.

Our proof is similar to that of [6], who proved a related theorem for the case of PCA (2.2).

Proof. We showed above that every stationary point of (2.3) has the form $XY = \sum_{i \in \Omega} u_i d_i v_i^T$, with $\Omega \subseteq \{1, \ldots, k'\}$, $|\Omega| \leq k$, and $d_i = \sigma_i - \gamma$.

We use the representative element from each stationary orbit described above, so each column of X is $u_i\sqrt{d_i}$ and each row of Y is $\sqrt{d_i}v_i^T$ for some $i \in \Omega$. The columns of X are orthogonal, as are the rows of Y.

If a stationary point is not the global minimum, then $\sigma_j > \sigma_i$ for some $i \in \Omega$, $j \notin \Omega$. Below, we show we can always find a descent direction if this condition holds, thus showing that the only local minimum is the global minimum.

Assume we are at a stationary point with $\sigma_j > \sigma_i$ for some $i \in \Omega$, $j \notin \Omega$. We will find a descent direction by perturbing XY in direction $u_j v_j^T$. Form $\tilde{X}$ by replacing the column of X containing $u_i\sqrt{d_i}$ by $(u_i + \epsilon u_j)\sqrt{d_i}$, and $\tilde{Y}$ by replacing the row of Y containing $\sqrt{d_i}v_i^T$ by $\sqrt{d_i}(v_i + \epsilon v_j)^T$. Now the regularization term increases slightly:

$$\begin{aligned}
&\gamma(\|\tilde{X}\|_F^2 + \|\tilde{Y}\|_F^2) - \gamma(\|X\|_F^2 + \|Y\|_F^2) \\
&\quad = \sum_{i' \in \Omega, i' \neq i} (2\gamma t_{i'}) + 2\gamma d_i(1+\epsilon^2) - \sum_{i' \in \Omega} 2\gamma t_{i'} \\
&\quad = 2\gamma d_i \epsilon^2.
\end{aligned}$$

Meanwhile, the approximation error decreases:

$$\begin{aligned}
&\|A - \tilde{X}\tilde{Y}\|_F^2 - \|A - XY\|_F^2 \\
&\quad = \|u_i\sigma_i v_i^T + u_j\sigma_j v_j^T - (u_i + \epsilon u_j)d_i(v_i + \epsilon v_j)^T\|_F^2 - (\sigma_i - d_i)^2 - \sigma_j^2 \\
&\quad = \|u_i(\sigma_i - d_i)v_i^T + u_j(\sigma_j - \epsilon^2 d_i)v_j^T - \epsilon u_i d_i v_j^T - \epsilon u_j d_i v_i^T\|_F^2 \\
&\qquad - (\sigma_i - d_i)^2 - \sigma_j^2 \\
&\quad = \left\| \begin{bmatrix} \sigma_i - d_i & -\epsilon d_i \\ -\epsilon d_i & \sigma_j - \epsilon^2 d_i \end{bmatrix} \right\|_F^2 - (\sigma_i - d_i)^2 - \sigma_j^2 \\
&\quad = (\sigma_i - d_i)^2 + (\sigma_j - \epsilon^2 d_i)^2 + 2\epsilon^2 d_i^2 - (\sigma_i - d_i)^2 - \sigma_j^2 \\
&\quad = -2\sigma_j \epsilon^2 d_i + \epsilon^4 d_i^2 + 2\epsilon^2 d_i^2 \\
&\quad = 2\epsilon^2 d_i(d_i - \sigma_j) + \epsilon^4 d_i^2,
\end{aligned}$$

where we have used the rotational invariance of the Frobenius norm to arrive at the third equality above. Hence the net change in the objective value in going from (X, Y) to $(\tilde{X}, \tilde{Y})$ is

$$\begin{aligned}
2\gamma d_i \epsilon^2 + 2\epsilon^2 d_i(d_i - \sigma_j) + \epsilon^4 d_i^2 &= 2\epsilon^2 d_i(\gamma + d_i - \sigma_j) + \epsilon^4 d_i^2 \\
&= 2\epsilon^2 d_i(\sigma_i - \sigma_j) + \epsilon^4 d_i^2,
\end{aligned}$$

which is negative for small ϵ. Hence we have found a descent direction, showing that any stationary point with $\sigma_j > \sigma_i$ for some $i \in \Omega$, $j \notin \Omega$ is not a local minimum. $\square$

data type	**loss**	$L(u,a)$	**params**	**dim**
real	quadratic	$(u-a)^2$		1
real	absolute value	$\lvert u-a \rvert$		1
real	huber	$\mathbf{huber}(u-a)$		1
real	quantile	$\alpha(a-u)_+ + (1-\alpha)(u-a)_+$	tilt α	1
boolean	logistic	$\log(1+\exp(-au))$		1
boolean	hinge	$(1-ua)_+$		1
boolean	weighted hinge	$(\delta(a=-1)+c\delta(a=1))(1-au)_+$	weight c	1
integer	poisson	$\exp(u)-au+a\log a-a$		1
ordinal	ordinal hinge	$\sum_{a'=1}^{a-1}(1-u+a')_+ +$ $\sum_{a'=a+1}^{d}(1+u-a')_+$	levels d	1
ordinal	multinomial ordinal	$\sum_{i=1}^{a-1} u_i - \sum_{i=a}^{d-1} u_i +$ $\log\left(\sum_{a'=1}^{d-1}\exp\left(\sum_{i=1}^{a'-1} u_i \sum_{i=a'}^{d-1} u_i\right)\right)$	levels d	$d-1$
categorical	one-vs-all	$(1-u_a)_+ + \sum_{a'\neq a}(1+u_{a'})_+$	levels d	d
categorical	hamming	$\delta(u_a \neq 1) + \sum_{a'\neq a}\delta(u_{a'} \neq -1)$	levels d	d
categorical	multinomial	$-\log\left(\frac{\exp(u_a)}{\sum_{a'=1}^{d}\exp(u_{a'})}\right)$	levels d	d

Table A.1: A few loss functions. Here δ is a function that returns 1 if its argument is true and 0 otherwise. **params** shows the parameters of the loss function, and **dim** gives its embedding dimension.

regularizer	$r(x)$
nothing	0
quadratic	$\|x\|_2^2$
ℓ_1	$\|x\|_1$
nonnegative	$\mathbf{I}_+(x)$
nonnegative ℓ_1 regularized	$\|x\|_1 + \mathbf{I}_+(x)$
orthogonal nonnegative	$\mathbf{I}_1(x) + \mathbf{I}_+(x)$
s-sparse	$\mathbf{card}(x) \leq s$
clustered	$\mathbf{I}_1(x) + \mathbf{I}(\sum_{l=1}^k x_l = 1)$
mixture	$\mathbf{I}_+(x) + \mathbf{I}(\sum_{l=1}^k x_l = 1)$

Table A.2: A few regularizers. Here $\mathbf{I}_+$ is the indicator of the nonnegative orthant, $\mathbf{I}_1$ is the indicator of the 1-sparse vectors, and $\mathbf{I}$ is a function that returns 0 if its argument is true and ∞ otherwise.

References

[1] J. Abernethy, F. Bach, T. Evgeniou, and J.-P. Vert. A new approach to collaborative filtering: Operator estimation with spectral regularization. *The Journal of Machine Learning Research*, 10:803–826, 2009.

[2] A. Agarwal, A. Anandkumar, P. Jain, and P. Netrapalli. Learning sparsely used overcomplete dictionaries via alternating minimization. *arXiv preprint arXiv:1310.7991*, 2013.

[3] P. K. Agarwal and N. H. Mustafa. k-means projective clustering. In *Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 155–165. ACM, 2004.

[4] M. Aharon, M. Elad, and A. Bruckstein. k-SVD: An algorithm for designing overcomplete dictionaries for sparse representation. *IEEE Transactions on Signal Processing*, 54(11):4311–4322, 2006.

[5] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[6] P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Networks*, 2(1):53–58, 1989.

[7] M. Berry, M. Browne, A. Langville, V. Pauca, and R. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational Statistics & Data Analysis*, 52(1):155–173, 2007.

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

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

[10] 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.

[11] J. Bolte, S. Sabach, and M. Teboulle. Proximal alternating linearized minimization for nonconvex and nonsmooth problems. *Mathematical Programming*, pages 1–36, 2013.

[12] J. Borwein and A. Lewis. *Convex analysis and nonlinear optimization: theory and examples*, volume 3. Springer Science & Business Media, 2010.

[13] R. Boyd, B. Drake, D. Kuang, and H. Park. Smallk is a C++/Python high-performance software library for nonnegative matrix factorization (NMF) and hierarchical and flat clustering using the NMF; current version 1.2.0. `http://smallk.github.io/`, June 2014.

[14] S. Boyd, C. Cortes, M. Mohri, and A. Radovanovic. Accuracy at the top. In *Advances in Neural Information Processing Systems*, pages 962–970, 2012.

[15] S. Boyd and J. Mattingley. Branch and bound methods. *Lecture notes for EE364b, Stanford University*, 2003.

[16] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

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

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

[19] S. Burer and R. Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming*, 95(2):329–357, 2003.

[20] S. Burer and R. D. C. Monteiro. Local minima and convergence in low-rank semidefinite programming. *Mathematical Programming*, 103, 2005.

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

[22] E. Candès and Y. Plan. Matrix completion with noise. *CoRR*, abs/0903.3131, 2009.

[23] E. Candès and B. Recht. Exact matrix completion via convex optimization. *CoRR*, abs/0805.4471, 2008.

[24] E. Candès and T. Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory*, 56(5):2053–2080, 2010.

[25] Raymond B Cattell. The scree test for the number of factors. *Multivariate behavioral research*, 1(2):245–276, 1966.

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

[27] J. Chen and A. Edelman. Parallel prefix polymorphism permits parallelization, presentation & proof. *arXiv preprint arXiv:1410.6449*, 2014.

[28] S. Chen, D. Donoho, and M. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61, 1998.

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

[30] 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.

[31] A. Damle and Y. Sun. Random projections for non-negative matrix factorization. *arXiv preprint arXiv:1405.4275*, 2014.

[32] D. Das and S. Das. Quadratic programing solver for non-negative matrix factorization with spark. In *Spark Summit 2014*, 2014.

[33] A. d'Aspremont, L. El Ghaoui, M. I. Jordan, and G. R. Lanckriet. A direct formulation for sparse PCA using semidefinite programming. In *Advances in Neural Information Processing Systems*, volume 16, pages 41–48, 2004.

[34] M. Davenport, Y. Plan, E. Berg, and M. Wootters. 1-bit matrix completion. *arXiv preprint arXiv:1209.3672*, 2012.

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

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

[37] 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.

[38] C. De Sa, K. Olukotun, and C. Ré. Global convergence of stochastic gradient descent for some nonconvex matrix problems. *CoRR*, abs/1411.1134, 2014.

[39] S. Diamond, E. Chu, and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization, version 0.2. `http://cvxpy.org/`, May 2014.

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

[41] C. Ding, T. Li, W. Peng, and H. Park. Orthogonal nonnegative matrix t-factorizations for clustering. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 126–135. ACM, 2006.

[42] A. Dinno. Implementing Horn's parallel analysis for principal component analysis and factor analysis. *Stata Journal*, 9(2):291, 2009.

[43] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering large graphs via the singular value decomposition. *Machine Learning*, 56(1-3):9–33, 2004.

[44] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936.

[45] E. Elhamifar and R. Vidal. Sparse subspace clustering. In *IEEE Conference on Computer Vision and Pattern Recognition, 2009*, pages 2790–2797. IEEE, 2009.

[46] 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.

[47] C. Févotte, N. Bertin, and J. Durrieu. Nonnegative matrix factorization with the Itakura-Saito divergence: With application to music analysis. *Neural Computation*, 21(3):793–830, 2009.

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

[49] N. Gillis. *Nonnegative matrix factorization: Complexity, algorithms and applications.* PhD thesis, UCL, 2011.

[50] N. Gillis and F. Glineur. Low-rank matrix approximation with weights or missing data is NP-hard. *SIAM Journal on Matrix Analysis and Applications*, 32(4):1149–1165, 2011.

[51] Nicolas Gillis and François Glineur. A continuous characterization of the maximum-edge biclique problem. *Journal of Global Optimization*, 58(3):439–464, 2014.

[52] 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.

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

[54] 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.

[55] S. Gunasekar, A. Acharya, N. Gaur, and J. Ghosh. Noisy matrix completion using alternating minimization. In *Machine Learning and Knowledge Discovery in Databases*, pages 194–209. Springer, 2013.

[56] M. Gupta, S. Bengio, and J. Weston. Training highly multiclass classifiers. *The Journal of Machine Learning Research*, 15(1):1461–1492, 2014.

[57] 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.

[58] M. Hardt. On the provable convergence of alternating minimization for matrix completion. *arXiv preprint arXiv:1312.0925*, 2013.

[59] M. Hardt and M. Wootters. Fast matrix completion without the condition number. *arXiv preprint arXiv:1407.4070*, 2014.

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

[61] J. Horn. A rationale and test for the number of factors in factor analysis. *Psychometrika*, 30(2):179–185, 1965.

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

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

[64] Z. Huang and M. Ng. A fuzzy k-modes algorithm for clustering categorical data. *IEEE Transactions on Fuzzy Systems*, 7(4):446–452, 1999.

[65] P. Huber. *Robust Statistics.* Wiley, New York, 1981.

[66] 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.

[67] I. Jolliffe. *Principal component analysis.* Springer, 1986.

[68] J. Josse and S. Wager. Stable autoencoding: A flexible framework for regularized low-rank matrix estimation. *arXiv preprint arXiv:1410.8275*, 2014.

[69] J. Josse, S. Wager, and F. Husson. Confidence areas for fixed-effects pca. *arXiv preprint arXiv:1407.7614*, 2014.

[70] M. Journée, F. Bach, P. Absil, and R. Sepulchre. Low-rank optimization on the cone of positive semidefinite matrices. *SIAM Journal on Optimization*, 20(5):2327–2351, 2010.

[71] L. Kaufman and P. J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

[72] R. Keshavan. *Efficient algorithms for collaborative filtering.* PhD thesis, Stanford University, 2012.

[73] R. Keshavan and A. Montanari. Regularization for matrix completion. In *2010 IEEE International Symposium on Information Theory Proceedings (ISIT)*, pages 1503–1507. IEEE, 2010.

[74] R. Keshavan, A. Montanari, and S. Oh. Matrix completion from noisy entries. In *Advances in Neural Information Processing Systems*, pages 952–960, 2009.

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

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

[77] H. Kim and H. Park. Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. *Bioinformatics*, 23(12):1495–1502, 2007.

[78] H. Kim and H. Park. Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. *SIAM Journal on Matrix Analysis and Applications*, 30(2):713–730, 2008.

[79] J. Kim, Y. He, and H. Park. Algorithms for nonnegative matrix and tensor factorizations: A unified view based on block coordinate descent framework. *Journal of Global Optimization*, 58(2):285–319, 2014.

[80] J. Kim and H. Park. Toward faster nonnegative matrix factorization: A new algorithm and comparisons. In *Eighth IEEE International Conference on Data Mining*, pages 353–362. IEEE, 2008.

[81] J. Kim and H. Park. Fast nonnegative matrix factorization: An active-set-like method and comparisons. *SIAM Journal on Scientific Computing*, 33(6):3261–3281, 2011.

[82] R. Koenker. *Quantile regression*. Cambridge University Press, 2005.

[83] R. Koenker and J. G. Bassett. Regression quantiles. *Econometrica: Journal of the Econometric Society*, pages 33–50, 1978.

[84] E. Lawler and D. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

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

[86] D. Lee and H. Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems*, pages 556–562, 2001.

[87] H. Lee, A. Battle, R. Raina, and A. Ng. Efficient sparse coding algorithms. In *Advances in Neural Information Processing Systems*, pages 801–808, 2006.

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

[89] 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.

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

[91] C. Lin. Projected gradient methods for nonnegative matrix factorization. *Neural Computation*, 19(10):2756–2779, 2007.

[92] Z. Liu and L. Vandenberghe. Interior-point method for nuclear norm approximation with application to system identification. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1235–1256, 2009.

[93] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[94] L. Mackey. Deflation methods for sparse PCA. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, 2009.

[95] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 689–696. ACM, 2009.

[96] J. Mairal, J. Ponce, G. Sapiro, A. Zisserman, and F. Bach. Supervised dictionary learning. In *Advances in Neural Information Processing Systems*, pages 1033–1040, 2009.

[97] I. Markovsky. *Low Rank Approximation: Algorithms, Implementation, Applications*. Communications and Control Engineering. Springer, 2012.

[98] 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.

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

[100] 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.

[101] T. Minka. Automatic choice of dimensionality for pca. In T.K. Leen, T.G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, pages 598–604. MIT Press, 2001.

[102] K. Mohan and M. Fazel. Reweighted nuclear norm minimization with application to system identification. In *Proceedings of the 2010 American Control Conference (ACC)*, pages 2953–2959. IEEE, 2010.

[103] P. Netrapalli, U. Niranjan, S. Sanghavi, A. Anandkumar, and P. Jain. Provable non-convex robust PCA. In *Advances in Neural Information Processing Systems*, pages 1107–1115, 2014.

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

[105] J. Nocedal and S. Wright. *Numerical optimization.* Springer Science & Business Media, 2006.

[106] B. Olshausen and D. Field. Sparse coding with an overcomplete basis set: A strategy employed by V1? *Vision Research*, 37(23):3311–3325, 1997.

[107] S. Osnaga. *Low Rank Representations of Matrices using Nuclear Norm Heuristics*. PhD thesis, Colorado State University, 2014.

[108] A. Owen and P. Perry. Bi-cross-validation of the svd and the non-negative matrix factorization. *The Annals of Applied Statistics*, pages 564–594, 2009.

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

[110] H.-S. Park and C.-H. Jun. A simple and fast algorithm for k-medoids clustering. *Expert Systems with Applications*, 36(2, Part 2):3336–3341, 2009.

[111] 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.

[112] 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.

[113] P. Perry. Cross-validation for unsupervised learning. *arXiv preprint arXiv:0909.3052*, 2009.

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

[115] K. Preacher and R. MacCallum. Repairing Tom Swift's electric factor analysis machine. *Understanding Statistics: Statistical Issues in Psychology, Education, and the Social Sciences*, 2(1):13–43, 2003.

[116] R. Raina, A. Battle, H. Lee, B. Packer, and A. Ng. Self-taught learning: Transfer learning from unlabeled data. In *Proceedings of the 24th International Conference on Machine Learning*, pages 759–766. ACM, 2007.

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

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

[119] 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.

[120] J. 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.

[121] W. Revelle and K. Anderson. Personality, motivation and cognitive performance: Final report to the army research institute on contract MDA 903-93-K-0008. Technical report, 1998.

[122] P. Richtárik, M. Takáč, and S. Ahipaşaoğlu. Alternating maximization: Unifying framework for 8 sparse PCA formulations and efficient parallel codes. *arXiv preprint arXiv:1212.4137*, 2012.

[123] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *The Journal of Machine Learning Research*, 5:101–141, 2004.

[124] A. Schein, L. Saul, and L. 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.

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

[126] F. Shahnaz, M. W. Berry, V. P. Pauca, and R. J. Plemmons. Document clustering using nonnegative matrix factorization. *Information Processing & Management*, 42(2):373–386, 2006.

[127] S. Shalev-Shwartz, A. Gonen, and O. Shamir. Large-scale convex minimization with a low-rank constraint. *arXiv preprint arXiv:1106.1622*, 2011.

[128] H. Shen and J. Huang. Sparse principal component analysis via regularized low rank matrix approximation. *Journal of Multivariate Analysis*, 99(6):1015–1034, 2008.

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

[130] R. Smith. Nuclear norm minimization methods for frequency domain subspace identification. In *Proceedings of the 2010 American Control Conference (ACC)*, pages 2689–2694. IEEE, 2012.

[131] M. Soltanolkotabi and E. Candes. A geometric analysis of subspace clustering with outliers. *The Annals of Statistics*, 40(4):2195–2238, 2012.

[132] M. Soltanolkotabi, E. Elhamifar, and E. Candes. Robust subspace clustering. *arXiv preprint arXiv:1301.2603*, 2013.

[133] N. Srebro. *Learning with Matrix Factorizations.* PhD thesis, Massachusetts Institute of Technology, 2004.

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

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

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

[137] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[138] H. Steck. Hinge rank loss and the area under the ROC curve. In J. N. Kok, J. Koronacki, R. L. Mantaras, S. Matwin, D. Mladenič, and A. Skowron, editors, *Machine Learning: ECML 2007*, volume 4701 of *Lecture Notes in Computer Science*, pages 347–358. Springer Berlin Heidelberg, 2007.

[139] D. L. Sun and C. Févotte. Alternating direction method of multipliers for non-negative matrix factorization with the beta-divergence. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2014.

[140] R. Sun and Z.-Q. Luo. Guaranteed matrix completion via nonconvex factorization. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 270–289. IEEE, 2015.

[141] 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.

[142] M. Tipping and C. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.

[143] N. Tishby, F. Pereira, and W. Bialek. The information bottleneck method. *arXiv preprint physics/0004057*, 2000.

[144] J. Tropp. *Topics in Sparse Approximation.* PhD thesis, The University of Texas at Austin, 2004.

[145] J. Tropp and A. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Transactions on Information Theory*, 53(12):4655–4666, 2007.

[146] P. Tseng. Nearest q-flat to m points. *Journal of Optimization Theory and Applications*, 105(1):249–252, 2000.

[147] M. Tweedie. An index which distinguishes between some important exponential families. In *Statistics: Applications and New Directions. Proceedings of the Indian Statistical Institute Golden Jubilee International Conference*, pages 579–604, 1984.

[148] N. Usunier, D. Buffoni, and P. Gallinari. Ranking with ordered weighted pairwise classification. In *Proceedings of the 26th annual International Conference on Machine Learning*, pages 1057–1064. ACM, 2009.

[149] S. Vavasis. On the complexity of nonnegative matrix factorization. *SIAM Journal on Optimization*, 20(3):1364–1377, 2009.

[150] R. Vidal. A tutorial on subspace clustering. *IEEE Signal Processing Magazine*, 28(2):52–68, 2010.

[151] T. Virtanen. Monaural sound source separation by nonnegative matrix factorization with temporal continuity and sparseness criteria. *IEEE Transactions on Audio, Speech, and Language Processing*, 15(3):1066–1074, 2007.

[152] V. Vu, J. Cho, J. Lei, and K. Rohe. Fantope projection and selection: A near-optimal convex relaxation of sparse PCA. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2670–2678. Curran Associates, Inc., 2013.

[153] J. Weston, S. Bengio, and N. Usunier. Large scale image annotation: Learning to rank with joint word-image embeddings. *Machine Learning*, 81(1):21–35, 2010.

[154] J. Weston, H. Yee, and R. J. Weiss. Learning to rank recommendations with the k-order statistic loss. In *Proceedings of the 7th ACM Conference on Recommender Systems*, RecSys '13, pages 245–248, New York, NY, USA, 2013. ACM.

[155] D. Witten, R. Tibshirani, and T. Hastie. A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. *Biostatistics*, page kxp008, 2009.

[156] 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.

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

[158] 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.

[159] 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.

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

[161] H. Zou, T. Hastie, and R. Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15(2):265–286, 2006.

[162] W. Zwick and W. Velicer. Comparison of five rules for determining the number of components to retain. *Psychological bulletin*, 99(3):432, 1986.