Integer Matrix Factorization and Its Application

Matthew M. Lin, Bo Dong, Moody T. Chu

Abstract—Matrix factorization has been of fundamental importance in modern sciences and technology. This work investigates the notion of factorization with entries restricted to integers or binaries, , where the "integer" could be either the regular ordinal integers or just some nominal labels. Being discrete in nature, such a factorization or approximation cannot be accomplished by conventional techniques. Built upon a basic scheme of rank one approximation, an approach that recursively splits (approximates) the underlying matrix into a sum of rank one matrices with discrete entries is proposed. Various computational issues involved in this kind of factorization, which must take into account the metric being used for measurement, are addressed. The mechanism presented in this paper can handle multiple types of data. For application purposes, the discussion emphasizes mainly on binary-integer factorizations. But the notion is readily generalizable with slight modifications to, for example, integer-integer factorizations. Applications to cluster analysis and pattern discovery are demonstrated through some real-world data. Of particular interest is the result on the ordering of rank one approximations which, in a remote sense, is the analogy for discrete data to the ordering of singular values for continuous data. As such, a truncated low rank factorization (for discrete data) analogous to the truncated singular value decomposition is also obtained.

Index Terms—matrix factorization, clustering, classification, pattern discovery

1 Introduction

Matrix factorization has been of fundamental importance in modern sciences and technology. In the past decades, theory and numerical methods for matrix factorization have been extensively studied. In an interesting appraisal of the two purposes of matrix factorization, Hubert, Meulman, and Heiser [1] commented that, on one hand,

> "Matrix factorization in the context of numerical linear algebra (NLA) generally serves the purpose of rephrasing through a series of easier subproblems a task that may be relatively difficult to solve in its original form."

Thus, the factorization of a square matrix A as the product of a lower triangular matrix L and an upper triangular matrix U, for example, "has no real importance in and of itself other than as a computationally convenient means for obtaining a solution to the original linear system." On the other hand, regarding a matrix $A \in \mathbb{R}^{m\times n}$ as a data matrix containing numerical observations on m objects (subjects) over n attributes (variables), or possibly $B \in \mathbb{R}^{n\times n}$ representing, say, the correlation between columns of A, they made the point that

> "The major purpose of a matrix factorization in this context is to obtain some form of lower-rank (and therefore simplified) approximation to A (or possibly to B) for understanding the structure of the data matrix, particularly the relationship within the objects and within the attributes, and how the objects relate to the attributes."

Thus, in the context of applied statistics/psychometrics (AS/P), "matrix factorizations are again directed toward the issue of simplicity, but now the actual components making up a factorization are of prime concern and not solely as a mechanism for solving another problem." We prefer to be able to "interpret the matrix factorization geometrically and actually present the results spatially through coordinates obtained from the components of the factorization". In such a factorization can be achieved, we might better understand the structure or retrieve the information hidden in the original

- *Matthew M. Lin is with Department of Mathematics, North Carolina State University, Raleigh, NC 27695-8205.
Email: mlin@ncsu.edu.*
This research was supported in part by the National Science Foundation under grant DMS-0505880.
- *Bo Dong is with Department of mathematics, Dalian University of Technology, Dalian, Liaoning, 116024.
Email: dongbodlut@gmail.comThis work is partially supported by Chinese Scholarship Council and DLUT(Dalian University of Technology) under grands 3004-893327 and 3004-842321.*
- *Moody T. Chu is with Department of Mathematics, North Carolina State University, Raleigh, NC 27695-8205.
Email: chu@math.ncsu.edu.*
This research was supported in part by the National Science Foundation under grants CCR-0204157 and DMS-0505880, and NIH Roadmap for Medical Research grant 1 P20 HG003900-01.

data matrix.

There are at least three different ways to formulate matrix factorization, depending on the types of constraints imposed on the factors. The most conventional notion is that the matrix A is equal to the product of two or more factors. In this case, the factorization can be considered as the sum of rank-one matrices. Indeed, a remarkably simple rank-one reduction formula due to Wedderburn is able to unify several fundamental factorization processes under one framework. It was shown in the review paper by Chu, Funderlic and Golub [2] that almost all matrix decompositions known in NLA could be obtained via this mechanism. Perhaps not noted to the NLA community, the conception of rank reduction, including the now well recognized singular value decomposition, has been prevalent for many decades in the AS/P community. See, for example, discussions in [3], [4], [5], [6], [7], [8] and also the time line for the appearance of the rank reduction results in [1]. The data involved in this type of calculation are mostly from the real or complex field of numbers.

There are situations in applications, including text mining [9], [10], image articulation [11], [12], bioinformatics [13],micro-array analysis [14] , cheminformatics [15], [16], air pollution research [16], and spectral data analysis [17], where the data to be analyzed is nonnegative. As such, the low rank data are also required to be comprised of nonnegative values only in order to avoid contradicting physical realities. This demand brings forth the notion of the so called nonnegative matrix factorization (NMF). That is, for a given nonnegative matrix $A \in \mathbb{R}^{m\times n}$ and a fixed integer k, we are interested in the minimization problem [15]

$$\min_{U\in\mathbb{R}_+^{m\times k}, V\in\mathbb{R}_+^{k\times n}} \|A - UV\|_F, \tag{1}$$

where $\mathbb{R}_+$ stands for the half-arry of nonnegative real numbers. Note that the equality $A = UV$ may never be true, but the two matrices U, V are still termed as low rank "factors" of A in the literature. Classical tools cannot guarantee to maintain the nonnegativity. Considerable research efforts have benn devoted to develop special techniques for NMF. See, for example, the multiplicative update algorithm by Lee and Seung [11], [18], the gradient methods [19], [20], and alternating least square methods [21], [22], [23], [24], [16]. Additional discussions can be found in [25], [26], [27], [28], [29], [30], and the many references contained therein.

The two types of factorizations mentioned above and their approaches are of distinct nature. The data they intend to handle, however, share a common feature in that they are from a continuum domain. To motivate the discrete type of data considered in this paper, envisage the scenario of switch manufacturing in the telecommunications industry [31]. Suppose that each switch cabinet consists of n slots which can be fitted with only one type of specified board options. Assume that there are t_i different board types for each slot. In all, there are $\prod_{i=1}^{n} t_i$ different models. It might be desirable to first build a few basic models so that, corresponding to different custom orders, we might meet a specific configuration more efficiently. The question is how many and what basic models should be built. One plausible approach is to "learn" from past experiences, that is, we try to obtain the basic model information from the matrix of past sales. Labeling each board type in slot by an integer (or any token) while the same integer for different slots (columns) may refer to different attributes, the data representing past m customer orders form an integer matrix of size $m \times n$. Let it be stressed again that, in this setting, entries in the data matrix are of discrete values and that these values may or may not reflect any ordinal scale. A factorization of such a data matrix therefore must be subject to certain specifically defined arithmetic rules, which is the main thrust of this paper.

Without causing ambiguity, let $\mathbb{Z}$ denote either the conventional set (or subset) of integers when ordinal scale matters, or the set of all possible tokens which are not totally preordered, for the system under consideration. For the simplicity of discussion, we shall assume that the entries of the given matrix A are from the same set $\mathbb{Z}$. In practice, different columns of A may be composed of elements from different subsets of $\mathbb{Z}$. Even the same element in $\mathbb{Z}$ may have different meaning in different columns. If a comparison between two observations (rows) is needed, entries at different locations in the rows might need to be measured differently. Before we can perform the factorization, a properly defined metric for the row vectors, such as the metric for the product space (of individual entries), therefore must be in place, which then induces a metric d to measure the "nearness" between two matrices. With this in mind, we define our integer matrix factorization (IMF) as follows.

(IMF) Given an integer matrix $A \in \mathbb{Z}^{m\times n}$, an induced metric d, and a positive integer[1] $k < \min\{m, n\}$, find a binary matrix U of size $m \times k$ with mutually orthogonal columns and $V \in$

1. The determination of such a rank k is itself an important question which thus far has no satisfactory answer yet.

$\mathbb{Z}^{k\times n}$ so that the functional

$$f(U,V) := d(A,UV), \qquad (2)$$

is minimized.

Each entry a_{ij} in A denotes, in a broad sense, the *score* obtained by entity i on variable j; the value of $u_{i\ell}$, being dichotomous, indicates whether entity i is influenced by "factor" ℓ; and $v_{\ell j}$ marks the *attribute* of variable j to the factor ℓ. Take the sale record of a certain electronic device as the matrix A, for example. Then rows of A denote the specifications by m customers with respect to the n different slots in the device; rows of V represent the basis models to be built; and the mutual exclusion of columns of U implies that each customer order corresponds to exactly one possible basic model. The factors in this case represent some abstract concepts by which the customer orders are divided into k clusters. Clearly, this is a typical classification problem written in factorization form. We also see that the number k plays an important role in the low rank approximation. In practice, we prefer to have as few factors as possible while keeping the objective value $f(U,V)$ low.

In the special case when the set $\mathbb{Z}$ is made of $\mathbb{Z}_2 := \{0,1\}$, i.e., binaries, a nonorthogonal binary decomposition by recursive partitioning has recently been proposed in [32]. The recursion continues until either the Hamming distance within a cluster is less than a preset threshold or all members within a cluster have reached homogeneity. The code PROXIMUS, written in the C language, takes into account efficient data structure, storage, and movement within the system. In this article, we generalize that method to handle integer matrices in general.

Needless, the metric d for measuring nearness or similarity plays critical roles. We need to carefully discern whether variables are ordinal (totally ordered) or nominal (unordered) to the categories. For the former, we also need to decide whether the spacing between the (categorial) values is the same across all levels of the variables or not. If the values are not equally spaced, using integers to represent these values has the advantage of numerically scaling in the spacing disparities. The commonly used Hamming distance or Euclidean distance can be refined to reflect the different situations. We limit our discussion in this paper to these two metrics, although our concept can easily be generalized to other means of measurement.

This presentation contains several interesting applications of IMF, but it is the details on the computational aspects of IMF that deserve our attention. To make our approach effectual, we need to analyze several essential components separately first before putting them to work in conjunction. We thus organize this paper as follows: We begin in Section 2 with a basic scheme that does alternating direction search for rank-one approximation. We show that in each alternating direction an optimal solution is attainable in closed form. This scheme serves as a building block by which we construct in Section 3 a divide-and-conquer strategy that ultimately factorize the given matrix A. The method gradually breaks down the data matrix to submatrices in a recursive way. Although it seems that we are dealing with numerals, our method actually can handle categorical variables. A suitable criterion for assessing the validity of the proposed splitting, therefore, must also be addressed. A prototype code is sketched in Algorithm 3 to demonstrate how these components should be assembled together. In practice, the code needs to be modified to reflect the different nature of the underlying data and the metric, but the general concept is the same. We have experimented our method with quite a few real-world data sets. We select to report only a few in Section 4. Our numerical experiences seem to suggest that our approach is easy to program, converges considerably fast even with a large data matrix, and can handle multiple types of data.

2 BASIC RANK-ONE APPROXIMATION

Our IMF algorithm is built upon a series of rank-one approximations. In this section we describe how such an approximation is achieved. Suppose that, after taking into account the data type, a metric d is already given. The general idea for computing the rank-one approximation is to employ the notion of alternating direction iterations (ADI) as follows:

Algorithm 1 Rank-one approximation with general metric d

```
Given matrix $A \in \mathbb{Z}^{m\times n}$, and initial vector $\mathbf{v} \in \mathbb{Z}^{1\times n}$.
return  Vectors $\mathbf{u} \in \mathbb{Z}_2^{m\times 1}$ and $\mathbf{v} \in \mathbb{Z}^{1\times n}$ that minimize $d(A, \mathbf{uv})$.
repeat
    For the fixed $\mathbf{v}$, find $\mathbf{u} \in \mathbb{Z}_2^{m\times 1}$ to minimize $d(A, \mathbf{uv})$
    For the fixed $\mathbf{u}$, find $\mathbf{v} \in \mathbb{Z}^{1\times n}$ to minimize $d(A, \mathbf{uv})$
until convergence
```

Obviously, the iterations can also start with an initial vector $\mathbf{u} \in \mathbb{Z}_2^{m\times 1}$. Note that Algorithm 1 is a descent method and that an optimal solution at each step can be found since there are only finitely many choices for $\mathbf{u}$ and $\mathbf{v}$. Thus the real issue at stake is to effectively find the minimizer in each step. We analyze the computation with regard to two commonly used metrics.

2.1 Approximation with Hamming metric

The *Hamming distance* between two arrays of the same length is defined as the minimum number of substitutions required to transform one array into the other. Here, we apply the same notion to measure the number of different entries between two matrices.

With respect to the Hamming metric, the following two results show that for each given vector $\mathbf{v}$ (or $\mathbf{u}$), the optimal solution $\mathbf{u}$ (or $\mathbf{v}$) for minimizing $d(A, \mathbf{uv})$ can be found in closed form. For convenience, we introduce three operators, $\mathsf{match}(\mathbf{x}, \mathbf{y})$, $\mathsf{zeros}(\mathbf{x})$ and $\mathsf{mode}(\mathbf{x})$, to count the number of matching elements between vectors $\mathbf{x}$ and $\mathbf{y}$, the number of zero entries in the vector $\mathbf{x}$, and the most frequent entry of the vector $\mathbf{x}$, respectively. Also, we adopt the notation $A(i, :)$ for the ith row of a matrix A.

Theorem 1: Given $A \in \mathbb{Z}^{m\times n}$ and $\mathbf{v} \in \mathbb{Z}^{1\times n}$, then among all possible $\mathbf{u} \in \mathbb{Z}_2^{m\times 1}$ the Hamming metric $d(A, \mathbf{uv})$ is minimized at $\mathbf{u}^*$ whose ith entry u_i^*, $i = 1, \ldots, m$, is defined by

$$u_i^* := \begin{cases} 1, & \text{if } \mathsf{match}(A(i,:), \mathbf{v}) \geq \mathsf{zeros}(A(i,:)), \\ 0, & \textit{otherwise}. \end{cases} \tag{3}$$

Proof: For any $\mathbf{u} \in \mathbb{Z}_2^{m\times 1}$, observe that

$$d(A, \mathbf{uv}) = \sum_{i=1}^{m} d(A(i,:), u_i \mathbf{v}). \tag{4}$$

Thus minimizing $d(A, \mathbf{uv})$ is equivalent to minimizing each individual term $d(A(i,:), u_i\mathbf{v})$ for $i = 1, 2, \cdots, m$. From the relationships that $\mathsf{match}(A(i,:), \mathbf{v}) = n - d(A(i,:), \mathbf{v})$ whereas $\mathsf{zeros}(A(i,:)) = n - d(A(i,:), \mathbf{0})$, it is clear that the choice in (3) is optimal. □

Theorem 2: Given $A \in \mathbb{Z}^{m\times n}$ and $\mathbf{u} \in \mathbb{Z}_2^{m\times 1}$ with $\mathbf{u} \neq \mathbf{0}$, then among all $\mathbf{v} \in \mathbb{Z}^{1\times n}$ the Hamming metric $d(A, \mathbf{uv})$ is minimized at $\mathbf{v}^*$ whose jth entry v_j^*, $j = 1, \ldots, n$, is defined by

$$v_j^* := \mathsf{mode}(A_2(:, j)), \tag{5}$$

where A_2 is the submatrix composed of all $A(i,:)$ whose corresponding u_i is 1.

Proof: With permutations if necessary, we may assume without loss of generality that

$$\mathbf{u} = \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{bmatrix}$$

with $\mathbf{u}_1 = [0, \cdots, 0]_{1\times s}^{\top}$ and $\mathbf{u}_2 = [1, \cdots, 1]_{1\times(m-s)}^{\top}$. Divide A accordingly into two parts

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

with $A_1 \in \mathbb{Z}^{s\times n}$ and $A_2 \in \mathbb{Z}^{(m-s)\times n}$. For any vector $\mathbf{v} \in \mathbb{Z}^{1\times n}$, observe that

$$\begin{aligned} & d(A, \mathbf{uv}) \\ & = d(A_1, \mathbf{u}_1\mathbf{v}) + d(A_2, \mathbf{u}_2\mathbf{v}) \\ & = d(A_1, \mathbf{0v}) + \textstyle\sum_{j=1}^{n}\sum_{i=1}^{m-s} d(A_2(i,j), v_j), \end{aligned}$$

Minimizing $d(A, \mathbf{uv})$ is equivalent to minimizing $\sum_{i=1}^{m-s} d(A_2(i,j), v_j)$ for each $j = 1, \ldots, n$. The optimal choice for v_j is certainly the most frequently occuring entry in the jth column of A_2. □

It is worth noting that in both theorems above, the optimal solutions may not be unique if there is a tie. For instance, we may assign $u_i^* = 0$ instead of 1 when $\mathsf{match}(A(i,:), \mathbf{v}) = \mathsf{zeros}(A(i,:))$. It will not affect the optimal objective value $d(A, \mathbf{u}^*\mathbf{v})$ in Theorem 1. Likewise, there will be multiple choices for v_j^* in Theorem 2 when there are more than one most frequent entries in the jth column of A_2.

Equipped with the formulas described in Theorems 1 and 2, the rank-one approximation with the Hamming metric as the measurement of nearness is now taking shape in the form of Algorithm 2. We name this algorithm VOTE because the choice of $\mathbf{u}$ (and $\mathbf{v}$ if A is a binary matrix) is a matter of majority rule between the dichotomy of zeros and matches. A function similar to the command find in Matlab, where $\mathbf{ind} = \mathsf{find}(\mathbf{x})$ locates all nonzero elements of array $\mathbf{x}$ and returns the linear indices of those elements in the vector $\mathbf{ind}$, proves handy in the coding.

In the event that A is a binary matrix, we now argue that the definition (5) for $\mathbf{v}$ is equivalent to the formula (3), if the roles of $\mathbf{u}$ and $\mathbf{v}$ and of rows and columns are interchanged, respectively. More precisely, we make the following observation whose proof is essentially the same as that for Theorem 1.

Lemma 1: Suppose $A \in \mathbb{Z}_2^{m\times n}$. Given $\mathbf{u} \in \mathbb{Z}_2^{m\times 1}$, Then among all $\mathbf{v} \in \mathbb{Z}_2^{1\times n}$ the Hamming metric $d(A, \mathbf{uv})$ is minimized at $\mathbf{v}^*$ whose jth entry v_j^* is given by

$$v_j^* = \begin{cases} 1, & \text{if } \mathsf{match}(A(:,j), \mathbf{u}) \geq \mathsf{zeros}(A(:,j)), \\ 0, & \text{otherwise}. \end{cases} \tag{6}$$

Another view of the equivalence between Theorem 2 and Lemma 1 is through the rela-

Algorithm 2 Rank-one factorization with Hamming metric: $[\mathbf{u}, \mathbf{v}] = \text{VOTE}(A)$

```
Given matrix $A \in \mathbb{Z}^{m\times n}$.
return Vectors $\mathbf{u} \in \mathbb{Z}_2^{m\times 1}$ and $\mathbf{v} \in \mathbb{Z}^{1\times n}$ such that $d(A, \mathbf{uv})$ is minimized.
$\mathbf{v} \leftarrow$ randomly selected from one row of matrix $A$
$\mathbf{z} \leftarrow$ numbers of zeros in $A$ per row
repeat
  vold $\leftarrow \mathbf{v}$
  $\mathbf{m} \leftarrow$ numbers of matches between $v$ and each row of $A$
  if $m_i \geq z_i$ then
    $u_i \leftarrow 1$
  else
    $u_i \leftarrow 0$
  end if
  if $u = \mathbf{0}$ then
    $\mathbf{v} \leftarrow \mathbf{1}$
  else
    ind $\leftarrow$ find($\mathbf{u}$)
    $v_i \leftarrow$ mode($A$(ind, $i$))
  end if
until vold $= \mathbf{v}$
```

tionships

$$\begin{aligned}&\mathsf{match}(A(:,j),\mathbf{u})\\&=\mathsf{match}(A_1(:,j),\mathbf{0})+\mathsf{match}(A_2(:,j),\mathbf{1}),\\&\mathsf{zeros}(A(:,j))\\&=\mathsf{zeros}(A_1(:,j))+\mathsf{zeros}(A_2(:,j)),\end{aligned}$$

whereas it is obviously that $\mathsf{match}(A_1(:,j),\mathbf{0}) = \mathsf{zeros}(A_1(:,j)$. Thus the choice of $\mathsf{mode}(A_2(:,j))$ in Theorem 2 is again a majority rule between $\mathsf{match}(A_2(:,j),\mathbf{1})$ and $\mathsf{zeros}(A_2(:,j))$.

Based on Lemma 1, we claim that the method VOTE applied to binary data is theoretically equivalent to the existing code PROXIMUS which has already been demonstrated to have a wide range of important applications [32]. To see the equivalence, recall that the basic rank-one approximation in PROXIMUS is based on the mechanism of minimizing the Euclidean distance $\|A - \mathbf{uv}\|_F^2$. Upon rewriting

$$\|A - \mathbf{uv}\|_F^2 = \|A\|_F^2 - 2\mathbf{u}^\top A\mathbf{v}^\top + \|\mathbf{u}\|_2^2\|\mathbf{v}\|_2^2,$$

we see that the functional

$$f(\mathbf{u},\mathbf{v}) = 2\mathbf{u}^\top A\mathbf{v}^\top - \|\mathbf{u}\|_2^2\|\mathbf{v}\|_2^2 \tag{7}$$

needs to be maximized. It is not difficult to conclude (see the proof for Theorem 4) that, given a binary vector $\mathbf{v}$, the optimal entries $\mathbf{u}$ must be defined by

$$u_i := \begin{cases} 1, & \text{if } 2(A\mathbf{v}^\top)_i - \|\mathbf{v}\|_2^2 \geq 0, \\ 0, & \text{otherwise }, \end{cases} \tag{8}$$

whereas, given a binary vector $\mathbf{u}$, the optimal entries $\mathbf{v}$ is given by

$$v_i := \begin{cases} 1, & \text{if } 2(\mathbf{u}^\top A)_i - \|\mathbf{u}\|_2^2 \geq 0, \\ 0, & \text{otherwise }. \end{cases} \tag{9}$$

Both (8) and (9) happen to be the rules adopted by PROXIMUS. On the other hand, observe that

$$\begin{aligned}&\mathsf{match}(A(i,:),\mathbf{v}) - \mathsf{zeros}(A(i,:))\\&=\{n-(\mathbf{v}-A(i,:))(\mathbf{v}-A(i,:))^\top\}\\&-\{n-A(i,:)A(i,:)^\top\}\\&=-\mathbf{v}\mathbf{v}^\top+2A(i,:)\mathbf{v}^\top\\&=2(A\mathbf{v}^\top)_i-\|\mathbf{v}\|_2^2,\end{aligned} \tag{10}$$

and

$$\begin{aligned}&\mathsf{match}(A(:,i),\mathbf{u}) - \mathsf{zeros}(A(:,i))\\&=\{m-(\mathbf{u}-A(:,i))^\top(\mathbf{u}-A(:,i))\}\\&-\{m-A(:,i)^\top A(:,i)\}\\&=-\mathbf{u}^\top\mathbf{u}+2\mathbf{u}^\top A(i,:)\\&=2(\mathbf{u}^\top A)_i-\|\mathbf{u}\|_2^2,\end{aligned} \tag{11}$$

implying that VOTE and PROXIMUS are employing exactly the same basic rank-one approximation in each sweep when dealing with binary data. However, note that our method is computationally simpler than PROXIMUS because VOTE avoids matrix-vector multiplications needed in (8) and (9). Furthermore, our approach VOTE can handle more general polychotomous data.

2.2 Approximation with Euclidean metric

In many applications, it is essential to differentiate the true discrepancies among variable values. That is, the values that a variable takes signifies levels of priority, weight, or worth. If the data are somehow represented in the Euclidean space, then the real distance between two points is meaningful and makes a difference in the interpretation. Under such a setting, we discuss in this section the rank-one approximation when the Euclidean metric is used as the measurement for nearness. For demonstrate, $\mathbb{Z}$ denotes all regular integers in this section.

Given $\mathbf{v}$, even if it is not binary, the definition (8) for $\mathbf{u}$ remains to be the optimizer. For completion, we restate the following theorem, but provide a slightly different proof.

Theorem 3: Given $A \in \mathbb{Z}^{m\times n}$ and $\mathbf{v} \in \mathbb{Z}^{1\times n}$, then among all $\mathbf{u} \in \mathbb{Z}_2^{m\times 1}$ the minimal value of $\|A - \mathbf{uv}\|_F^2$ is attained at $\mathbf{u}^*$ whose ith entry is defined by

$$u_i = \begin{cases} 1, & \text{if } 2(A\mathbf{v}^\top)_i - \|\mathbf{v}\|_2^2 \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

Proof: Since

$$\|A - \mathbf{uv}\|_F^2 = \sum_{i=1}^m \|A(i,:) - u_i\mathbf{v}\|_2^2,$$

it is clear that in order to minimize each individual term in the summation we should choose

$$u_i := \begin{cases} 1, & \text{if } \|A(i,:)\|_2^2 \geq \|A(i,:) - \mathbf{v}\|_2^2, \\ 0, & \text{otherwise }, \end{cases}$$

which is equivalent to (12). □

Let $\mathsf{round}(\mathbf{x})$ denote the operator that rounds every entry of $\mathbf{x}$ to its nearest integer. Given $\mathbf{u}$, the next result nicely generalizes the selection criterion in PROXIMUS for general integer vector $\mathbf{v}$.

Theorem 4: Given $A \in \mathbb{Z}^{m\times n}$ and $\mathbf{u} \in \mathbb{Z}_2^{m\times 1}$ with $\mathbf{u} \neq \mathbf{0}$, then among all $\mathbf{v} \in \mathbb{Z}^{1\times n}$, the minimal value of $\|A - \mathbf{uv}\|_F^2$ is attained at $\mathbf{v}^*$ defined by

$$\mathbf{v}^* := \mathsf{round}\left(\frac{\mathbf{u}^\top A}{\|\mathbf{u}\|_2}\right). \tag{13}$$

Proof: We rewrite (7) as

$$\begin{aligned} &2\mathbf{u}^\top A\mathbf{v}^\top - \|\mathbf{u}\|_2^2\|\mathbf{v}\|_2^2 \\ &= \sum_i \left[-\|\mathbf{u}\|_2^2\left(v_i - \frac{(\mathbf{u}^\top A)_i}{\|\mathbf{u}\|_2}\right)^2 + \left(\frac{(\mathbf{u}^\top A)_i}{\|\mathbf{u}\|_2}\right)^2\right]. \end{aligned} \tag{14}$$

It is now clear that the only option for $\mathbf{v}$ to minimize $\|A-\mathbf{uv}\|_F^2$ while keeping $\mathbf{v}$ an integer vector is the definition (13). □

It takes only some slight modifications in the code VOTE to reflect the Euclidean metric, so we shall skip the particulars here. Also, in the event that $\mathbb{Z}$ is restricted to only a subset of integers, the operator round might return a value outside $\mathbb{Z}$. Instead, we can use (14) to define a proper value $v_i \in \mathbb{Z}$ that is nearest to $\frac{(\mathbf{u}^\top A)_i}{\|\mathbf{u}\|_2}$.

We conclude this section with two remarks. Firstly, in the event that $\mathbf{u} = \mathbf{0}$, we can simply assign $\mathbf{v}$ arbitrarily, say, $\mathbf{v} = \mathbf{1}$, without affecting the iteration. Secondly and most importantly, the reason we insist on restricting $\mathbf{u}$ to $\mathbb{Z}_2^{m\times 1}$ is only for the purpose of establishing mutually exclusive clusters, as we shall see in the next section. Our emphasis in this paper is on the binary-integer matrix factorization. However, the proof of Theorem 4 certainly can be applied to $\mathbf{u}$ if $\mathbf{u}$ is to be an integer vector. In this case, the optimal integer vector $\mathbf{u}$ for minimizing $\|A - \mathbf{uv}\|_F^2$ with a fixed $\mathbf{v}$ is given by

$$\mathbf{u}^* := \mathsf{round}\left(\frac{A\mathbf{v}^\top}{\|\mathbf{v}\|_2}\right). \tag{15}$$

If we continue building a sequence of integer rank-one approximations to A in the same way as the IMF algorithm to be described in the next section, the term "integer matrix factorization" is then justified.

3 ALGORITHMS OF IMF

We now describe the procedure for constructing the IMF of a given matrix A. The main idea is to compose A via a sequence of rank-one approximations, but not in the traditional sense of *additive* manner as we shall explain in this section. Three essential issues must be resolved before we can move this idea forward. These are — how to assess the quality of an approximation, how to recursively deflating the matrix, and how to determine the optimal rank. We address each issue separately in the sequel.

For the convenience of reference, after a rank-one approximation $\mathbf{uv}$ for A is established, we say collectively that those rows $A(i,:)$ of A corresponding to $u_i = 1$ are *active* and call $\mathbf{v}$ the *pattern vector* or *representative* of these active rows.

3.1 Communal rule

Although in each sweep of our Algorithm 1 we are able to find the "global" minimizer in each direction, the product of the one-sided minimizers does not necessarily give rise to the global minimizer for two-sided objective function. The quality of a rank-one approximation $A \approx \mathbf{uv}$ depends highly on the initial value. This dependence limits the ADI to find only local interpretable patterns. It is therefore reasonable to set up a checking criterion to decide whether active rows are adequately represented by the pattern vector $\mathbf{v}$.

To put it differently, the assignment of u_i for each $i = 1, \ldots, m$ thus far is based solely on a comparison of $A(i,:)$ individually with $\mathbf{v}$. The active rows themselves have not been juxtaposed with each other. Qualifying $A(i,:)$ into the active group by self-justification has the danger that the differences between the pattern vector and corresponding active rows of A might vary immensely and, thus, the homogeneity within the collective of all active rows cannot be warranted. To exclude outliers, the conventional approach by fixing a neighborhood around $\mathbf{v}$ is not effective because $\mathbf{v}$ is not necessarily the mean and every single active row, though preliminarily counted as active, could be far away from $\mathbf{v}$. Instead, we propose to check the communality of the active rows more dynamically by using a simple statistical rule.

Firstly, let the vector $\mathbf{r}$ denote the collection of "distances" between active rows of A and $\mathbf{v}$, that is,

$$r_j = d(A(i_j,:), \mathbf{v}), \tag{16}$$

whenever $u_{i_j} = 1$. Secondly, we calculate the mean μ and standard deviation σ of $\mathbf{r}$. For a fixed $\beta > 0$, we adopt a communal rule that whenever

$$|r_j - \mu| > \beta\sigma, \tag{17}$$

the membership of $A(i_j, :)$ is rejected from the cluster by resetting $u_{i_j} = 0$. It is possible that all active rows will be rejected based on (17), leading to the so called *cluster death* and causing the algorithm to break down. To avoid such a situation, we keep $u_{i_\ell} = 1$ whenever $r_\ell = \min \mathbf{r}$.

The parameter β in (17) serves as a threshold which can affect the ultimate partitions of data matrix. The larger the threshold β is, the more inclusive the cluster become, and the less the number of partitions of A will be. For data with large noises, for instance, increasing β lead to fewer representatives for the rows of A which, in turn, might rid of some of the unwanted substances in the data. On the other hand, if pairwise matching is more important than pairwise distance, we might decrease the β value to increase the uniformity. In our algorithm, the user can adjust the parameter value β according to the need.

3.2 Recursive decomposition

The purpose of the rank-one approximation is to separate rows of A into two mutually exclusive clusters according to innate attributes of each row while trying to find a representative for the active rows. The motive of a filtering by the communal rule is to further refine the active rows to form a tighter cluster. These objectives are repeatedly checked through the following divide-and-conquer procedure:

Step 1. Given a matrix A, assess a possible cluster $\mathbf{u}$ of active rows and its representative $\mathbf{v}$.

Step 2. Based on available $\mathbf{u}$, deflate the matrix A into two submatrices composed of all active and inactive rows, respectively.

Step 3. Recursively apply Step 1 to each submatrix in Step 2 until no more splitting is possible or a predesignated number of iteration is reached.

Step 4. Whenever an action in Step 3 is terminated, record the corresponding $\mathbf{u}$ and $\mathbf{v}$ as an extra column and row in the matrix U and V, respectively.

Algorithm 3 sketches how a simple integer factorization $A \approx UV$ can be executed through VOTE without controlling the size p in the final output matrices $U \in \mathbb{Z}_2^{m\times p}$ and $V \in \mathbb{Z}^{p\times n}$. Strictly speaking, this is not the IMF we have defined in (2) because the ultimate p obtained by Algorithm 3 could be exceedingly large, but it does represent an exhaustive search for factors. In the extreme case, each cluster contains only one member, that is, we have $U = I$ and $V = A$, which of course is of little interest.

Algorithm 3 Integer matrix factorization by VOTE: $[U, V] = \text{IMF}\textsc{vote}(A, active, \beta)$

```
Given
  $A$ = matrix in $\mathbb{Z}^{m\times n}$ to be decomposed,
  $active$ = array of indices identifying
             submatrices of $A$ being analyzed,
  $\beta$ = threshold for communal rule.
return Matrices $U \in \mathbb{Z}_2^{m\times p}$ and $V \in \mathbb{Z}^{p\times n}$ for
some integer $p$ such that $A \approx UV$.
$A \leftarrow A(active, :)$
$\mathbf{u}, \mathbf{v} \leftarrow$ VOTE($A$)
% Check the communality
$uactive \leftarrow find(\mathbf{u})$
for each row $A(i, :)$ of $A(uactive, :)$ do
  if $A(i, :)$ does not meet communal rule then
    $u_{uactive_i} \leftarrow 0$
  end if
end for
% Keep decomposing matrix
$ZeroCheck \leftarrow find(\mathbf{u} = 0)$
if $ZeroCheck$ is not empty then
  $active1 \leftarrow active(find(\mathbf{u} = 1))$
  if $actove1$ is not empty then
    IMFVOTE($A, active1, \beta$)
  end if
  $active0 \leftarrow active(ZeroCheck)$
  IMFVOTE($A, active0, \beta$)
else
  Augment $\mathbf{u}$ and $\mathbf{v}$ in $U$ and $V$ as a column
  and a row, respectively
end if
```

A few remarks about Algorithm 3 are worth noting. Observe that the splitting of A according to $\mathbf{u}$ automatically guarantees that the resulting matrix U has mutually orthogonal columns, which also means that each row of A is assigned to one and only one group. Observe also that the code IMFVOTE invokes itself recursively. Such a feature, allowable in most modern programming languages, makes the code particularly efficient. Finally, be aware of the selection mechanism embedded in the code that determines the final membership $\mathbf{u}$ and representative $\mathbf{v}$ through multiple levels of screening.

3.3 Optimal low rank approximation

In the application of low rank approximations, one of the most challenging issues even to this date is the predetermination of the low rank k. Algorithm 3 calculates an approximation $A \approx UV$ without any restriction on the sizes of

$U \in \mathbb{Z}_2^{m\times p}$ and $V \in \mathbb{Z}^{p\times n}$. Suppose that such a factorization is now at hand. It is natural to ask whether there is a way to pick up two submatrices $U_k \in \mathbb{Z}_2^{m\times k}$ and $V_k \in \mathbb{Z}^{k\times n}$ from U and V such that $A \approx U_k V_k$ and $k < p$. In this section, we make an interesting observation on how to choose the best pair of submatrices (U_k, V_k) so that $d(A, U_k V_k)$ is minimal among all possible submatrices of compatible sizes. Although this is a postscript to the main computation already done by Algorithm 3, it sheds a remarkable insight into the optimal low rank approximation.

Denote the columns and rows of U and V by

$$U = [\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_p], \quad V = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_p \end{bmatrix},$$

respectively. Note that these rank-one approximations $\mathbf{u}_i \mathbf{v}_i$ are found, say, by Algorithm 3, successively without any specific ordering. For $\ell = 1, \ldots, p$, define $\mathcal{S}_\ell$ to be the collection

$$\mathcal{S}_\ell := \left\{ (i_1, \ldots, i_\ell) \,\middle|\, d\left(A, [\mathbf{u}_{i_1}, \cdots, \mathbf{u}_{i_\ell}] \begin{bmatrix} \mathbf{v}_{i_1} \\ \vdots \\ \mathbf{v}_{i_\ell} \end{bmatrix}\right) = \min_{(j_1, \cdots, j_\ell)} d\left(A, [\mathbf{u}_{j_1}, \cdots, \mathbf{u}_{j_\ell}] \begin{bmatrix} \mathbf{v}_{j_1} \\ \vdots \\ \mathbf{v}_{j_\ell} \end{bmatrix}\right) \right\}, \tag{18}$$

where d is either the Hamming or the Euclidean metric and the ℓ-tuple $(j_1, \ldots, j_\ell)$ is made of distinct indices from $\{1, \ldots, p\}$. The folliwng nesting effect among $\mathcal{S}_\ell$'s is rather surprising.

Theorem 5: Suppose that the matrix $A \in \mathbb{Z}^{m\times n}$ has been factorized into $A \approx UV$ with $U \in \mathbb{Z}_2^{m\times p}$ and $V \in \mathbb{Z}^{p\times n}$ by Algorithm 3. Then every element in $\mathcal{S}_s$ must appear as a segment in some element of $\mathcal{S}_t$, if $s < t$.

Proof: It suffices to prove the assertion for the case $s = 1$ and $t = 2$. The following argument can be generalized to other cases. Suppose that there exists an integer $i_1 \in \mathcal{S}_1$ but $\{i_1, i_2\} \notin \mathcal{S}_2$, for any $i_2 \in \{1, 2, \cdots, p\}$. Given any $\{j_1, j_2\} \in \mathcal{S}_2$, then we have

$$d\left(A, [\mathbf{u}_{j_1}, \mathbf{u}_{j_2}] \begin{bmatrix} \mathbf{v}_{j_1} \\ \mathbf{v}_{j_2} \end{bmatrix}\right) < d\left(A, [\mathbf{u}_{i_1}, \mathbf{u}_{j_2}] \begin{bmatrix} \mathbf{v}_{i_1} \\ \mathbf{v}_{j_2} \end{bmatrix}\right). \tag{19}$$

We want to prove that a contradiction arises.

Note that the numeral "1" appears at mutually disjoint positions within the vectors $\mathbf{u}_{i_1}$, $\mathbf{u}_{j_1}$ and $\mathbf{u}_{j_2}$. Simultaneously permuting rows if necessary, we may assume without loss of generality that rows of A have been divided into four blocks

$$\begin{bmatrix} A_1 \\ A_2 \\ A_3 \\ A_4 \end{bmatrix} \tag{20}$$

where rows of A_1, A_2 and A_3 correspond to 1's in $\mathbf{u}_{i_1}$, $\mathbf{u}_{j_1}$ and $\mathbf{u}_{j_2}$, respectively, and A_4 corresponds to the common zeros of all $\mathbf{u}_{i_1}$, $\mathbf{u}_{j_1}$ and $\mathbf{u}_{j_2}$. Then it becomes clear that

$$\begin{aligned} d(A, \mathbf{u}_{i_1}\mathbf{v}_{i_1} + \mathbf{u}_{j_2}\mathbf{v}_{j_2}) &= d(A_1, \mathbf{v}_{i_1}) + d(A_2, 0) \\ &+ d(A_3, \mathbf{v}_{j_2}) + d(A_4, 0), \\ d(A, \mathbf{u}_{j_1}\mathbf{v}_{j_1} + \mathbf{u}_{j_2}\mathbf{v}_{j_2}) &= d(A_1, 0) + d(A_2, \mathbf{v}_{j_1}) \\ &+ d(A_3, \mathbf{v}_{j_2}) + d(A_4, 0), \\ d(A, \mathbf{u}_{i_1}\mathbf{v}_{i_1}) &= d(A_1, \mathbf{v}_{i_1}) + d(A_2, 0) \\ &+ d(A_3, 0) + d(A_4, 0), \\ d(A, \mathbf{u}_{j_1}\mathbf{v}_{j_1}) &= d(A_1, 0) + d(A_2, \mathbf{v}_{j_1}) \\ &+ d(A_3, 0) + d(A_4, 0). \end{aligned}$$

Upon substitution, it follows from (19) that

$$d(A, \mathbf{u}_{j_1}\mathbf{v}_{j_1}) < d(A, \mathbf{u}_{i_1}\mathbf{v}_{i_1}),$$

which contradicts with respect to the assumption that $i_1 \in \mathcal{S}_1$. □

An application of the preceding theorem enables us to generate a reduced version of the UV approximation of A. The procedure is outlined in the following theorem.

Theorem 6: Suppose that the matrix $A \in \mathbb{Z}^{m\times n}$ has been factorized into $A \approx UV$ with $U \in \mathbb{Z}_2^{m\times p}$ and $V \in \mathbb{Z}^{p\times n}$ by Algorithm 3. Sort through the rank-one approximations and rearrange the rows if necessary, assume that

$$d(A, \mathbf{u}_1\mathbf{v}_1) \le d(A, \mathbf{u}_2\mathbf{v}_2) \le \cdots \le d(A, \mathbf{u}_p\mathbf{v}_p). \tag{21}$$

Then, for $k = 1, 2, \cdots, p$, the product $U_k V_k$ where U_k and V_k are submatrices of U and V given by

$$U_k := [\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_k], \quad V_k = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_k \end{bmatrix}, \tag{22}$$

is the best possible approximation to A in the sense that the k-tuple $(1, 2, \cdots, k)$ is in $\mathcal{S}_k$.

Proof: We prove the assertion by induction on k.

Obviously, the case $k = 1$ is already done due to the rearrangement specified in (21). Assume therefore that $(1, 2, \cdots, k-1) \in \mathcal{S}_{k-1}$. We want to show that $(1, 2, \cdots, k) \in \mathcal{S}_k$. The following argument is essentially parallel to that in Theorem 5, except that we work on blocks.

By Theorem 5, there exists an integer $q \in \{k, k+1, \cdots, p\}$ such that $(1, 2, \cdots, k-1, q) \in \mathcal{S}_k$. If $q = k$, then we are done. Assume, by contradiction, that $q \neq k$. Consider the pair of submatrices

$$\tilde{U}_k := [U_{k-1}, \mathbf{u}_q], \quad \tilde{V}_k := \left[\begin{array}{c} V_{k-1} \\ \mathbf{v}_q \end{array}\right].$$

Still, columns of $\tilde{U}_k$ are mutually exclusive. Because $(1, 2, \cdots, k-1, k) \notin \mathcal{S}_k$, we know

$$d(A, \tilde{U}_k\tilde{V}_k) < d(A, U_kV_k),$$

which is equivalent to

$$d(A, U_{k-1}V_{k-1} + \mathbf{u}_q\mathbf{v}_q) < d(A, U_{k-1}V_{k-1} + \mathbf{u}_k\mathbf{v}_k). \tag{23}$$

Again, without loss of generality, we may partition A into blocks such as that in (20) where rows of A_1, A_2 and A_3 correspond to 1's in U_{k-1}, $\mathbf{u}_q$ and $\mathbf{u}_k$, respectively, and A_4 corresponds to the common zeros of all U_{k-1}, $\mathbf{u}_q$ and $\mathbf{u}_k$. Then, clearly we have the following expressions:

$$\begin{aligned}
&d(A, U_{k-1}V_{k-1} + \mathbf{u}_q\mathbf{v}_q) \\
&= d(A_1, V_{k-1}) + d(A_2, \mathbf{v}_q) + d(A_3, 0) + d(A_4, 0), \\
&d(A, U_{k-1}V_{k-1} + \mathbf{u}_k\mathbf{v}_k) \\
&= d(A_1, V_{k-1}) + d(A_2, 0) + d(A_3, \mathbf{v}_k) + d(A_4, 0), \\
&d(A, \mathbf{u}_q\mathbf{v}_q) \\
&= d(A_1, 0) + d(A_2, \mathbf{v}_q) + d(A_3, 0) + d(A_4, 0), \\
&d(A, \mathbf{u}_k\mathbf{v}_k) \\
&= d(A_1, 0) + d(A_2, 0) + d(A_3, \mathbf{v}_k) + d(A_4, 0).
\end{aligned}$$

It follows from (23) that

$$d(A, \mathbf{u}_q\mathbf{v}_q) < d(A, \mathbf{u}_k\mathbf{v}_k),$$

which contradicts the assumption that

$$d(A, \mathbf{u}_q\mathbf{v}_q) \geq d(A, \mathbf{u}_k\mathbf{v}_k),$$

for $q \in \{k+1, \cdots, p\}$. □

In short, even though we still do not know how to find the overall optimal rank, the above discussion suggests that something meaningful can be done after we have "completely" factorize A as $A \approx UV$. That is, by ranking the rank-one approximations as in (21), we can build some lower rank approximations in a controlled way. This byproduct is interesting enough that we summarize it in Algorithm 4.

Algorithm 4 Low_Rank_Optimization: $[U, V] =$ LowRankApprox(A, k)

```
Given matrix $A \in \mathbb{Z}_2^{m\times n}$ and integer $k$.
return Low rank factors $U \in \mathbb{Z}_2^{m\times k}$ and $V \in \mathbb{Z}^{k\times n}$.
$U, V \leftarrow$ IMFvote$(A, active, \beta)$
for all $i$ from 1 to $p$ do
    $\mathbf{r} \leftarrow [\mathbf{r}; \mathsf{match}(A, U(:, i)V(i, :))]$
end for
$index\_rankings \leftarrow sort(\mathbf{r})$
$U \leftarrow U(:, index\_rankings(1:k))$
$V \leftarrow V(index\_rankings(1:k), :)$
```

4 Numerical experiments

In this section, we report some interesting applications of our IMF techniques. We carry out experiments on five data sets, two of which related to cluster analysis are well documented in the literature, another two demonstrate the ability of our algorithms in discovering latent patterns, and the last one is randomly generated to assess the overall performance.

4.1 Cluster analysis

The general purpose of cluster analysis is to partition a set of observations into subsets, called clusters, so that observations in the same cluster share some common features. The most difficult part in cluster analysis is the interpretation of the resulting clusters. Our purpose here is not to compare the performance of our method with the many cluster analysis algorithm already developed in the literature. Rather, we merely want to demonstrate that IMFvote naturally produces clusters and their corresponding integer representatives. If needed, we can also employ LowRankApprox to pick up the most relevant low rank representation of original data set.

We report experiments on two real data sets, mushroom and Wisconsin breast cancer (original), from the *Machine Learning Repository* maintained by the Computer Science Department at the University of California at Irvine [33]. To gauge the effectiveness of our algorithm, we employ four parameters, RowErrorRate, CompressionRatio, Precision, and Recall whose meanings are outline below [34].

RowErrorRate, defined by

$$\text{RowErrorRate} := \frac{d(A, UV)}{m}, \tag{24}$$

refers to the average difference per row between the retrieved data UV and the original data A. This value reflects how effective row vectors in V are representing the original matrix A.

CompressionRatio [35], defined by

$$\begin{aligned}
&\text{CompressionRatio} \\
&:= \frac{\sharp \text{ of entries in matrices } U \text{ and } V}{\sharp \text{ of entries in matrices } A} \\
&= \frac{(m+n)k}{mn}
\end{aligned} \tag{25}$$

measures how efficiently the original data A has been compressed by the low rank representation UV. It is hoped that through the compression, less relevant data or redundant information is removed.

Precision and Recall, defined by

$$\begin{aligned}\text{Precision} &:= \frac{\text{relevant data}\cap\text{retrieved data}}{\text{retrieved data}},\\ \text{Recall} &:= \frac{\text{relevant data}\cap\text{retrieved data}}{\text{relevant data}},\end{aligned}$$

compute the percentages of the retrieved data that are relevant and relevant data that are retrieved, respectively. In the context of information retrieval, "relevant data" usually refer to documents that are relevant to a specified inquiry. Since our task at present is simply to divide observations into disjoint clusters based on innate attributes, we do not know the exact meaning of each cluster. In other words, we do have at hand a representative for each cluster, but we do not have an interpretation of the representative. Still, considering points in a cluster as retrieved data relative to their own representative, it might be interesting to compare the corresponding Precision and Recall with some known training data. In this way, a high correlation between a particular cluster and the training data might suggest an interpretation for the cluster.

Mushroom Data Set. This Agaricus and Lepiota Family data set consists of biological for 8124 hypothetical species of gilled mushroom. Each species is characterized by 23 attributes such as its cap shape, cap surface, odor, and so on. Each attribute has different nominal values. For example, the cap shape may be belled, conical, convex, flat, knobbed, or sunken, while the cap surface may be fibrous, grooved, scaly, or smooth. To fit into our scheme, we convert the attribute information into a list of integer values. These integers should not be regarded as numerals, but only labels. Since attributes are independent of each other, the same integer for different attributes has different meanings. Our input data A is an 8124×23 integer matrix. Of particularly noticeable is its first column which is dichotomous and indicates whether the mushroom is edible or poisonous. The Hamming metric is more suitable than the Euclidean metric for this problem.

In our first experiment, we simply want to investigate how different β values affect the quantities of approximation. As is expected, a smaller β value would lead to a larger number p of rows in the matrix V after a complete factorization. For mushroom data set, we find that corresponding to $\beta = 1, 2, 3$ the numbers of rows in V are $p = 1418, 15, 1$, respectively. For each β, construct V_k according to Theorem 6 after sorting the corresponding V. We plot in Figure 1 the value RowErrorRate versus k. Note the rapidly decreasing behavior, especially in the cases $\beta = 1, 2$, suggesting that the sorting in LowRankApprox is capable of identifying the first few most important clusters and their representatives. A comparison with the CompressionRatio$\approx 0.0436k$ in this case is also worth noting. To achieve RowErrorRate$= 9.6128$, we just need one pattern vector with $\beta = 3$, but we will need many more representatives with $\beta = 1$ or 2. On the other hand, with $\beta = 3$ we cannot possibly improve the RowErrorRate, but with more restrictive β values there is a chance to improve the RowErrorRate.

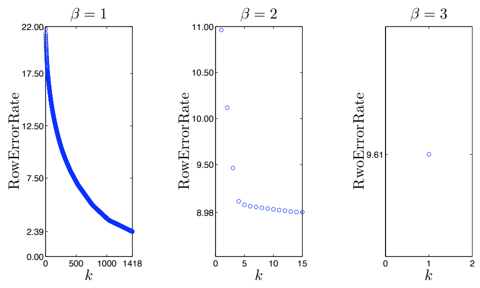

Fig. 1. Performance of algorithms on the mushroom data set

In our second experiment, we apply LowRankApprox to the 8124×22 submatrix after removing the first column from A. Rather surprisingly, by deleting merely one column of A, the resulting IMF behaves very differently. With $\beta = 2$, for example, the complete factorization returns only three clusters with RowErrorRate$= 9.4623$. In an attempt to provide some meaning to these clusters, we construct the so called *confusion matrix* in Table 1 with respect to the attribute of whether the mushroom is edible or poisonous. As is seen, the first cluster represented by $\mathbf{v}_1$ contains large amounts of mushrooms in both kinds, resulting in relative high Recall rates. But the Precision values by $\mathbf{v}_1$ are about the same as the distribution rates in the original data, indicating that the cluster by $\mathbf{v}_1$ does not differentiate edible mushrooms from poisonous ones. The exact meaning of these clusters is yet to be understood.

Wisconsin Breast Cancer Data Set. In a similar way, we experiment our method the Breast Cancer Wisconsin Data. This data set A contains 699 samples, with 458 (65.5%) benign and 241 (34.5%) malignant cases. Each sample is characterized by 11 attributes. The first attribute is dichotomous with labels 2 or 4, indicating benign and malignant tumors. The remaining ten attributes are of integer values ranging from 1 to 10, but some entries in this part of A have missing values for which we assign the value 0 in our computation. It is commonly known that aspects such as the thickness of clumps or

	Retrieved data set		
	$\mathbf{v}_1$	$\mathbf{v}_2$	$\mathbf{v}_3$
Edible	4128	80	0
Poisonous	3789	119	8
Cardinality of $\mathbf{v}_i$	7917	199	8
Precision of edible mushrooms	0.5214	0.4020	0
Recall of edible mushrooms	0.9810	0.0190	0
Precision of poisonous mushrooms	0.4786	0.5980	1
Recall of poisonous mushrooms	0.9676	0.0304	0.0020

TABLE 1
Precision and Recall of edible or poisonous mushrooms

the uniformity of cell shape affect the prognosis. Thus the ordinal of values in attributes matters. It is more appropriate to use the Euclidean metric.

Again, we try out three different communal rules by varying β. With CompressionRatio$\approx 0.1125k$, meaning a reduction of 11% memory space per one less cluster, we plot RowErrorRate versus k in Figure 2. It is seen for this breast cancer data that increasing k would improve RowErrorRate only modestly. Since the RowErrorRate is already low to begin with, a low rank IMF should serve well in approximating the original A.

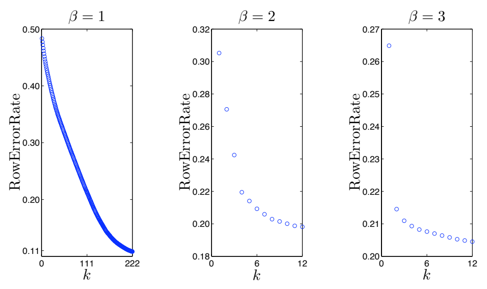

Fig. 2. Performance of algorithms on Wisconsin breast cancer data set

We then apply LowRankApprox with $\beta = 3$ to the 699×10 submatrix by removing the first column of A. A complete factorization returns 11 pattern vectors with most samples being included in the first two clusters. We construct the confusion matrix with respect to the first attribute of A in Table 2. Judging from the Recall and Precision values, we have high level of confidence that the two patterns $\mathbf{v}_1$ and $\mathbf{v}_2$ should be good indicators of whether the tumor is malignant or benigh, respectively. Such a classification of the original 699 samples into 2 major indicators will be extremely useful in medical science.

4.2 Pattern discovery

Most modern image processing techniques treats images as a two-dimensional array composed of pixels. A digital image is an instruction of how to color each pixel. In a gray scale image, for example, the instruction for every element is an integer between 0 and 255 (or a nonnegative decimal number between $[0, 1]$ which requires more storage and often is converted to integers) indicating the intensity of brightness at the corresponding pixel. In this section, let $A \in \mathbf{Z}^{m\times n}$ denote a collection of n images each of which is represented by a column of m pixels. Consider the scenario that images in this library are composite objects of many basic parts which are latent at present. The factorization $A = UV$ then might be suggested as a way to identify and classify those "intrinsic" parts that make up the object being imaged by multiple observations. More specifically, columns of U are the basis elements while each row of V can be thought of as an identification sequence representing the corresponding image in A in terms of the basis elements. This idea has been extensively exploited by NMF techniques. See the many references mentioned earlier. The point to make, nonetheless, is that the NMF techniques cannot guarantee the resulting images to have integer-valued pixels whereas our IMF can.

Needless to say, the same idea can be applied to an extended field of applications, such as the quantitative structure-activity relationship (QSAR) discovery in chemoinformatics. Due to space limitation, we shall illustrate the pattern discovery ability of IMF by using images.

Swimmer Data Set. The "Swimmer" data set characterized in [26] contains a set of black-and-white stick figures satisfying the so called *separable factorial articulation criteria*. Each figure, placed in a frame of 32×32 pixels, is made of "torso" of 12 pixels in the center and four "limbs" of six pixels. Each limb points to one of four directions at its articulation position, so totally there are 256 figures in the collection. Sample images from the Swimmer data set are depicted in Figure 3.

After vectorizing each image into a column, our target matrix A is of size 1024×256. The original data matrix contains only two integer values, 0 and 255, which we convert without loss of generality to binaries. The question is whether we are able to recover the 17 basic parts that make up these swimmers. Taking everything into account, we should also expect one additional part for the background. Such an expectation of $U \in \mathbf{Z}_2^{1024\times 18}$ appears to be problematic because the original matrix A has

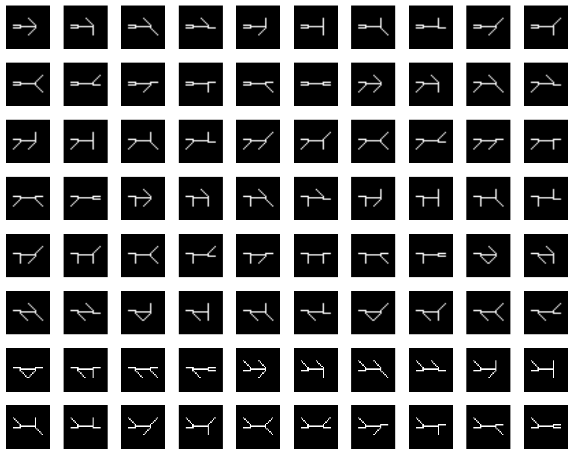

Fig. 3. 80 sample images from the swimmer database.

numerical rank of only 13. In this context, the notion of "low" rank approximation to A really is not appropriate.

After using the Hamming metric and $\beta = 0.5$ in our code IMFVote to carry out a complete factorization, we reshape the 18 columns of the resulting binary U into 32×32 matrices. We recover all 16 limbs, one torso, plus the background as depicted in Figure 4. Note that these 17 recovered "body" parts are completely disjointed from each other and suffer from no blurring at all — a result cannot be accomplished by NMF techniques. In fact, the factors U and V returned from our IMF satisfy $A = UV$ exactly.

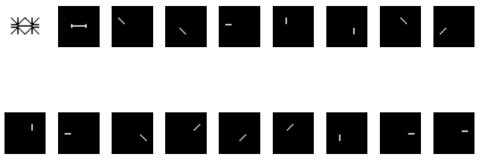

Fig. 4. Basic elements recovered from the swimmer data set.

Block Matrix Data Set. To demonstrate that our method can recognize patterns more complicated than one-dimensional sticks, consider a 5×5 block matrix with each block of size 5×5. Randomly select 2 out of the 25 blocks and assign the value 1 to their entries while keeping all other entries 0. Border this 25×25 matrix with 4 pixels on each side and call the resulting 33×33 matrix an image. Totally there are 300 images. Collect these images into the 1089×300 binary matrix A with each column representing a vectorized 33×33 image. Apply IMFVOTE with Euclidean metric and $\beta = 1$ to A. The resulting complete factorization returns 27 clusters shown in Figure 5. It is interesting to note that 25 basic patterns containing exactly one 5×5 block are completely discovered. Additionally, one pattern representing almost the entire border except the point on the upper left corner is found.

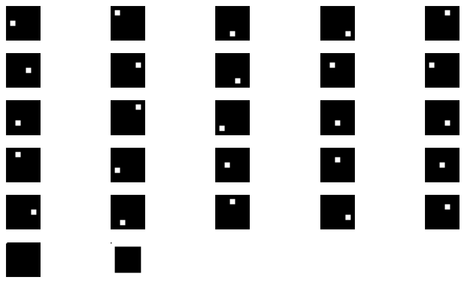

Fig. 5. Bases elements recovered from block matrix data set.

4.3 Random performance test

To further test the capability of the IMF in recovering random clusters or patterns, for a given triplet (m, n, k) of integers we randomly generate $W \in \mathbb{Z}_2^{m\times k}$ and $H \in \mathbb{Z}_{14}^{k\times n}$, where columns of W are kept mutually exclusive and $Z_{14} = \{0, 1, \ldots, 13\}$. Define $A = WH$ and apply IMFVOTE with Euclidean metric and $\beta = 1$ to A.

Let (U, V) denote the pair of factors returned by our calculation. We wonder how likely (U, V) would be the same as the original (W, H) after some permutations. When this happens, we say that our method has reached its optimal performance. Recall that the basic rank one approximation in our scheme is only a local minimizer. We expect that pushing our algorithm, foredoomed just like most optimization techniques for nonlinear problems, to reach its optimal performance would be an extremely challenging task.

For each given (m, n, k), we repeat the above-described experiment 1000 times and tally the rate of success, denoted by OptRate, in reaching the optimal performance. We also measure the CPU time needed for each experiment on a PC running Windows XP and Matlab R2009a with Intel(R)Core(TM)2 Duo CPU T8300@2.4GHz and 3.5GB of RAM. The average CPU time, denoted by AvgTime, then serves as an across-the-board reference for the computational overhead. Test results for a few selected triplets are summarized in Table 3. It seems possible to draw a few general rules from this table. For problems of the same size (m, n), larger k means more complexity and deeper recursion which, in turn, reduce OptRate and cost more AvgTime. For problems of the same (m, k), increasing n costs only AvgTime, but has modest effect on OptRate. For problem of the same (n, k), increasing m also costs only AvgTime and effects little on OptRate. The overall speed of our method seems reasonable, even though our code is yet to be further polished for efficiency by taking into account data structure, storage, and movement within the system.

5 Conclusions

Matrix factorization has many important applications. In this paper, we investigate the notion of factorization with entries restricted to integers or binaries. Being discrete in nature, such a factorization or approximation cannot be accomplished by conventional techniques. Built upon a basic scheme of rank one approximation, we propose an approach that recursively splits (or more correctly, approximates) the underlying matrix into a sum of rank one matrices with discrete entries. We carry out a systematic discussion to address the various computational issues involved in this kind of factorization. The ideas are implemented into an algorithm using either the Hamming or the Frobenius metric, but using other types of metrics is possible.

If the underlying data are binary, our idea is in line with the existing code PROXIMUS. But our formulation is readily generalizable to other types of data. For example, for application purposes we have mainly concentrated on binary-integer factorizations, where the "integer" could be either the regular ordinal integers or just some nominal labels. If the underlying data are regular integers, we have developed the mechanism to perform integer-integer factorizations.

Five different testing data are used to demonstrate the working of our IMF algorithm. Of particular interest is the result in Theorem 6 where we show how an optimal lower rank can be selected after a simple sorting. We think, in a remote sense, the ordering in (21) for discrete data is analogous to the ordering of singular values for continuous data. Similarly, the truncated product $U_k V_k$ defined in (22) is analogous to the truncated singular value decomposition.

We hope that our discussion in this paper offers a unified and effectual avenue of attack on more general factorization problem. There is plenty of room for future research, including a refinement of our algorithm for more efficient data management and a generalization to more complex data types.

Acknowledgment

The authors wish to thank Professor Robert E Hartwig and Mr. Yen-Wei Li for helpful discussions.

References

[1] Lawrence Hubert, Jacqueline Meulman, and Willem Heiser. Two purposes for matrix factorization: a historical appraisal. *SIAM Rev.*, 42(1):68–82 (electronic), 2000.

[2] Moody T. Chu, Robert E. Funderlic, and Gene H. Golub. A rank-one reduction formula and its applications to matrix factorizations. *SIAM Rev.*, 37(4):512–530, 1995.

[3] Louis Guttman. General theory and methods for matric factoring. *Psychometrika*, 9(1):1–16, 1944.

[4] Louis Guttman. Multiple group methods for common-factor analysis: Their basis, computation, and interpretation. *Psychometrika*, 17(2):209–222, 1952.

[5] Louis Guttman. A necessary and sufficient formula for matrix factoring. *Psychometrika*, 22(1):79–81, 1957.

[6] Paul Horst. *Factor Analysis of Data Matrices*. Holt, Rinehart and Winston, New York, 1965.

[7] Harry H. Harman. *Modern Factor Analysis*. University of Chicago Press, Chicago, 1976.

[8] Andrew L. Comrey and Howard B. Lee. *A First Course in Factor Analysis*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1992.

[9] C. Ding, X. He, and H.D. Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *In: Proceedings of the Fifth SIAM International Conference on Data Mining*, Newport Beach, CA, 2005.

[10] W. Xu, X. Liu, and Y. Gong. Document-clustering based on non-negative matrix factorization. In *In: Proceedings of SIGIR03*, pages 267–273, Toronto, CA, July 28–August 1 2003.

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

[12] A. Pascual-Montano, J. M. Carzzo, K. Lochi, D. Lehmann, and R. D. Pascual-Marqui. Nonsmooth nonnegative matrix factorization (nsnmf). *IEEE Transactions on, Pattern Analysis and Machine Intelligence*, 28:403–415, 2006.

[13] Z. Chen, A. Cichocki, and T. M. Rutkowski. Constrained non-negative matrix factorization method for eeg analysis in early detection of alzheimers disease. In *In IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP2006*, Toulouse, France, 2006.

[14] M. Dugas, S. Merk, S. Breit, and P. Dirschedl. Mdclust-exploratory microarray analysis by multidimensional clustering. *Bioinformatics*, 20:931–936, 2004.

[15] P. Paatero and U. Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error. *Environmetrics*, 5:111–126, 1994.

[16] P. Paatero. The multilinear engine–a table-driven, least squares program for solving multilinear problems, including the n-way parallel factor analysis model. *J. of Computational and Graphical Statistics*, 8(4):854–888, 1999.

[17] Michael W. Berry, Murray Browne, Amy N. Langville, Paul V. Pauca, and Robert J. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Computational Statistics and Data Analysis*, 52(1):155–173, 2007.

[18] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems*, volume 13, pages 556–562, 2001.

[19] M. Chu, F. Diele, R. Plemmons, and S. Ragni. Optimality, computation and interpretation of non-negative matrix factorizations. *Available online at http://www4.ncsu.edu/ mtchu/Research/Papers/nnmf.ps*, 2005.

[20] Patrik O. Hoyer. Non-negative matrix factorization with sparseness constraints. *The Journal of Machine Learning Research*, 5:1457–1469, 2004.

[21] D. P. Bertsekas. *Nonlinear programming (2nd)*. Belmont, MA: Athena Scientific., 1999.

[22] Rasmus Bro and Sijmen De Jong. A fast non-negativity-constrained least squares algorithm. *Journal of Chemometrics*, 11(5):393–401, 1997.

[23] Hyunsoo Kim and Haesun Park. Nonnegative matrix factorization based on alternating nonnegativity constrained least squares and active set method. *SIAM J. Matrix Anal. Appl.*, 30(2):713–730, 2008.

[24] Charles L. Lawson and Richard J. Hanson. *Solving least squares problems*, volume 15 of *Classics in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1995. Revised reprint of the 1974 original.

[25] Moody T. Chu and Matthew M. Lin. Low-dimensional polytope approximation and its applications to nonnegative matrix factorization. *SIAM J. Sci. Comput.*, 30(3):1131–1155, 2008.

[26] D. Donoho and V. Stodden. When does nonnegative matrix factorization give a correct decomposition into parts? In *Proc. 17th Ann. Conf. Neural Information Processing Systems*, NIPS, Stanford University, Stanford, CA, 2003, 2003.

[27] P. K. Hopke. *Receptor Modeling for Air Quality Management*. Elsevier, Amsterdam, 1991.

[28] Patrik O. Hoyer. Nonnegative sparse coding. In *Proc. IEEE Workshop Neural Networks for Signal Processing*, Martigny, 2002.

[29] T. Kawamoto, K. Hotta, T. Mishima, J. Fujiki, M. Tanaka, and T Kurita. Estimation of single tones from chord sounds using non-negative matrix factorization. *Neural Network World*, 3:429–436, 2000.

[30] Suvrit Sra and Inderjit S. Dhillon. Nonnegative matrix approximation: Algorithms and applications. Technical report, Deptartment of Computer Sciences, University of Texas at Austin, 2006.

[31] Shona D. Morgan. Cluster analysis in electronic manufacturing. *Ph.D. dissertation, North Carolina State University, Raleigh, NC 27695.*, 2001.

[32] Mehmet Koyutürk, Ananth Grama, and Naren Ramakrishnan. Nonorthogonal decomposition of binary matrices for bounded-error data compression and analysis. *ACM Trans. Math. Software*, 32(1):33–69, 2006.

[33] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[34] G. G Chowdhury. *Introduction to modern information retrieval. 2nd ed.* facet publishing, 2004.

[35] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley-Interscience [John Wiley & Sons], Hoboken, NJ, second edition, 2006.

	Retrieved data set										
	$\mathbf{v}_1$	$\mathbf{v}_2$	$\mathbf{v}_3$	$\mathbf{v}_4$	$\mathbf{v}_5$	$\mathbf{v}_6$	$\mathbf{v}_7$	$\mathbf{v}_8$	$\mathbf{v}_9$	$\mathbf{v}_{10}$	$\mathbf{v}_{11}$
Benign	33	411	1	5	3	0	1	1	1	1	1
Malignant	225	13	0	0	1	1	1	0	0	0	0
Cardinality of $\mathbf{v}_i$	258	424	1	5	4	1	2	1	1	1	1
Precision of benign cancer	0.1279	0.9693	1	1	0.75	0	0.5	1	1	1	1
Recall of benign cancer	0.0721	0.8974	0.0022	0.0109	0.0066	0	0.0022	0.0022	0.0022	0.0022	0.0022
Precision of malignant cancer	0.8721	0.0307	0	0	0.25	1	0.5	0	0	0	0
Recall of malignant cancer	0.9336	0.0539	0	0	0.0041	0.0041	0.0041	0	0	0	0

TABLE 2
Precision and Recall of benign or recall patients

m	500	500	500	500	500	500	500	500	500	1000	1000	1000	1000	1000	1000
n	30	40	50	30	40	50	60	80	100	30	40	50	60	80	100
k	5	6	7	10	12	14	5	6	7	10	12	14	5	6	7
OptRate	.5530	0.3200	.1820	0.0480	.0110	.0010	.5280	.3130	.1450	.0470	.0100	.0020	.5440	.3520	.1740
AvgTime	.0133	.0158	.0178	.0167	.0196	.0224	.0177	.0213	.0252	.0468	.0524	.0575	.0529	.0627	.0724

TABLE 3
Global convergence rate and average time per factorzation (in seconds) on randomly generated data sets.