Assessing Data Mining Results on Matrices with Randomization

Markus Ojala
Helsinki Institute for Information Technology, Aalto University, FI-00076 Aalto, Finland

Abstract—**Randomization is a general technique for evaluating the significance of data analysis results. In randomization-based significance testing, a result is considered to be interesting if it is unlikely to obtain as good result on random data sharing some basic properties with the original data. Recently, the randomization approach has been applied to assess data mining results on binary matrices and limited types of real-valued matrices. In these works, the row and column value distributions are approximately preserved in randomization. However, the previous approaches suffer from various technical and practical shortcomings. In this paper, we give solutions to these problems and introduce a new practical algorithm for randomizing various types of matrices while preserving the row and column value distributions more accurately. We propose a new approach for randomizing matrices containing features measured in different scales. Compared to previous work, our approach can be applied to assess data mining results on different types of real-life matrices containing dissimilar features, nominal values, non-Gaussian value distributions, missing values and sparse structure. We provide an easily usable implementation that does not need problematic manual tuning as theoretically justified parameter values are given. We perform extensive experiments on various real-life datasets showing that our approach produces reasonable results on practically all types of matrices while being easy and fast to use.**

I. Introduction

Large amount of data are nowadays collected from various sources. The data are commonly stored in matrices containing multiple measurements for various items. Often, these measurements are in different scales, making the analysis hard. Nevertheless, the data mining community has produced various methods for finding patterns in such data. Assessing the quality of these patterns is an important question that has lately obtained an increasing interest by data miners.

Traditional statistics has long been considering the issue of significance testing. However, most of the conventional methods are not applicable to assess data mining patterns on matrices. In this paper, we use the randomization-based significance testing approach, where the original structural measure, such as the clustering error of the matrix, is compared to the distribution of structural measures of randomized matrices sharing some basic statistics with the original matrix. If the original result clearly differs from the results on random data, it is considered as interesting; otherwise, it is seen as a random artefact in the data.

Recently, the randomization approach has been applied to binary matrices [1] and real-valued matrices [2]. In these works, the row and column value distributions are preserved in randomization. In binary matrices this equals the number of ones in rows and columns. Only patterns that deviate from this background knowledge are considered to be interesting. For example, a row having high values is not interesting, whereas a large correlation between two observations can be.

In this paper, we generalize the approach further to numerical matrices with dissimilar features measured in different scales. We give solutions to the various problems in the previous approaches and introduce an improved algorithm. The previous real-valued randomization [2] could be used only on datasets having features with similar and smooth value distributions without long tails, missing values or sparse structure. For example, data containing medical measurements, such as height, weight and blood pressure, from several subjects was not supported. This paper generalizes the previous work [1], [2] to a practical method for assessing patterns on various types of real-life matrices.

II. Background

A. Randomization-Based Significance Testing

Let A be an $n \times d$ matrix. Assume that we have performed some data mining task such as clustering on A. We would like to know whether the found pattern, in this case the clustering structure, is just a random artefact in the data or something more interesting. To assess the pattern, we assume that its quality can be summarized with a single number $\mathcal{S}(A)$, called the *structural measure* of A. It can be, e.g., the clustering error or the number of frequent itemsets in A. Any measure can be used as far as larger (or smaller) values of $\mathcal{S}(A)$ correspond to stronger presence of the pattern.

To assess the quality of $\mathcal{S}(A)$, we produce a set $\widehat{\mathcal{A}}$ of k independent and identically distributed randomized samples $\widehat{A}$ sharing some properties with the original matrix. The same data mining method is then applied on each sample $\widehat{A} \in \widehat{\mathcal{A}}$, giving structural measure $\mathcal{S}(\widehat{A})$. The significance of the original result $\mathcal{S}(A)$ is summarized with an *empirical* p*-value*,

$$p = \frac{|\{\widehat{A} \in \widehat{\mathcal{A}} \mid \mathcal{S}(\widehat{A}) \geq \mathcal{S}(A)\}| + 1}{k+1}. \tag{1}$$

The empirical p-value is the proportion of randomized samples $\widehat{A}$ containing a stronger pattern than the original matrix A. Here it is assumed that large values of $\mathcal{S}(A)$ correspond to strong patterns. If the empirical p-value is less than a given threshold α, say $\alpha = 0.05$, the original result is regarded as significant and independent of the basic properties of the data preserved in randomization.

B. Relation to Previous Work

Randomization has been widely applied in significance testing [3], [4]. The randomization of binary matrices is studied by Gionis *et al.* [1]. A result on a binary matrix is considered to be interesting if it is not explained by the row and column sums. To produce random binary matrices, so called *swaps* are used that change a pair of ones with a pair of zeros preserving the constraints. The method given in this paper reduces on binary matrices to the same method as in [1].

The binary approach is generalized to real-valued matrices by Ojala *et al.* [2] by preserving the row and column value distributions approximately in randomization. The main restriction is that the features have to be measured in the same scale and the value distributions should be similar and smooth. In practice, datasets satisfying these constraints are rare. Randomized real-valued matrices are produced in [2] either by controlling the error in the row and column statistics by using Metropolis-Hastings method and local modifications, or by the *SwapDiscretized* method that first discretizes the matrix and performs swaps preserving the discretization, generalizing the idea of binary swaps.

III. Methods

Next, we introduce a new method *SwapConstrained* for solving the following tasks with different types of data.

Task 1 (Similar features)**.** *Given an $n \times d$ real-valued matrix A where the features (columns) are of similar type, generate a matrix $\widehat{A}$ chosen independently and uniformly from the set of $n \times d$ real-valued matrices having approximately the same values in each row and column as A.*

Features being of similar type means that they are measured using the same scale and their value distributions are fairly similar. This is the same task as studied in [2]. The new method preserves the original values in the entire matrix thus, e.g., an integer-valued matrix is integer-valued also after randomization. The following two tasks are new.

Task 2 (Dissimilar features)**.** *Given an $n \times d$ real-valued matrix A where the features are of dissimilar type, generate a matrix $\widehat{A}$ chosen independently and uniformly from the set of $n \times d$ real-valued matrices having approximately the same values in each column and approximately the same column-wise ranks in each row as A.*

The *column-wise rank* of a value is its ordinal number inside the same column. The randomization preserves the distribution of extreme values in each row when the features have equal importance. In this form, Task 2 assumes a correspondence between large values in different features.

Task 3 (Nominal matrix)**.** *Given an $n \times d$ nominal matrix A where the features are of similar type, generate a matrix $\widehat{A}$ chosen independently and uniformly from the set of $n \times d$ nominal matrices having exactly the same nominal values in each row and column as A.*

$$\begin{array}{c|cc|} & i & j \\ \hline & \vdots & \vdots \\ x & \dots\widehat{A}_{xi}\dots & \widehat{A}_{xj}\dots \\ & \vdots & \vdots \\ y & \dots\widehat{A}_{yi}\dots & \widehat{A}_{yj}\dots \\ & \vdots & \vdots \\ \hline \end{array} \quad \Longrightarrow \quad \begin{array}{c|cc|} & i & j \\ \hline & \vdots & \vdots \\ x & \dots\widehat{A}_{yi}\dots & \widehat{A}_{xi}\dots \\ & \vdots & \vdots \\ y & \dots\widehat{A}_{yj}\dots & \widehat{A}_{xj}\dots \\ & \vdots & \vdots \\ \hline \end{array}$$

(a) Swap rotation of four elements $\widehat{A}_{xi}, \widehat{A}_{xj}, \widehat{A}_{yi}, \widehat{A}_{yj}$.

$$\begin{array}{c|cc|} & i & j \\ \hline x & \widehat{R}^i_x \cap \widehat{C}^x_i & \widehat{R}^j_x \cap \widehat{C}^x_j \\ y & \widehat{R}^i_y \cap \widehat{C}^y_i & \widehat{R}^j_y \cap \widehat{C}^y_j \\ \hline \end{array} \quad \Longrightarrow \quad \begin{array}{c|cc|} & i & j \\ \hline x & \widehat{R}^j_x \cap \widehat{C}^y_i & \widehat{R}^i_x \cap \widehat{C}^y_j \\ y & \widehat{R}^j_y \cap \widehat{C}^x_i & \widehat{R}^i_y \cap \widehat{C}^x_j \\ \hline \end{array}$$

(b) Reordered row and column constraints before and after rotation.

Figure 1. A swap rotation with row and column constraints. The rotation of the four elements as shown in (a) is accepted if $\widehat{A}_{xi} \in \widehat{C}^y_j$, $\widehat{A}_{yj} \in \widehat{C}^x_i$, $\widehat{A}_{xj} \in \widehat{R}^i_y$ and $\widehat{A}_{yi} \in \widehat{R}^j_x$. Such modification preserves the given row and column constraints exactly. Constraints are reordered as shown in (b).

A. Constrained Swaps

We introduce a new Markov chain Monte Carlo method for solving Task 1. In Section III-C, we solve Task 2 and Task 3 by using the same algorithm with different preprocessing. The method is based on performing random swaps on the original $n \times d$ matrix A while preserving precalculated constraints for row and column value distributions. The steps form a Markov chain [4] in the space of $n \times d$ matrices satisfying the row and column constraints. A state of the chain after a large number of steps is used as a random sample.

To approximately preserve the values in each row and column of a matrix $A \in \mathbb{R}^{n \times d}$ in randomization, we allow A_{xi} to be replaced in the row x by a value in a tolerance range R^i_x and in the column i by a value in a tolerance range C^x_i. These ranges can be, e.g., $R^i_x = [A_{xi} - \varepsilon_r, A_{xi} + \varepsilon_r]$ and $C^x_i = [A_{xi} - \varepsilon_c, A_{xi} + \varepsilon_c]$ where ε_r and ε_c are parameters for the row and column tolerances. This allows us to preserve the row and column value distributions with different accuracy. Note that the original matrix satisfies $A_{xi} \in R^i_x \cap C^x_i$.

The task is to produce randomized matrices $\widehat{A} \in \mathbb{R}^{n \times d}$ satisfying given row and column constraints R^i_x and C^x_i. A matrix $\widehat{A}$ satisfies the constraints if for each row x we can reorder the original row constraints $R^1_x, \dots, R^d_x$ in an order $\widehat{R}^1_x, \dots, \widehat{R}^d_x$ such that $\widehat{A}_{x1} \in \widehat{R}^1_x, \dots, \widehat{A}_{xd} \in \widehat{R}^d_x$, and similarly for each column i and its constraints $C^1_i, \dots, C^n_i$. Thus, an acceptable randomization $\widehat{A}$ satisfies $\widehat{A}_{xi} \in \widehat{R}^i_x \cap \widehat{C}^x_i$ for some reordered constraints $\widehat{R}^i_x$ and $\widehat{C}^x_i$. At each step we keep record of the ordering of row and column constraints $\widehat{R}^i_x$ and $\widehat{C}^x_i$ that the current randomized matrix $\widehat{A}$ is satisfying.

To obtain such randomizations we use swap rotation depicted in Figure 1 that preserves the row and column constraints while relocating four elements. First, we choose four elements in the intersections of rows x and y and columns i and j as shown in Figure 1a. The basic idea

Algorithm 1 *SwapConstrained*(D, R, C, I)

```
Input: Rank matrix $D$, constraints $R$ and $C$, attempts $I$
Output: Randomization $\widehat{D}$, reordered rank constraints $\widehat{R}, \widehat{C}$
1: $\widehat{D} \leftarrow D$, $\widehat{R} \leftarrow R$, $\widehat{C} \leftarrow C$
2: for $l \leftarrow 1$ to $I$ do
3:     Choose $\widehat{D}_{xi}$ randomly
4:     Choose $\widehat{D}_{yj} \in \widehat{C}^x_i$ randomly
5:     if $x \neq y \wedge i \neq j \wedge \widehat{D}_{xi} \in \widehat{C}^y_j \wedge \widehat{D}_{xj} \in \widehat{R}^i_y \wedge \widehat{D}_{yi} \in \widehat{R}^j_x$ then
6:         $\{\widehat{D}, \widehat{R}, \widehat{C}\} \leftarrow$ SwapRotate$(\widehat{D}, \widehat{R}, \widehat{C}, x, y, i, j)$
7:     end if
8: end for
9: return $\widehat{D}, \widehat{R}, \widehat{C}$
```

behind a swap rotation is that if $\widehat{A}_{xi} \approx \widehat{A}_{yj}$ and $\widehat{A}_{xj} \approx \widehat{A}_{yi}$, the values are approximately preserved in each row and column after the rotation. To use this observation to preserve the row and column constraints exactly, we accept a swap rotation if the elements satisfy the following conditions:

$$\widehat{A}_{xi}, \widehat{A}_{yj} \in \widehat{C}^x_i \cap \widehat{C}^y_j \qquad \text{and} \qquad \widehat{A}_{yi}, \widehat{A}_{xj} \in \widehat{R}^j_x \cap \widehat{R}^i_y.$$

This guarantees that also after a swap rotation the rows x and y and the columns i and j have elements satisfying the given row and column constraints. If the swap rotation is accepted, the ordering of row and column constraints is updated as shown in Figure 1b, i.e., $\widehat{A}_{xi}$ and $\widehat{A}_{yj}$ swap their column constraints and $\widehat{A}_{xj}$ and $\widehat{A}_{yi}$ swap their row constraints. Note that the constraints $\widehat{A}_{xi} \in \widehat{R}^i_x$, $\widehat{A}_{yj} \in \widehat{R}^j_y$, $\widehat{A}_{xj} \in \widehat{C}^x_j$ and $\widehat{A}_{yi} \in \widehat{C}^y_i$ are satisfied both before and after swap rotation.

We can improve the performance by replacing the values by ranks. Let $D \in \mathbb{Z}^{n \times d}$ contain the ordinal numbers, *ranks*, of the values in A sorted in increasing order, i.e., A_{xi} is the D_{xi}:th smallest value in A. The constraints R^i_x and C^x_i are also replaced by the lower and upper bound ranks of the values satisfying the corresponding tolerance ranges. This allows to quickly sample a random element satisfying a given constraint. We keep track on the locations of the elements to be able to recover the current location (x, i) for a given rank. The final method is given in Algorithm 1.

SwapConstrained starts from the original rank matrix D and attempts to perform I swap rotations. The running time is linear in the number of attempts I. Note that the element $\widehat{D}_{yj}$ is chosen in constant time satisfying $\widehat{D}_{yj} \in \widehat{C}^x_i$. As we keep track of the locations, we can first sample a rank uniformly from $\widehat{C}^x_i$ and then recover its location (y, j). *SwapConstrained* produces samples uniformly from all the samples reachable with swap rotations. The proof is similar as for *SwapDiscretized* [2] and is omitted for space.

To produce a set of randomized samples, we form the rank constraints for the original matrix $A \in \mathbb{R}^{n \times d}$ once and randomize it k times with *SwapConstrained*. The randomized samples depend on each other unless the chain has fully converged. We use the technique by Besag and Clifford [5] that guarantees the validity of the empirical p-values regardless of independence. First, we run the chain backwards I steps to produce a new starting state $\widehat{A}_0$ and then start k independent randomizations from $\widehat{A}_0$. In our case running the chain backwards equals running it forwards.

B. Forming Rank Constraints for a Matrix

SwapConstrained can randomize a matrix while preserving given row and column tolerance ranges. Next, we describe approaches for defining the constraints. Good constraints should preserve the values in each row and column accurately while allowing wide flexibility for the method to work. To preserve the row and column value distributions accurately, we study the properties of Kolmogorov-Smirnov test [6] that is a non-parametric test for deciding if a set of values comes from a reference distribution.

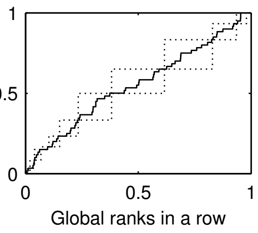
The Kolmogorov-Smirnov test verifies if the values in the original reference column (or row) and in a randomized column (row) are approximately the same. In our case, the test statistic D_n is the maximum absolute difference between two empirical cumulative distribution functions (CDF) each consisting of n observations. The value of a CDF of n observations at point x is the fraction of the n observations that are less than or equal to x. The value of D_n depends only the ordering of the values, not on their scale. The null-hypothesis of the randomized CDF being equal to the reference CDF is rejected with significance level α if $D_n > c_\alpha/\sqrt{n}$, where c_α is a constant depending on α, e.g., $c_{0.05} = 1.36$, $c_{0.01} = 1.63$, $1 = c_{0.27}$, $0.5 = c_{0.96}$ and $0.25 = c_{1.00}$. Now a band of width $c_\alpha/\sqrt{n}$ around the reference CDF contains a CDF of n random samples from the reference distribution with probability $1 - \alpha$. Thus to preserve the value distributions accurately, the tolerance ranges should contain approximately $c_\alpha/\sqrt{n}$ of all the values.

Based directly on the properties of Kolmogorov-Smirnov test, we suggest using constraints where the column and row rank tolerance ranges of value v with global rank e in A consist of fixed number of global neighboring ranks in A, i.e.,

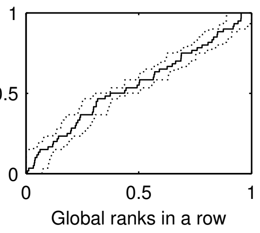
$$C_{\mathit{GNbr}}(e) = \left[e - \frac{sd\sqrt{n}}{2}, e + \frac{sd\sqrt{n}}{2}\right],$$
$$R_{\mathit{GNbr}}(e) = \left[e - \frac{sn\sqrt{d}}{2}, e + \frac{sn\sqrt{d}}{2}\right], \qquad (2)$$

where s is a scaling constant comparable to c_α. If there are equal values, the tolerance ranges are extended to contain all the equal values and the ranges are made equal for ranks with equal value. In the experiments we use $s = 1$, guaranteeing the randomized matrices to have the same values in each row and column with high accuracy. These constraints also work well in the algorithmic respect of *SwapConstrained*.

In the previous approach [2], the discretization where the values are assigned to equal-width bins is studied (*DEps*). It is not justified in a statistical manner for preserving the value distributions, and it does not work if the value distributions



(a) *DEps*: Shared discrete tolerance ranges with fixed global width ε.



(b) *GNbr*: Unique tol. ranges with fixed number of global neighbors.

Figure 2. Cumulative distributions functions of original values (solid) and their tolerance ranges (dotted). The CDFs of rows and columns after randomization are guaranteed to be inside the corresponding tolerance ranges.

have long tails as then practically all values are assigned to the same bin. In Figure 2 we give examples of the *DEps* and *GNbr* constraining approaches. The previous approach could support only discrete bins where the tolerance ranges are shared. The new *SwapConstrained* can also handle unique tolerance ranges for each value thus allowing smoother bounds for the randomized CDF. The *GNbr* constraining approach does not have any problems with non-smooth distributions having narrow peaks or long tails.

C. Dissimilar Features and Nominal Values

So far, we have considered only Task 1. Solving Task 3 with nominal values and similar features is simple. We replace elements having the same nominal values by consecutive ranks and define their row and column tolerance ranges to contain all elements with the same value. After randomizing the original nominal values are returned. For this method to work in practice, the number of different nominal values should be fairly small—otherwise, the matrix cannot be randomized sufficiently, i.e., the corresponding null-hypothesis is too strict. If all the values are unique, only the original matrix satisfies the row and column value distributions exactly, and no data mining result is statistically significant.

To randomize a matrix $A \in \mathbb{R}^{n\times d}$ with dissimilar features (Task 2) we first transform the matrix A into a rank matrix $B \in \mathbb{Z}^{n\times d}$ by replacing the values in each feature (column) by their ordinal numbers inside the corresponding feature. Next, we randomize this new matrix B by using Task 1. That is, we first use constraining approach *GNbr* on B, then call *SwapConstrained*, and finally, return the values to the resulting matrix to obtain $\widehat{B} \in \mathbb{Z}^{n\times d}$. This randomized $\widehat{B}$ can contain some rank multiple times inside one column. We replace each rank with the corresponding value inside the feature thus some values may be repeated and some may be left out, resembling bootstrapping.

D. Sparse Matrices and Missing Values

SwapConstrained can be generalized to handle also sparse matrices and missing values. The missing values are assigned with consecutive high ranks and with equal constraints covering all missing values. Then the number of missing values is preserved in each row and column.

In randomizing sparse matrices, i.e., matrices where most of the values are zeros (or missing values), we preserve the number of zeros in each row and column exactly (sparsity) as well as the given constraints for the nonzero elements (value distributions). The zero and nonzero elements can still change places with each other. We modify the Algorithm 1 to keep track only on the nonzero elements in the matrix. In the tolerance ranges of *GNbr* in Equation (2) $\sqrt{n}$ and $\sqrt{d}$ are replaced by square roots of average number of nonzero elements in columns and rows, respectively. The rank matrix D is replaced by an associative array. For speedup, we choose first a nonzero element $\widehat{D}_{xi}$. To compensate for this, we need to do rotations also in the other direction. To support dissimilar features, column-wise proportional ranks of the nonzero elements should be used in the matrix B.

IV. Analysis

We analyze the mixing time of *SwapConstrained*, i.e., the number of attempts I needed for convergence. Due to space limitations, we omit the details. Let the *GNbr* constraining approach be used with the neighborhood size for the middle elements being $r = sn\sqrt{d}$ for row constraints and $c = sd\sqrt{n}$ for column constraints. Then the acceptance probability of line 5 in Algorithm 1 is $\rho_{GNbr} \approx 4r/9m$, where $m = nd$, if the elements are in random locations. However, in the beginning of randomization the acceptance rate is $\rho^0_{GNbr} \approx r/m$.

We can approximate that *SwapConstrained* has converged when each element has been swapped at least once. Since each swap rotation relocates four elements, and the acceptance rate of *SwapConstrained* with *GNbr* constraints varies between $4r/9m \dots r/m$, sufficient number of attempts is approximately $I_{GNbr} \approx m^2 \ln m/r$. This follows from the properties of Coupon collector's problem. It allows a simple optimization because I_{GNbr} does not depend on the column constraints. If $r < c$, we can apply *SwapConstrained* to the transpose of the matrix, thus reducing the number of attempts to $m^2 \ln m/c$. Therefore, the final running time is $O(\min(\sqrt{n}, \sqrt{d}) nd \ln(nd))$ with *GNbr* constraints.

For sparse matrices the acceptance rate is larger as most of the elements are zeros. Each swap is accepted approximately with probability $2/3$, giving $I^{\text{sparse}}_{GNbr} \approx m \ln m$, where m is now the number of nonzero elements. In practice, we use attempts between I^{sparse}_{GNbr} and I_{GNbr} in proportion to the sparsity.

V. Experiments

A. Convergence and Performance

We analyze the performance of *SwapConstrained* empirically on various real datasets. The datasets Random and Gene were studied in the previous randomization paper [2]. Random is an artificial random data, whereas Gene is a real gene-expression data. Many commonly used datasets are taken from UCI machine learning repository [7]. The sparse dataset Jester contains real-valued ratings for jokes by various users [8]. The different sized sparse MovieLens

Table I
DATASETS AND STATISTICS OF RANDOMIZATION. TYPE: THE RANDOMIZATION TASK APPLIED. ATT.: THE NUMBER OF ATTEMPTS PER ELEMENT. SW.: ACCEPTED ATTEMPTS PER ELEMENT. TIME: RUNNING TIME TO PRODUCE ONE RANDOMIZED SAMPLE. DIST.: DISTANCE MEASURE [2] APPLIED TO RANK MATRICES. DIFF.: PROPORTION OF CHANGED ELEMENTS. RE OR CE: AVERAGE L_1-CDF RANK ERROR IN ROW OR COLUMN VALUE DISTRIBUTIONS MULTIPLIED BY 1000.

Dataset	Rows	Cols	Elms	Type	Att.	Sw.	Time	Dist.	Diff.	RE	CE
RANDOM	100	100	Full	Sim.	94	4.2	0.1s	1.40	1.00	11.7	11.8
GENE	1.4k	60	Full	Sim.	93	6.1	1.8s	1.18	1.00	15.8	1.8
IRIS	150	4	Full	Diss.	14	2.4	0.0s	1.08	1.00	90.1	13.2
WINE	178	13	Full	Diss.	30	3.4	0.0s	1.30	1.00	46.9	8.0
WATER	527	38	Full	Diss.	66	4.5	0.2s	1.29	1.00	20.6	5.7
BREAST	699	9	Full	Nom.	34	9.8	0.0s	0.67	0.99	0.0	0.0
WDBC	569	30	Full	Diss.	56	5.1	0.1s	1.10	1.00	26.5	2.8
WINEQL	6.5k	11	Full	Diss.	40	4.3	0.7s	1.34	1.00	47.5	2.3
LETTER	20k	16	Full	Nom.	130	12.8	28s	0.97	1.00	0.0	0.0
JESTER	64k	150	1.8M	Sim.	21	1.3	43s	1.14	0.98	23.9	0.3
MVLNSS	943	1.6k	100k	Nom.	13	6.4	1.3s	1.33	1.00	0.0	0.0
MVLNSM	6.0k	3.7k	1M	Nom.	15	8.0	25s	1.34	1.00	0.0	0.0
MVLNSL	72k	11k	10M	Nom.	17	8.9	6m1s	1.35	1.00	0.0	0.0

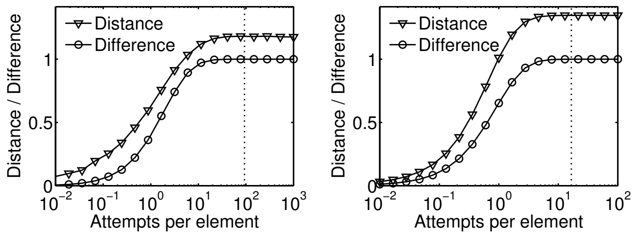
datasets contain 1–5 ratings for movies by users (available at http://www.grouplens.org/node/73).

In Table I, the basic characteristics of the datasets are given with some performance measures of *SwapConstrained*. The datasets GENE (2.0%), WATER (3.0%) and BREAST (0.3%) contain missing values. If the dataset contains only few different values, Task 3 is applied. If the features are dissimilar or the column value distributions differ substantially, Task 2 is applied. The number of attempts I as analyzed in Section IV is used. Producing one randomized sample with *SwapConstrained* is fast even on large matrices; Java implementation on 2.83GHz machine was used.

To analyze the randomness of the samples, we use two measures: rank distance and difference between the original and randomized matrix. The rank distance is the normalized root mean square distance used in [2] applied to the rank matrices ($\approx \sqrt{2}$ for full permutation). Difference is the proportion of elements that has been relocated. The L_1-CDF error for rank matrices is a generalization of the error measure used in [2] to rank matrices. For example, the row error of 0.0158 with GENE dataset means that in average the ranks in a row of randomized matrix have changed 1.58% from the original row rank distribution. We can conclude that *SwapConstrained* produces well-randomized matrices while preserving row and column value distributions accurately.

In Figure 3, convergence of the rank distance and difference randomness measures are shown on datasets GENE and MVLNSL. We note that *SwapConstrained* converges well on the number of attempts analyzed in Section IV. Note that convergence on sparse matrices needs notably less attempts.

Next, we apply different approaches on dataset GENE to compare their performance. The running time on the original dataset is 1.8 seconds. When the implementation for sparse matrices is applied, the running time is 3.9 seconds. The original rank distance is 1.18 and the row and column



(a) GENE (b) MVLNSL

Figure 3. Convergence of two randomness measures with *SwapConstrained*. Dashed line corresponds to the default number of attempts.

errors are 0.0158 and 0.0018. When the approach for dissimilar features is applied, the corresponding values are 1.20, 0.0280, 0.0014, i.e., very close to the original results.

We conclude with short comparison to the results in [2]. The new *SwapConstrained* is faster than the previous methods and preserves the row and column distributions more accurately. Furthermore, no manual tuning of parameters is needed as the new method contains theoretically justified default values for the tolerance ranges and the number of attempts. On all the datasets studied in [2], the new method produced similar significance testing results. Note that the previous methods are not intended to be used on practically any of the datasets studied in this paper as they contain either dissimilar features, too different value distributions for features, non-smooth value distributions, missing values or sparse structure. In practice, very few real-life datasets satisfy the requirements; in [2] mainly artificial datasets are studied to show the usefulness of the concept.

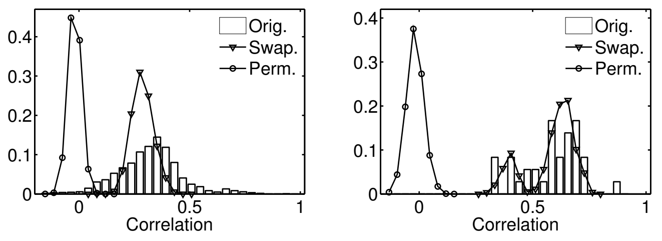
B. Clustering and Principal Component Analysis

Next, we apply some basic data mining methods to the non-sparse datasets and assess the results using *SwapConstrained*. In Table II we give the significance testing results for simple k-means clustering, for k-means clustering on whitened data and for principal component analysis (PCA). The missing values are replaced by column medians after randomization. In whitening, each column is normalized to zero mean and unit variance. As the structural measure we use the L_2 clustering error with k clusters and the fraction of variance explained by the first $\widehat{d}$ principal components.

All the clustering and PCA results are insignificant with the artificial RANDOM dataset. Most of the results on the real datasets are regarded as significant. However, the plain k-means results on BREAST, WINE and WATER are insignificant, i.e., they are explained by the row and column distributions. These datasets have dissimilar features; there is just some feature with large values that describes the obtained clustering on its own. After whitening the clustering structures are clearly significant, i.e., there exists some dependency between the features that disappears in randomization but is important for the clustering structure. This demonstrates well how *SwapConstrained* helps us to remove unreasonable patterns from consideration.

Table II
SIGNIFICANCE TESTING RESULTS FOR k-MEANS AND PCA. THE ORIGINAL STRUCTURAL MEASURES AND THE AVERAGE MEASURES OF 999 RANDOMIZED SAMPLES ARE GIVEN WITH THE p-VALUES.

		k-means			Whit. + k-means			PCA			
Dataset	k	Orig.	Rnd.	p-val.	Orig.	Rnd.	p-val.	$\widehat{d}$	Orig.	Rnd.	p-val.
RANDOM	10	148	147	0.923	8.46k	8.48k	0.252	5	0.17	0.17	0.381
GENE	10	96.7k	115k	0.001	46.7k	55.5k	0.001	5	0.58	0.39	0.001
IRIS	3	78.9	213	0.001	140	338	0.001	2	0.96	0.64	0.001
WINE	3	2.37M	2.36M	0.523	1.27k	1.91k	0.001	5	0.80	0.53	0.001
WATER	10	867M	821M	0.943	11.4k	15.7k	0.001	5	0.61	0.26	0.001
BREAST	2	19.7k	19.7k	0.422	2.80k	2.81k	0.001	4	0.85	0.83	0.001
WDBC	2	77.9M	119M	0.001	11.6k	12.6k	0.001	5	0.85	0.49	0.001
WINEQL	7	1.80M	2.72M	0.001	34k	52k	0.001	5	0.80	0.48	0.001
LETTER	26	616k	1.03M	0.001	122k	205k	0.001	5	0.69	0.43	0.001



(a) GENE (b) BREAST

Figure 4. Distributions of pairwise correlations between the columns in the original and randomized data with *SwapConstrained* and *PermuteColumns*.

C. Pairwise Correlations

Lastly, we study the number of significant correlations between the features. On sparse datasets, we calculate the pairwise correlations between the 100 columns having the highest number of elements. For each original correlation, we calculate an empirical p-value by comparing it to the null distribution of correlations on 99 randomized datasets with *SwapConstrained*. For comparison, we also use the permutation of columns, *PermuteColumns*, that does not preserve the row distributions but preserves the column value distributions exactly. In Figure 4, the distributions of pairwise correlations on original data and randomized data are given. *SwapConstrained* explains most of the pairwise correlations.

In Table III, we give the number of significant pairwise correlations. To correct for multiple hypotheses, we use the approach by Benjamini-Hochberg [9]. It controls the false discovery rate (FDR), i.e., the expected proportion of results incorrectly regarded as significant. We restrict the FDR to 0.05. *PermuteColumns* regards much more pairwise correlations as significant than *SwapConstrained*.

VI. CONCLUSIONS

We have considered the problem of assessing data mining results on various type of matrices by using randomization-based significance testing. A pattern is regarded as interesting if it is not explained by the row and column value distributions. The main contribution of this paper is the generalization of the previous approaches [1], [2] to a practical tool for studying real-life matrices. We proposed a new null-model for matrices containing features measured in different scales; in such matrices, the value distributions of features and the feature-wise rank distributions of observations are preserved. We introduced a new algorithm *SwapConstrained* for randomizing various types of matrices while preserving the constraints more accurately. Compared to the previous approaches, the new method can be used to assess patterns on matrices containing dissimilar features, nominal values, non-smooth value distributions, missing values and sparse structure, thus making it practical for many types of matrices. In addition, the *SwapConstrained* method does not need any manual tuning as theoretically justified parameter values are given. The experiments showed that the method produces reasonable results on real datasets. *SwapConstrained* is a practical and easily usable tool for assessing data mining results on various types of real-life data. The implementation is available at http://www.cis.hut.fi/mrojala/randomization/.

Table III
THE NUMBER OF SIGNIFICANT PAIRWISE CORRELATIONS BETWEEN THE FEATURES. MEDIUM CORRELATION IN THE ORIGINAL AND RANDOMIZED DATA ARE GIVEN. THE B-H THRESHOLD IS THE SMALLEST SIGNIFICANT / THE LARGEST INSIGNIFICANT CORRELATION.

	Orig.	*SwapConstrained*			*PermuteColumns*		
Dataset	Med.	Med.	B-H thresh.	#Sign.	Med.	B-H thresh.	#Sign.
RANDOM	0.00	0.00	— / 0.32	0	0.00	— / 0.32	0
GENE	0.35	0.30	0.42 / 0.42	428	0.00	0.05 / 0.04	1735
IRIS	0.35	0.22	0.82 / -0.11	3	0.00	0.82 / -0.11	3
WINE	0.13	0.09	0.24 / 0.22	30	0.00	0.16 / 0.14	37
WATER	0.07	0.09	0.21 / 0.21	156	0.00	0.09 / 0.09	320
BREAST	0.62	0.63	0.91 / 0.76	1	0.00	0.34 / —	36
WDBC	0.35	0.33	0.43 / 0.42	176	0.00	0.07 / 0.07	368
WINEQL	0.01	0.02	0.06 / 0.04	23	0.00	0.03 / 0.01	27
LETTER	0.03	0.06	0.49 / 0.30	17	0.00	0.01 / 0.01	64
JESTER	0.38	0.33	0.36 / 0.36	3030	0.00	0.04 / 0.03	4944
MVLNSS	0.15	0.14	0.45 / 0.44	53	0.00	0.28 / 0.27	800
MVLNSM	0.14	0.12	0.22 / 0.22	796	0.00	0.08 / 0.08	3812
MVLNSL	0.18	0.14	0.17 / 0.17	2612	0.00	0.03 / 0.02	4718

REFERENCES

[1] A. Gionis, H. Mannila, T. Mielikäinen, and P. Tsaparas, "Assessing data mining results via swap randomization," *ACM Trans. Knowl. Discov. Data*, vol. 1, no. 3, p. 14, 2007.

[2] M. Ojala, N. Vuokko, A. Kallio, N. Haiminen, and H. Mannila, "Randomization methods for assessing data analysis results on real-valued matrices," *Statistical Analysis and Data Mining*, vol. 2, no. 4, pp. 209–230, 2009.

[3] P. Good, *Permutation tests: A Practical Guide to Resampling Methods for Testing Hypotheses.* Springer, 2000.

[4] J. Besag, "MCMC methods for statistical inference," 2004, http://www.ims.nus.edu.sg/Programs/mcmc/files/besag_tl.pdf.

[5] J. Besag and P. Clifford, "Generalized Monte Carlo significance tests," *Biometrika*, vol. 76, no. 4, pp. 633–642, 1989.

[6] F. Massey, "The Kolmogorov-Smirnov test for goodness of fit," *J. Amer. Statistical Assoc.*, vol. 46, no. 253, pp. 68–78, 1951.

[7] A. Asuncion and D. J. Newman, "UCI machine learning repository," http://www.ics.uci.edu/~mlearn/MLRepository.html.

[8] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, "Eigentaste: A constant time collaborative filtering algorithm," *Inf. Retr.*, vol. 4, no. 2, pp. 133–151, 2001.

[9] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: A practical and powerful approach to multiple testing," *J. Roy. Statistical Society*, vol. 57, no. 1, pp. 289–300, 1995.