HELSINKI UNIVERSITY OF TECHNOLOGY
Faculty of Information and Natural Sciences
Degree Programme in Engineering Physics

Markus Ojala

Randomization of real-valued matrices for assessing the significance of data mining results

Master's Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.

Espoo, February 27, 2008

Supervisor: Academy Professor Heikki Mannila
Instructor: Doctor Kai Puolamäki

HELSINKI UNIVERSITY OF TECHNOLOGY ABSTRACT OF MASTER'S THESIS
Faculty of Information and Natural Sciences

Author:	Markus Ojala
Degree Programme:	Engineering Physics
Major subject:	Mathematics
Minor subject:	Computer and Information Science
Title:	Randomization of real-valued matrices for assessing the significance of data mining results
Title in Finnish:	Reaalimatriisien satunnaistaminen tiedonlouhinnan tulosten merkitsevyyden määrittämiseksi
Chair:	T-61 Computer and Information Science
Supervisor:	Academy Professor Heikki Mannila
Instructor:	Doctor Kai Puolamäki

Data mining is the process of analyzing large amounts of data to find out relevant information. Many data mining methods are suitable for analyzing real-valued matrices, which arise naturally in various application areas such as in bioinformatics. In this Master's Thesis the significance testing of data mining results on real-valued matrices is studied.

A randomization-based significance testing is used. A result is considered to be significant if it is unlikely to obtain such a result on randomized data which share some characteristics with the original data. An approach is adopted where the mean values and variances of the rows and columns of a matrix are preserved in randomization. Thus a data mining result is interesting if it is not explained purely by the row and column means and variances of the matrix.

In this thesis, three methods for generating such randomized matrices are developed. The methods are analyzed both theoretically and empirically, and they are shown to be efficient in practice. The performance of the methods is evaluated both on real and generated data. The results imply that the methods are usable in assessing the significance of data mining results.

Pages: vi + 72	Keywords: data mining, significance testing, real-valued matrix, randomization, gene expression

Faculty fills

Approved:	Library code:

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ
Informaatio- ja luonnontieteiden tiedekunta

Tekijä:	Markus Ojala
Koulutusohjelma:	Teknillinen fysiikka
Pääaine:	Matematiikka
Sivuaine:	Informaatiotekniikka
Työn nimi:	Reaalimatriisien satunnaistaminen tiedonlouhinnan tulosten merkitsevyyden määrittämiseksi
Title in English:	Randomization of real-valued matrices for assessing the significance of data mining results
Professuuri:	T-61 Informaatiotekniikka
Työn valvoja:	Akatemiaprofessori Heikki Mannila
Työn ohjaaja:	Tohtori Kai Puolamäki

Tiedonlouhinta on tapa analysoida suuria määriä tietoaineistoa oleellisen tiedon löytämiseksi. Monet tiedonlouhinnan menetelmät soveltuvat reaaliarvoisten matriisien tutkimiseen. Tällaisia matriiseja esiintyy luonnostaan useissa sovelluskohteissa kuten bioinformatiikassa. Tässä diplomityössä tutkitaan reaalimatriiseista saatujen tiedonlouhinnan tulosten merkitsevyyden testausta.

Työssä käytetään satunnaistukseen perustuvaa merkitsevyystestausta. Tulosta pidetään merkitsevänä, jos on epätodennäköistä saada vastaava tulos satunnaistetulla aineistolla, jolla on joitain yhteisiä ominaisuuksia alkuperäisen aineiston kanssa. Työssä omaksutaan lähestymistapa, jossa matriisin rivien ja sarakkeiden keskiarvot ja varianssit säilytetään satunnaistuksessa. Täten tiedonlouhinnan tulos on kiinnostava, jos se ei selity pelkästään matriisin rivien ja sarakkeiden keskiarvoilla ja variansseilla.

Tässä diplomityössä kehitetään kolme menetelmää tällaisten satunnaisten matriisien tuottamiseksi. Menetelmiä analysoidaan sekä teoreettisesti että kokeellisesti, ja niiden näytetään olevan tehokkaita käytännössä. Menetelmien toimintakykyä arvioidaan sekä todellisella että keinotekoisella aineistolla. Työn tulokset näyttävät, että kehitetyt menetelmät ovat käyttökelpoisia tiedonlouhinnan tulosten merkitsevyyden määrittämisessä.

Sivumäärä: vi + 72 Avainsanat: tiedonlouhinta, merkitsevyystestaus, reaalimatriisi, satunnaistus, geenilekspressio

Täytetään tiedekunnassa
Hyväksytty: Kirjasto:

Acknowledgments

This Master's Thesis has been done at the Department of Information and Computer Science at Helsinki University of Technology.

First, I would like to thank my supervisor Academy Professor Heikki Mannila for suggesting this interesting research topic for me and sharing his useful insights. I also wish to thank M.Sc. Aleksi Kallio and M.Sc. Niina Haiminen for collaboration and especially M.Sc. Niko Vuokko for co-operation in developing and analyzing the methods. I am grateful to my instructor Dr. Kai Puolamäki for his support and ideas during the writing process. I also want to thank my colleagues for the pleasant and inspiring working environment.

Finally, I would like to thank my parents, family and friends for their support during my life and studies. Most of all, I want to thank my lovely wife Paula Vahermaa for encouraging me and for making my life worthwhile.

Espoo, February 27, 2008

Markus Ojala

Contents

Chapter 1

Introduction

The technological revolution in the last century has produced many efficient computing and measuring devices. Everyday, a large amount of data is produced and collected in the world. To analyze the data automatically, powerful computer systems and algorithms are needed.

Data mining is the field of research which studies discovering knowledge in databases. Hand *et al.* [22] has given the following definition for it:

> Data mining is the analysis of (often large) observational data sets to find unsuspected relationships and to summarize the data in novel ways that are both understandable and useful to the data owner.

Data mining research has developed several efficient algorithms for automatically extracting knowledge from large masses of data. They are widely used in many other fields of science such as bioinformatics where the traditional methods are not sufficient.

Defining the significance of the obtained results is an important part of science. There exist various traditional statistical methods for significance testing. They are commonly used, for example, to assess the quality of poll results or to decide whether a new drug relieves the symptoms of a disease. However, the significance testing has been given less attention in the data mining community. There are two reasons for this: firstly, the data mining research is still quite a new research field, and secondly the traditional statistical tests are not directly applicable with all data mining tasks.

In this thesis we study and develop methods for significance testing of data mining results [12,16,27,30,31,35,47,49]. In general, the statistical methods can be divided into two categories: analytical and randomization tests. In analytical tests the significance is assessed by theoretical calculations of the distributions. However, the analytical tests are generally applicable only to reasonably simple problems. Thus we will concentrate on randomization tests [6,7,20] which suit in more general tasks and are easier to use.

The basic idea in randomization tests is to compare the original result to results obtained on randomized data which share some statistics with the original data. If the original result differs substantially from the results on randomized data, the original result is assessed to be significant. A simple example of randomization tests is permutation tests which are widely used, for example, in control studies in medical genetics. In such tests the variable describing whether the patient belongs to the case or to the control group is permuted randomly, and the original data analysis is repeated. The original findings are accepted if they are stronger than, for example, 99% of the findings on the randomized datasets. [19]

In this thesis, we restrict our consideration to real-valued matrices. Additionally we require that the rows of a matrix are similar with each other as well as the columns. Thus also the values in the matrix are similar with each other and measured in the same units. Such matrices arise naturally in many research areas such as in bioinformatics. For example, gene expression measurements produce matrices whose rows correspond to genes and columns to sample tissues, and the values in the matrix are the corresponding gene expression levels in the sample tissues. Thus, in this thesis, we study how to assess the significance of data mining results on real-valued matrices by randomization tests.

For the randomization approach we need to choose the uninteresting properties of a matrix, that is, the properties we want to preserve in a randomization. We adopt the approach where the means and variances of the rows and columns in a matrix are considered to be fixed. These describe the general characteristics of the underlying phenomenon while not fixing the special features of the original data. Thus we consider the data mining results on a real-valued matrix to be interesting if it is highly unlikely to obtain as good results on random matrices having the same row and column means and variances as the original matrix.

Therefore, the results are viewed insignificant if they can be explained by the first and second order statistics of the rows and columns of the matrix. Hence, the main computational problem in this thesis is the following:

Problem 1. *Given an $m \times n$ real-valued matrix A, generate a random matrix $\hat{A}$ chosen independently and uniformly from the set of $m \times n$ real-valued matrices having the same row and column means and variances as A.*

We introduce three methods for solving Problem 1 approximately. The methods are based on doing local modifications on the original matrix while preserving the row and column sums and variances. The methods actually produce randomizations which are dependent on each other and share the first and second order statistics only approximately. However, we show that these disadvantages can be compensated and that the methods are usable in practice. We evaluate the performance of the methods both on real and artificial data, and use the approach to measure the significance of a few data mining results. The results imply that the methods work efficiently and are usable in the significance testing in practice.

The idea of randomizing matrices for assessing the significance of data mining results is not a completely new idea. However, as far as we know, no one has studied randomization of real-valued matrices. On the contrary, the binary matrix case, where the values in the matrix are zeros and ones, is extensively studied in statistics, theoretical computer science and application areas [9, 13, 14, 17, 28, 37, 41], and it is computationally quite challenging. Also randomization of contingency tables, where the values are positive integers, is widely studied [10, 13–15, 44, 48]. Our approach of randomizing real-valued matrices while preserving the first and second order statistics of rows and columns is a generalization of swap randomization task of binary matrices where the number of ones in each row and column is preserved [14, 19, 40].

The rest of this thesis is structured as follows. In Chapter 2 we introduce application areas where real-valued matrices arise naturally and where our randomization approach is suitable. Gene expression analysis is discussed in detail and a gene expression dataset used in the experiments is introduced. Additionally, three general data mining methods working on real-valued matrices are introduced. Notice, however that our approach is very general and it can be used

almost with any data mining method. In Chapter 3 we review traditional statistical tests and discuss the problems arising in significance testing. After that the randomization approach in significance testing is discussed. Then we give theoretical backgrounds of Markov chain Monte Carlo methods for sampling from probability distributions. Finally we discuss the problem of randomization of real-valued matrices and give a simple example why the adopted approach is useful.

After the theoretical introductory parts, our main contributions are introduced. In Chapter 4 we present three methods for randomizing real-valued matrices while preserving the row and column sums and variances, and discuss their properties. For consistency, we also introduce a method for binary matrices whose ideas we have generalized to real-valued case [19]. In Chapter 5 our main experimental results are presented. First, some visual examples are shown. After that we study the properties of the methods empirically. Finally, significance testing of various data mining results are performed. In Chapter 6 we summarize the thesis and give conclusions.

Chapter 2

Data mining of real-valued matrices

In this chapter we discuss some research areas where real-valued matrices are used and explain a few data mining methods on them.

2.1 Fields of applications

In this section we introduce some fields of research where our randomization methods are applicable. The main application we are using is gene expression. Thus we explain it in detail in Subsection 2.1.1 and introduce in Subsection 2.1.2 the gene expression dataset GENE we are using in the experiments. Finally, we discuss other fields of applications.

2.1.1 Gene expression

Next we introduce the biological concept *gene expression*, explain how it is measured using microarray technology and discuss general analysis phases of such data. [4]

The phenomenon

Every cell in an individual contains the same set of chromosomes and genes. However, in each cell only a fraction of the genes are active. Those genes are *expressed* in the corresponding cell and they determine the properties of the cell. Different

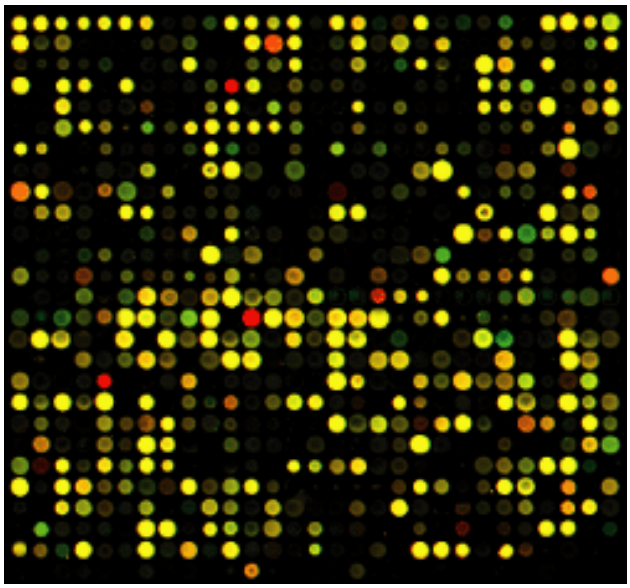

Figure 2.1: A hybridized microarray.

cell types have different genes expressed. Thus, *gene expression* is the process where the information of a gene is converted into the structures and functions of a cell.

The operation of cells is controlled by proteins. The translation from DNA to proteins contains two phases. First, the information in DNA is transcripted into messenger RNA (mRNA). Then, the mRNA controls the production of proteins. The amount of mRNA has a direct impact to the amount of the corresponding protein produced.

The proper expression of a large number of genes is critical to the normal growth and health of an individual. Disruptions in gene expression levels are responsible for many diseases. Identifying the abnormally working genes, for example, in a cancer tissue, can help in developing suitable medication which affects directly the operation of the abnormal genes.

Microarray measurements

The expression of a gene is measured indirectly by measuring the amount of the corresponding mRNA in the cell. In the past decade, the new microarray technology has made it possible to measure the expression levels of a large amount of

genes. Earlier, the scientists were able to measure the expression levels of only few genes.

A *microarray* is a small glass slide containing samples of multiple known genes arranged in a regular pattern. Typically, microarrays are used for measuring relative gene expression levels of multiple genes. First, mRNA is extracted from test and reference samples. The reference sample is taken from normal, healthy tissue. The samples are labeled with different fluorescent dyes, for example, the reference sample with Cy3 (green) and the test sample with Cy5 (red).

Then the samples are combined and hybridized to a microarray; an mRNA molecule has ability to *hybridize*, that is, to bind in the microarray to the specific DNA template from which it originated. After that the microarray is scanned and the fluorescence intensities are measured for each spot. The ratios between the average red and green intensities within a spot are used as numerical values for the expression levels of the corresponding genes.

Thus the result of a microarray is a relative expression level of a gene compared to a normal expression level. This is useful since we want to find the abnormalities in the test sample. In Figure 2.1 an artificial example of microarray measurement is presented. A green spot corresponds to down-regulation in the test sample and, vice versa, a red corresponds to up-regulation. In the yellow spots the test and reference samples are equally expressed. In the black spots the genes are not expressed in either of the samples.

Gene expression analysis

In gene expression analysis usually multiple tissue samples are studied. For each sample the gene expression levels are measured by microarray technology. From the measurements a real-valued matrix is formed where the rows correspond to genes and the columns to different samples. The values in the matrix are the corresponding gene expression levels of the samples [11].

After the microarray measurements the gene expression analysis is carried out in multiple steps. In Figure 2.2 a general analysis pipeline is presented. First the data from microarray measurements is normalized, which contains the calculation of the ratios between the test and reference average intensities. The missing values

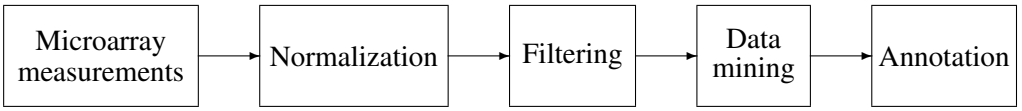

Figure 2.2: The phases of a general gene expression analysis.

are commonly replaced with the average values of the expression levels of the corresponding genes. Standard technique to equalize the influence of up- and down-regulated genes is to take logarithms of the ratios. Also scaling of the values can be done, either globally or locally in genes or in tissues. [43, 46]

Filtering is used to reduce the number of genes. Usually genes with almost no measurements or genes without significant up- or down-regulation in any sample are ignored. This can make the data mining step faster and the results more reliable. However, filtering should not be the "data mining", that is, the purpose is to drop only clearly irrelevant genes.

Data mining is the procedure where new information is extracted from the data. The idea is to find either global or local structures from the gene expression matrix. The data mining methods are discussed more in Section 2.2. However, in this thesis, for simplicity, we are mainly studying global structures. Annotation is the final step where the data mining results are interpreted and classification of genes or samples is done.

Significance testing of the results from a gene expression analysis is an important but usually highly under-weighted part. We are using the randomization methods to assess the significance of the data mining step of the analysis pipeline. However, the methods presented in this thesis could be used to assess the normalization and filtering steps as well.

2.1.2 Gene expression dataset GENE

Next, we introduce a real gene expression dataset GENE which we are using in the experiments in Chapter 5. In addition to this dataset we are using four artificial datasets introduced in Section 5.3. The GENE dataset is a publicly available data[1] which was first studied by Scherf *et al.* [42].

[1] See http://discover.nci.nih.gov/nature2000/.

Scherf *et al.* used actually multiple datasets, but we are studying only the normalized and filtered gene expression data which have 1375 genes and 60 samples. The samples are taken from 60 human cancer cell lines used in a drug discovery screen by the National Cancer Institute. In the original paper also drug activity data was studied and combined with the gene expression data.

The original data contained 9703 genes. The normalization procedure involved identifying missing values: By visual examination the spots contaminated with dust were treated as missing values. Also the spots whose intensity was lower than 1.5 times the local background intensity were considered as missing values. The genes with more than four missing values were filtered. Finally, only the genes with at least four of the red-green ratios being > 2.6 or < 0.38 were accepted. The data was equalized by taking base 2 logarithm.

Around 2% of the values in the normalized and filtered dataset were missing. We replaced them by the average of the values in the corresponding rows. In Figure 2.3 the matrix of GENE dataset is plotted as a heat map. We notice that some samples and genes outstand from the data. In Figure 2.4 the distribution of the values in dataset GENE is presented. As the values of the matrix were equalized by taking base 2 logarithm, the expression level of zero means that the corresponding gene has a normal expression level. We notice that the distribution is centered around zero and is close to a normal distribution. For needs of the randomization methods we finally scaled the values in the matrix linearly into the range [0, 1].

2.1.3 Other applications

Although the only real data we are using in this thesis is the gene expression dataset GENE, there exist various other application areas where real-valued matrices arise naturally. The only restrictions our randomization methods have are that the rows as well as the columns of the matrix should be comparable with each other. Also the distribution of the values should be close to normal.

For example, we could have prices for different products in different stores. However, the products should be similar, for example, fruits. Then the price would be the price per kilogram. The rows of a matrix could correspond to the products,

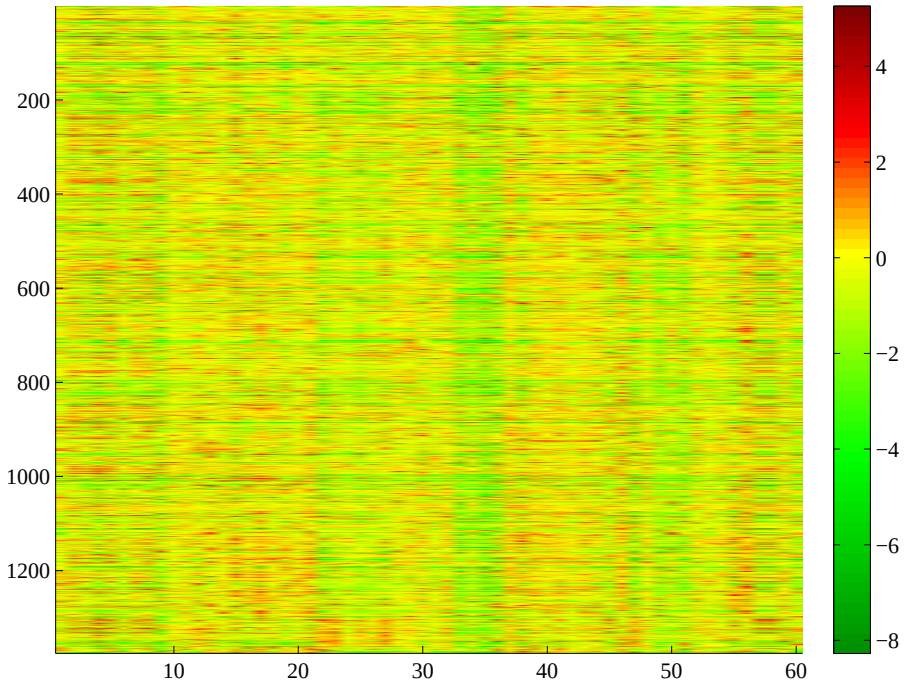

Figure 2.3: The dataset GENE plotted as a heat map.

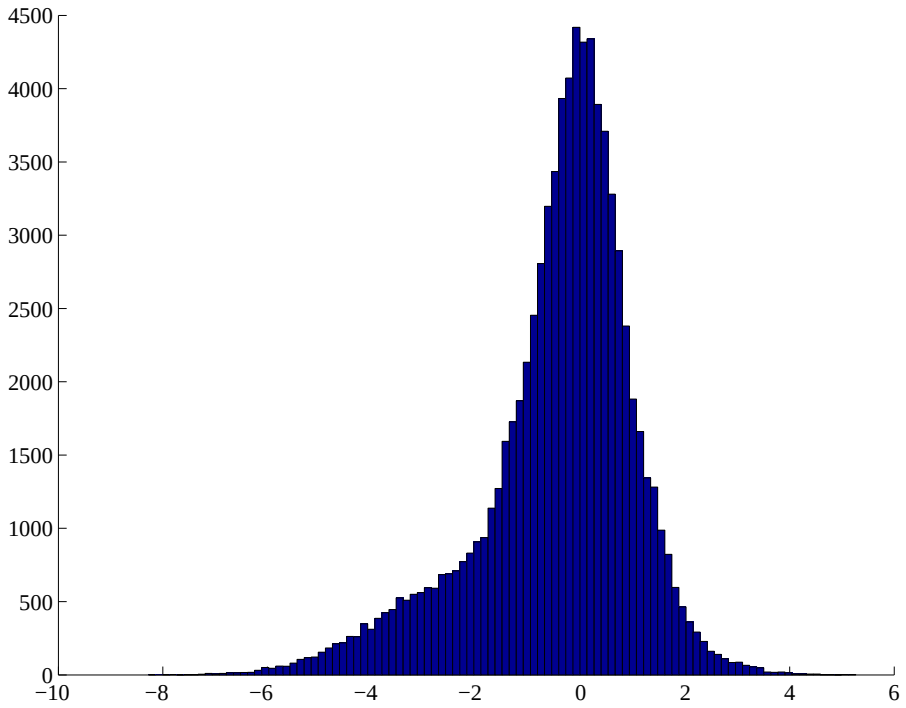

Figure 2.4: The histogram of the values in the dataset GENE with 100 bins.

the columns to the stores and the entries in the matrix to the corresponding prices. The further study of other fields of applications is left for future work.

2.2 Data mining methods

In this section, we introduce some traditional data mining methods which work on real-valued data matrices. Usually, however, the matrix is thought as a set of points: rows correspond to points and columns to dimensions. Thus an $m \times n$ matrix has m points in n-dimensional space. However, in this section we give some examples with GENE dataset using the columns as the data points. Hence the data mining methods work on a set of points; a matrix is just a convenient way to present the data.

In the thesis, we are interested in assessing the data mining results. Generally, data mining methods find structures in matrices. As we want to be able to compare the results, we insist that the structuredness can be expressed in a single real number. However, a full-ordering between structures would be enough but assuming the measures of structuredness to be real-values makes the notation clearer. We give a general definition for the structuredness of a matrix:

Definition 2.1. The *structural measure* of a matrix A, denoted as $\mathcal{S}(A)$, is a real-valued function measuring the amount of the structure in the matrix A found by a data mining method.

We introduce three data mining methods which we use also in the experiments: clustering, correlation and principal component analysis. We explain how they work and form a structural measure $\mathcal{S}$ for each. Usually the structural measure $\mathcal{S}$ is a cost function. Notice, however, that the approach for significance testing explained in this thesis is applicable with almost any data mining method.

2.2.1 Clustering

Clustering is maybe the most classical and generally used data mining method. Assume that $\mathcal{X}$ is a set of m points in $\mathbb{R}^n$. The idea of clustering is to partition $\mathcal{X}$ into subsets so that in each subset the points are somehow similar. The subsets are

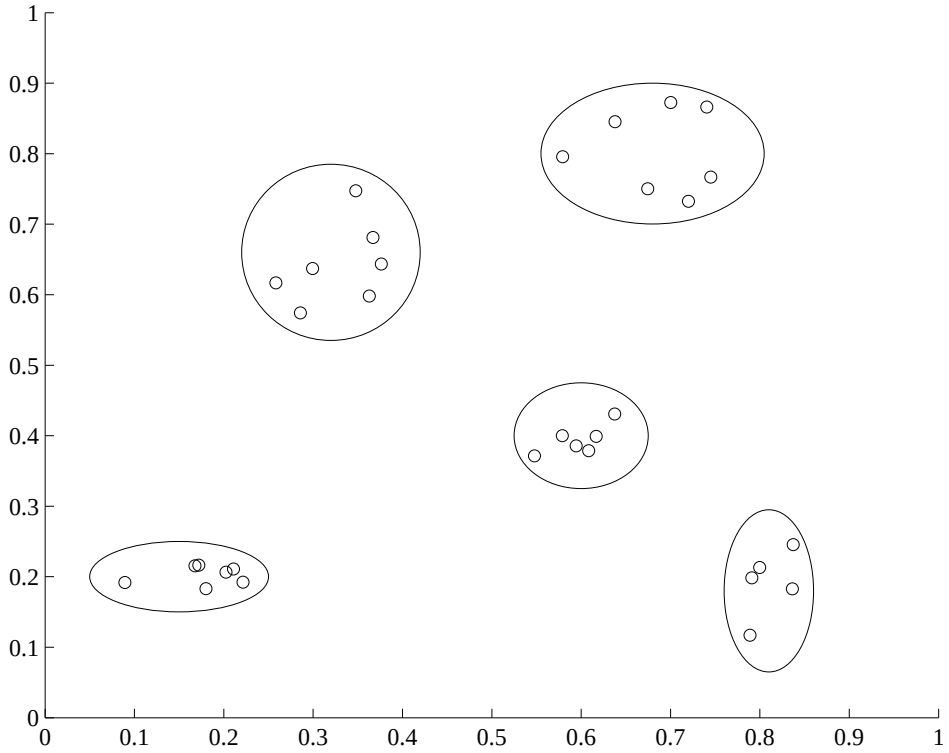

Figure 2.5: An example of a clustering with five clusters in two dimensional space.

called *clusters*. Generally, any distance measure can be used for similarity, but we will adhere to the traditional Euclidean distance. The quality of the clustering is used as the structural measure of the set $\mathcal{X}$ corresponding to an $m \times n$ matrix A.

The number of clusters k is often decided beforehand. There exist methods for finding a suitable k by calculating the clustering for many different values of k and selecting the best k by validation. However, in the experiments, we will use only a fixed k. In Figure 2.5 a clustering with five clusters of artificial data is presented.

We introduce two different clustering approaches, K-means and hierarchical clustering, and give various algorithms for calculating them. In the experiments we are using only the K-means clustering with ten clusters computed by K-means++ algorithm introduced in the following. There exists also so called *bi-clustering* algorithms where the rows and columns of a matrix are clustered simultaneously [34]. To learn more about clustering algorithms, see references [5, 26].

K-means

K-means is a specific problem of clustering. It is also a name of a simple algorithm introduced by Lloyd in 1957 [32] for solving the problem. The task is to find a set $C \subset R^n$ of k points minimizing the error function

$$\phi = \sum_{x \in \mathcal{X}} \min_{c \in C} \|x - c\|^2. \tag{2.1}$$

The clustering error (2.1) can be used directly as the structural measure $\mathcal{S}$. Notice that with such structural measure, smaller value of $\mathcal{S}$ means that the matrix contains actually more structure.

In Algorithm 1 the basic K-means method is presented. It starts by randomly selecting k cluster centers c_i. It repeatedly associates the points into the clusters C_i such that their distances to the corresponding cluster centers c_i are minimized. After that the cluster centers c_i are updated to be the new average of the points in the corresponding cluster C_i.

Algorithm 1 K-means

Input: Set $\mathcal{X}$ of m points in $\mathbb{R}^n$, number of clusters k

1: Randomly pick k cluster centers $C = \{c_1, \ldots, c_k\}$
2: **while** not converged **do**
3: **for** $i \leftarrow 1, k$ **do**
4: $C_i \leftarrow \left\{x \in \mathcal{X} \mid c_i = \arg\min_{c \in C} \|x - c\|\right\}$
5: **end for**
6: **for** $i \leftarrow 1, k$ **do**
7: $c_i \leftarrow \frac{1}{|C_i|} \sum_{x \in C_i} x$
8: **end for**
9: **end while**
10: **return** C

Traditionally, the set C of cluster centers is initialized to be a random subset of the set $\mathcal{X}$. This guarantees that the starting centers are positioned reasonably. Purely random points in $\mathbb{R}^n$ can be used as well but that can lead to empty clusters. However, there exist a better initialization of C which leads to K-means++ algorithm discussed later.

The convergence condition in line 2 of the algorithm is that the partition of

the points into the clusters has not changed in the last iteration. The clustering error (2.1) decreases in each iteration [3], thus guaranteeing the convergence. However, the method usually converges only to a local optimum. Sometimes the convergence is pretty slow [2] and the process is stopped after some predefined number of iterations. Since the result of clustering varies from run to run, the clustering is usually repeated some fixed number of times and the best clustering found, that is, the best according to the clustering error (2.1), is output.

K-means++

K-means++ is a simple modification of the original K-means algorithm which, however, provides a significant improvement to the performance of the clustering. There exist also other methods for solving the K-means problem but they have not gained substantial popularity in practice [23, 33]. The K-means++ was invented by Arthur and Vassilvitskii in 2007 [3], and they have even said: "Friends don't let friends use K-means!". This means that the K-means++ is generally better than the traditional version of the method by Lloyd.

The idea is to modify the initialization of the cluster centers. We choose initial cluster centers step by step. Let $D(x)$ denote the shortest distance from data point $x \in \mathcal{X}$ to the closest center c_i which is already chosen. At each step, we assign a probability for each point in $\mathcal{X}$ for being the next new cluster center. The K-means++ is presented in the Algorithm 2.

Algorithm 2 K-means++

Input: Set $\mathcal{X}$ of m points in $\mathbb{R}^n$, number of clusters k

1: Choose the center c_1 uniformly at random from $\mathcal{X}$
2: **for** $i \leftarrow 2, k$ **do**
3: Choose the center c_i to be $x \in \mathcal{X}$ with probability $\frac{D(x)^2}{\sum_{x \in \mathcal{X}} D(x)^2}$
4: **end for**
5: Proceed as with the standard K-means algorithm from the line 2 on.

The authors of K-means++ have conducted various experiments to show that their algorithm generally produces better clustering and converges faster than the original K-means. Furthermore, for K-means++ there exist theoretical results on the quality of the clustering. With the traditional K-means such results do not

exist and it may, indeed, produce arbitrarily bad results compared to the optimal result. Hence, for the K-means++ we have the following theorem:

Theorem 2.2. *The expected value of the clustering error ϕ of a clustering constructed with K-means++ algorithm satisfies*

$$\mathrm{E}(\phi) \leq 8(\ln k + 2)\phi_{opt}, \tag{2.2}$$

where ϕ_{opt} is the optimal value of the clustering error.

The proof can be found in literature [3]. The initialization procedure just guarantees that we distribute the initial centroids smartly among the points in $\mathcal{X}$.

Hierarchical clustering

The approach of hierarchical clustering differs significantly from K-means. Instead of finding a good clustering with fixed k, hierarchical clustering finds a clustering for *all values* of k. Hence, the number of clusters wanted can be decided afterwards. Hierarchical clustering is especially good for visualizing the clustering structure as explained later.

There exist two kinds of methods for performing hierarchical clustering: agglomerative and divisive. The former builds the clustering by combining current clusters to form new ones whereas the latter divides the clusters to make new ones. Usually only the agglomerative hierarchical clustering methods are used since they are computationally more effective. Therefore, we are not discussing divisive algorithms.

General agglomerative hierarchical clustering is presented in Algorithm 3. The set $\mathcal{C}^k$ is the set of k clusters. The method starts by associating each point in its own cluster thus forming the clustering $\mathcal{C}^m$. At each iteration a clustering $\mathcal{C}^k$ is formed from the previous clustering $\mathcal{C}^{k+1}$ by combining the two clusters which are closest to each other according to some distance measure d.

The only choice in hierarchical clustering is the distance measure d between two clusters. There exist three commonly used measures: minimum, average and maximum distance. The method is called in those cases as single-linkage,

Algorithm 3 Agglomerative hierarchical clustering

Input: Set $\mathcal{X}$ of m points in $\mathbb{R}^n$

1: Initialize $\mathcal{C}^m = \big\{\{x_1\}, \ldots, \{x_m\}\big\}$
2: **for** $k \leftarrow m-1, 1$ **do**
3: Find $C_i, C_j \in \mathcal{C}^{k+1}$ minimizing the distance $d(C_i, C_j)$
4: $\mathcal{C}^k \leftarrow \mathcal{C}^{k+1} \setminus C_i \setminus C_j$
5: $\mathcal{C}^k \leftarrow \mathcal{C}^k \cup \big\{C_i \cup C_j\big\}$
6: **end for**
7: **return** $\mathcal{C}^1, \ldots, \mathcal{C}^m$

average-linkage and complete-linkage hierarchical clustering, respectively. They are defined as

$$d_{\mathrm{SL}}(C_i, C_j) = \min_{x \in C_i, y \in C_j} \|x - y\|, \tag{2.3}$$

$$d_{\mathrm{AL}}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{x \in C_i} \sum_{y \in C_j} \|x - y\|, \tag{2.4}$$

$$d_{\mathrm{CL}}(C_i, C_j) = \max_{x \in C_i, y \in C_j} \|x - y\|. \tag{2.5}$$

The hierarchical clustering process forms a binary tree where the nodes correspond to the clusters and edges to combination relations. Such a tree is called a *dendrogram*. The height of a node corresponds to the distance between the two clusters which were combined to form the new cluster. A dendrogram is an easy way to visualize the clustering. It helps also in deciding the correct number of clusters.

In Figure 2.6 we present an example of a dendrogram which was calculated from the GENE dataset by using the columns as the points. Similar hierarchical clustering was done in the original paper [42]. Remember that the columns correspond to different cell lines, and there are 60 of those in total. We notice that similar cell lines have been clustered together.

With hierarchical clustering there exist a few suitable structural measures. Naturally, we can use the clustering error (2.1) as the structural measure as was done with K-means. However, then we have to fix the number of clusters. Another option is to use the dendrogram to form a structural measure: $\mathcal{S}$ can be the

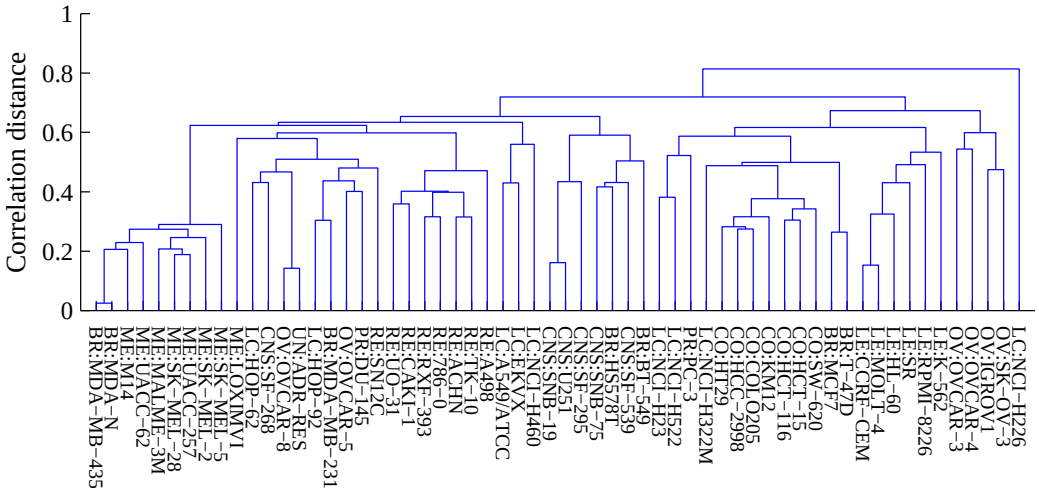

Figure 2.6: A dendrogram of an average-linkage hierarchical clustering of the columns of the GENE dataset with one minus Pearson correlation coefficient as the distance measure.

sum of the heights of the nodes in a dendrogram. This is then a general measure where we do not have to fix the number of clusters.

2.2.2 Correlation

Correlation is an important concept in statistics but it can be seen also as a simple data mining tool. The *correlation* (a.k.a. *correlation coefficient*) indicates the strength of the linear dependence between two random variables X and Y. The Pearson correlation coefficient is defined as

$$\rho_{X,Y} = \frac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{\mathrm{E}\left((X-\mu_X)(Y-\mu_Y)\right)}{\sigma_X \sigma_Y}, \tag{2.6}$$

where μ_X and μ_y are the expected values and σ_X and σ_Y are the standard deviations of X and Y, respectively.

Usually we have just sample measurements and we need to use the Pearson product-moment correlation coefficient for two vectors x and y in $\mathbb{R}^n$:

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{(n-1)s_x s_y} \tag{2.7}$$

where $\bar{x}$, $\bar{y}$, s_x and s_y are the corresponding sample estimates of μ_X, μ_Y, σ_X and σ_Y, respectively. Correlation coefficient varies in the range $[-1, 1]$. If the vectors x and y have a high linear dependence then the correlation coefficient is near to one and they are said to be highly correlated.

The correlation coefficient can be used directly as a structural measure between two vectors. There are couple ways to use the correlation as a structural measure of a real-valued matrix A. First of all, the matrix A is again thought to consist of a set $\mathcal{X}$ of m points in R^n. We define two structural measures, the average absolute correlation $S_{\mathrm{AC}}(A)$ and the maximum correlation $S_{\mathrm{MC}}(A)$ of the matrix A, by using the set $\mathcal{X}$:

$$\mathcal{S}_{\mathrm{AC}}(A) = \frac{1}{m(m-1)} \sum_{\substack{x,y \in \mathcal{X} \\ x \neq y}} |r_{xy}| \tag{2.8}$$

$$\mathcal{S}_{\mathrm{MC}}(A) = \max_{\substack{x,y \in \mathcal{X} \\ x \neq y}} r_{xy} \tag{2.9}$$

In the experiments we are using the maximum correlation $\mathcal{S}_{\mathrm{MC}}$.

2.2.3 Principal component analysis

Principal component analysis (PCA) is again not a pure data mining method. It is a classical linear dimensionality reduction method [25, 39]. However, the quality of the dimensionality reduction can be used as a structural measure. We present how to calculate the projection and to evaluate the quality. For more information about dimensionality reduction methods see references [29].

In derivation of PCA it is useful to present the set of points as a matrix $A \in \mathbb{R}^{m \times n}$ corresponding to the m points in $\mathbb{R}^n$. The goal is to find the best linear projection of the points into a lower dimensional space $\mathbb{R}^d$ in some sense. Usually d is much smaller than n. The PCA can be derived from different view points but we derive it by maximizing the preserved variance under some constraints as introduced by Hotelling in 1933 [25].

We assume that the columns of A have a zero mean. The object is to find an orthonormal axis change $W \in \mathbb{R}^{n \times d}$, that is, $W^T W = I_d$, such that the points

in the projected data $\hat{A} = AW$ are uncorrelated while preserving as much of the variance as possible. The covariance matrix of the projected data $\hat{A}$ is now

$$\begin{aligned} C_{\hat{A}} &= \frac{1}{m}\hat{A}^T\hat{A} \\ &= \frac{1}{m}W^T A^T A W \\ &= W^T C_A W \\ &= W^T V \Lambda V^T W, \end{aligned}$$

where $C_A = V\Lambda V^T$ is the eigenvalue decomposition [45] of the covariance matrix of A. We assume that the eigenvalues are sorted in decreasing order in the diagonal of Λ. Then the maximum variance is obtained by choosing the first d eigenvectors of C_A as the projection, that is,

$$W = V I_{n\times d}, \tag{2.10}$$

which gives

$$C_{\hat{A}} = I_{d\times n}\Lambda I_{n\times d}. \tag{2.11}$$

The method for PCA is presented in Algorithm 4. In the line 1 the columns of A are centered, that is, they are made to have zero mean. Then the eigenvalue decomposition is calculated and the projection is formed. The method outputs also the fraction ρ of variance explained. If it is close to one, the data contains clear inner structure and the intrinsic dimension is really close to d. The value of ρ can be used directly as a structural measure.

Algorithm 4 Principal component analysis

Input: Matrix A in $\mathbb{R}^{m\times n}$, new dimension $d < n$

1: $A \leftarrow A - \frac{1}{m}1_m 1_m^T A$
2: $A = V\Lambda V^T$
3: $W \leftarrow V I_{n\times d}$
4: $\hat{A} \leftarrow AW$
5: $\rho \leftarrow \left(\sum_{i=1}^{d}\lambda_i\right)/\left(\sum_{i=1}^{n}\lambda_i\right)$
6: **return** $\hat{A}$, ρ

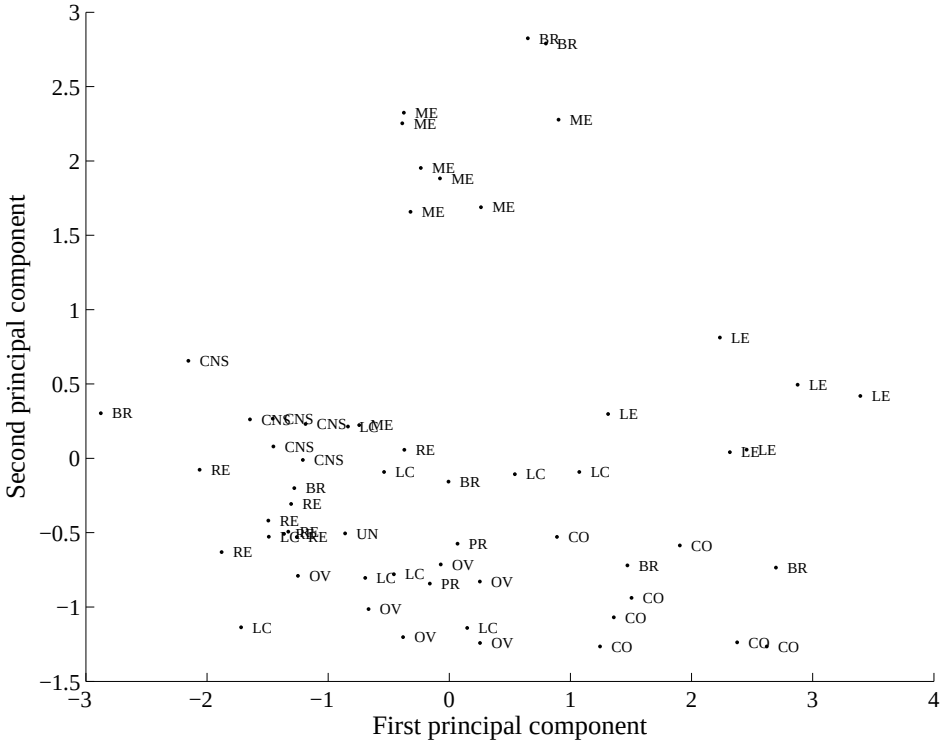

Figure 2.7: The linear projection of the columns of GENE dataset to two dimensions with principal component analysis. The names of the cell lines are abbreviated for visual clarity.

To get some idea of the power of PCA, we have projected columns of the GENE data matrix into two dimensions. The result is shown in Figure 2.7. Remember that the original dimension was 1375. To reduce overlapping, we give only part of the names of the cell lines. Comparing the result with the results of hierarchical clustering in Figure 2.6 suggests that some of the structure is really preserved in the projection. However, only $\rho = 11.5\%$ of variance was explained by the first two principal components.

Chapter 3

Significance testing

In this chapter we first introduce traditional statistical tests and randomization tests for assessing the significance of a result. A result is called *significant* if it is unlikely to have occurred purely by chance. We use a *p-value* to measure the significance level. Then, we discuss generally how to produce random samples for needs of statistical tests. Finally, we study how the significance testing of data mining results on real-valued matrices should be done, and give an example of usefulness of randomization. The discussion is based on references [1, 6, 18, 21].

3.1 Traditional statistical tests

First, we present the general structure of traditional statistical tests. Then we give an example of using the approach in a simple case. Finally, we discuss the problems occurring in significance testing: the two different types of statistical error and the multiple-testing problem.

3.1.1 Overview

Traditional significance testing is based on assessing hypothesis of simple models. In practice, we have to make generalizing assumptions of the true phenomenon to fit it into the theoretical models that we can study by traditional methods.

Consider an example where the expression levels of two genes A and B are studied. We want to know whether the expression levels of genes A and B differ

significantly from each other. For that purpose, we have measured the gene expression levels of A and B in n similar tissue samples. In a simple approach, we can calculate the average expression levels of A and B and compare them directly with each other. However, if, for instance, the average levels are 2.1 and 2.5, respectively, it is incorrect to say that B is more expressed as A without further study, since that may occur purely by chance.

Instead, we have to formulate a precise hypothesis that we will test. We are interested whether the expression levels of A and B differ in all possible samples, not only in the sample set which we have for solving the problem. Thus the question is that how probable it is that the expression levels of A and B equal based on the sample set collected. This is expressed formally in a *null-hypothesis* H_0, which can be tested empirically:

H_0: The two sample sets are taken from collections with equal means.

The *alternative hypothesis* H_1 constitutes the opposite claim:

H_1: The sample sets are taken from collections with different means.

This does not state which one of A and B is larger. Another alternative hypothesis H_1 fixing this is

H_1: The sample set of A is taken from a collection with larger mean.

The validity of a null-hypothesis is assessed by applying an appropriate statistical test. There exist various statistical tests for different purposes. A couple of common statistical tests for testing the equality of means are summarized in Table 3.1. In each test a *test statistic* is calculated which is then compared to its theoretical distribution under the null-hypothesis. In Table 3.1 two different types of tests are presented: z- and t-test. The test statistic z follows a standard normal distribution whereas t follows a Student's t-distribution [21], which resembles a normal distribution in its shape. Student's t-distribution has a parameter f which is called the number of *degrees of freedom*.

From the value of the test statistic we can calculate a *p-value* which gives the probability of getting a result as extreme as the one obtained under the null-hypothesis. If the p-value is less than a *significance level* α decided in advance,

Name	Test statistic	Assumptions
One-sample z-test	$z = \frac{\bar{x}-\mu_0}{\frac{\sigma}{\sqrt{n}}}$	Normal distribution or $n > 30$, σ known
Two-sample z-test	$z = \frac{(\bar{x}_A-\bar{x}_B)-(\mu_A-\mu_B)}{\sqrt{\frac{\sigma_A^2}{n_A}+\frac{\sigma_B^2}{n_B}}}$	Normal distribution, σ_A and σ_B known
One-sample t-test	$t = \frac{\bar{x}-\mu_0}{\frac{s}{\sqrt{n}}}$, $f = n-1$	Normal distribution and σ unknown
Two-sample t-test	$t = \frac{\bar{x}_A-\bar{x}_B}{\sqrt{\frac{s_A^2}{n_A}+\frac{s_B^2}{n_B}}}$, $f = \frac{(s_A^2/n_A+s_B^2/n_B)^2}{\frac{(s_A^2/n_A)^2}{n_A-1}+\frac{(s_B^2/n_B)^2}{n_B-1}}$	Normal distribution, σ_A and σ_B unknown and unequal

Table 3.1: Test statistics for testing whether means are equal. The variable z follows standard normal distribution and t follows Student's t-distribution with degree of freedom f. Variables $\bar{x}$ and s are the sample mean and standard deviation, respectively, and μ and σ are the corresponding true values, respectively.

we can conclude that the result is unlikely to happen under null-hypothesis, and thus we can reject the null-hypothesis H_0 and accept the alternative hypothesis H_1. The significance level is commonly chosen as 0.05, 0.01 or 0.001. The test can be either one-tailed or two-tailed depending on the alternative hypothesis. If we make no difference between $A > B$ and $B > A$ the test is two-tailed, and conversely.

The outline of traditional statistical significance testing is summarized below:

1. Collect data
2. State a null-hypothesis H_0 and an alternative hypothesis H_1
3. Choose a significance level α
4. Select a test statistic with valid assumptions
5. Calculate a p-value
6. If $p < \alpha$, reject the null-hypothesis H_0

3.1.2 Example

We continue the example introduced in the previous subsection. Assume we have measured the gene expression levels of genes A and B from $n = 10$ tissue samples. Let the measured expression values for gene A and B be

$$A: \quad 2.42, 1.21, 2.13, 1.72, 1.25, 2.65, 2.83, 2.22, 1.55, 3.31$$
$$B: \quad 2.68, 2.20, 2.07, 2.83, 3.24, 2.09, 1.86, 3.36, 2.55, 2.10.$$

The mean values of the measured expression levels of genes A and B are $\bar{x}_A = 2.13$ and $\bar{x}_B = 2.50$, respectively. We want to test whether the difference in their means is significant. Thus we formulate a null-hypothesis H_0 whose significance we shall test:

$$H_0: \quad \mu_A = \mu_B,$$

where the μ_A and μ_B denotes the true, hidden means of gene expression levels of A and B. We shall use one-tailed test, where the alternative hypothesis is

$$H_1: \quad \mu_B > \mu_A.$$

We choose a significance level $\alpha = 0.05$. We may assume that the distribution of the gene expression levels is a normal distribution. The normality assumption may not be true but usually it is justified. As we do not know the true variances of expression levels of A and B, we must use two-sample t-test. The sample deviations are $s_A = 0.70$ and $s_B = 0.52$, thus we get the following values for the test statistics by using the equations in Table 3.1: $t = -1.34$, $f = 16.67$.

The p-value is obtained from the cumulative distribution function of Student's t-distribution with degree of freedom $f = 16.67$. At the point $t = -1.34$, the cumulative distribution has a value $p = 0.099$ which is the one-tailed p-value of our test. Since $p = 0.099 > 0.05 = \alpha$, we cannot reject the null-hypothesis. Thus we cannot say that the means $\bar{x}_A$ and $\bar{x}_B$ differ significantly. However, we can say that with p-value 0.099 the gene expression level of B is higher than A.

As the two datasets were artificial, we know their true, hidden distributions. Both were drawn from normal distributions with parameters $\mu_A = 2.1$, $\sigma_A = 0.5$, $\mu_B = 2.5$ and $\sigma_B = 0.5$. Thus we made a wrong conclusion! The difference was just not significant enough that we could statistically be sure about it.

3.1.3 Statistical error

An error produced by randomness is called *statistical error*. In statistical significance testing we distinguish two types of error: type I and type II. They are defined as rejecting or failing to reject a null-hypothesis incorrectly, respectively.

Type I error is also known as *false positive*: we reject the null-hypothesis when it is actually true. Thus we notice a difference in the test variables although it is only produced by chance. For example, if we claim a drug to relieve symptoms of a disease when it actually does not affect at all, we make an error of Type I.

Error of type II, also known as *false negative*, is just the opposite: we accept a null-hypothesis when the alternative would be true, thus we fail to reject the null hypothesis. The difference may be so small that our test statistic cannot distinguish it, or the observed difference is by chance too small. In the example in the previous subsection we made a type II error. Notice, however, that generally we cannot know whether our conclusion is right or wrong as we do not have any additional hidden knowledge.

The selection of significance level α of a test affects directly the probability of errors of type I and II. If the significance level is $\alpha = 0.05$, in average one out of 20 statistical tests, where the null hypothesis is true, is incorrectly regarded as significant. With significance level $\alpha = 0.01$ the same happens in average in one out of 100 tests.

However, when the probability of type I error is decreased by decreasing the significance level α, the probability of type II error is increased at the same time. Thus the selection of significance level is a compromise between probabilities of type I and II errors. Nevertheless, increasing the sample size also decreases the probability of making type I or type II error. The easiest way to make the results more reliable is to use a larger sample set.

3.1.4 Multiple testing

Consider a data mining task where we are interested in finding a pair of genes which have a high correlation between them. It is tempting to try all pairs, find the pair having the highest correlation and apply directly some statistical test for

assessing the significance of the correlation between them. Furthermore, it is likely that the pair is found to be significantly correlated when the statistical test is applied as described in Subsection 3.1.1.

The problem in this approach is that we are actually doing multiple tests at the same time and we should take this into account in the statistical test. For example, if we test 1000 independent null-hypotheses, it is likely that at least one of them is found to be false with significance level 0.001 even if they all are correct. If we test n independent, correct null-hypotheses with significance level α, the probability of getting at least one false positive is $1-(1-\alpha)^n$. With $n = 1000$ and $\alpha = 0.001$ the probability is 0.6323. In general, a random sample set is likely to contain some samples which seem significantly different from others.

Thus we have to pay attention when combining data mining and statistical analysis. If the null-hypothesis is formed based on the data mining task, a plain significance test may produce incorrect results. If, however, the data mining and the null-hypothesis are independent from each other, statistical tests can be applied freely.

There are a couple of different approaches to overcome the multiple-testing problem. If we want to do multiple tests, we can do a Bonferroni correction to the significance level α: If we do n independent tests, we should use significance level $\alpha' = \alpha/n$ instead of α. This guarantees that the probability of getting a false positive is less than α since

$$1 - (1 - \alpha')^n < 1 - (1 - \alpha' n) = \alpha.$$

The problem with Bonferroni correction is that it gives very pessimistic results. Often it is also hard to find the correct number of independent hypotheses.

If we are not interested in a specific hypothesis, we can combine the multiple tests in a single test and use standard significance testing for the new test statistic. For example, we can measure the *average* correlation value between pairs of genes. If this differs significantly from expected, we can conclude that the sample set in itself contains significant correlation. However, deciding which pairs of genes are significantly correlated is still impossible.

Randomization tests partly overcome the problem of multiple testing as the

same data mining task is performed also to randomized samples. Randomization approach is discussed in the next section.

3.2 Applying randomization in significance testing

In this section we introduce how randomization can be used in significance testing [6, 7, 20]. In traditional tests the distribution of the underlying phenomenon is studied theoretically. Even with fairly simple probability distributions this can be impossible without using harsh approximations. In randomization, the basic idea is to use *Monte Carlo simulation*, that is, to draw random, independent samples from the probability distribution of the null-hypothesis and to use the sample set as an approximation of the underlying probability distribution.

Although Monte Carlo simulation is applicable to various other cases, we use it only to approximately calculate the p-value. To get an idea of other applications, Monte Carlo simulation can be used, for example, to calculate the mean and variance of a probability distribution π just by approximating them by the mean and variance of a random sample set drawn from π, respectively.

3.2.1 Empirical p-values

Let A be our original sample. We want to assess the significance of a data mining result on A. We assume that the data mining result can be described by one number which we call the *structural measure* of A, written $\mathcal{S}(A)$. Some structural measures were introduced in Section 2.2. The idea is to compare the original result against the structural measures of randomized samples. For the moment, we assume that we are able to draw independent random samples $\hat{A}$ which share some statistics with A.

Let $\hat{\mathcal{A}} = \{\hat{A}_1, \ldots, \hat{A}_k\}$ be a set of independent randomizations of A. Then the one-tailed *empirical p-value* of the structural measure $\mathcal{S}(A)$, with the hypothesis of $\mathcal{S}(A)$ being small, is

$$\frac{\left|\left\{\hat{A} \in \hat{\mathcal{A}} \mid \mathcal{S}(\hat{A}) \leq \mathcal{S}(A)\right\}\right| + 1}{k+1}. \tag{3.1}$$

This captures the fraction of randomized samples that have a smaller value of the structural measure than the original data A. The one-tailed empirical p-value with the hypothesis of $\mathcal{S}(A)$ being large, and the two-tailed empirical p-value are defined similarly. The expectation value of the empirical p-value equals the traditional p-value. Thus, if the p-value is small, we can say that the structural measure of the original data is significant as it differs substantially from structural measures of random samples.

More formally, we define a probability distribution π where the randomizations are drawn. We are testing a null-hypothesis that A is from π. In the real-valued matrix case, the original data A corresponds to the original real-valued matrix. The π is ideally a uniform probability distribution among all the real-valued matrices sharing the same row and column sums and variance with A. The null-hypothesis corresponds to the case that the structural measure of A can be explained purely by the first and second order statistics of rows and columns. A small p-value implies that the structural measure is not due to the row and column sums and variances.

In Section 3.3 we describe some general approaches for sampling from probability distribution π by using Markov chains and discuss how the independence requirement for the samples can be relaxed. In Chapter 4 we further develop the general approaches into randomization of real-valued matrices.

3.2.2 Sequential tests

Occasionally, we could accept the null-hypothesis by using only a couple of the first samples instead of all the k samples. In those cases, there is no reason to produce all the k samples as it can be time consuming. We introduce a sequential version of calculation of the p-value as developed by Besag *et al.* [6, 8].

In addition to specifying the maximum number k of samples, a minimum number h of samples, typically 10–20, is given. Assume that the hypothesis is that $\mathcal{S}(A)$ is small. Then random samples $\hat{A}_1, \hat{A}_2, \ldots$ are drawn until either h of the structural measures $\mathcal{S}(\hat{A}_i)$ of the drawn randomized samples $\hat{A}_i$ are smaller than $\mathcal{S}(A)$ or until all k samples are drawn. In the former case the p-value is $(h + 1)/(l + 1)$ where l is the number of samples drawn. In the latter case the

normal definition for p-value in Equation 3.1 can be used.

To see that the truncation produces correct p-values, let L be the random variable of the number of samples needed for h exceedings. Then the corresponding random variable for the p-value is $P = (h+1)/(L+1)$. Now under the null-hypothesis

$$\Pr\left(P \leq \frac{h+1}{l+1}\right) = \Pr(L \geq l) = \Pr(L > l-1) = \frac{h+1}{l+1},$$

since $\mathcal{S}(A)$ has to be among the bottom $h+1$ structural measures of l randomized samples and the original sample, in total $l+1$ samples. Thus under the null-hypothesis the random variable P gets correct values. Hence, the sequential calculation of p-value is correct.

For example, if $k = 1000$ and $h = 20$, the expected sample size is reduced to 98 if the null-hypothesis is correct [8]. Notice, however, that sequential tests produce speed up only when the null-hypothesis holds. Furthermore, they are best suited for preliminary analysis of the data. Thus, in the experiments we are fully producing the samples.

3.3 Sampling from probability distributions

In this chapter, we discuss how to generate samples from a given probability distribution π for calculation of empirical p-values. We introduce a stochastic concept *Markov chain*, and develop sampling methods based on it. Finally, we discuss how the sampling should be done in order to get valid empirical p-values. For excellent introduction to Markov chain and its applications, consult references [1,6].

3.3.1 Markov chains

A Markov chain is a discrete-time stochastic process where the next state depends only on the current state. More formally, a sequence of random variables $X_1, X_2, \ldots$ is called a *Markov chain* if it fulfills the *Markov property*

$$\Pr(X_{n+1} = x \mid X_n = x_n, \ldots, X_1 = x_1) = \Pr(X_{n+1} = x \mid X_n = x_n). \quad (3.2)$$

Usually we are only interested in *time-homogeneous* Markov chains where the transition probabilities do not depend on time, that is,

$$\Pr(X_{n+1} = x \mid X_n = y) = \Pr(X_n = x \mid X_{n-1} = y) \tag{3.3}$$

for all n. Later on, we will implicitly assume all Markov chains to be time-homogeneous. In case of finite state space S we use the *transition probability matrix* P whose element P_{ij} is the probability of moving from state i to state j. A Markov chain is said to be *connected* (or *irreducible*) if every state is reachable in finite number of steps from every other state. A connected chain is called *aperiodic* (or *acyclic*) if for all two states i and j there exist a time n_{ij} such that $P_{ij}^{(n)} > 0$ for all $n \geq n_{ij}$, where $P_{ij}^{(n)} = (P^n)_{ij}$ is the probability to be in state j after n steps when starting from state i. The *stationary distribution* of a time-homogeneous Markov chain is the distribution where the process converges. The probability vector π of the stationary distribution fulfills *general balance*

$$\pi^T = \pi^T P. \tag{3.4}$$

An important special case is a *time-reversible* Markov chain for which we cannot identify whether the chain is running forwards or backwards. Note that the *reversed chain*, for example, $X_{m+1}, X_m, X_{m-1}, \ldots$, also fulfills the Markov property 3.2 and is thus a Markov chain itself, since if X is independent of Y then Y is also independent of X. Let R be the transition probability matrix of the reversed Markov chain. Then a Markov chain is *time-reversible* if $R_{ij} = P_{ij}$.

For the transition probabilities R_{ij} of a reversed Markov chain we have

$$\begin{aligned} R_{ij} &= \Pr(X_m = j | X_{m+1} = i) \\ &= \frac{\Pr(X_m = j)\Pr(X_{m+1} | X_m = j)}{\Pr(X_{m+1} = i)} = \frac{\pi_j P_{ji}}{\pi_i}. \end{aligned} \tag{3.5}$$

Thus a time-reversible Markov chain fulfills *detailed balance*

$$\pi_i P_{ij} = \pi_j P_{ji}. \tag{3.6}$$

We have the following lemma:

Lemma 3.1. *If π is a probability vector and it fulfills the detailed balance equation* (3.6) *for all states i and j of a connected and aperiodic Markov chain, then the process is time-reversible and π is the stationary distribution of the chain.*

Proof. Connectedness and aperiodicity of the Markov chain guarantees that the process converges and all states have a positive final probability. Time-reversibility follows directly from detailed balance and from Equation (3.5). By using detailed balance we get:

$$\sum_{i \in S} \pi_i P_{ij} = \pi_j \sum_{i \in S} P_{ji} = \pi_j, \tag{3.7}$$

thus π fulfills the general balance condition and is then the stationary distribution of the chain. □

If instead of detailed balance (3.6) we demand symmetry of transition probabilities, $P_{ij} = P_{ji}$, we have

Theorem 3.2. *The stationary distribution of a connected, aperiodic Markov chain with symmetric transition probabilities, $P^T = P$, is a uniform distribution and the corresponding chain is time-reversible.*

Proof. Let π be a uniform distribution. Since $P^T = P$, the distribution π trivially fulfills the detailed balance (3.6). Thus Lemma 3.1 implies that π is the stationary distribution of the chain and the chain is time-reversible. □

The definitions can easily be generalized also to the case of continuous state space. Then, instead of transition probability matrix P we have *transition kernel* P defined in the continuous state space. For clarity and to be consistent with the notation of continuous state space, we will later use notations $P(x, y)$ and $\pi(x)$ also for finite state space to mean P_{ij} and π_i, respectively.

3.3.2 Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) is a general concept for methods for sampling from probability distributions. MCMC is based on constructing a Markov

chain with the desired distribution as its stationary distribution. The states of the Markov chain are then used as samples from the desired distribution.

Optimally, we could draw the starting state randomly from the stationary distribution. However, this is usually not possible and we have to start just somewhere. Thus, one of the most important steps of the MCMC method is the *mixing time* (or *burn-in time*). It describes the number of steps after which the state distribution of the Markov chain has approximately converged to the stationary distribution. Only samples obtained after the mixing time of the chain should be accepted as random samples from the stationary distribution.

The mixing time is usually hard to evaluate theoretically. In practice, we can use some distance measure to approximate the mixing time, that is, when the distance between the starting state and the current state has converged, we can assume that the distribution has converged. Often it is just enough to be sure that the chain is *functionally mixed* which means that the distribution of the values of some relevant function of the samples has converged.

There exist various ways to draw samples with MCMC methods. We can take a prescribed number of successive samples after the mixing time. However, then the samples are likely to depend strongly on each other, but if we will use a lot of samples this may not be a problem. Nevertheless, we usually prefer fewer but more independent samples. This can be achieved by taking every kth sample after the mixing time where k is as large as we can afford, close to the mixing time. Another way is to produce each sample by starting a new chain from the beginning state and take the first sample after the mixing time.

However, we will use none of the ways described above to draw samples since they do not guarantee the *exchangeability* condition discussed in Subsection 3.3.4. Shortly put, we want our original state to be comparable with the randomized samples when calculating p-values. In Subsection 3.3.4 we describe how to achieve this.

3.3.3 Metropolis-Hastings

Metropolis-Hastings algorithm [24,36] is one of the most used MCMC methods to sample from probability distributions. It is a rejection sampling algorithm which

uses a *proposal density* $Q(y|x)$ that gives the proposal probability of the new state y given the current state x. It is assumed that sampling from the proposal density $Q(\cdot|x)$ is easy, whereas it suffices that the desired probability $\pi(x)$ can be calculated only up to a constant factor.

At each step a new proposal y is drawn from the distribution $Q(\cdot|x)$. It is accepted as the new state if u sampled randomly from uniform distribution $U(0, 1)$ fulfills

$$u < \frac{\pi(y)Q(x|y)}{\pi(x)Q(y|x)}, \tag{3.8}$$

otherwise the chain stays in the current state x, which is then the new state. As the symbol π for the desired probability distribution suggests, we have the following theorem:

Theorem 3.3. *The Markov chain produced by Metropolis-Hastings algorithm is time-reversible, and if the chain is connected and aperiodic, the stationary distribution equals the desired probability distribution π.*

Proof. We will prove that π satisfies detailed balance (3.6) in which case we can apply Lemma 3.1. Due to the rejection sampling defined in Equation (3.8), the true transition probabilities $P(x, y)$ are

$$P(x, y) = Q(y|x) \min\left\{1, \frac{\pi(y)Q(x|y)}{\pi(x)Q(y|x)}\right\} \quad \text{if } x \neq y, \tag{3.9}$$

and $P(x, x)$ is defined by subtraction of $P(x, y)$, $y \neq x$. Then $P(x, x) \geq 0$ since $P(x, y) \leq Q(x, y)$ and $Q(x, y)$ sums to one over y. Now if $x \neq y$

$$\pi(x)P(x, y) = \min\{\pi(x)Q(y|x), \pi(y)Q(x|y)\} = \pi(y)P(y, x) \tag{3.10}$$

It also holds trivially for $x = y$, thus π satisfies the detailed balance condition. □

The proposal distribution Q has a huge impact on the mixing time. It should be as global as possible while allowing a high acceptance rate in Equation (3.8). The optimal acceptance rate under some reasonable assumptions is around 25% [18]. However, the best case would be if $Q(\cdot|x)$ equalled π for all x. Then the acceptance rate would be 100%, but then there is no need for Metropolis-Hastings sam-

pling. Often symmetrical proposal distribution is used, that is, $Q(y|x) = Q(x|y)$, in which case Equation (3.8) reduces to

$$u < \frac{\pi(y)}{\pi(x)}. \tag{3.11}$$

This is the original version of the method by Metropolis [36], and we are using it in our randomization methods described in Chapter 4.

3.3.4 MCMC p-values

Assume that our task is to validate whether an observation A has been drawn from a distribution π or not. If we cannot produce samples from π directly, a basic approach is to produce a Markov chain with the stationary distribution π and use A to initialize the chain. Samples from the chain are then used for calculating an empirical p-value as described in Subsection 3.2.1.

However, as discussed in Section 3.3.2, producing independent samples with MCMC methods is hard. Thus calculating an empirical p-value using samples produced with any of the sampling schemes introduced in 3.3.2 may not produce a legitimate p-value. Nevertheless, we can produce a valid p-value without relying on independence if we can produce *exchangeable* samples as suggested by Besag *et al.* [7] and explained in more detail by Besag [6, p. 46].

Instead of starting the chain from A and running it forwards, we produce a new starting state A_0 by running the chain first I steps *backwards* from A. Then samples $A_1, \ldots, A_k$ are each produced by starting a new chain from A_0 and running it I steps forwards. Running the chain backwards is implemented by reversing the Markov chain as explained in Section 3.3.1 and in Equation 3.5.

If A is indeed from π, then the samples $A_1, \ldots, A_k$ are also. Furthermore, the samples A and $A_1, \ldots, A_k$ have an underlying joint distribution which is *exchangeable*, i.e, we cannot distinguish any of the samples A and $A_1, \ldots, A_k$ from each other since they all could be produced with I steps from A_0. Thus if A is really from π, the rank of $\mathcal{S}(A)$ among $\mathcal{S}(A)$ and $\mathcal{S}(A_1), \ldots, \mathcal{S}(A_k)$ is distributed uniformly and the empirical p-value calculated is correct. On the other hand, if A is not from π, sufficiently large I allows the chain to converge and we are able to

confirm that A is not from π.

Thus the exchangeability condition guarantees that we calculate the p-value correctly even if the samples are dependent. Large number of steps I is still good for allowing the chain to convergence, and traditional tests can be used for finding an appropriate mixing time. However, if we use too small I, we get just more conservative results.

There exists also a serial version of the approach which actually does not guarantee exchangeability condition but generally converges faster. In the experiments we use the parallel version explained above. For more about the serial version, see references [6, 7]. The approach of sequential tests explained in Subsection 3.2.2 can be used also in calculation of MCMC p-values.

3.4 Significance testing on real-valued matrices

As the main interest in this thesis is the significance testing of real-valued matrices, we discuss it further. In Chapter 4 we introduce methods based on the MCMC approach discussed in the previous section, and use them to generate random samples for calculating p-values as explained in 3.3.4. However, we first have to fix our null-hypothesis, that is, to form a probability distribution π from which the random real-valued matrices are drawn.

First we discuss generally what kind of randomness we want and how to achieve it. Then we study specifically the case of randomizing a matrix while preserving its row and column means and variances. Finally, we give an example why preserving these statistics is useful in some applications.

3.4.1 Problem definition

In introduction, we formulated the problem as randomly sampling real matrices with given row and column means and variances. The choice of means and variances is purely arbitrary: they are just natural characteristics explaining most of the variation in a matrix. They may also capture the essential parts of the underlying phenomenon, but that depends on the application.

As it is ambiguous what sort of randomization we prefer, we formulate a general problem for randomization of real-valued matrices:

Problem 2. *Given an $m \times n$ real-valued matrix A, generate a random matrix $\hat{A}$ from probability distribution*

$$\pi(\hat{A}) = \begin{cases} c \exp(-w E(A, \hat{A})), & \hat{A} \in S, \\ 0, & \hat{A} \notin S, \end{cases} \tag{3.12}$$

where S is the set of allowed matrices, $E(A, \hat{A})$ is an error measure of $\hat{A}$ respect to A, w is an error scaling constant and c is a normalizing constant.

The error function E captures the characteristics we want to preserve. The probability $\pi(\hat{A})$ is high for matrices containing small error. The exponential term in π effectively eliminates matrices with large error. The constant w controls how steep the distribution is. Notice that the probability distribution π is uniform among all the matrices having the same error measure. In the next subsection 3.4.2 we develop a function $E(A, \hat{A})$ measuring the error in the row and column means and variances of $\hat{A}$ compared to A. However, the methods developed in Chapter 4 can be used with an arbitrary error function. Nevertheless, we are only studying the error function explained in the next subsection.

In addition to the error measure E, we are restricting the allowed matrices to a predefined set S. Naturally, S can be the set of all real-valued matrices having the same size than A, thereby allowing all real-valued matrices as outcomes. Furthermore, we can restrict the values in randomized matrices into some predefined range, for instance, into $[0, 1]$ by suitable S. This can be important if, for example, only positive values are meaningful.

Sometimes also the distribution of the values in the matrix is important. Of course, this could be embedded in the error measure, but it can be controlled by S as well. In some of our methods we use a simple way to obtain this: S is the set of all matrices containing the values in A permuted randomly.

In Chapter 4 we introduce three different methods for randomizing real-valued matrices. They all use the same error function defined in the next subsection 3.4.2; only the set S is different for each.

3.4.2 Measuring the error of a randomized matrix

Next, we define an error measure on the difference in row and column means and variances between two matrices. Let A be the original $m \times n$ real-valued matrix whose row and column means and variances we wish to preserve. Let $\hat{A}$ be another $m \times n$ real-valued matrix, for example an output of one of our algorithms. Let r_i be the sum of the values in the ith row of A and c_j the sum of the values in the jth column of A. Let R_i and C_j be the corresponding sums of squares of the values in row i and column j. Thus

$$\begin{aligned} r_i &= \sum_{j=1}^{n} A_{ij}, & c_j &= \sum_{i=1}^{m} A_{ij}, \\ R_i &= \sum_{j=1}^{n} A_{ij}^2, & C_j &= \sum_{i=1}^{m} A_{ij}^2. \end{aligned} \tag{3.13}$$

Let $\hat{r}_i$, $\hat{c}_j$, $\hat{R}_i$ and $\hat{C}_j$ be the corresponding values for the randomized matrix $\hat{A}$. Now let $E(r_i)$, $E(c_j)$, $E(R_i)$, $E(C_j)$ be the row sum, column sum, row square sum and column square sum errors respectively, that is,

$$\begin{aligned} E(r_i) &= |r_i - \hat{r}_i|, & E(c_j) &= |c_j - \hat{c}_j|, \\ E(R_i) &= |R_i - \hat{R}_i|, & E(C_j) &= |C_j - \hat{C}_j|. \end{aligned} \tag{3.14}$$

To obtain algorithms for our task, we need to combine the sum and square sum errors, corresponding to differences in means and variances between A and $\hat{A}$. A general approach is to allow the importance of rows vs. columns and means vs. variances to be defined by separate weight parameters. Let w_r and w_s be row and square sum weights, respectively. We define the general error function as

$$\begin{aligned} E(A, \hat{A}) = w_r \sum_{i=1}^{m} \Big(E(r_i)^2 + w_s E(R_i)^2 \Big) \\ + \sum_{j=1}^{n} \Big(E(c_j)^2 + w_s E(C_j)^2 \Big). \end{aligned} \tag{3.15}$$

This measures the distance in means and variances between the given matrix $\hat{A}$ and the original matrix A. In our experiments, we use parameter values $w_r = m/n$ and $w_s = 1$, treating each of the row and column sum and square sum errors as equally important.

Note that the error measures are not scale invariant, that is, multiplying A and $\hat{A}$ both by the same constant changes the value of $E(A, \hat{A})$. However, we assume the values $A(i, j)$ to be in the interval [0, 1]. To obtain this, the values are linearly scaled into [0, 1].

Similarly, one could define error measures for higher moments or for some other specific features. The only restriction is that the error in the matrix $\hat{A}$ has to be interpretable with a single number, being a combination of partial errors.

3.4.3 Example

Most of the existing randomization techniques for real-valued matrices are based on simply permuting the values in a single column (or row). To show why this is not necessarily enough, consider the two 10×5 real-valued matrices A and B shown in Figure 3.1. They share their first two columns, and the correlation between these columns is high, 0.92. However, in matrix B the values on each row are tightly distributed around the mean of the row, whereas in matrix A the variance of each row is high. If the test of significance of correlation between columns x and y would consider only the first two columns, the results for the two matrices would be identical. However, it seems plausible that the high correlation between the first and second columns in matrix B is due to the general structure of the matrix, and not some interesting local structure involving the two columns, as might be the case with matrix A. More specifically, it seems that the correlation of x and y in B is due to the small variance of each row in B.

To test this observation, we generated randomized matrices: sets $\mathcal{A}$ and $\mathcal{B}$ each contain 1000 independent random matrices having approximately the same row and column means and variances as A and B. Then the correlations between the x and y in the randomized matrices are as follows: for the matrices in set $\mathcal{A}$ the smallest correlation between x and y is -0.38, the maximum 0.89, average 0.34 and standard deviation 0.25, while in set $\mathcal{B}$ corresponding values were 0.82, 0.99,

x	y			
.46	.36	.21	.68	.45
.44	.29	.64	.21	.04
.74	.87	.32	.84	.03
.04	.06	.96	.63	.31
.75	.66	.73	.13	.01
.85	.81	.41	.21	.38
.80	.98	.74	.61	.68
.70	.72	.27	.63	.09
.30	.37	.44	.37	.04
.57	.41	.93	.58	.61

Matrix A

x	y			
.46	.36	.56	.51	.53
.44	.29	.49	.52	.38
.74	.87	.90	.79	.80
.04	.06	.03	.11	.05
.75	.66	.68	.75	.71
.85	.81	.83	.81	.90
.80	.98	.88	.90	.81
.70	.72	.67	.79	.63
.30	.37	.37	.35	.43
.57	.41	.46	.44	.41

Matrix B

Figure 3.1: Examples with two real-valued data matrices sharing the first two columns x and y having high correlation. The values on each row of the matrix B are close to each other whereas in A the variance of each row is large. The high correlation between x and y is significant in A but not significant in B when tested using the methods introduced in Chapter 4.

0.93 and 0.03, respectively. This gives empirical p-values of 0.001 for matrix A and 0.4156 for matrix B. Thus we may conclude that the high correlation between the first and second columns in A is indeed not due to the row and column sums and variances, unlike in B.

The example shows that the structure of the entire matrix can have a strong effect on the significance of even the basic data mining results. The randomization approach presented in this paper is applicable in assessing the significance of structural measures conditional on the knowledge of row and column sums and variances, but it can be modified directly to other conditions as well.

Chapter 4

Randomization methods

In this chapter, we introduce methods for randomizing real-valued matrices while preserving row and column sums and variances. However, we first explain randomization of binary matrices as introduced by Gionis *et al.* [19]. We further develop the ideas of randomizing binary matrices to be applicable with real-valued case. All methods are based on local transformations. The series of operations in each method forms a Markov chain, in either discrete or continuous space.

4.1 Methods for 0–1 matrices

In this section, we introduce a randomization method for binary matrices developed by Gionis *et al.* [19]. It is based on swapping matrix elements. The idea of swapping matrix elements as a randomization technique has a long history [14]. Our methods for real-valued matrices, which are introduced in Section 4.2, get their basic idea from the method introduced next. The method is called the *Self-loop* method by the original authors [19] but we will refer it as *SwapBinary*. There exist also various other methods for randomizing binary matrices [9,13,14,17] but they are not introduced in this thesis.

Binary matrices consist of zeros and ones. A real world example of such matrix is a market basket data. The rows contain customers and the columns contain products. A cell contains one if the corresponding customer bought the corresponding product, and zero otherwise. Typical data mining tasks for binary

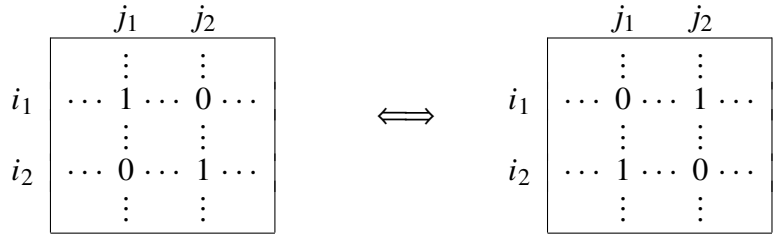

Figure 4.1: An example of a binary swap. The four elements shown are rotated and rest of the matrix is kept fixed. The number of ones in each row and column do not change.

matrices are finding the frequent item sets, that is, the sets of products which many customers bought, and finding the associative rules, that is, rules such that if customer bought product A he is likely to buy product B as well.

In randomization of binary matrices, we want to sample matrices uniformly from all binary matrices with the same number of ones in each row and column as in the original matrix. Thus we are actually preserving row and column sums (and also higher moments) and restricting the values to zeros and ones.

The basic idea of randomization of binary matrices is to select four elements as in Figure 4.1 and swap them to form a new matrix. The swap procedure retains the number of ones in each row and column. All the binary matrices with the same margins are reachable with such swap operations from any such matrix.

In Algorithm 5 the *SwapBinary* method is presented. It starts from the original matrix and iteratively performs the swap operation. At each step, it chooses randomly four elements with ones in the two opposite corners and swaps them if the two other corners contain zeros.

The first two corners are insisted to contain ones as the binary matrices are usually sparse, that is, they contain mainly zeros, to make the process faster. The method *SwapBinary* samples uniformly from all the binary matrices with the same margins as the original matrix. In the experiments by Gionis *et al.* [19], the authors found that usually a couple of times the number of ones in the matrix is sufficient to guarantee the convergence.

Algorithm 5 SwapBinary

Input: Binary matrix A, number of attempts k

1: $\hat{A} \leftarrow A$
2: **for** $i \leftarrow 1, k$ **do**
3: Pick i_1 and j_1 randomly s.t. $\hat{A}_{i_1 j_1} = 1$
4: Pick i_2 and j_2 randomly s.t. $\hat{A}_{i_2 j_2} = 1$
5: **if** $\hat{A}_{i_1 j_2} = 0$ and $\hat{A}_{i_2 j_1} = 0$ **then**
6: $\hat{A} \leftarrow \text{Swap}(\hat{A}, i_1, i_2, j_1, j_2)$
7: **end if**
8: **end for**
9: **return** $\hat{A}$

4.2 Methods for real-valued matrices

In this section we introduce three algorithms for performing sampling from the set of real-valued matrices with given row and column sums and variances. All methods output a randomized version $\hat{A}$ of the original $m \times n$ matrix A. They sample from probability distribution π defined in Problem 2 in Subsection 3.4.1.

The methods use the error function $E(A, \hat{A})$ defined in Subsection 3.4.2. We study three different sets of allowed matrices S and introduce a separate algorithm for each of them. The sets of allowed matrices (or equivalently the proposed algorithms) differ in the amount of structure that is maintained in the value distribution: in the first set the *discretized* distribution of the values in the rows and columns is preserved, in the second set the distribution of the values in the whole matrix is preserved whereas in the third set only the range of the values is preserved.

The algorithms are based on doing local modifications on the matrices. The idea for the type of the modifications comes from binary swaps explained in the previous section 4.1. Here we use a concept of swap rotations as shown in Figure 4.2, which degenerates to conventional binary swaps as in Figure 4.1 in the case of 0–1 data. At each step we randomly choose from the current matrix four elements a, b, a', and b', located at the intersections of two rows i_1 and i_2 and two columns j_1 and j_2. A new matrix is produced by rotating those four elements clockwise, while keeping the other elements unchanged.

The smaller the difference between (a, b) and (a', b'), the smaller the change

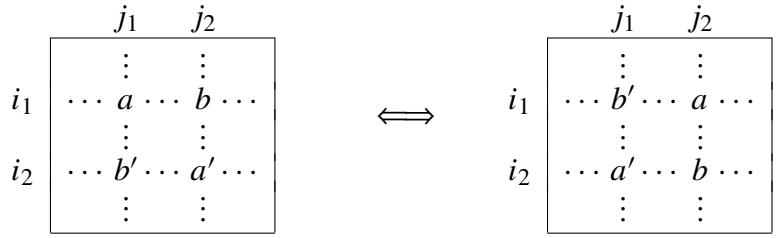

Figure 4.2: An example of a swap rotation in a real-valued matrix. The four elements shown are rotated and rest of the matrix is kept fixed. If $a = a'$ and $b = b'$ then the row and column statistics do not change.

in the row and column statistics will be. If $a = a'$ and $b = b'$, the row and column statistics do not change at all, corresponding to binary swaps.

In the following discussion, the data is assumed to be scaled into the unit interval $[0, 1]$. The data can always be scaled linearly into $[0, 1]$ and after randomization returned into the original range of the values.

4.2.1 Discrete swaps

Our first method is a fairly crude generalization of the *SwapBinary* algorithm to real valued data. First the values in the original matrix are discretized into a predefined number of classes N. Then swaps are performed requiring that a and a' as well as b and b' belong to the same class. Finally, the data is "undiscretized" by mapping the discretized values back to the original ones. The pseudocode of this approach is presented in Algorithm 6.

The *Discretize* method returns a matrix C, where the matrix values have been replaced by their class labels. At the end of the algorithm, the *Undiscretize* method replaces the class labels with real values. The *Swap* method implements the operation shown in Figure 4.2.

In our experiments the data matrix is discretized by dividing the range of A's values into N intervals of equal length. As the values are assumed to be in $[0, 1]$, the division intervals are $[0, 1/N], \ldots, (1 - 1/N, 1]$. The undiscretization may either restore the original matrix elements in their new places, or replace them with the average value of the elements in the corresponding class. The latter produces matrices with a smaller error. However, the former preserves the values of the

Algorithm 6 SwapDiscretized

Input: Matrix A, number of attempts I and classes N

1: $C \leftarrow$ Discretize(A, N)
2: **for** $i \leftarrow 1, I$ **do**
3: Pick i_1 and j_1 randomly
4: Pick i_2 and j_2 randomly with $C_{i_1 j_1} = C_{i_2 j_2}$
5: **if** $i_1 \neq i_2$ and $j_1 \neq j_2$ and $C_{i_1 j_2} = C_{i_2 j_1}$ **then**
6: $C \leftarrow$ Swap(C, i_1, i_2, j_1, j_2)
7: **end if**
8: **end for**
9: $\hat{A} \leftarrow$ Undiscretize(C)
10: **return** $\hat{A}$

original matrix, and we apply it in the experiments.

The selection procedure in line 4 can be done in constant time by keeping track of the locations of elements of each type. Compared to the *SwapBinary* method the first selection in *SwapDiscretized* is not restricted in anyway as there in general can be more than two classes.

SwapDiscretized is a simple method for approximately preserving all the row and column moments of the original data. The moments are exactly maintained in the discretized space. However, contrary to binary case, all valid permutations of matrix elements are not reached by this method, as shown in Subsection 4.3.1. Choosing the value for N involves making a compromise between efficiency of mixing, and the error induced in the row and column statistics.

4.2.2 Metropolis with swaps

Next we introduce a method based on the Metropolis algorithm introduced in Subsection 3.3.3. It allows us to generate samples $\hat{A}$ directly from the probability distribution π defined in the Problem 2:

$$\pi(\hat{A}) = \begin{cases} c \exp(-w E(A, \hat{A})), & \hat{A} \in S, \\ 0, & \hat{A} \notin S, \end{cases}$$

Then the matrices $\hat{A}$ for which $E(A, \hat{A})$ is small have a high probability of being generated. We let S be the set of all the matrices containing the values of the original matrix A permuted randomly. For the Metropolis algorithm we need also a proposal distribution Q. We use the uniform distribution among all the matrices reachable from the current matrix with one swap rotation. A direct implementation of the Metropolis approach is presented in Algorithm 7.

Algorithm 7 SwapMetropolis

Input: Matrix A, number of attempts I, error limiter $w > 0$

1: $\hat{A} \leftarrow A$
2: **for** $i \leftarrow 1, I$ **do**
3: Pick $i_1 \neq i_2$ and $j_1 \neq j_2$ randomly
4: $A' \leftarrow \text{Swap}(\hat{A}, i_1, i_2, j_1, j_2)$
5: $u \leftarrow \text{Uniform}(0,1)$
6: **if** $u < \exp\{-w(E(A, A') - E(A, \hat{A}))\}$ **then**
7: $\hat{A} \leftarrow A'$
8: **end if**
9: **end for**
10: **return** $\hat{A}$

The difference in error induced on line 4 can be calculated in constant time, provided we keep track of the row and column sums and square sums. This holds because the swapped matrix A' differs from $\hat{A}$ only on rows i_1 and i_2 and on columns j_1 and j_2.

The *SwapMetropolis* algorithm can attain all permutations of the input matrix. The value for the constant w involves making a compromise between efficiency of mixing and the error induced in the row and column statistics: increasing w decreases the chances of accepting transitions that induce additional error.

4.2.3 Metropolis with masking

The Metropolis algorithm can also be applied with a different set S of allowed matrices. We define $S = [0, 1]^{m \times n}$ thus restricting only the range of the values to be the same with the original matrix. The next method works also with $S = \mathbb{R}^{m \times n}$ with little modifications. We cannot use the swap rotation as the local modification as it does not change the values. Thus we introduce a new local modification:

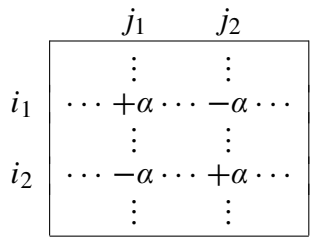

Figure 4.3: The addition operation in *MaskMetropolis*. The addition mask preserves the original row and column sums.

addition mask. A new matrix is created from the current one by selecting rows i_1, i_2 and columns j_1, j_2 at random, and adding the mask presented in Figure 4.3 to the four intersection elements. The same sampling scheme as in *SwapMetropolis* is then applied. The *MaskMetropolis* method is given in Algorithm 8.

Algorithm 8 MaskMetropolis

Input: Matrix A, attempts I, limiter $w > 0$, scale $s > 0$

1: $\hat{A} \leftarrow A$
2: **for** $i \leftarrow 1, I$ **do**
3: Pick $i_1 \neq i_2$ and $j_1 \neq j_2$ randomly
4: $\alpha \leftarrow$ Uniform($-s$,s)
5: $A' \leftarrow$ AddMask($\hat{A}$, α, i_1, i_2, j_1, j_2)
6: **if** for all $i, j : A'_{ij} \in [0, 1]$ **then**
7: $u \leftarrow$ Uniform(0,1)
8: **if** $u < \exp\{-w(E(A, A') - E(A, \hat{A}))\}$ **then**
9: $\hat{A} \leftarrow A'$
10: **end if**
11: **end if**
12: **end for**
13: **return** $\hat{A}$

The auxiliary method *AddMask* adds the mask presented in Figure 4.3. The parameter transition scale s defines the range $[-s, s]$, from which α is selected uniformly at random. Other distributions, such as normal distribution, could also be used for choosing α. As with *SwapDiscretized*, the error difference in line 8 can be calculated in constant time. However, this algorithm never changes the row and column sums of the matrix, so only the square sum parts of the error function need to be considered. Thus, actually the addition mask restrict the S to contain only

matrices having exactly the same row and column sums as the original matrix.

MaskMetropolis changes the matrix values directly, whereas *SwapDiscretized* and *SwapMetropolis* only reorder the entries. Thus the distribution of values in the output matrix $\hat{A}$ may differ significantly from the original distribution. However, *MaskMetropolis* preserves the row and column sums exactly, and as seen in our experimental results, it also preserves the variances quite accurately. Choosing the values for the parameters w and s involve a compromise between efficiency of mixing and the error induced in the variances.

4.2.4 Other methods

In our studies we have also tested several variations of the three algorithms and a couple of other approaches [38]. There were a few alternatives that gave intuitive results, but were inferior to these three in theoretical or practical aspects.

A simple approach based on swap rotations is to allow a swap if (a, b) differ from (a', b') by at most a small ε each. However, the error $E(A, \hat{A})$ keeps growing during the randomization and the outputs are useless. One particularly promising algorithm starts from a random matrix, selects a cell at random, and replaces its value with a value from [0, 1] that minimizes locally the error in Equation (3.15). The technique produces matrices with small errors, but the distribution of resulting matrices is unknown.

Other alternatives are obtained by changing the error function $E(A, \hat{A})$ or the set of allowed matrices S. The further study of these approaches is left for future work.

4.2.5 Applying the algorithms

Our methods are instantiations of the general MCMC approach. Simply starting the chain from the original data and running the chain does not guarantee the exchangeability condition. Thus we will use the techniques introduced in Subsection 3.3.4. First, the chain is run *backwards* for I steps, resulting in a new dataset A_0. After that, the samples are created one by one, always starting from A_0 and running the chain forwards for I steps. The p-value calculated by using these

samples is then valid.

In *SwapDiscretized* the Markov chain is time-reversible, because the transition probabilities are uniform across the candidates. Due to the Theorem 3.3 the Markov chains of the two Metropolis algorithms are also time-reversible. Therefore running the backward phase can be done by running the basic algorithm with all the three methods.

4.3 Analysis of the methods

In this section we discuss some of the key properties regarding the distribution of matrices produced by our methods.

4.3.1 Discrete swaps

The success probability of a swap on line 6 in Algorithm 6 is discussed below. Let n_l be the number of elements with label l, thus $\sum_{l=1}^{N} n_l = mn$. If $A_{i_1 j_2}$ has label l then the probability of success of a swap is $\frac{n_l - 1}{mn-1} \approx \frac{n_l}{mn}$, assuming the locations of labels in matrix A are randomly distributed. Using Chebyshev's sum inequality or Cauchy-Schwarz inequality we obtain the following approximate lower bound for the acceptance probability:

$$\sum_{l=1}^{N} \left(\frac{n_l}{mn}\right)^2 \geq N \left(\frac{\sum_{l=1}^{N} \frac{n_l}{mn}}{N} \right)^2 = \frac{1}{N}. \tag{4.1}$$

The algorithm is unable to reach all possible matrices with the same distribution of row and column labels as in the original matrix. This may not be a problem in practice, though it implies that more conservative significance results are obtained. Consider the counterexample shown in Figure 4.4 with $N = 3$. The matrices have the same number of entries of each type per row and column, but neither of them contains four elements which could be swapped. Thus they cannot be transformed to each other. The counterexample can be generalized directly to all matrices with odd number of rows or columns.

The chain of swap operations has a uniform stationary distribution in the space

$$\begin{array}{|ccc|} \hline 1 & 2 & 3 \\ 2 & 3 & 1 \\ 3 & 1 & 2 \\ \hline \end{array} \qquad \not\leftrightarrow \qquad \begin{array}{|ccc|} \hline 1 & 2 & 3 \\ 3 & 1 & 2 \\ 2 & 3 & 1 \\ \hline \end{array}$$

Figure 4.4: A counterexample of connectedness. The two matrices have the same row and column statistics, but they cannot be transformed to each other by using swap rotation, since there does not exist any swappable quartet.

of all reachable permutations, when the selection of candidate elements is done as on lines 3–5. This follows from Theorem 3.2: The state space is connected as we have restricted the space to all reachable permutations. The chain is aperiodic since there is a positive probability of staying in the current state, and finally, the transition probabilities are symmetric since the swap can be undone by another swap. The theorem also guarantees the time-reversibility of the Markov chain of *SwapDiscretized* as stated in Subsection 4.2.5.

SwapDiscretized does not strictly follow the definition of Problem 2 in Subsection 3.4.1. Since in the discretized space all the matrices $\hat{A}$ have the same row and column statistics, the error $E(A, \hat{A})$ stays constant. Only error is made in the discretization process. If the undiscretization is done by replacing class labels with the average values, the uniform stationary distribution equals the π in Equation 3.12. However, if we replace the labels with the original values, the error $E(A, \hat{A})$ is not constant for each output $\hat{A}$ and the error distribution does not strictly follow the π in Equation 3.12.

4.3.2 Metropolis with swaps

Consider the error distribution of matrices produced by Algorithm 7. Since there exist many more matrices with notable error than matrices with almost zero error, the stationary distribution of the error of resulting matrices $\hat{A}$ is concentrated far from zero. More precisely, the distribution of the resulting error $\hat{E} = E(A, \hat{A})$ is

$$\Pr(\hat{E} = x) \propto \exp(-wx)q(x), \tag{4.2}$$

where $q(x)$ is the distribution of error x of matrices in S, that is, matrices with the values in A permuted randomly [24]. The distribution $q(x)$ is a sum of hyper-

exponential distributions that can be approximated with a Gamma distribution. It results that Equation (4.2) can be approximated by another Gamma distribution. Additional theoretical studies on the error distributions are left for further work.

We show that the state space is connected when $m, n \geq 3$. To move from state A_1 to A_2 we can clearly use the swap rotation to get one element at a time in the correct place until only 3×3 square is incorrectly placed. This is a special case which can be shown to have solution in all cases by brute force. However, if the error limiter ω is too large, the state space can in practice be unconnected. Nevertheless, the results will be then just more conservative as explained in Subsection 3.3.4. Approximately $3mn$ successful steps are needed at most to move from any state to any other state.

4.3.3 Metropolis with masking

The discussion in the previous subsection of the error distribution of matrices produced by *SwapMetropolis* also applies directly to the *MaskMetropolis* method. The only change is in the distribution $q(x)$ where now the set S consists of all matrices in $[0, 1]^{m \times n}$ having the same row and column sums as the original matrix.

The state space S can easily be seen to be connected with MaskMetropolis. To move from state A_1 to A_2 we can change one element at a time to the new value by using the addition mask. As the sums of the rows and columns are fixed also the remaining values are guaranteed to be correct. Approximately mn/s successful steps suffices to move from any state to any other state.

For a simplistic analysis of the practical convergence speed of *MaskMetropolis* method, suppose we start running Algorithm 8 with error limiter $\omega = 0$ on an $m \times n$ matrix, but this time accepting matrices with elements outside the unit interval. In practice this means accepting all attempts. Any given element will be changed on average every $2/m \cdot 2/n = 4/(mn)$ attempts, each change coming from the uniform distribution $U(-s, s)$. Thus after $I = \Omega(mn)$ attempts each element's total change will approximately follow the normal distribution $N(0, 4Is^2/(3mn))$, because we have $\mathrm{Var}(U(-s, s)) = s^2/3$. Now if we want the changes to come from the standard normal distribution when using, for example, $s = 0.1$ we get $I = 75mn$, which is close to the number of attempts used in the experiments.

Chapter 5

Experiments

We performed various experiments and tests with the three methods, *SwapDiscretized*, *SwapMetropolis* and *MaskMetropolis*, on real and artificial datasets. First we give visual examples of randomizations to justify the usefulness of the methods. After that we study empirically the convergence properties of the methods and choose appropriate parameter values. Finally, we perform significance testing on five different datasets of the three structural measures introduced in Section 2.2: K-means, maximum correlation and principal components.

5.1 Examples of randomizations

In this section we give some examples of the results produced by each of the three methods, and motivate the usefulness of preserving also row and column variances in addition to means.

5.1.1 Randomization of topographical data

First we show results on a 100×100 matrix resembling a hilly surface, shown in Figure 5.1. The results shown come from applying our algorithms with the parameters described in Section 5.2. One randomized sample is presented for each method.

The *SwapDiscretized* method tends to create rectangular shapes, which may be seen from the image. The approximation of the error in *SwapMetropolis* makes

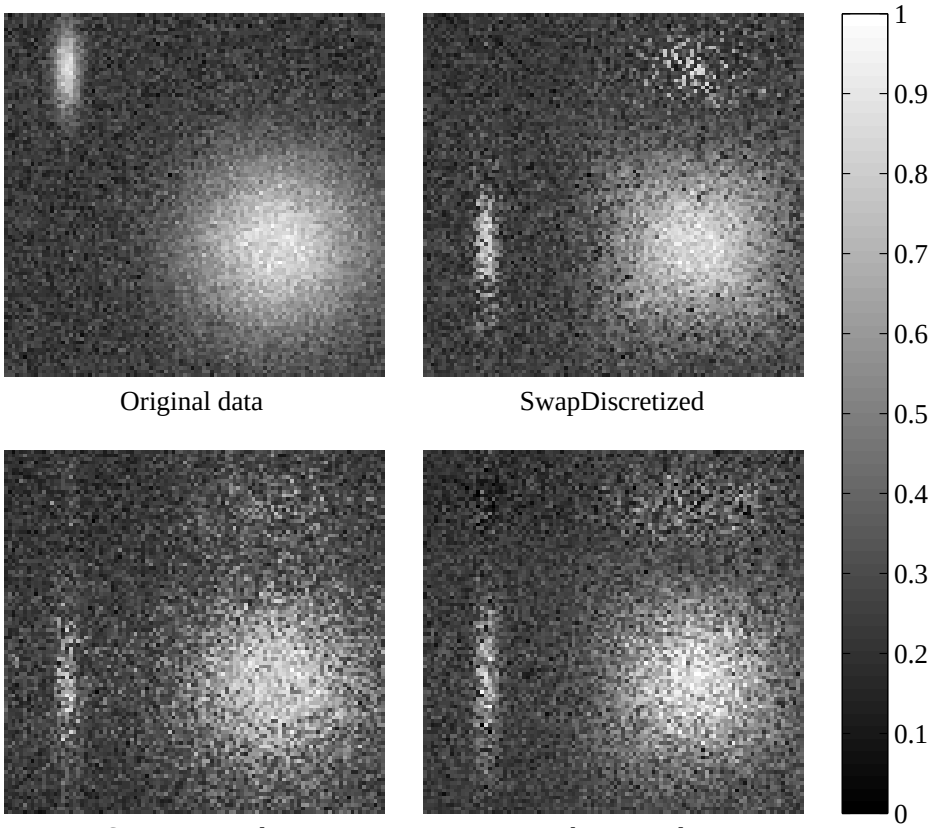

Figure 5.1: Original topographic data and results of randomization of the original data with our three methods. The small top left artifact in the original matrix has disappeared in the randomizations, which have produced "shadows" of the artifact to top right and bottom left regions.

its results look a bit noisy. Finally, the existence of the small hill on top left in the original data results in shadow shapes emerging in the top right and bottom left regions with all methods.

In all the randomized matrices the massive bottom right hill remains, but the smaller top left artifact disappears. *MaskMetropolis* even introduces a hole in the place of the small hill. Making an analogy to real data, the bottom right patch is something inherent to the data source, probably obvious and not very exciting, while the top left patch could be something more delicate and interesting. This fits with the idea that the patterns that disappear in randomization, with respect to some structural measure, are the significant ones.

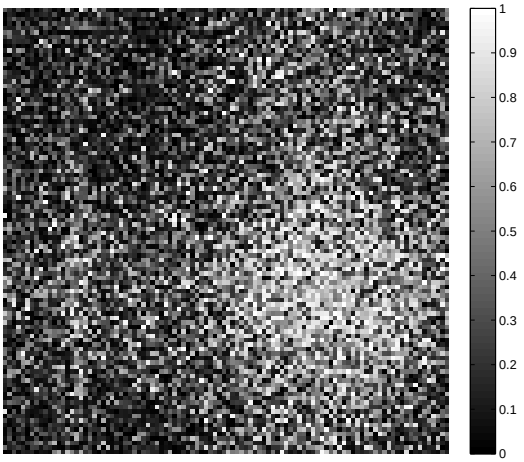

Figure 5.2: Topographic data randomized by preserving only row and column sums. Most of the structure has disappeared but the bottom right corner is lighter than rest of the matrix.

5.1.2 Importance of preserving variances

We also randomized the same topographical data by preserving only row and column sums, see Figure 5.2. It was obtained by applying *MaskMetropolis* with the parameter w set to zero, effectively accepting all valid transitions.

Figure 5.2 shows the importance of maintaining variances when randomizing real-valued data. In the figure we may still see the effect of preserving sums: the lower right corner is lighter than the rest of the matrix. However, much less of the original structure is maintained in the randomization process.

For example, suppose we are interested in the presence of large subrectangles of the data with high average and small variance corresponding to "high and flat hills". The original data clearly has one such pattern. If we assess the significance of this by comparing against randomizations such as in Figure 5.2, maintaining only means, the finding seems significant. However, if we maintain also variances, Figure 5.1 indicates that the presence of such rectangles is explained by the means and variances. Hence the discovered rectangle is not significant under that hypothesis.

Higher moments or other characteristics could also be included in the error function, but this would most likely imply difficulties in attaining a high enough

acceptance rate among the attempted modifications. Fortunately, the *SwapDiscretized* method preserves also higher moments in a discretized space.

5.2 Evaluating the methods

In this section we study the convergence, performance and error rate of each of our methods as well as the independence of randomized samples. We evaluate the methods on GENE dataset introduced in Subsection 2.1.2, which was linearly scaled into [0, 1].

We ran the *SwapDiscretized* algorithm with the class count $N = 30$. *SwapMetropolis* was run with the error limiter $w = 10$, and *MaskMetropolis* with $w = 1000$ and the transition scale $s = 0.1$. The values of w were chosen based on finding a suitable acceptance rate for which the methods converged fast enough, and were strict enough on error. Different parameters could be used for different datasets depending on the size and the type of the data. However, the chosen parameter values produced good results with all datasets used. The erroneous is discussed in detail in Subsection 5.2.2.

5.2.1 Convergence and performance

We performed various experiments to measure the convergence and performance of the methods. Finding the mixing times of Markov chains such as the ones we use is a theoretically hard issue; here we concentrate on some simple diagnostics for detecting convergence. Note that randomization-based tests can also be used even if it is not certain that the chain is able to cover all of the state space, the result will just be a more conservative test as explained in Subsection 3.3.4. We will apply the algorithms as explained in Subsection 4.2.5.

To assess the convergence of the methods, we monitor the Frobenius distance between the original matrix A and the randomized matrix $\hat{A}$. The Frobenius distance $\|A - \hat{A}\|_F$ is defined as

$$\|A - \hat{A}\|_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} (A_{ij} - \hat{A}_{ij})^2. \tag{5.1}$$

Method	Acceptance rate	Time (s)
SwapDiscretized	0.111	5.87
SwapMetropolis	0.149	5.03
MaskMetropolis	0.270	9.01

Table 5.1: Performance of the methods with GENE dataset. Acceptance rate is the number accepted swaps or additions divided by the number of attempts. Randomization was done with $100mn$ attempts for *SwapMetropolis* and *SwapDiscretized*, and with $200mn$ attempts for *MaskMetropolis*, where $mn = 82500$ in the GENE data matrix. Time is the time needed to produce one sample matrix, that is, the time needed in a forward phase.

Notice that it does not measure the error in the row and column statistics, but the dissimilarity between the two matrices.

Figure 5.3 shows the Frobenius distance as a function of attempts when randomizing the GENE data matrix. Other datasets and the structural measure functions discussed in the next section gave similar convergence results, although the methods converged faster with artificial data. Each data point represented in the figure is a result from an independent single randomization started from the original matrix first by running I steps back and then I steps forward, where I is the number of attempts presented in the x-axis. From the figure we observe that all methods converged to approximately the same Frobenius distance. *MaskMetropolis* converged the slowest as a high error limiter w was used for it. Based on these convergence tests, we used in the following experiments $100mn$ attempts for *SwapDiscretized* and *SwapMetropolis*, and $200mn$ attempts for *MaskMetropolis*. We found that usually around 2–3 times Nmn, where N is the number of discrete classes, was an appropriate number of attempts for *SwapDiscretized*.

In Table 5.1 we present the acceptance rate of attempts and the running times in the forward phase for each method on GENE dataset. We used C++ implementations integrated with MATLAB on a 2.2GHz Opteron with 4GB of memory. We noticed that the methods performed fast enough for all practical use with each attempt taking constant time, and the space requirement being a few times the size of the matrix. In practice our methods are efficient and scale very well to large matrices.

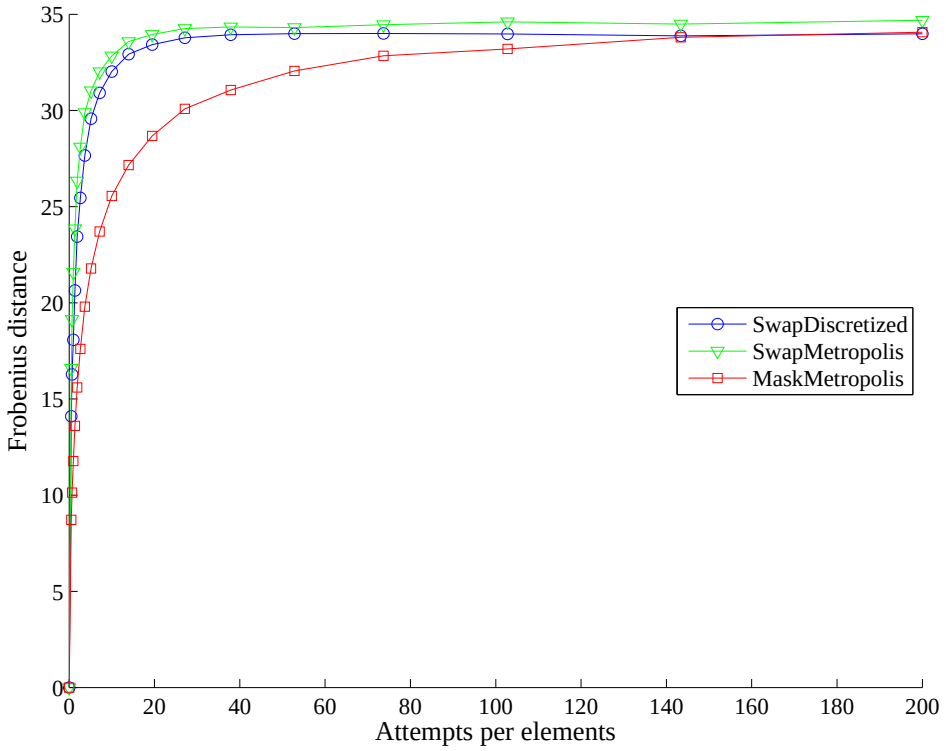

Figure 5.3: The Frobenius distance between the original and randomized matrix as a function of iterations used for backward and forward run with GENE dataset. The number of attempts needed for the convergence of Frobenius distance can be used as an approximation of the mixing times of the methods.

5.2.2 Error rate

Figure 5.4 shows the error as defined in (3.15) as a function of attempted modifications per element, $I/(mn)$. We notice that the error of *SwapDiscretized* grows as it is calculated according to the original values of the matrix, and is therefore not truly discretized. *MaskMetropolis* produces the most accurate matrices.

Table 5.2 summarizes the average values of the error in row and column means and standard deviations for each method. *MaskMetropolis* produces clearly the smallest errors. We notice that errors are minimal per element. The differences in means and in standard deviations of rows and columns are around 0.001 with all methods.

With all methods the parameters affect directly to the error rate. By increasing the number N of classes with *SwapDiscretized* or increasing the error scaling

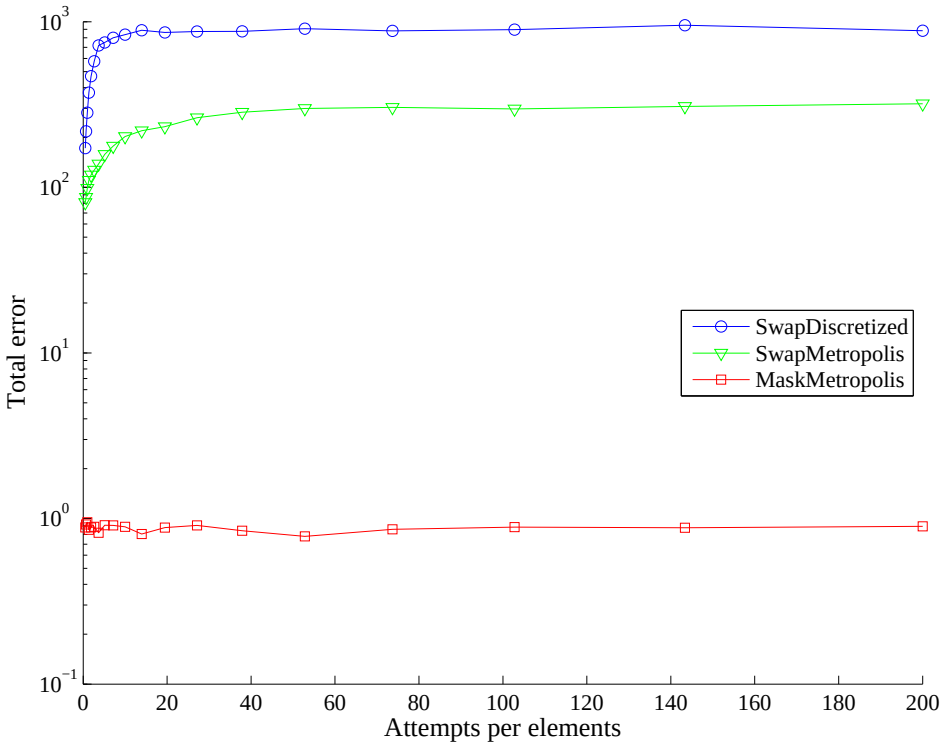

Figure 5.4: The error defined in Equation (3.15) as a function of attempts used for randomizing GENE data.

	Mean		Std	
Method	Rows	Cols	Rows	Cols
SwapDiscretized	1.45	0.31	1.46	0.33
SwapMetropolis	0.92	0.35	9.96	3.14
MaskMetropolis	0.00	0.00	0.49	0.11

Table 5.2: Average values of absolute errors in row and column means and standard deviations in randomized matrices with GENE dataset. Values are multiplied by 1000.

Method	From original	Pairwise
SwapDiscretized	34.03 (0.08)	33.91 (0.09)
SwapMetropolis	34.55 (0.08)	34.71 (0.08)
MaskMetropolis	33.95 (0.08)	33.76 (0.08)

Table 5.3: Frobenius distances of the randomized matrices from the original data matrix and from each other with GENE dataset. The values in parentheses are the standard deviations.

constant w with *SwapMetropolis* and *MaskMetropolis*, the amount of error decreases. However, at the same time the mixing becomes harder. Thus we cannot use arbitrarily large parameter values for N and w. The values for the parameters were chosen such that the Frobenius distance converged in reasonable time while giving as small error rate as possible.

5.2.3 Independence

To confirm that the methods actually produce different random matrices, we calculated the pairwise Frobenius distances between 1000 randomized samples for each method. The results are shown in Table 5.3. The pairwise Frobenius distances almost equal the Frobenius distances from the original data, thus we may conclude that the methods indeed produce different randomizations.

To confirm that the randomized matrices differ sufficiently from the original matrix, we studied how the ranks of the values in the original matrix differ from the ranks of the values in the randomized matrix in the corresponding locations. The results are presented in Figure 5.5. The rank of an element in a matrix is the index of the corresponding element in the list of the matrix values sorted in increasing order (not to be confused with the rank of a matrix). If the randomization method did not randomize the matrix at all, the difference in ranks would be zero in all locations. On the other hand, if the method had produced a totally random permutation of the original values, the expected number of elements having a rank difference d is $2(1 - \frac{d}{mn})$ for $d > 0$ and 1 for $d = 0$.

As we see from the Figure 5.5, the methods have produced randomization where the distribution of rank difference is close to the distribution of rank difference of a random permutation. However, as we are preserving the first and

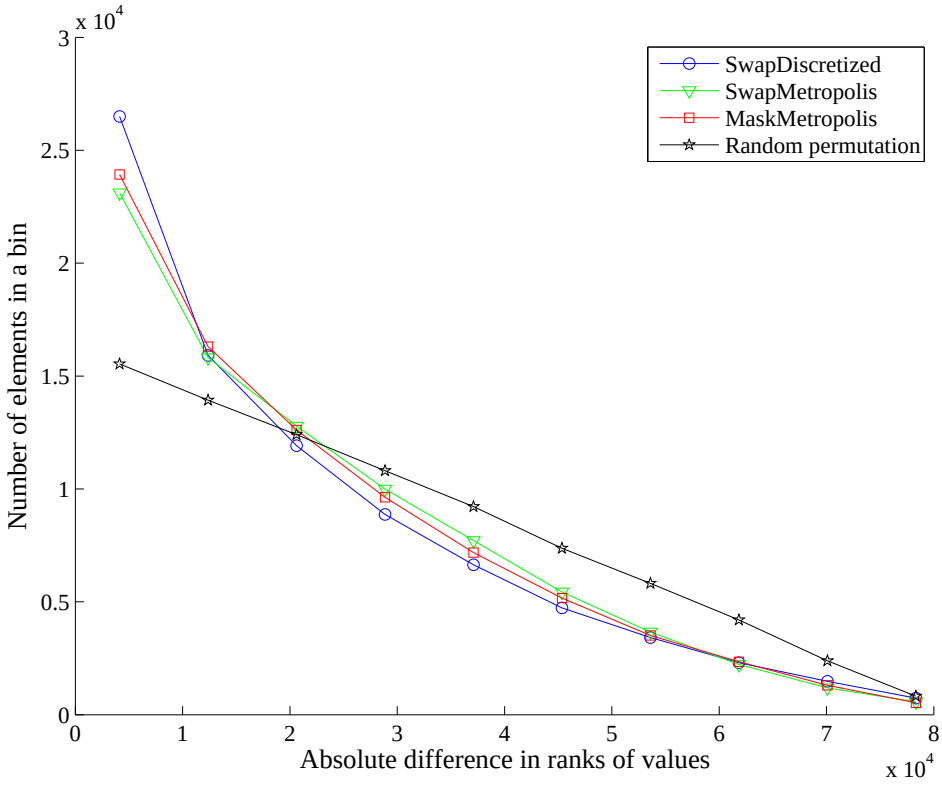

Figure 5.5: Histograms of the absolute differences in the ranks of the values in the original matrix and randomized matrix between the elements in the same location with each method. The histograms contain ten bins and for clarity they are plotted as curves. The original data is GENE data matrix with 82500 elements. For comparison the histogram is plotted also for a matrix containing a random permutation of the original values.

second order statistics of rows and columns, the randomized matrices contain more structure than a random permutation. Nevertheless, we can conclude that the randomizations really differ from the original matrix.

5.3 Significance testing of structural measures

In this section we used the three methods for assessing the significance of structural measures on generated datasets and on a real gene expression dataset GENE. We used three different structural measures: clustering error, maximum correlation between matrix rows, and variance explained by the main principal compo-

Dataset	Rows	Columns	Mean	Std
RANDOM	100	100	0.473	0.132
CLUSTER	1117	100	0.509	0.081
GAUSSIAN	1000	10	0.529	0.142
COMPONENT	1000	50	0.278	0.116
GENE	1375	60	0.578	0.110

Table 5.4: The number of rows and columns as well as the average values and standard deviations occurring in the five datasets.

nents.

We used four types of artificial data in our experiments. To assess that the methods produce reasonable significance levels on pure random data, we generated dataset RANDOM containing 100 rows and 100 columns, with each entry independently generated from the normal distribution with zero mean and unit variance. The three other artificial datasets CLUSTER, GAUSSIAN and COMPONENT are each generated for one of the structural measures and are thus explained in the corresponding subsections. Table 5.4 shows some properties of the datasets. The values were linearly scaled into [0, 1]. In the following, rows denote data points and columns dimensions.

We used the same parameters for the methods for all datasets as in Section 5.2. We generated 1000 randomized samples with each method for all datasets using the Besag approach introduced in Subsection 4.2.5. From these samples the structural measures were calculated. Next, we give results for each of the three structural measures.

5.3.1 Clustering error

For clustering purpose, we generated a dataset CLUSTER which has a clear Gaussian cluster structure with 10 clusters, each with 10–200 points. Cluster centers were drawn from $\mathcal{N}_{100}(0, 1)$ and cluster points were produced by adding random values from $\mathcal{N}_{100}(0, 1)$ to the cluster center. The clustering structure in CLUSTER should be significant given the row and column means and variances.

We calculated the K-means clustering error defined in Equation (2.1) with 10 clusters for each sample produced by the three methods. Finally, the p-values

Dataset	Method	Measure	p-value
RANDOM	Original data	147.02	
	SwapDiscretized	146.74 (0.52)	0.702
	SwapMetropolis	146.71 (0.55)	0.713
	MaskMetropolis	147.35 (0.54)	0.261
CLUSTER	Original data	457.33	
	SwapDiscretized	659.47 (0.77)	0.001
	SwapMetropolis	661.95 (0.63)	0.001
	MaskMetropolis	656.31 (0.93)	0.001
GENE	Original data	525.53	
	SwapDiscretized	592.38 (1.24)	0.001
	SwapMetropolis	610.70 (0.99)	0.001
	MaskMetropolis	592.29 (1.24)	0.001

Table 5.5: K-means clustering errors with 10 clusters calculated for each original dataset and randomizations with each method. The average clustering error in 1000 randomizations is given. The values in parentheses are the standard deviations. The p-values are calculated for the original data matrices with the hypothesis that the original data contain cluster structure.

were calculated for the original data with the hypothesis that the original data have smaller clustering error than the randomized matrices. The results for the K-means clustering are shown in Table 5.5.

For the actual calculation of K-means clustering, we used a C++ implementation of the K-means++ algorithm [3] described in Subsection 2.2.1 integrated with MATLAB. To make the results more stable, we repeated the calculation ten times for each sample and used the minimum value for the clustering error.

We notice that randomizations of dataset RANDOM contain around the same amount of clustering structure as the original data. Thus, the p-values are around 0.5 and we can conclude that the clustering structure in RANDOM dataset depends purely on the row and columns sums and variances.

However, with CLUSTER dataset we see that the clustering structure has disappeared totally. All of the randomizations had larger clustering error than the original data. Thus the dataset CLUSTER contains significant clustering structure which cannot be explained only by the row and column statistics. We got similar results for the GENE dataset.

Dataset	Method	Measure	p-value
RANDOM	Original data	0.363	
	SwapDiscretized	0.361 (0.028)	0.430
	SwapMetropolis	0.361 (0.029)	0.407
	MaskMetropolis	0.360 (0.029)	0.406
GAUSSIAN	Original data	0.993	
	SwapDiscretized	0.992 (0.002)	0.398
	SwapMetropolis	0.992 (0.002)	0.395
	MaskMetropolis	0.992 (0.002)	0.373
GENE	Original data	0.995	
	SwapDiscretized	0.737 (0.046)	0.001
	SwapMetropolis	0.644 (0.026)	0.001
	MaskMetropolis	0.657 (0.024)	0.001

Table 5.6: Maximum correlation values between the rows calculated for each original dataset and randomizations with each method. The average of maximum correlation in 1000 randomizations is given. The values in parentheses are the standard deviations. The p-values are calculated for the original data matrices with the hypothesis that the original data contains a high correlation.

5.3.2 Maximum correlation

The second structural measure we studied was the maximum correlation between rows. We generated dataset GAUSSIAN containing 1000 points taken from 10-dimensional normal distribution with unit variance and with center drawn from $\mathcal{N}_{10}(0, 1)$. The data has high correlation between the rows, but, however, it should be totally explained by row and column statistics.

We calculated the correlation between each two rows as defined in Equation (2.7) for each sample and used the maximum of them as a structural measure for the corresponding sample. Finally, p-value was calculated for the original data with the hypothesis that the original data has a higher maximum correlation than the randomizations. Results are presented in Table 5.6.

Notice that assessing the significance of the maximum correlation between rows is an instance of the multiple-test problem discussed in Subsection 3.1.4. However, as we are performing the same data mining task also to the randomized matrices, we get valid p-values. Thus we are not fixing the pair of rows whose correlation we are studying. Instead we are calculating correlations between all

pairs of rows in randomized matrices as well.

We notice that the results with RANDOM dataset are as expected. However, this time there is not almost any fluctuation in the results between the three methods. With GAUSSIAN dataset we notice that although there is a high maximum correlation value in the original data, it remains also in the randomized matrices. Thus our methods have been able to classify the notable structure in the original data to depend only on the row and column statistics. With GENE dataset we again assess that its high original maximum correlation value between rows is really significant.

5.3.3 Principal components

As the last structural measure we used the fraction of variance explained by the five main principal components as explained in Subsection 2.2.3. We generated dataset COMPONENT containing 1000 random points with 5 intrinsic dimensions. The points were linearly transformed into a 50-dimensional space and Gaussian noise was added to the points. The COMPONENT dataset contains a clear intrinsic dimension which should not be explained by the row and column statistics.

We calculated the fraction of variance explained by the first five principal components for the original datasets and randomized samples. The p-value was calculated with the hypothesis that the original data has higher fraction of variance explained. The results are presented in Table 5.7.

For RANDOM dataset we observe that the randomized matrices have very similar structural measures as the original matrix. For COMPONENT dataset we see that the randomizations do not contain as small intrinsic dimension as the original data although large amount of variance is explained by the main five principal components. As with the other two structural measures, the structure in dataset GENE is independent from the row and column statistics, and therefore interesting.

Dataset	Method	Measure	p-value
RANDOM	Original data	0.173	
	SwapDiscretized	0.174 (0.003)	0.625
	SwapMetropolis	0.173 (0.003)	0.486
	MaskMetropolis	0.174 (0.003)	0.607
COMPONENT	Original data	0.941	
	SwapDiscretized	0.765 (0.001)	0.001
	SwapMetropolis	0.736 (0.001)	0.001
	MaskMetropolis	0.769 (0.000)	0.001
GENE	Original data	0.605	
	SwapDiscretized	0.454 (0.001)	0.001
	SwapMetropolis	0.433 (0.001)	0.001
	MaskMetropolis	0.456 (0.001)	0.001

Table 5.7: The fraction of variance explained by the first five principal components calculated for each original dataset and randomizations with each method. The average of fraction of variance in 1000 randomizations is given. The values in parentheses are the standard deviations. The p-values are calculated for the original data matrices with the hypothesis that the original data contains a high fraction of variance explained.

Chapter 6

Conclusions and discussion

In this thesis, we have studied the problem of significance testing of data mining results. We restricted our consideration to real-valued matrices and viewed the data mining results as interesting if it is unlikely to obtain as good results on randomized matrices having the same row and column means and variances as the original matrix. This randomization approach can be used to give empirical p-values to data mining results on real-valued matrices and therefore to assess the significance of the results.

We gave three iterative methods, *SwapDiscretized*, *SwapMetropolis* and *MaskMetropolis*, for randomizing real-valued matrices while preserving the row and column means and variances. The methods were analyzed both theoretically and empirically. Visual examples of randomizations were given for each method. Finally, the methods were used for significance testing of three different data mining methods on four artificial datasets and on one real dataset. Our empirical tests imply that the obtained p-values clearly show whether the data mining result is significant or not.

The methods were fast in practice, and around $100mn$ attempted local modifications were needed with each method to produce a randomization of an $m \times n$ matrix. Thus the convergence properties of the methods were similar. However, they had differences in preserving the row and column sums and variances. *MaskMetropolis* produced the most accurate matrices and it also preserved the means exactly. The two other methods, *SwapDiscretized* and *SwapMetropolis*,

produced matrices with similar error. However, in discretized space the method *SwapDiscretized* preserved the row and column statistics exactly. The selection of parameters for the methods affected the error rates but with reasonable parameters *MaskMetropolis* produced the least error.

The most important difference between the methods was the distribution of the values of randomized matrices. *MaskMetropolis* changed the values directly whereas *SwapDiscretized* and *SwapMetropolis* only rearranged the values. The resulting distribution of the values with *MaskMetropolis* was Gaussian regardless of the original distribution. However, the results obtained in significance testing were similar with each method. Thus the differences in the methods did not affect the results.

The three methods presented are obviously not the only choices to solve the defined problem. We shortly presented also a couple of other approaches we have tested. However, they were not good in practice nor in theory. Nevertheless, the Metropolis-Hastings approach can be used with virtually any local modification. Our methods used two different modification templates: swap rotations and addition masks. Both of them affects four elements in the matrix concurrently. However, the local modification could be more drastic or conservative or even global.

Proving theoretical properties of the convergence of the methods is hard. As theoretical results of convergence of even the 0–1 case of our problem do not exist, it seems quite hard to obtain methods for randomization of real-valued matrices which would have provable convergence times. However, it would be interesting to know the properties of the space of the matrices having the same row and column sums and variances in more detail. If we could understand the shape of that space better, we might be able to produce more efficient methods for sampling from it.

To overcome the problem of dependency between the randomized samples in significance testing, we applied the approach suggested by Besag *et al.* [6–8]. We used the parallel version of the approach where a new starting state was first produced by running the chain backwards and after that each randomized sample was independently produced by starting a new chain from that initial state and running it forwards. Applying the more efficient serial version of this test could

produce notable speedup in convergence.

However, we can produce independent samples also by starting the chain each time from a new random matrix. For example with *SwapMetropolis* the starting state could be a random permutation of the values. The problem with this approach is that we cannot guarantee the exchangeability condition with the original matrix. Nevertheless, this approach has some good properties and studying it theoretically could be beneficial. With this approach the methods would converge faster and the randomized matrices could have less error in row and column sums and variances as the independence is guaranteed.

The final important open issue is the choice of statistics that are preserved in randomization-based significance testing. Our approach of preserving the row and column means and variances is quite natural as they describe the central features of the underlying phenomenon. On the other hand, in some applications we could need other weaker or stronger statistics. One should notice, that the statistics that are preserved in randomization are the properties that are considered to be uninteresting — everything else is then interesting and thus significant.

For example, we could try to preserve the whole distribution of the values in rows and columns instead of only means and variances. If the distribution of the underlying phenomenon is not a normal distribution, this approach could produce more reliable results than our approach, which tends to normalize everything. However, the study of other approaches is left for future work.

In summary, we have introduced the problem of randomizing real-valued matrices for assessing the significance of data mining results. We gave three methods to solve the problem. The experimental results were promising, but more study is needed before the approach can be recommended to practioners.

Bibliography

[1] Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael I. Jordan. An Introduction to MCMC for Machine Learning. *Machine Learning*, 50(1):5–43, 2003.

[2] David Arthur and Sergei Vassilvitskii. How slow is the k-means method? In *SCG '06: Proceedings of the twenty-second annual symposium on Computational geometry*, pages 144–153, New York, NY, USA, 2006. ACM.

[3] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[4] Pierre Baldi and G. Wesley Hatfield. *DNA Microarrays and Gene Expression: From Experiments to Data Analysis and Modeling*. Cambridge University Press, 2002.

[5] Pavel Berkhin. Survey of Clustering Data Mining Techniques. Technical report, Accrue Software, San Jose, CA, 2002.

[6] Julian Besag. Markov chain Monte Carlo methods for statistical inference. http://www.ims.nus.edu.sg/ Programs/mcmc/files/besag_tl.pdf, 2004.

[7] Julian Besag and Peter Clifford. Generalized Monte Carlo significance tests. *Biometrika*, 76(4):633–642, 1989.

[8] Julian Besag and Peter Clifford. Sequential Monte Carlo p-values. *Biometrika*, 78(2):301–304, 1991.

[9] Ivona Bezáková, Nayantara Bhatnagar, and Eric Vigoda. Sampling binary contingency tables with a greedy start. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 414–423. SIAM, 2006.

[10] Ivona Bezáková, Alistair Sinclair, Daniel Stefankovic, and Eric Vigoda. Negative examples for sequential importance sampling of binary contingency tables. http://arxiv.org/abs/math.ST/0606650, 2006.

[11] Alvis Brazma and Jaak Vilo. Gene expression data analysis. *Microbes Infect*, 3:823–829, Aug 2001.

[12] Sergey Brin, Rajeev Motwani, and Craig Silverstein. Beyond market baskets: Generalizing association rules to correlations. In *SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA*, pages 265–276. ACM, 1997.

[13] Yuguo Chen, Persi Diaconis, Susan P. Holmes, and Jun S. Liu. Sequential Monte Carlo methods for statistical analysis of tables. *Journal of the American Statistical Association*, 100(469):109–120, 2005.

[14] George W. Cobb and Yung-Pin Chen. An application of Markov chain Monte Carlo to community ecology. *The American Mathematical Monthly*, 110:265–288, Apr 2003.

[15] Persi Diaconis and Anil Gangolli. Rectangular arrays with fixed margins. In *Discrete Probability and Algorithms*, pages 15–41, 1995.

[16] William DuMouchel and Daryl Pregibon. Empirical Bayes screening for multi-item associations. In *Knowledge Discovery and Data Mining*, pages 67–76, 2001.

[17] Martin Dyer. Approximate counting by dynamic programming. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9–11, 2003, San Diego, CA, USA*, pages 693–699. ACM, 2003.

[18] A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis*. Chapman & Hall, New York, 2nd edition, 2003.

[19] Aristides Gionis, Heikki Mannila, Taneli Mielikäinen, and Panayiotis Tsaparas. Assessing data mining results via swap randomization. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 167–176, New York, NY, USA, 2006. ACM Press.

[20] Phillip Good. *Permutation Tests: A Practical Guide to Resampling Methods for Testing Hypotheses*. Springer, 2nd edition, 2000.

[21] Øyvind Hammer and David A. T. Harper. *Paleontological Data Analysis*. Wiley-Blackwell, 2005.

[22] David J. Hand, Heikki Mannila, and Padhraic Smyth. *Principles of data mining*. MIT Press, Cambridge, MA, USA, 2001.

[23] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *Applied Statistics*, 28(1):100–108, 1979.

[24] W. Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[25] Harold Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441,498–520, 1933.

[26] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[27] Szymon Jaroszewicz and Dan A. Simovici. A general measure of rule interestingness. In *5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2001)*, pages 253–265, 2001.

[28] Nadav Kashtan, Shalev Itzkovitz, Ron Milo, and Uri Alon. Efficient sampling algorithm for estimating subgraph concentrations and detecting network motifs. *Bioinformatics*, 20(11):1746–1758, 2004.

[29] John A. Lee and Michel Verleysen. *Nonlinear Dimensionality Reduction*. Information Science and Statistics. Springer, 1st edition, October 2007.

[30] Bing Liu, Wynne Hsu, and Yiming Ma. Pruning and summarizing the discovered associations. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August 15-18, 1999, San Diego, CA, USA*, pages 125–134. ACM, 1999.

[31] Bing Liu, Wynne Hsu, and Yiming Ma. Identifying non-actionable association rules. In *Knowledge Discovery and Data Mining*, pages 329–334, 2001.

[32] S. Lloyd. Least Squares Quantization in PCM. Technical report, Bell Laboratories, 1957.

[33] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, Berkeley*, pages 281–297. University of California Press, 1967.

[34] S. C. Madeira and A. L. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, 2004.

[35] Nimrod Megiddo and Ramakrishnan Srikant. Discovering predictive association rules. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98), August 27-31, 1998, New York City, New York, USA*, pages 274–278. AAAI Press, 1998.

[36] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Mici Teller, and Edward Teller. Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[37] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298:824–827, 2002.

[38] Markus Ojala. Sampling Real Matrices with Given Margins, 2007. Special assignment, Laboratory of Computer and Information Science, Helsinki University of Technology.

[39] K Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 1901.

[40] Herbert J. Ryser. Combinatorial properties of matrices of zeros and ones. *Canadian J. Math*, 9:371–377, 1957.

[41] James G. Sanderson. Testing ecological patterns. *American Scientist*, 88:332–339, 2000.

[42] Uwe Scherf *et al.* A gene expression database for the molecular pharmacology of cancer. *Nature Genetics*, 24:236–244, 2000.

[43] J. Schuchhardt, D. Beule, A. Malik, E. Wolski, H. Eickhoff, H. Lehrach, and H. Herzel. Normalization strategies for cDNA microarrays. *Nucleic Acids Research*, 28(10):e47, 2000.

[44] Tom A.B. Snijders. Enumeration and simulation methods for 0–1 matrices with given marginals. *Psychometrika*, 56:397–417, Sep 1991.

[45] Lloyd N. Trefethen and David Bau III. *Numerical Linear Algebra*. SIAM: Society for Industrial and Applied Mathematics, June 1997.

[46] Olga Troyanskaya, Michael Cantor, Gavin Sherlock, Pat Brown, Trevor Hastie, Robert Tibshirani, David Botstein, and Russ B. Altman. Missing value estimation methods for DNA microarrays. *Bioinformatics*, 17(6):520–525, 2001.

[47] Antti Ukkonen and Heikki Mannila. Finding outlying items in sets of partial rankings. In *PKDD*, pages 265–276, 2007.

[48] Bo-Ying Wang and Fuzhen Zhang. Precise number of $(0, 1)$-matrices in $U(R, S)$. *Discrete Mathematics*, 187:211–220, 1998.

[49] Hui Xiong, Shashi Shekhar, Pang-Ning Tan, and Vipin Kumar. Exploiting a support-based upper bound of Pearson's correlation coefficient for efficiently identifying strongly correlated pairs. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 334–343. ACM, 2004.