UNIVERSITÄT DES SAARLANDES

DOCTORAL THESIS

Matrix Factorization over Dioids and its Applications in Data Mining

Sanjar Karaev

Dissertation
zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Fakultät für Mathematik und Informatik
der Universität des Saarlandes

Saarbrücken, September 2019

Day of Colloquium	10 July 2019
Dean of the Faculty	Univ.-Prof. Dr. Sebastian Hack
Chair of the Committee	Univ.-Prof. Dr. Kurt Mehlhorn
First Reviewer	Prof. Dr. Pauli Miettinen
Second Reviewer	Univ.-Prof. Dr. Gerhard Weikum
Third Reviewer	Prof. Dr. Matthijs van Leeuwen
Academic Assistant	Dr. Oana Balalau

Abstract

Matrix factorizations are an important tool in data mining, and they have been used extensively for finding latent patterns in the data. They often allow to separate structure from noise, as well as to considerably reduce the dimensionality of the input matrix. While classical matrix decomposition methods, such as *nonnegative matrix factorization* (NMF) and *singular value decomposition* (SVD), proved to be very useful in data analysis, they are limited by the underlying algebraic structure. NMF, in particular, tends to break patterns into smaller bits, often mixing them with each other. This happens because overlapping patterns interfere with each other, making it harder to tell them apart.

In this thesis we study matrix factorization over algebraic structures known as dioids, which are characterized by the lack of additive inverse ("negative numbers") and the idempotency of addition $(a + a = a)$. Using dioids makes it easier to separate overlapping features, and, in particular, it allows to better deal with the above mentioned pattern breaking problem.

We consider different types of dioids, that range from continuous (subtropical and tropical algebras) to discrete (Boolean algebra). Among these, the Boolean algebra is perhaps the most well known, and there exist methods that allow one to obtain high quality Boolean matrix factorizations in terms of the reconstruction error. In this work, however, a different objective function is used – the description length of the data, which enables us to obtain compact and highly interpretable results.

The tropical and subtropical algebras, on the other hand, are much less known in the data mining field. While they find applications in areas such as job scheduling and discrete event systems, they are virtually unknown in the context of data analysis. We will use them to obtain idempotent nonnegative factorizations that are similar to NMF, but are better at separating the most prominent features of the data.

Kurzfassung

Matrix-Faktorisierungen sind ein wichtiges Werkzeug in Data-Mining und wurden umfangreich zum Auffinden latenter Muster in den Daten verwendet. Oft erlauben sie, die Struktur vom Rauschen zu trennen, sowie Dimensionalität von der Eingabematrix wesentlich zu reduzieren. Obwohl klassische Methoden für die Matrix-Zerlegung, wie z.B. *nicht negative Matrixfaktorisierung* (NMF) und *Singulärwertzerlegung* (SVD), in der Datenanalyse sich als sehr nützlich erwiesen haben, sind sie durch die zugrunde liegende algebraische Struktur eingeschränkt. Insbesondere neigt NMF dazu, Muster in kleinere Bits zu brechen, und vermischt sie oft miteinander. Das passiert, weil überschneidende Muster sich gegenseitig stören, sodass es schwieriger ist, sie auseinander zu halten.

In dieser Dissertation werden Matrix-Faktorisierungen über algebraische Strukturen, sogenannte Dioiden, untersucht, die sich durch die fehlende additive Inverse ("negative Zahlen") und Idempotenz der Addition $(a + a = a)$ auszeichnen. Mit Dioiden ist es einfacher überschneidende Merkmale zu trennen. Insbesondere erlauben sie besser mit dem erwähnten Musterbrechenproblem umzugehen.

Es werden unterschiedliche Dioiden untersucht, die von kontinuierlichen (subtropische und tropische Algebren) bis zu diskreter (Boolesche Algebra) reichen. Unter diesen, die Boolesche Algebra ist wahrscheinlich die bekannteste, und es gibt Methoden, die ermöglichen hochwertiger Matrix-Faktorisierungen in Bezug auf den Rekonstruktionsfehler zu erzielen. In dieser Arbeit aber wird eine andere Zielfunktion verwendet: Die Länge der Beschreibung von den Daten. Die Zielfunktion ermöglicht uns kompakte und hochinterpretierbare Ergebnisse zu erzielen.

Andererseits sind die tropische und subtropische Algebren viel weniger im Bereich Data-Mining bekannt. Sie finden zwar Anwendungen in Bereichen wie Job-Scheduling und diskrete Ereignissysteme, jedoch sind sie im Kontext von Datenanalyse nahezu unbekannt. Hier werden sie verwendet, um idempotente, nicht negative Faktorisierungen zu erhalten, die NMF ähneln, aber die wichtigsten Merkmale der Daten besser voneinander trennen.

Summary

Matrix factorization methods provide an efficient way of analysing complex, high dimensional data, and often yield highly interpretable results. The idea is, given the input data in a form of a matrix, to (approximately) represent it as a product of two or more lower-dimensional factor matrices. This allows to summarize the data with a relatively small number of latent features (patterns), that are then usually easier to interpret. In classical methods, such as, for example, *nonnegative matrix factorization* (NMF), each element of the input matrix is represented as the sum of the corresponding elements of each pattern. This leads to the so called *component interpretation*, that allows for independent analysis of patterns.

While the component interpretation has proven to be an effective data analysis technique, it is considerably less useful when many patterns contribute to the same data point. More specifically, since all features are summed together, it becomes hard to identify which one made the greatest contribution. In turn, this makes the interpretation of each pattern dependent on other patterns, thus preventing decoupling of the data.

The purpose of this thesis is to address the above problem by changing the way patterns are aggregated. From the previous discussion, it is apparent that the culprit is using the summation as our aggregation operation. In order to be able to determine what pattern is the most dominant at any given point, we will use *dioid* data structures, meaning that the addition operation has to satisfy $a + a = a$ for any element a. This property, which is called $idempotency$, allows for much easier separation of patterns. In fact, we will require an even stronger property that for any elements a and b, their sum should be either a or b. This will guarantee that for every element in the data there is a single pattern (winner) completely determining its value. Furthermore, all other patterns do not make any contribution at this point, which allows us to interpret it as being "represented" by the dominant pattern. This thesis covers several dioid types and corresponding matrix factorization problems.

- **Subtropical Matrix Factorization.** The first dioid type is the *subtropical algebra*, which is defined over the set of nonnegative real numbers and characterized by using the maximum operation instead of the addition. Its

structure is somewhat similar to NMF due to its nonnegative factors, but because it is also a dioid, it ensures that for each element in the input matrix there is a single pattern determining it. Compared to the NMF model, it improves the interpretability by allowing patterns to directly explain the data. We proposed two different algorithms for solving the corresponding matrix factorization problem. These algorithms were published in [Karaev and Miettinen, 2016b] and [Karaev and Miettinen, 2016a]. The first algorithm is optimized to work on the data with discrete noise (when some elements get flipped to random values), while the second one is designed for continuous (e.g. Gaussian) noise. We demonstrated that these methods yield interpretable results on real-world data, and are often complementary to NMF. We later published a more general approach, that encompasses both of the above discussed methods [Karaev and Miettinen, 2019].

- **Mixed Model: Combining Nonnegative and Subtropical Matrix Factorization.** Both NMF and subtropical matrix factorization have their advantages: while the former can effectively recover smooth transitions from low to high values in the data, the latter helps to identify the most prominent features. In the real world data rarely follows any specific model exactly – rather it can often be a mix of several models and noise. Instead of trying to force a model on a particular piece of data, we proposed an approach that combines NMF and subtropical matrix factorization. It represents the input matrix as a convex combination of both models, allowing for more flexibility than either of them. By conducting experiments on real-world data, we demonstrated that this approach produces highly interpretable results. This work was published in [Karaev et al., 2018a].

- **Boolean Matrix Factorization with Minimum Description Length.** Dioids are not restricted to continuous data – in fact, the Boolean algebra, that is defined on the set $\{0, 1\}$ of binary numbers, is an example of a dioid. In Boolean matrix factorization (BMF) the task is to represent the input matrix as a Boolean product of two lower-dimensional binary factor matrices. The binary error (the number of falsely reconstructed entries) is traditionally used as the optimization objective. While this choice of a metric seems quite intuitive, there is an implicit assumption that 0 is as likely to be turned to 1 as 1 is to 0. Given that much of the real-world binary data is of presence-absence nature, this does not seem very reasonable. It appears that in most cases failing to observe something that is actually present is more likely than reporting something that is not. For example, an animal species not having been reported in a certain geographical area does not necessarily mean that it is actually absent there – rather the lack of observation might be due to some other factor, such as the remoteness of a location. On the other hand, since such observations are usually made by professionals, false positives seem much less likely [see e.g. MacKenzie et al., 2002, for an in depth discussion]. Instead of optimizing the binary error, we use the *minimum description length (MDL)* principle [Rissanen, 1978, Grünwald, 2007]. MDL postulates that a model that yields a shorter representation of the data should be preferred. In our work [Karaev et al., 2015] we proposed a BMF algorithm that directly optimizes the description length, and that is highly resistant to destructive noise (failing to report positives). We further demonstrated that it can find intuitive results in real-world data.

- **Nested Subgraph Identification Using the Tropical Algebra.** While there are BMF algorithms that are highly effective at representing a binary matrix as a collection of rank-1 submatrices (cliques), it is important to note that not all binary data lends itself easily to clique summarization. For example, many communities in social networks have a densely interconnected core, that is only loosely connected to the remaining nodes. They are known to follow the so called *core-periphery* model [Borgatti and Everett, 1999]. An extreme example of this model, and one that is, in a sense, opposite to a clique, is a star. A star is a community that only has one node that is connected to all other nodes. All these community types are examples of *nested* matrices. It is therefore of interest to study the problem of summarizing a given binary matrix as a collection of nested matrices. It has been studied before, but the nested submatrices were almost always assumed to be non-overlapping. In [Karaev et al., 2018b] we remove this restriction by making use of another dioid, the *tropical algebra*, and propose a highly scalable algorithm for solving the corresponding matrix factorization problem.

Acknowledgments

First and foremost, I would like to thank my supervisor, Prof. Pauli Miettinen, who provided me with all the necessary guidance during my time as a PhD student. While paying very close attention to every aspect of my work, he gave me the freedom to explore my ideas and get a hands-on experience in doing research.

I would also like to express my deep gratitude to Prof. Gerhard Weikum for giving me the opportunity to pursue my PhD degree at the Databases and Information Systems group of the Max-Planck Institute for Informatics.

This work was done during my membership at the Max-Planck Institute for Informatics and Saarland University. I would like to thank the International Max-Planck Research School for Computer Science and the Saarbrücken Graduate School of Computer Science for their financial support, which enabled me to dedicate myself fully to research.

Like any long term commitment, PhD has many ups and downs, and it is vital to have peers that one can always rely on and share experiences with. I want to thank all of my friends for their support, and especially Rashid Ibragimov, Aleksander Torguzov, and Sergiy Belozorov, with whom I had countless interesting discussions. Rashid also provided assistance when I first arrived to Germany and helped to translate the abstract of this thesis, for which I am deeply grateful.

Last, but not least, I would like to thank my parents, who were always there to support me.

Contents

CHAPTER 1

Introduction

Finding simple patterns that can be used to describe the data is one of the main problems in data mining. Many different approaches are known for this general task, but one of the most common pattern finding techniques rarely gets classified as such. Matrix factorizations (or decompositions, these two terms are used interchangeably in this thesis) represent a given input matrix $\boldsymbol{A}$ as a product of two (or more) factor matrices, $\boldsymbol{A} \approx \boldsymbol{BC}$. This standard formulation of matrix factorizations makes their pattern mining nature less obvious, but let us write the matrix product $\boldsymbol{BC}$ as a sum of rank-1 matrices, $\boldsymbol{BC} = \boldsymbol{F}_1 + \boldsymbol{F}_2 + \cdots + \boldsymbol{F}_k$, where $\boldsymbol{F}_i$ is the outer product of the ith column of $\boldsymbol{B}$ and the ith row of $\boldsymbol{C}$. Now it becomes clear that the rank-1 matrices $\boldsymbol{F}_i$ are "simple patterns", and the matrix factorization produces k such patterns whose sum is a good approximation of the original data matrix.

This so-called "component interpretation" [Skillicorn, 2007] is more appealing with some factorizations than with others. For example, classical singular value decomposition (SVD) does not easily admit such an interpretation, as the components are not easy to interpret without knowing the earlier components. On the other hand, the motivation for nonnegative matrix factorization (NMF) often comes from the component interpretation, as can be seen, for example, in the famous "parts of faces" figures of Lee and Seung [1999]. The "parts-of-whole" interpretation is at the heart of NMF: every rank-1 component adds something to the overall decomposition, and never removes anything. This aids with the interpretation of the components, and is also often claimed to yield sparse factors, although this latter point is more contentious [see e.g. Hoyer, 2004].

Perhaps the reason why matrix factorization algorithms are not often considered as pattern mining methods is that the rank-1 matrices are summed together to build the full data. Due to this summation, it is rare for any rank-1 component to explain any part of the input matrix alone since it would be added together with other components. But the use of summation as a way of aggregating rank-1 components can be considered to be "merely" a consequence of the underlying algebraic structure. If we change the algebra – in particular, if the way components are aggregated is altered – it might be possible to have more prominent patterns that would (locally) explain the data.

In this thesis we tackle the above problem by changing the algebraic operation used for the elementwise component aggregation. Let us denote this operation by $\Box$. In order for element $\boldsymbol{A}_{ij}$ of the input matrix $\boldsymbol{A}$ to be completely determined by a single component, we must have $(\boldsymbol{F}_1 \Box \boldsymbol{F}_2 \Box \cdots \Box \boldsymbol{F}_k)_{ij} = (\boldsymbol{F}_p)_{ij}$ for some p. This can be enforced by requiring that $\Box$ satisfies

$$a \Box b \in \{a, b\} \tag{1.1}$$

for any a and b from its domain of definition. By placing this constraint on the aggregation operation, we ensure that each element in the matrix product is completely explained by the most dominant pattern at this point. Since neither of the remaining components influences the outcome, we call this the "winner-takes-it-all" interpretation. This is in stark contrast to standard matrix factorization methods, such as NMF, where each pattern would only make a "contribution" to the final value.

Any algebraic operation satisfying (1.1) is automatically *idempotent*, that is for any value a we have $a \Box a = a$. This also means that, together with the multiplication operation, $\Box$ forms a *dioid*, which is a semiring with an idempotent addition. Throughout this thesis we will only consider dioids that also satisfy (1.1) since this is necessary to guarantee that each matrix element is explained by a single pattern.

A prominent example of a dioid is the so called *subtropical algebra* that is defined over the set of nonnegative real numbers and is characterized by using the maximum operation instead of addition. Correspondingly, in the subtropical matrix factorization rank-1 components are combined using the maximum operation instead of the standard elementwise summation, which means that for every element there is always a component that is the "winner", and explains its value. The following example, where we use $\boxtimes$ to define the subtropical matrix factorization, illustrates this point.

$$\begin{aligned}
\begin{pmatrix} 1 & 2 & 0 \\ 2 & 4 & 1 \\ 0 & 4 & 2 \end{pmatrix} &= \begin{pmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 2 \end{pmatrix} \boxtimes \begin{pmatrix} 1 & 2 & 0 \\ 0 & 2 & 1 \end{pmatrix} \\
&= \underbrace{\begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix} \times \begin{pmatrix} 1 & 2 & 0 \end{pmatrix}}_{\boldsymbol{F}_1} \max \underbrace{\begin{pmatrix} 0 \\ 1 \\ 2 \end{pmatrix} \times \begin{pmatrix} 0 & 2 & 1 \end{pmatrix}}_{\boldsymbol{F}_2} .
\end{aligned}$$

Here each element of the input matrix can be explained by the corresponding entry from either $\boldsymbol{F}_1$ or $\boldsymbol{F}_2$. While for the standard algebra multiple rank-1 components are needed to explain a data point, with the subtropical algebra it suffices to consider only the component with the largest value.

The winner-takes-it-all interpretation of subtropical matrix factorization can be seen as the opposite of the parts-of-whole nature of NMF. Moreover, both models can be applied to any nonnegative matrix, which raises the question of which approach should be used on a dataset at hand. While in some cases one of the methods is clearly better than the other, real-world data rarely follows any specific model exactly. It might happen that, while some parts of the data exhibit more NMF-like features, the rest of it is better explained using the more discrete subtropical algebra. Rather than trying to force a method on the data, we can represent the input matrix as a mix of both structures, which leads to what we call the mixed linear-tropical model. By learning which parts of the matrix are better represented using NMF and where to use the subtropical algebra, this model allows to fit the data more precisely and recover more patterns than either of the constituting methods.

Another important dioid is the *Boolean* algebra. It is defined over the the set $\{0, 1\}$ of binary numbers and uses the operations of logical `OR` and `AND`. While it might seem mathematically trivial, it is a useful tool for analyzing binary datasets. In particular, Boolean matrix factorization (or BMF) provides a natural way of extracting binary patterns and generally leads to more interpretable results than if the normal addition was used [Miettinen, 2009]. This is because, when rank-1 components are combined to obtain an approximation of the input matrix, the result would still be binary.

While there are many cost functions that BMF methods can optimize, traditionally the emphasis has been on minimizing the binary reconstruction error, that is, the total number of wrongly reconstructed entries. This seems natural since, when assessing the quality of a decomposition, we intuitively assume that there is no difference between misclassifying 1 as 0 and 0 as 1. In the real world, however, this assumption is often incorrect. Consider, for example, a matrix representing distributions of different animal species. If a species can be found at a specific geographical location, then the corresponding entry is 1, otherwise it is 0. It appears entirely possible that in very remote places data might be underreported, resulting in many spurious 0s. On the other hand, mistakenly reporting animals as present seems less likely. Since a lot of the real-world binary data is of the presence/absence form, it seems reasonable to assume that the balance of false positives and false negatives is often skewed. To tackle this problem, the use of the minimum description length principle (MDL) as the measure of the quality of a BMF decomposition was proposed by Miettinen and Vreeken [2012]. MDL postulates that the model that produces the shortest encoding of the data should be preferred, and can be seen as a formalization of Occam's razor principle. In this thesis we will introduce a BMF algorithm that directly optimizes the description length, and verify that it produces highly interpretable results on real-world data.

BMF can be thought of as summarizing a binary matrix with a collection of (quasi-)cliques. While this approach is quite effective and usually produces readily interpretable results, restricting patterns to cliques limits what kind of structure can be found. For example, if the input data represents connections in a social network, we would only be able to identify communities that are highly interconnected. It is known, however, that many real-world communities are not cliques. One of the best studied non-clique community types is the so called *core-periphery* model [see e.g. Borgatti and Everett, 1999], where there is a dense core that is only weakly connected to the remaining nodes. Another common community type is a star, which can be considered to be the opposite of a clique as only a single node is connected to the rest of the community. Even though these pattern shapes might seem very diverse and unrelated, they are all particular cases of a more general concept, a so called *nested* matrix. This leads naturally to the problem of summarizing a binary matrix with a set of nested submatrices. While it was studied before, earlier work almost always required patterns to be non-overlapping, which is a major limitation – for example, for a social network, it would mean that a person can only belong to a single community. In this thesis we lift this restriction by making use of another dioid, known as the *tropical algebra*, and formulate a binary matrix factorization problem that allows nested submatrices to overlap.

Thesis roadmap. We will start by giving a formal definition of the dioid as well as presenting some associated theoretical results in Chapter 2. The remaining chapters present matrix factorization methods over different types of dioids and their applications to data mining. First we will cover the subtropical matrix factorization in Chapter 3. Then in Chapter 4 a mixed approach is presented, where the subtropical and NMF structures are combined in a single model. The last part of the thesis deals with binary data. In Chapter 5 we will see how dioid methods can be applied to community mining in graphs, and Chapter 6 will present a new Boolean matrix factorization method.

Contributions. The thesis is based on the following 6 papers:

- Sanjar Karaev, Pauli Miettinen, and Jilles Vreeken. Getting to know the unknown unknowns: Destructive-noise resistant boolean matrix factorization. In *15th SIAM International Conference on Data Mining (SDM)*, pages 325–333, 2015
- Sanjar Karaev and Pauli Miettinen. Capricorn: An algorithm for subtropical matrix factorization. In *16th SIAM International Conference on Data Mining (SDM)*, pages 702–710, 2016b
- Sanjar Karaev and Pauli Miettinen. Cancer: Another algorithm for subtropical matrix factorization. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 576–592, 2016a
- Sanjar Karaev and Pauli Miettinen. Algorithms for approximate subtropical matrix factorization. *Data Min. Knowl. Discov.*, 33(2):526–576, 2019. doi: https://doi.org/10.1007/s10618-018-0599-1
- Sanjar Karaev, James Hook, and Pauli Miettinen. Latitude: A model for mixed linear–tropical matrix factorization. In *18th SIAM International Conference on Data Mining (SDM)*, pages 360–368, 2018a
- Sanjar Karaev, Saskia Metzler, and Pauli Miettinen. Logistic-tropical decompositions and nested subgraphs. In *4th International Workshop on Mining and Learning with Graphs*, 2018b

Chapters 2 and 3, that cover theoretical aspects of dioids and algorithms for subtropical matrix factorization, contain the material from papers Karaev and Miettinen [2016b], Karaev and Miettinen [2016a], and Karaev and Miettinen [2019]. Chapter 4, which discusses the mixed tropical-NMF model, corresponds to the paper Karaev et al. [2018a]. Finally, applications of dioids to binary data, discussed in Chapters 5 and 6, are based on Karaev et al. [2018b] and Karaev et al. [2015], respectively.

CHAPTER 2

Matrix Factorization over Dioids

2.1 Notation and Basic Definitions

Vectors, matrices, and sets. Throughout this work, we will denote a matrix by upper-case boldface letters ($\boldsymbol{A}$), and vectors by lower-case boldface letters ($\boldsymbol{a}$). Elements of a matrix will be denoted by the same letter as the matrix itself, so for example the element on the ith row and jth column of matrix $\boldsymbol{A}$ will be denoted by $\boldsymbol{A}_{ij}$.

The ith row of matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}_i$ and the jth column by $\boldsymbol{A}^j$. Matrix $\boldsymbol{A}$ with the ith column removed is denoted by $\boldsymbol{A}^{-i}$, and $\boldsymbol{A}_{-i}$ is the respective notation for $\boldsymbol{A}$ with a removed row. Many matrices and vectors in this work will come from the set of nonnegative real numbers, and we will use a handy notation $\mathbb{R}_+ = [0, \infty)$. Often we will need to approximate a given matrix $\boldsymbol{A} \in \mathbb{R}_+^{n \times m}$ by another matrix $\boldsymbol{A'} \in \mathbb{R}_+^{n \times m}$. We say that $\boldsymbol{A'}$ *overcovers* $\boldsymbol{A}$ at some point (i, j) if $\boldsymbol{A'}_{ij} > \boldsymbol{A}_{ij}$.

We use the shorthand $[n]$ to denote the set $\{1, 2, \ldots, n\}$.

Matrix norms and matrix factorization. In the course of this thesis we will often need to measure the quality of various matrix approximations using some metric. The most common choices are the *Frobenius* and L_1 matrix norms.

Definition 1. The *Frobenius* norm of a matrix $\boldsymbol{A} \in \mathbb{R}_+^{n \times m}$ (also known as the L_2 norm) is defined as follows:

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} \boldsymbol{A}_{ij}^2}\ . \tag{2.1}$$

Definition 2. For a matrix $\boldsymbol{A} \in \mathbb{R}_+^{n \times m}$, the expression

$$\|\boldsymbol{A}\|_1 = \sum_{i=1}^{n} \sum_{j=1}^{m} |\boldsymbol{A}_{ij}| \tag{2.2}$$

defines its L_1 matrix norm.

A standard matrix factorization problem is defined as follows:

Problem 1 (Approximate rank-k matrix factorization)**.** Given a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times m}$ and an integer $k > 0$, find factor matrices $\boldsymbol{B} \in \mathbb{R}^{n \times k}$ and $\boldsymbol{C} \in \mathbb{R}^{k \times m}$ minimizing

$$E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) = \|\boldsymbol{A} - \boldsymbol{B}\boldsymbol{C}\| \ . \tag{2.3}$$

Here $\|\cdot\|$ is an arbitrary matrix norm.

Throughout this thesis we will draw multiple comparisons to the so called *nonnegative matrix factorization*.

Problem 2 (Approximate nonnegative rank-k matrix factorization)**.** Given a matrix $\boldsymbol{A} \in \mathbb{R}_+^{n \times m}$ and an integer $k > 0$, find factor matrices $\boldsymbol{B} \in \mathbb{R}_+^{n \times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k \times m}$ minimizing

$$E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) = \|\boldsymbol{A} - \boldsymbol{B}\boldsymbol{C}\| \ . \tag{2.4}$$

Here again $\|\cdot\|$ can be any matrix norm, but the Frobenius norm $\|\cdot\|_F$ is by far the most common.

Dioids. In this thesis we consider matrix factorization over idempotent semirings, also known as *dioids*. Formally, we use the following definition:

Definition 3. A semiring $(\mathbb{S}, +, \times)$ with addition $+$ and multiplication $\times$ is a *dioid* if the addition is *idempotent*, that is, for any $a \in \mathbb{S}$ we have

$$a + a = a \ . \tag{2.5}$$

Since every dioid is also a semiring, its multiplication operation distributes over the addition, and hence the definitions of matrix product and factorization follow naturally.

Definition 4. Given a dioid $(\mathbb{S}, +, \times)$ and matrices $\boldsymbol{B} \in \mathbb{S}^{n \times k}$ and $\boldsymbol{C} \in \mathbb{S}^{k \times m}$, their *dioid matrix product* is defined as

$$(\boldsymbol{B} \times \boldsymbol{C})_{ij} = \sum_{s=1}^{k} \boldsymbol{B}_{is} \times \boldsymbol{C}_{sj} \ , \tag{2.6}$$

where the summation uses the additive operation of $\mathbb{S}$.

Problem 3 (Approximate rank-k matrix factorization over dioids)**.** Given a dioid $(\mathbb{S}, +, \times)$, a matrix $\boldsymbol{A} \in \mathbb{S}^{n \times m}$, and an integer $k > 0$, find factor matrices $\boldsymbol{B} \in \mathbb{S}^{n \times k}$ and $\boldsymbol{C} \in \mathbb{S}^{k \times m}$ minimizing

$$E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) = \|\boldsymbol{A} - \boldsymbol{B} \times \boldsymbol{C}\| \ . \tag{2.7}$$

As before, we deliberately did not specify any particular norm. Depending on the circumstances, different matrix norms can be used, of which we will primarily consider the two most natural choices – the Frobenius and L_1 norms.

Examples of dioids. We will consider various types of dioids and the corresponding matrix factorization problems. The most central for this thesis is the so called *subtropical* (or *max-times*) *algebra*, whose main property is that the addition is replaced with the maximum operation.

Definition 5. The *subtropical* (or *max-times*) algebra is the set $\mathbb{R}_+$ of nonnegative real numbers together with operations $a \boxplus b = \max\{a, b\}$ (addition) and $a \boxdot b = ab$ (multiplication), defined for any $a, b \in \mathbb{R}_+$. The identity element for addition is 0 and for multiplication it is 1.

In the future we will use the notation $a \boxplus b$ and $\max\{a, b\}$ and the names *subtropical* and *max-times* interchangeably. It is straightforward to see that the max-times algebra is a *dioid*, that is, the addition satisfies $(a \boxplus a = a)$. It is important to note that the subtropical algebra is anti-negative, that is, there is no subtraction operation.

The matrix product over the subtropical algebra is defined in the natural way:

Definition 6. The *subtropical matrix product* of two matrices $\boldsymbol{B} \in \mathbb{R}_+^{n \times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k \times m}$ is defined as

$$(\boldsymbol{B} \boxtimes \boldsymbol{C})_{ij} = \max_{s=1}^{k} \boldsymbol{B}_{is} \boldsymbol{C}_{sj} \ . \tag{2.8}$$

Now that we have sufficient notation, we can formally introduce the subtropical matrix factorization problem.

Problem 4 (Approximate subtropical rank-k matrix factorization)**.** Given a matrix $\boldsymbol{A} \in \mathbb{R}_+^{n \times m}$ and an integer $k > 0$, find factor matrices $\boldsymbol{B} \in \mathbb{R}_+^{n \times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k \times m}$ minimizing

$$E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) = \|\boldsymbol{A} - \boldsymbol{B} \boxtimes \boldsymbol{C}\| \ . \tag{2.9}$$

The *matrix rank* over the subtropical algebra can be defined in many ways, depending on which definition of the normal matrix rank is taken as the starting point. We will discuss different subtropical ranks in detail in Section 2.2.4. Here we give the main definition of the rank that will be used throughout the thesis, the so-called *Schein* (or *Barvinok*) *rank* of a matrix.

Definition 7. The *subtropical (Schein or Barvinok) rank* of a matrix $\boldsymbol{A} \in \mathbb{R}_+^{n \times m}$ is the least integer k such that $\boldsymbol{A}$ can be expressed as an element-wise maximum of k rank-1 matrices, $\boldsymbol{A} = \boldsymbol{F}_1 \boxplus \boldsymbol{F}_2 \boxplus \cdots \boxplus \boldsymbol{F}_k$. Matrix $\boldsymbol{F} \in \mathbb{R}_+^{n \times m}$ has subtropical (Schein/Barvinok) rank of 1 if there exist column vectors $\boldsymbol{x} \in \mathbb{R}_+^n$ and $\boldsymbol{y} \in \mathbb{R}_+^m$ such that $\boldsymbol{F} = \boldsymbol{x}\boldsymbol{y}^T$. Matrices with subtropical Schein (or Barvinok) rank of 1 are called *blocks*.

When it is clear from the context, we will use the term *rank* (or *subtropical rank*) without other qualifiers to denote the subtropical Schein/Barvinok rank.

Another example of a dioid is the *tropical* (or *max-plus*) *algebra* [see e.g. Akian et al., 2007], which is very closely related to the subtropical algebra.

Definition 8. The *tropical* (or *max-plus*) algebra is defined over the set of extended real numbers $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty\}$ with operations $a \oplus b = \max\{a, b\}$ (addition) and $a \odot b = a + b$ (multiplication). The identity elements for addition and multiplication are $-\infty$ and 0, respectively.

Matrix multiplication and factorization over the tropical algebra are defined analogously to their subtropical counterparts.

Definition 9. For two matrices $\boldsymbol{B} \in \bar{\mathbb{R}}^{n\times k}$ and $\boldsymbol{C} \in \bar{\mathbb{R}}^{k\times m}$, their *tropical matrix product* is defined as

$$(\boldsymbol{B} \diamond \boldsymbol{C})_{ij} = \max_{s=1}^{k}\{\boldsymbol{B}_{is} + \boldsymbol{C}_{sj}\} \ . \tag{2.10}$$

Problem 5 (Approximate tropical rank-k matrix factorization)**.** Given a matrix $\boldsymbol{A} \in \bar{\mathbb{R}}^{n\times m}$ and an integer $k > 0$, find factor matrices $\boldsymbol{B} \in \bar{\mathbb{R}}^{n\times k}$ and $\boldsymbol{C} \in \bar{\mathbb{R}}^{k\times m}$ minimizing

$$E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) = \|\boldsymbol{A} - \boldsymbol{B} \diamond \boldsymbol{C}\| \ . \tag{2.11}$$

In fact, the tropical and subtropical algebras are isomorphic [Blondel et al., 2000], which can be seen by taking the logarithm of the subtropical algebra or the exponent of the tropical algebra (with the conventions that $\log 0 = -\infty$ and $\exp(-\infty) = 0$). Thus, most of the results we prove for the subtropical algebra can be extended to their tropical analogues, although caution should be used when dealing with approximate matrix factorizations. The latter is because, as we will see in Theorem 6, the *reconstruction error* of an approximate matrix factorization under the two different algebras does not transfer directly.

A very closely related *min-plus* algebra is defined analogously to the tropical algebra, except that it uses $\min$ as the addition operation, and the domain of its definition is $\mathbb{R} \bigcup \infty$. It is isomorphic to the tropical algebra via the map $x \mapsto -x$, and since this map preserves any p-norm, it is also isometric [see e.g. Hook, 2017]. The min-plus algebra was used, for example, for network structure approximation by low-rank matrix factorization Hook [2017]. Namely, it was shown that it is possible to simplify some bipartite networks by representing them as a min-plus product of two lower rank matrices.

The last dioid type considered in this thesis is the Boolean algebra, which can be seen as a restriction of the subtropical algebra to the set $\{0, 1\}$. As with the subtropical and tropical algebras, we can define matrix product and matrix factorization in the standard way.

Definition 10. For two matrices $\boldsymbol{B} \in \{0, 1\}^{n\times k}$ and $\boldsymbol{C} \in \{0, 1\}^{k\times m}$, their *Boolean matrix product* is defined as

$$(\boldsymbol{B} \circ \boldsymbol{C})_{ij} = \bigvee_{l=1}^{k} \boldsymbol{B}_{il}\boldsymbol{C}_{lj} \ . \tag{2.12}$$

Problem 6 (Approximate Boolean rank-k matrix factorization)**.** Given a matrix $\boldsymbol{A} \in \{0, 1\}^{n\times m}$ and an integer $k > 0$, find factor matrices $\boldsymbol{B} \in \{0, 1\}^{n\times k}$ and $\boldsymbol{C} \in \{0, 1\}^{k\times m}$ minimizing

$$E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) = \|\boldsymbol{A} - \boldsymbol{B} \circ \boldsymbol{C}\| \ . \tag{2.13}$$

Special matrices. In the final part of this section we introduce special types of matrices that will be used throughout the thesis.

Definition 11. A *mask* of a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times m}$ is an n-by-m binary matrix $\boldsymbol{M}$ such that $\boldsymbol{M}_{ij} = 0$ if and only if $\boldsymbol{A}_{ij} = 0$, and otherwise $\boldsymbol{M}_{ij} = 1$. We denote the mask of $\boldsymbol{A}$ by $m(\boldsymbol{A})$.

Definition 12. Let $\boldsymbol{A}$ and $\boldsymbol{X}$ be matrices of the same size, and let Γ be a subset of their indices. Then if for all indices $(i, j) \in \Gamma$, $\boldsymbol{X}_{ij} \geq \boldsymbol{A}_{ij}$, we say that $\boldsymbol{X}$ *dominates* $\boldsymbol{A}$ *within* Γ. If Γ spans the entire size of $\boldsymbol{A}$ and $\boldsymbol{X}$, we simply say that $\boldsymbol{X}$ *dominates* $\boldsymbol{A}$. Correspondingly, $\boldsymbol{A}$ is said to be *dominated by* $\boldsymbol{X}$.

In order to introduce the next special matrix type, we will need the following auxiliary function definition.

Definition 13. A function $s \colon [n] \to [m]$ is a *step function* if $s(i) \geq s(j)$ for all $i \in [n]$ and $j \in [m]$ such that $j < i$.

We can now define a *nested matrix* as follows:

Definition 14. A binary matrix $\boldsymbol{A} \in \{0, 1\}^{n \times m}$ is *directly nested* if there exists a step function s such that on each row $i \in [n]$ of $\boldsymbol{A}$, $\boldsymbol{A}_{ij} = 1$ if $j \leq s(i)$ and $\boldsymbol{A}_{ij} = 0$ if $j > s(i)$. $\boldsymbol{A}$ is *nested* if there exists a way to permute its rows and columns such that the permuted matrix $\boldsymbol{A}'$ is directly nested.

2.2 Theory

While we will discuss various types of dioids, the subtropical algebra will play a major role in this thesis. In this section we cover some important theoretical aspects of the subtropical algebra, as well as delve deeper into how it is related to other dioids. We will start by studying the computational complexity of Problem 4, showing that it is NP-hard even to approximate. After that, we will show that dominated subtropical factorizations of sparse matrices are sparse. Then we compare the subtropical factorizations to decompositions over other algebras, analyzing how the error of an approximate decomposition behaves when moving from tropical to subtropical algebra. Finally, we briefly summarize different ways to define the subtropical rank, and how these different ranks can be used to bound each other, and the Boolean rank of a binary matrix, as well.

2.2.1 Computational complexity

The computational complexity of different matrix factorization problems varies. For example, SVD can be computed in polynomial time [Golub and Van Loan, 2012], while NMF is NP-hard [Vavasis, 2009]. Unfortunately, the subtropical factorization is also NP-hard.

Theorem 1. *Computing the max-times matrix rank is an* NP*-hard problem, even for binary matrices.*

The theorem is a direct consequence of the following theorem by Kim and Roush [2005]:

Theorem 2 (Kim and Roush, 2005)**.** *Computing the max-plus (tropical) matrix rank is* NP*-hard, even for matrices that take values only from* $\{-\infty, 0\}$.

While computing the rank deals with exact decompositions, its hardness automatically makes any approximation algorithm with provable multiplicative guarantees unlikely to exist, as the following corollary shows.

Corollary 3. *It is* NP*-hard to approximate Problem 4 to within any polynomially computable factor.*

Proof. Any algorithm that can approximate Problem 4 to within a factor α must find a decomposition of error $\alpha \cdot 0 = 0$ if the input matrix has an exact max-times rank-k decomposition. As this implies solving the max-times rank, per Theorem 1 it is only possible if P=NP. □

2.2.2 Sparsity of the factors

When the input data matrix is sparse, it is often desirable to obtain sparse factor matrices. In fact, sparsity of factors is frequently mentioned as one of the benefits of using NMF [see, e.g. Hoyer, 2004]. While in general factors obtained by NMF might not be sparse, it was shown by Gillis and Glineur [2010] that in case of *dominated* decompositions, the sparsity of the factors cannot be less than the sparsity of the original matrix.

The proof of Gillis and Glineur [2010] relies on the anti-negativity, and hence can be easily adapted to the max-times setting. Let the *sparsity* of an n-by-m matrix $\boldsymbol{A}$ be defined as

$$s(\boldsymbol{A}) = \frac{nm - \eta(\boldsymbol{A})}{nm} , \tag{2.14}$$

where $\eta(\boldsymbol{A})$ is the number of nonzero elements in $\boldsymbol{A}$. Now we have the following

Theorem 4. *Let matrices* $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$ *and* $\boldsymbol{C} \in \mathbb{R}_+^{k\times m}$ *be such that their max-times product is dominated by an* n*-by-*m *matrix* $\boldsymbol{A}$*. Then the following bound holds:*

$$s(\boldsymbol{B}) + s(\boldsymbol{C}) \geq s(\boldsymbol{A}) . \tag{2.15}$$

Proof. The proof follows that of Gillis and Glineur [2010]. We first prove (2.15) for $k = 1$. Let $\boldsymbol{b} \in \mathbb{R}_+^n$ and $\boldsymbol{c} \in \mathbb{R}_+^m$ be such that $\boldsymbol{b}_i \boldsymbol{c}_j^T \leq \boldsymbol{A}_{ij}$ for all $1 \leq i \leq n$, $1 \leq j \leq m$. Since $(\boldsymbol{b}\boldsymbol{c}^T)_{ij} > 0$ if and only if $\boldsymbol{b}_i > 0$ and $\boldsymbol{c}_j > 0$, we have

$$\eta(\boldsymbol{b}\boldsymbol{c}^T) = \eta(\boldsymbol{b})\,\eta(\boldsymbol{c}) . \tag{2.16}$$

By (2.14) we have $\eta(\boldsymbol{b}\boldsymbol{c}^T) = nm(1 - s(\boldsymbol{b}\boldsymbol{c}^T))$, $\eta(\boldsymbol{b}) = n(1 - s(\boldsymbol{b}))$ and $\eta(\boldsymbol{c}) = m(1 - s(\boldsymbol{c}))$. Plugging these expressions into (2.16) we obtain $(1 - s(\boldsymbol{b}\boldsymbol{c}^T)) = (1 - s(\boldsymbol{b}))(1 - s(\boldsymbol{c}))$. Hence, the sparsity in a rank-1 dominated approximation of $\boldsymbol{A}$ is

$$s(\boldsymbol{b}) + s(\boldsymbol{c}) \geq s(\boldsymbol{b}\boldsymbol{c}^T) . \tag{2.17}$$

From (2.17) and the fact that the number of nonzero elements in $\boldsymbol{b}\boldsymbol{c}^T$ is no greater than in $\boldsymbol{A}$, it follows that

$$s(\boldsymbol{b}) + s(\boldsymbol{c}) \geq s(\boldsymbol{A}) . \tag{2.18}$$

Now let $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k\times m}$ be such that $\boldsymbol{B} \boxtimes \boldsymbol{C}$ is dominated by $\boldsymbol{A}$. Then $\boldsymbol{B}_{il}\boldsymbol{C}_{lj} \le \boldsymbol{A}_{ij}$ for all $i \in [n]$, $j \in [m]$, and $l \in [k]$, which means that for each $l \in [k]$, $\boldsymbol{B}^l\boldsymbol{C}_l$ is dominated by $\boldsymbol{A}$. To complete the proof observe that $s(\boldsymbol{B}) = k^{-1}\sum_{l=1}^k \boldsymbol{B}^l$ and $s(\boldsymbol{C}) = k^{-1}\sum_{l=1}^k \boldsymbol{C}_l$ and that for each l estimate (2.18) holds. □

2.2.3 Relationship between the subtropical algebra and other algebra types

Let us now study how the subtropical algebra relates to other algebras, namely the standard, the Boolean, and the tropical algebras. For the first two, we compare the ranks, and for the last, the reconstruction error.

Let us start by considering the Boolean rank of a binary matrix. The *Boolean (Schein or Barvinok) rank* is the following problem:

Problem 7 (Boolean rank)**.** Given a matrix $\boldsymbol{A} \in \{0,1\}^{n\times m}$, what is the smallest positive integer k such that there exist matrices $\boldsymbol{B} \in \{0,1\}^{n\times k}$ and $\boldsymbol{C} \in \{0,1\}^{k\times m}$ that satisfy $\boldsymbol{A} = \boldsymbol{B} \circ \boldsymbol{C}$.

Lemma 5. *If $\boldsymbol{A}$ is a binary matrix, then its Boolean and subtropical ranks are the same.*

Proof. We will prove the claim by first showing that the Boolean rank of a binary matrix is no less than the subtropical rank, and then showing that it is no larger, either. For the first direction, let the Boolean rank of $\boldsymbol{A}$ be k, and let $\boldsymbol{B}$ and $\boldsymbol{C}$ be binary matrices such that $\boldsymbol{B}$ has k columns and $\boldsymbol{A} = \boldsymbol{B} \circ \boldsymbol{C}$. It is easy to see that $\boldsymbol{B} \circ \boldsymbol{C} = \boldsymbol{B} \boxtimes \boldsymbol{C}$, and hence, the subtropical rank of $\boldsymbol{A}$ is no more than k.

For the second direction, we will actually show a slightly stronger claim: Let $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$ and let $m(\boldsymbol{A})$ be its mask. Then the Boolean rank of $m(\boldsymbol{A})$ is never more than the subtropical rank of $\boldsymbol{A}$. As $m(\boldsymbol{A}) = \boldsymbol{A}$ for a binary $\boldsymbol{A}$, the claim follows. To prove the claim, let $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$ have subtropical rank of k and let $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k\times m}$ be such that $\boldsymbol{A} = \boldsymbol{B} \boxtimes \boldsymbol{C}$. Let (i,j) be such that $\boldsymbol{A}_{ij} = 0$. By definition, $\max_{l=1}^k \boldsymbol{B}_{il}\boldsymbol{C}_{lj} = 0$, and hence

$$\max_{l=1}^{k} m(\boldsymbol{B})_{il} m(\boldsymbol{C})_{lj} = \bigvee_{l=1}^{k} m(\boldsymbol{B})_{il} m(\boldsymbol{C})_{lj} = 0 \;. \tag{2.19}$$

On the other hand, if (i,j) is such that $\boldsymbol{A}_{ij} > 0$, then there exists l such that $\boldsymbol{B}_{il}, \boldsymbol{C}_{lj} > 0$ and consequently,

$$\max_{l=1}^{k} m(\boldsymbol{B})_{il} m(\boldsymbol{C})_{lj} = \bigvee_{l=1}^{k} m(\boldsymbol{B})_{il} m(\boldsymbol{C})_{lj} = 1 \;. \tag{2.20}$$

Combining (2.19) and (2.20) gives us

$$m(\boldsymbol{A}) = m(\boldsymbol{B}) \circ m(\boldsymbol{C}) \;, \tag{2.21}$$

showing that the Boolean rank of $m(\boldsymbol{A})$ is at most k. □

Notice that Lemma 5 also furnishes us with another proof of Theorem 1, as the computation of the Boolean rank is an NP-hard problem [see, e.g. Miettinen, 2009]. Notice also that while the Boolean rank of the mask is never more than the subtropical rank of the original matrix, it can be much less. This is easy to see by considering a matrix with no zeros: it can have arbitrarily large subtropical rank, but its mask has Boolean rank 1.

Unfortunately, the Boolean rank does not help us with effectively estimating the subtropical rank, as its computation is an NP-hard problem. The standard rank is (relatively) easy to compute, but the standard rank and the max-times rank are incommensurable, that is, there are matrices that have smaller max-times rank than standard rank and others that have higher max-times rank than standard rank. Let us consider an example of the first kind,

$$\begin{pmatrix} 1 & 2 & 0 \\ 2 & 4 & 1 \\ 0 & 4 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 2 & 1 \\ 0 & 2 \end{pmatrix} \boxtimes \begin{pmatrix} 1 & 2 & 0 \\ 0 & 2 & 1 \end{pmatrix} .$$

As this decomposition shows, this matrix has a max-times rank of 2, while its normal rank is easily verified to be 3. Another example is the complement of the n-by-n identity matrix $\bar{\boldsymbol{I}}_n$, that is, the matrix that has 0s at the diagonal and 1s everywhere else, whose max-times rank is $O(\log n)$, while its standard rank is n (this follows from similar results regarding the Boolean rank, see, e.g. Miettinen, 2009).

As we have discussed earlier, tropical and subtropical algebras are isomorphic, and consequently for any matrix $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$ its max-times rank agrees with the max-plus rank of the matrix $\log(\boldsymbol{A})$. Yet, the errors obtained in approximate decompositions do not have to (and usually do not) agree. In what follows we characterize the relationship between max-plus and max-times errors.

Theorem 6. *Let $\boldsymbol{A} \in \overline{\mathbb{R}}^{n\times m}$, $\boldsymbol{B} \in \overline{\mathbb{R}}^{n\times k}$ and $\boldsymbol{C} \in \overline{\mathbb{R}}^{k\times m}$. Let $M = \exp\{N\}$, where*

$$N = \max_{\substack{i\in[n] \\ j\in[m]}} \left\{ \max\{\boldsymbol{A}_{ij}, \max_{1\le d\le k}\{\boldsymbol{B}_{id} + \boldsymbol{C}_{dj}\}\} \right\} .$$

If an error can be bounded in max-plus algebra as

$$\|\boldsymbol{A} - \boldsymbol{B} \diamond \boldsymbol{C}\|_F^2 \le \lambda , \tag{2.22}$$

then the following bound holds with respect to the max-times algebra:

$$\|\exp\{\boldsymbol{A}\} - \exp\{\boldsymbol{B}\} \boxtimes \exp\{\boldsymbol{C}\}\|_F^2 \le M^2\lambda . \tag{2.23}$$

Proof. Let $\alpha_{ij} = \max_{d=1}^{k}\{\boldsymbol{B}_{id} + \boldsymbol{C}_{dj}\}$. From (2.22) it follows that there exists a set of numbers $\{\lambda_{ij} \ge 0 : i \in [n], j \in [m]\}$ such that for any i, j we have $(A_{ij} - \alpha_{ij})^2 \le \lambda_{ij}$ and $\sum_{ij} \lambda_{ij} = \lambda$. By the mean-value theorem, for every i and j we obtain

$$|\exp\{\boldsymbol{A}_{ij}\} - \exp\{\alpha_{ij}\}| = |\boldsymbol{A}_{ij} - \alpha_{ij}| \exp\{\alpha^*_{ij}\} \le \sqrt{\lambda_{ij}} \exp\{\alpha^*_{ij}\} ,$$

for some $\min\{\boldsymbol{A}_{ij}, \alpha_{ij}\} \le \alpha^*_{ij} \le \max\{\boldsymbol{A}_{ij}, \alpha_{ij}\}$. Hence,

$$(\exp\{\boldsymbol{A}_{ij}\} - \exp\{\alpha_{ij}\})^2 \le \lambda_{ij}(\exp\{\max\{\boldsymbol{A}_{ij}, \alpha_{ij}\}\})^2 .$$

The estimate for the max-times error now follows from the monotonicity of the exponent:

$$\begin{aligned}\|\exp\{\boldsymbol{A}\} - \exp\{\boldsymbol{B}\} \boxtimes \exp\{\boldsymbol{C}\}\|_F^2 &\leq \sum_{ij} \left(\exp\{\alpha_{ij}^*\}\right)^2 \lambda_{ij} \\ &\leq \sum_{ij} \left(\exp\{\max\{\boldsymbol{A}_{ij}, \alpha_{ij}\}\}\right)^2 \lambda_{ij} \leq M^2 \lambda ,\end{aligned}$$

proving the claim. □

2.2.4 Different tropical and subtropical matrix ranks

The definition of the subtropical rank we use in this work is the so-called Schein (or Barvinok) rank (see Definition 7). Like in the standard linear algebra, this is not the only possible way to define the (subtropical) rank. Here we will review a few other forms of subtropical rank that can allow us to bound the Schein/Barvinok rank of a matrix. Unless otherwise mentioned, the definitions are by Guillon et al. [2015]; as always, all results without citations are ours. Following Guillon et al., we will present the definitions in this section over the tropical algebra. Recall that due to isomorphism, these definitions transfer directly to the subtropical case.

We begin with the tropical equivalent of the subtropical Schein/Barvinok rank:

Definition 15. The *tropical Schein/Barvinok rank* of a matrix $\boldsymbol{A} \in \overline{\mathbb{R}}^{n\times m}$, denoted $\mathrm{rank}_{\mathrm{S/B}}(\boldsymbol{A})$, is defined to be the least integer k such that there exist matrices $\boldsymbol{B} \in \overline{\mathbb{R}}^{n\times k}$ and $\boldsymbol{C} \in \overline{\mathbb{R}}^{k\times m}$ for which $\boldsymbol{A} = \boldsymbol{B} \diamond \boldsymbol{C}$.

Analogously to the standard case, we can also define the rank as the number of linearly independent rows or columns. The following definition of linear independence of a family of vectors in a tropical space is due to Gondran and Minoux [1984b].

Definition 16. A set of vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ from $\overline{\mathbb{R}}^n$ is called *linearly dependent* if there exist disjoint sets $I, J \subset \{1, \ldots, k\}$ and scalars $\{\lambda_i\}_{i\in I\cup J}$, such that $\lambda_i \neq -\infty$ for all i and

$$\max_{i\in I}\{\lambda_i + \boldsymbol{x}_i\} = \max_{j\in J}\{\lambda_j + \boldsymbol{x}_j\} . \tag{2.24}$$

Otherwise the vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$ are called *linearly independent*.

This gives rise to the so-called *Gondran–Minoux ranks*:

Definition 17. The *Gondran–Minoux* row (column) rank of a matrix $\boldsymbol{A} \in \overline{\mathbb{R}}^{n\times m}$ is defined as the maximal k such that $\boldsymbol{A}$ has k independent rows (columns). They are denoted by $\mathrm{rank}_{\mathrm{G\text{–}M;rw}}(\boldsymbol{A})$ and $\mathrm{rank}_{\mathrm{G\text{–}M;cl}}(\boldsymbol{A})$ respectively.

Another way to characterize the rank of the matrix is to consider the space its rows or columns can span.

Definition 18. A set $X \subset \overline{\mathbb{R}}^n$ is called *tropically convex* if for any vectors $\boldsymbol{x}, \boldsymbol{y} \in X$ and scalars $\lambda, \mu \in \overline{\mathbb{R}}$, we have $\max\{\lambda + \boldsymbol{x}, \mu + \boldsymbol{y}\} \in X$.

Definition 19. The *convex hull* $H(\boldsymbol{x}_1, \dots \boldsymbol{x}_k)$ of a finite set of vectors $\{\boldsymbol{x}_i\}_{i=1}^k \in \overline{\mathbb{R}}^n$ is defined as follows

$$H(\boldsymbol{x}_1, \dots \boldsymbol{x}_k) = \left\{ \max_{i=1}^{k} \{\lambda_i + \boldsymbol{x}_i\} \ : \ \lambda_i \in \overline{\mathbb{R}} \right\} .$$

Definition 20. The *weak dimension* of a finitely generated tropically convex subset of $\overline{\mathbb{R}}^n$ is the cardinality of its minimal generating set.

We can define the rank of the matrix by looking at the weak dimension of the (tropically) convex hull its rows or columns span.

Definition 21. The *row rank* and the *column rank* of a matrix $\boldsymbol{A} \in \overline{\mathbb{R}}^{n \times m}$ are defined as the weak dimensions of the convex hulls of the rows and the columns of $\boldsymbol{A}$ respectively. They are denoted by $\operatorname{rank}_{\mathrm{rw}}(\boldsymbol{A})$ and $\operatorname{rank}_{\mathrm{cl}}(\boldsymbol{A})$.

None of the above definitions coincide [see Akian et al., 2009], unlike in the standard algebra. We can, however, have a partial ordering of the ranks:

Theorem 7 (Guillon et al., 2015, Akian et al., 2009). *Let $\boldsymbol{A} \in \overline{\mathbb{R}}^{n \times m}$. Then the the following relations are true for the above definitions of the rank of $\boldsymbol{A}$:*

$$\left.\begin{array}{l} \operatorname{rank}_{G\text{–}M;rw}(\boldsymbol{A}) \\ \operatorname{rank}_{G\text{–}M;cl}(\boldsymbol{A}) \end{array}\right\} \leq \operatorname{rank}_{S/B}(\boldsymbol{A}) \leq \left\{\begin{array}{l} \operatorname{rank}_{rw}(\boldsymbol{A}) \\ \operatorname{rank}_{cl}(\boldsymbol{A}) \end{array}\right. . \tag{2.25}$$

The row and column ranks of an n-by-n tropical matrix can be computed in $O(n^3)$ time [Butkovič, 2010], allowing us to bound the Schein/Barvinok rank from above. Unfortunately, no efficient algorithm for the Gondran–Minoux rank is known. On the other hand, Guillon et al. [2015] presented what they called the *ultimate tropical rank* that lower-bounds the Gondran–Minoux rank and can be computed in time $O(n^3)$. We can also check if a matrix has full Schein/Barvinok rank in time $O(n^3)$ [see Butkovič and Hevery, 1985], even if computing any other value is NP-hard.

These bounds, together with Lemma 5, yield the following corollary for the *Boolean rank* of a square matrix:

Corollary 8. *Given an n-by-n binary matrix $\boldsymbol{A}$, its Boolean rank can be bound from below, using the ultimate rank, and from above, using the tropical column and row ranks, in time $O(n^3)$.*

2.3 Related Work

Here we present earlier research that is related to matrix factorization over dioids. We start by discussing classic methods, such as SVD and NMF, that have long been used for various data analysis tasks, and then present an overview of tropical methods. Some of the more technical related work (e.g. ranks) is described in Section 2.2. We postpone the discussion of the related work for Boolean matrix factorization until Section 6.4. Similarly, previous research in the area of community mining in graphs is discussed in Chapter 5.

2.3.1 Matrix factorization in data mining

Matrix factorization methods play a crucial role in data analysis as they help to find low-dimensional representations of the data and uncover the underlying latent structure. A classic example of a real-valued matrix factorization is the singular value decomposition (SVD) [see e.g. Golub and Van Loan, 2012], which is very well known and finds extensive applications in various disciplines, such as for example signal processing and natural language processing. The SVD of a real n-by-m matrix $\boldsymbol{A}$ is a factorization of the form $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$, where $\boldsymbol{U} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{V} \in \mathbb{R}^{m\times m}$ are orthogonal matrices, and $\boldsymbol{\Sigma} \in \mathbb{R}^{n\times m}$ is a rectangular diagonal matrix with nonnegative entries. An important property of SVD is that it provides the best low-rank approximation of a given matrix with respect to the Frobenius norm [Golub and Van Loan, 2012], giving rise to the so called truncated SVD. This property is frequently used to separate important parts of data from the noise. For example, it was used by Jha and Yadava [2011] to remove the noise from sensor data in electronic nose systems. Another prominent usage of the truncated SVD is in dimensionality reduction [see for example Sarwar et al., 2000, Deerwester et al., 1990].

Despite SVD being so ubiquitous, there are some restrictions to its usage in data mining due to the possible presence of negative elements in the factors. In many applications negative values are hard to interpret, and thus other methods have to be used. Nonnegative matrix factorization (NMF) is a way to tackle this problem. For a given nonnegative real matrix $\boldsymbol{A}$, the NMF problem is to find a decomposition of $\boldsymbol{A}$ into two matrices $\boldsymbol{A} \approx \boldsymbol{B}\boldsymbol{C}$ such that $\boldsymbol{B}$ and $\boldsymbol{C}$ are also nonnegative. This area drew considerable attention after a publication by Lee and Seung [1999], where they provided an efficient algorithm for solving the NMF problem. It is worth mentioning that even though the paper by Lee and Seung is perhaps the most famous in NMF literature, it was not the first one to consider this problem. Earlier works include Paatero and Tapper [1994] [see also Paatero, 1997], Paatero [1999], and Cohen and Rothblum [1993].

The reason for NMF's popularity is that it excels at extracting latent structure from the data, and is often able to predict unseen data. One of its more prolific applications is in face recognition, where it was shown to be a highly effective method for dealing with various occlusions, such as subjects wearing glasses or other obscuring items [Guillamet and Vitrià, 2002]. Another computer vision related application is the spectral unmixing problem, where the objective is, given spectral reflectance data of an unknown object and spectral signatures of various materials, to identify the composition of this object. Solving this problem effectively is crucial for safety in space as it helps to identify potentially hazardous objects, such as rockets, satellites, and asteroids [Keshava, 2003]. Another area of application that has been driving the development of NMF related methods is text mining [Pauca et al., 2004]. It was used, for example, for online extraction of topics from streaming social media [Saha and Sindhwani, 2012]. Other applications of NMF include document clustering [Xu et al., 2003] and pattern discovery [Brunet et al., 2004]. Overviews of NMF algorithms and their applications can be found in e.g. [Berry et al., 2007, Gillis, 2014].

There exist various flavours of NMF that impose different constraints on the factors; for example Hoyer [2004] used sparsity constraints. Though both NMF and SVD perform approximations of a fixed rank, there are other ways to enforce a compact representation of the data. For example, in maximum-margin matrix factorization constraints are imposed on the norms of factors. This approach was exploited by Srebro et al. [2004], who showed it to be a good method for predicting unobserved values in a matrix. The authors also indicate that posing constraints on the factor norms, rather than on the rank, yields a convex optimization problem, which is easier to solve.

2.3.2 Tropical algebra

Whereas the subtropical algebra has received relatively little attention, its close cousin, the tropical algebra, has become an important mathematical tool. Despite its theory being relatively young, it has been thoroughly studied in recent years. The reason for this is that it finds extensive applications in various areas of mathematics and other disciplines. It was especially useful for so called discrete event systems (DES), i.e., dynamic systems with asynchronously firing events [Cassandras and Lafortune, 2008]. An example of such a system is a production line, where in order to manufacture the final product, certain events have to occur (such as preparation and assembly of its various parts). In addition there might be constraints that in order to start a given task, certain events must have already occured – for example a car cannot be assembled before its components become available. This is known as scheduling and can be naturally expressed using the tropical algebra [see e.g. Baccelli et al., 1992, Cohen et al., 1999]. Other mathematical disciplines where the tropical algebra plays a crucial role are optimal control [Gaubert, 1997], asymptotic analysis [Dembo and Zeitouni, 2010, Maslov, 1992, Akian, 1999], and decidability [Simon, 1978, 1994]. There are also potential applications of the tropical algebra to non-linear image processing [see e.g. Angulo and Velasco-Forero, 2017]. Namely, a collection of images often has to be analysed by applying the same morphological operator to each image. If a collection has many images with similar content, it might be possible to project them onto a lower dimensional image space and simultaneously learn a dictionary of atom images. A potential benefit of using tropical algebras is that morphological operators become linear [Angulo and Velasco-Forero, 2017]. Another interesting application of the tropical algebras (or more precisely its min-plus cousin) is the all-pairs shortest path problem. Since it was established that this problem is equivalent to the min-plus closure of the adjacency matrix [see e.g. Munro, 1971], the min-plus algebra garnered considerable attention.

Although in this thesis we do not propose any algorithms for the pure tropical matrix factorization problem (it will only be used indirectly for the mixed linear-tropical and logistic-tropical decomposition), it is still closely related. Hence, it is worth mentioning that in the general case this problem is NP-complete [see e.g. Shitov, 2014]. De Schutter and De Moor [2002] demonstrated that if the max-plus algebra is extended in such a way that there is an additive inverse for each element, then it is possible to solve many of the standard matrix decomposition problems. Among other results, the authors obtained max-plus analogues of QR and SVD. They also claimed that the techniques they propose can be readily extended to other types of classic factorizations (e.g. Hessenberg and LU decompositions).

The problem of solving tropical linear systems of equations arises naturally in numerous applications, and is also closely related to matrix factorization. In order to illustrate this connection, assume that we are given a tropical matrix $\boldsymbol{A} \in \overline{\mathbb{R}}^{n\times m}$ and one of the factors, say $\boldsymbol{B} \in \overline{\mathbb{R}}^{n\times k}$. Then the other factor $\boldsymbol{C} \in \overline{\mathbb{R}}^{k\times m}$ can be found by solving the following set of problems

$$\boldsymbol{C}_j = \underset{\boldsymbol{c}\in\overline{\mathbb{R}}^k}{\arg\min} \|\boldsymbol{B} \diamond \boldsymbol{c} - \boldsymbol{A}_j\|_F,\ j = 1, \ldots, m\ . \tag{2.26}$$

Each problem in (2.26) requires “approximately” solving a system of tropical linear equations. The minus operation in (2.26) does not belong to the tropical semiring, so the approximation here should be understood in terms of minimizing the classical distance. The general form of tropical linear equations

$$\boldsymbol{A}\boldsymbol{x} \oplus \boldsymbol{b} = \boldsymbol{C}\boldsymbol{x} \oplus \boldsymbol{d} \tag{2.27}$$

is not always solvable [see e.g. Gaubert, 1997]; however various techniques exist for checking the existence of the solution for particular cases of (2.27).

For equations of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ the feasibility can be established, for example, through the so called *matrix residuation*. There is a general result that for an n-by-m matrix $\boldsymbol{A}$ over a complete idempotent semiring, the existence of the solution can be checked in $O(nm)$ time [see Gaubert, 1997]. Although the tropical algebra is not complete, there is an efficient way of finding if the solution exists [Cuninghame-Green, 1979, Zimmermann, 2011]. It was shown by Butkovič [2003] that this type of tropical equations is equivalent to the set cover problem, which is known to be NP-hard. This directly affects the max-times algebra through the above-mentioned isomorphism and makes the problem of precisely solving max-times linear systems of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ infeasible for high dimensions.

Homogeneous equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{B}\boldsymbol{x}$ can be solved using the *elimination* method, which is based on the fact that the set of solutions of a homogeneous system is a finitely generated semimodule [Butkovič and Hegedüs, 1984] [independently rediscovered by Gaubert, 1992]. If only a single solution is required, then according to Gaubert [1997], a method by Walkup and Borriello [1998] is usually the fastest in practice.

Now let $\boldsymbol{A}$ be a tropical square matrix of size $n \times n$. For complete idempotent semirings a solution to the equation $\boldsymbol{x} = \boldsymbol{A}\boldsymbol{x} \oplus \boldsymbol{b}$ is given by $\boldsymbol{x} = \boldsymbol{A}^*\boldsymbol{b}$ [see e.g. Salomaa and Soittola, 2012], where the operator $\boldsymbol{A}^*$ is defined as

$$\boldsymbol{A}^* = \oplus_{k=1}^{\infty} \boldsymbol{A}^k\ .$$

Since the tropical semiring is not complete (it is missing the ∞ element), $\boldsymbol{A}^*$ can not always be computed. Baccelli et al. [1992] provide a concise characterization for when the elements of $\boldsymbol{A}^*$ belong to the tropical semiring. Since $\boldsymbol{A}$ can be viewed as an adjacency matrix of some directed graph $\boldsymbol{G}$, and $\boldsymbol{A}^*_{ij}$ is the maximum weight of all paths between vertices i and j, it can be shown that $\boldsymbol{A}^*_{ij} < \infty$ if and only if there are no cycles of positive weight in $\boldsymbol{G}$. Moreover, when this condition is satisfied, $\boldsymbol{A}^*$ can be computed in a finite form, $\boldsymbol{A}^* = \boldsymbol{A}^0 \oplus \ldots \oplus \boldsymbol{A}^{n-1}$ [Baccelli et al., 1992]. Computing the operator $\boldsymbol{A}^*$ takes time $O(n^3)$ [see e.g. Gondran and Minoux, 1984a, Gaubert, 1997].

Another important direction of research is the eigenvalue problem $\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x}$. Tropical analogues of the Perron–Frobenius theorem [see e.g. Vorobyev, 1967, Maslov, 1992], and Collatz–Wielandt formula [Bapat et al., 1995, Gaubert, 1992] were developed. For a general overview of results in the $(\max, +)$ spectral theory, see for example Gaubert [1997].

Tropical algebra and tropical geometry were used by Gärtner and Jaggi [2008] to construct a tropical analogue of an SVM. Unlike in the classical case, tropical SVMs are localized, in the sense that the kernel at any given point is not influenced by all the support vectors. Their work also utilizes the fact that tropical hyperplanes are somewhat more complex than their counterparts in the classical geometry, which makes it possible to do multiple category classification with a single hyperplane.

CHAPTER 3

Algorithms for Subtropical Matrix Factorization

The problem of subtropical matrix factorization has some unique challenges that stem from the lack of linearity and smoothness of the max-times algebra. One of such issues is that dominated elements in a decomposition have no impact on the final result. Namely, if we consider the subtropical product of two matrices $\boldsymbol{B} \in \mathbb{R}_+^{n \times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k \times m}$, we can see that each entry $(\boldsymbol{B} \boxtimes \boldsymbol{C})_{ij} = \max_{1 \leq s \leq k} \boldsymbol{B}_{is}\boldsymbol{C}_{sj}$ is completely determined by a single element with index $\arg\max_{1 \leq s \leq k} \boldsymbol{B}_{is}\boldsymbol{C}_{sj}$. This means that all entries t with $\boldsymbol{B}_{it}\boldsymbol{C}_{tj} < \max_{1 \leq s \leq k} \boldsymbol{B}_{is}\boldsymbol{C}_{sj}$ do not contribute at all to the final decomposition. To see why this is a problem, observe that many optimization methods used in matrix factorization algorithms rely on local information to choose the direction of the next step (e.g. various forms of gradient descent). In the case of the subtropical algebra, however, the local information is practically absent, and hence we need to look elsewhere for effective optimization techniques.

A common approach to matrix decomposition problems is to update factor matrices alternatingly, which utilizes the fact that the problem $\min_{\boldsymbol{B},\boldsymbol{C}} \|\boldsymbol{A} - \boldsymbol{B}\boldsymbol{C}\|_F$ is biconvex. Unfortunately, the subtropical matrix factorization problem does not have the biconvexity property, which makes alternating updates less useful.

3.1 Algorithm (`Equator`)

Instead of trying to adapt the more conventional matrix factorization techniques to our problem, which lose a lot of their appeal due to the issues outlined above, we will try to exploit the covering nature of the subtropical algebra. Recall that in the subtropical world only the largest element makes a contribution at every given point, rendering all the smaller elements insignificant. It thus seems reasonable to "cover" the data by adding a single rank-1 matrix at a time. We are loosely following the idea by Kolda and O'Leary [2000], where the data is reconstructed by iteratively finding a rank-1 matrix to be added to the decomposition. On each step their algorithm finds a rank-1 matrix that would be a good addition to the current factorization, and then subtracts it from the data matrix. Since there is no minus operation in the subtropical algebra, subtracting newly found patterns is not possible. We will instead mark the elements in the data that got covered, so that they would not influence the discovery of subsequent patterns. We will next present an algorithm, that we called `Equator` (Algorithm 1), that implements the above approach.

To formalize these ideas, let us start with rewriting the subtropical matrix product in the following form.

$$\boldsymbol{B} \boxtimes \boldsymbol{C} = \max_{1 \le s \le k} \boldsymbol{B}^s \boldsymbol{C}_s \, . \tag{3.1}$$

Here each component $\boldsymbol{B}^s \boldsymbol{C}_s$ is a rank-1 pattern, and the maximum is taken elementwise across all patterns. It now becomes apparent that Problem 4 can be split into k subproblems of the following form: given a rank-$(l-1)$ decomposition $\boldsymbol{B} \in \mathbb{R}_+^{n \times (l-1)}$, $\boldsymbol{C} \in \mathbb{R}_+^{(l-1) \times m}$ of a matrix $\boldsymbol{A} \in \mathbb{R}_+^{n \times m}$, find a column vector $\boldsymbol{b} \in \mathbb{R}_+^{n \times 1}$ and a row vector $\boldsymbol{c} \in \mathbb{R}_+^{1 \times m}$ such that the error

$$\| \boldsymbol{A} - \max \{ \boldsymbol{B} \boxtimes \boldsymbol{C}, \boldsymbol{bc} \} \| \tag{3.2}$$

is minimized. Combining vectors $\boldsymbol{b}$ and $\boldsymbol{c}$ with factors $\boldsymbol{B}$ and $\boldsymbol{C}$ now yields a rank-l decomposition of $\boldsymbol{A}$. We have intentionally not used any specific norm in (3.2) since various error norms will be optimized later in this chapter. We assume by definition that the rank-0 decomposition is an all zero matrix of the same size as $\boldsymbol{A}$. The problem of rank-k subtropical matrix factorization is then reduced to solving (3.2) k times. One should of course remember that this scheme is just a heuristic, and finding optimal blocks on each iteration does not guarantee converging to a global minimum.

One prominent issue with the above approach is that an optimal rank-$(k-1)$ decomposition might not be very good when considered as a part of a rank-k decomposition. This is because for smaller ranks we generally have to cover the data more crudely, whereas when the rank increases, we can afford to use smaller and more refined blocks. In order to deal with this problem, we find and then update the blocks repeatedly, in a cyclic fashion. That means that after discovering the last block, we go all the way back to block one. The input parameter M defines the number of full cycles we make.

On a high level `Equator` (Algorithm 1) works as follows. First the factor matrices are initialized to all zeros (line 2). Since the algorithm makes iterative changes to the current solutions that might in some cases lead to worsening of the results, it also stores the best reconstruction error and the corresponding factors found so far. They are initialized with the starting solution on lines 3–4. The main work is done in the loop on lines 5–12, where on each iteration we update a single rank-1 matrix in the current decomposition using the `UpdateBlock` routine (line 7), and then check if the update improves the best result (lines 8–10).

We will present two versions of the `UpdateBlock` function, one called `Capricorn` and the other one `Cancer`. `Capricorn` is designed to work with discrete (or flipping) noise, when some of the elements in the data are randomly changed to different values. In this setting the level of noise is the proportion of the flipped elements relative to the total number of nonzeros. `Cancer`, on the other hand, is robust with continuous noise, when many elements are affected (e.g. Gaussian noise). We will discuss both of them in detail in the following sections. In the following, especially when presenting the experiments, we will use names

Algorithm 1 Equator

```
Input: $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$, $k > 0$, $M > 0$
Output: $\boldsymbol{B}^* \in \mathbb{R}_+^{n\times k}$, $\boldsymbol{C}^* \in \mathbb{R}_+^{k\times m}$
 1: function Equator($\boldsymbol{A}, k, M$)
 2:     $\boldsymbol{B} \leftarrow 0^{n\times k}$, $\boldsymbol{C} \leftarrow 0^{k\times m}$
 3:     $\boldsymbol{B}^* \leftarrow \boldsymbol{B}, \boldsymbol{C}^* \leftarrow \boldsymbol{C}$
 4:     $bestError \leftarrow E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$
 5:     for $count \leftarrow 1$ to $k \times M$ do
 6:         $l \leftarrow (count - 1) \pmod k + 1$            ▷ Index of the current block
 7:         $[\boldsymbol{B}^l, \boldsymbol{C}_l] \leftarrow$ UpdateBlock($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, count$)
 8:         if $E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) < bestError$ then
 9:             $\boldsymbol{B}^* \leftarrow \boldsymbol{B}, \boldsymbol{C}^* \leftarrow \boldsymbol{C}$
10:             $bestError \leftarrow E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$
11:         end if
12:     end for
13:     return $\boldsymbol{B}^*$, $\boldsymbol{C}^*$
14: end function
```

Capricorn and Cancer not only for a specific variation of the UpdateBlock function, but also for the Equator algorithm that uses it.

3.2 Algorithm (Capricorn)

We first describe Capricorn, which is designed to solve the subtropical matrix factorization problem in the presence of discrete noise, and minimizes the $L1$ norm of the error matrix. The main idea behind the algorithm is to spot potential blocks by considering ratios of matrix rows. Consider an arbitrary rank-1 block $\boldsymbol{X} = \boldsymbol{bc}$, where $\boldsymbol{b} \in \mathbb{R}_+^{n\times 1}$ and $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$. For any indices i and j such that $\boldsymbol{b}_i > 0$ and $\boldsymbol{b}_j > 0$, we have $\boldsymbol{X}_j = \frac{\boldsymbol{b}_j}{\boldsymbol{b}_i}\boldsymbol{X}_i$. This is a characteristic property of rank-1 matrices – all rows are multiples of one another. Hence, if a block $\boldsymbol{X}$ dominates some region Γ of a matrix $\boldsymbol{A}$, then rows of $\boldsymbol{A}$ should all be multiples of each other within Γ. These rows might have different lengths due to block overlap, in which case the rule only applies to their common part.

UpdateBlock (Algorithm 2) starts by identifying the index of the block that has to be updated at the current iteration (line 2). In order to find the best new block, we need to take into account that some parts of the data have already been covered, and we must ignore them. This is accomplished by replacing the original matrix with a residual $\boldsymbol{R}$ that represents what there is left to cover. The building of the residual (line 3) reflects the winner-takes-it-all property of the max-times algebra: if an element of $\boldsymbol{A}$ is approximated by a smaller value, it appears as such in the residual; if it is approximated by a value that is at least as large, then the corresponding residual element is NaN, indicating that this value is already covered. We then select a seed row (line 4), with the intention of growing a block around it. We choose the row with the largest sum as this increases the chances of finding the most prominent block. In order to find the best block $\boldsymbol{X}$ that the seed row passes through, we first find a binary matrix $\boldsymbol{H}$ that represents the mask of $\boldsymbol{X}$ (line 5). Next, on lines 6–9 we choose an approximation of the block mask with index sets b_idx and c_idx, which define

Algorithm 2 `UpdateBlock (Capricorn)`

```
Input: $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$, $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$, $\boldsymbol{C} \in \mathbb{R}_+^{k\times m}$, $count > 0$
Output: $\boldsymbol{b} \in \mathbb{R}_+^{n\times 1}$, $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$
Parameters: $bucketSize > 0$, $\delta > 0$, $\theta > 0$, $\tau \in [0,1]$
 1: function UpdateBlock($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, count$)
 2:     $l \leftarrow (count - 1) \pmod{k} + 1$                         ▷ Index of the current block
 3:     $\boldsymbol{R}_{ij} \leftarrow \begin{cases} \boldsymbol{A}_{ij} & (\boldsymbol{B}^{-l} \boxtimes \boldsymbol{C}_{-l})_{ij} < \boldsymbol{A}_{ij} \\ NaN & \text{otherwise} \end{cases}$       ▷ Residual matrix
 4:     $idx \leftarrow \arg\max_i \sum_j r_{ij}$
 5:     $\boldsymbol{H} \leftarrow$ CorrelationsWithRow($\boldsymbol{R}, idx, bucketSize, \delta, \tau$)
 6:     $r \leftarrow \arg\max_i \sum_j h_{ij}$
 7:     $c \leftarrow \arg\max_j \sum_i h_{ij}$
 8:     $b\_idx \leftarrow \{i : \boldsymbol{H}_{ic} = 1\}$
 9:     $c\_idx \leftarrow \{i : \boldsymbol{H}_{ri} = 1\}$
10:     $[\boldsymbol{b}, \boldsymbol{c}] \leftarrow$ RecoverBlock($\boldsymbol{R}, b\_idx, c\_idx$)
11:     $\boldsymbol{b} \leftarrow$ AddRows($\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{A}, \theta, bucketSize, \delta$)
12:     $\boldsymbol{c} \leftarrow$ AddRows($\boldsymbol{c}^T, \boldsymbol{b}^T, \boldsymbol{A}^T, \theta, bucketSize, \delta)^T$
13:     return $\boldsymbol{b}, \boldsymbol{c}$
14: end function
```

what elements of $\boldsymbol{b}$ and $\boldsymbol{c}$ should be nonzero. The next step is to find the actual values of elements within the block with the function `RecoverBlock` (line 10). Finally, we inflate the found core block with `ExpandBlock` (line 11).

The function **`CorrelationsWithRow`** (Algorithm 3) finds the mask of a new block. It does so by comparing a given seed row to other rows of the matrix and extracting sets where the ratio of the rows is almost constant. As was mentioned before, if two rows locally represent the same block, then one should be a multiple of the other, and the ratios of their corresponding elements should remain level. **`CorrelationsWithRow`** processes the input matrix row by row using the function `FindRowSet`, which for every row outputs the most likely set of indices, where it is correlated with the seed row (lines 4–6). Since the seed row is obviously the most correlated with itself, we compensate for this by replacing its mask with that of the second most correlated row (lines 8–9). Finally, we drop some of the least correlated rows after comparing their correlation value ϕ to that of the second most correlated row (after the seed row). The correlation function ϕ is defined as follows:

$$\phi(\boldsymbol{H}, idx, i) = \frac{\langle \boldsymbol{H}_i, \boldsymbol{H}_{idx} \rangle}{\langle \boldsymbol{H}_i, \boldsymbol{H}_i \rangle + 1} . \tag{3.3}$$

The parameter τ is a threshold determining whether a row should be discarded or retained. The auxiliary function `FindRowSet` (Algorithm 4) compares two vectors and finds the biggest set of indices where their ratio remains almost constant. It does so by sorting the log-ratio of the input vectors into buckets of a fixed size and then choosing the bucket with the most elements. The notation $\boldsymbol{u} \,./\, \boldsymbol{v}$ on line 2 means elementwise ratio of vectors $\boldsymbol{u}$ and $\boldsymbol{v}$.

`Capricorn` has two additional parameters: $bucketSize$ and δ. If the largest bucket has fewer than $bucketSize$ elements, the function will return an empty set – this is done because very small blocks do not reveal much structure and

Algorithm 3 `CorrelationsWithRow`

```
Input: $\boldsymbol{R} \in \mathbb{R}_+^{n\times m}$, $idx \in [n]$, $bucketSize > 0$, $\delta > 0$, $\tau \in [0,1]$
Output: $\boldsymbol{H} \in \{0,\,1\}^{n\times m}$
 1: function CorrelationsWithRow($\boldsymbol{R}, idx, bucketSize, \delta, \tau$)
 2:     turn all $NaN$ elements of $\boldsymbol{R}$ to 0
 3:     $\boldsymbol{H} \leftarrow 0^{n\times m}$
 4:     for $i \leftarrow 1$ to $n$ do
 5:         $V_i \leftarrow$ FindRowSet($\boldsymbol{R}_{idx}, \boldsymbol{R}_i, bucketSize, \delta$)
 6:         $\boldsymbol{H}(i, V_i) \leftarrow 1$
 7:     end for
 8:     $s \leftarrow \arg\max_{i\,:\,i\neq idx} \sum_j h_{ij}$
 9:     $\boldsymbol{H}_{idx} \leftarrow \boldsymbol{H}_s$
10:     for $i \leftarrow 1$ to $n$ do
11:         if $\phi(\boldsymbol{H}, idx, i) < \phi(\boldsymbol{H}, idx, s) - \tau$ then
12:             $\boldsymbol{H}_i \leftarrow 0$
13:         end if
14:     end for
15:     return $\boldsymbol{H}$
16: end function
```

Algorithm 4 `FindRowSet`

```
Input: $\boldsymbol{u} \in \mathbb{R}_+^m, \boldsymbol{v} \in \mathbb{R}_+^m$, $bucketSize > 0, \delta > 0$
Output: $V \subset [m]$
 1: function FindRowSet($\boldsymbol{u}, \boldsymbol{v}, bucketSize, \delta$)
 2:     $\boldsymbol{r} \leftarrow \log(\boldsymbol{u}\,./\,\boldsymbol{v})$
 3:     $nBuckets \leftarrow \lceil(\max\{r\} - \min\{r\})/\delta\rceil$
 4:     for $i \leftarrow 0$ to $nBuckets$ do
 5:         $V_i \leftarrow \{idx \in [m] \,:\, \min\{\boldsymbol{r}\} + i\delta \leq r_{idx} < \min\{\boldsymbol{r}\} + (i+1)\delta\}$
 6:     end for
 7:     $V \leftarrow \arg\max\{|V_i| \,:\, i = 1, \ldots, nBuckets\}$
 8:     if $|V| < bucketSize$ then
 9:         $V \leftarrow \emptyset$
10:     end if
11:     return $V$
12: end function
```

are mostly accidental. The width of the buckets is determined by the parameter δ.

At this point we know the mask of the new block, that is, the locations of its non-zeros. To fill in the actual values, we consider the submatrix defined by indices where the mask is non-zero, and find the best rank-1 approximation of it. We do this using the `RecoverBlock` function (Algorithm 5). It begins by setting all elements outside of the non-zero index set of the mask to 0 as they are irrelevant to the block (line 2). Then it chooses one row to represent the block (lines 3–4), which will be used to find a good rank-1 cover.

Finally, we find the optimal column vector for the block by computing the best weights to be used for covering different rows of the block with its representing row (line 5). Here we optimize with respect to the Frobenius norm, rather than L_1 matrix norm, since it allows to solve the optimization problem in closed form.

Algorithm 5 `RecoverBlock`

```
Input: $\boldsymbol{R} \in \mathbb{R}_+^{n \times m}$, $bIdx \subset [n]$, $cIdx \subset [m]$
Output: $\boldsymbol{b} \in \mathbb{R}_+^{n \times 1}$, $\boldsymbol{c} \in \mathbb{R}_+^{1 \times m}$
1: function RecoverBlock($\boldsymbol{R}$, $bIdx$, $cIdx$)
2:     turn $\boldsymbol{R}$ to 0 except elements with indices $(bIdx, cIdx)$
3:     $p \leftarrow$ RowRepresentingBlock($\boldsymbol{R}$, $bIdx$)
4:     $\boldsymbol{c} \leftarrow \boldsymbol{R}_p$
5:     $\boldsymbol{b} \leftarrow \arg\min_{\boldsymbol{t} \in \mathbb{R}_+^{n \times 1}} \|\boldsymbol{R} - \boldsymbol{tc}\|_F$
6:     return $\boldsymbol{b}$, $\boldsymbol{c}$
7: end function
```

Algorithm 6 `AddRows`

```
Input: $\boldsymbol{b} \in \mathbb{R}_+^{n \times 1}$, $\boldsymbol{c} \in \mathbb{R}_+^{1 \times m}$, $\boldsymbol{A} \in \mathbb{R}_+^{n \times m}$, $\theta > 0$, $bucketSize > 0$, $\delta > 0$
Output: $\boldsymbol{b} \in \mathbb{R}_+^{n \times 1}$
1: function AddRows($\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{A}, \theta, bucketSize, \delta$)
2:     $b\_idx \leftarrow \{t : \boldsymbol{b}_t > 0\}$
3:     for $i \in [n] \setminus b\_idx$ do
4:         $V_i \leftarrow$ FindRowSet($\boldsymbol{c}, \boldsymbol{R}_i, bucketSize, \delta$)
5:         if $V_i = \emptyset$ then
6:             continue
7:         end if
8:         $\alpha \leftarrow mean(\boldsymbol{R}_{iV_i} ./ \boldsymbol{c}_{V_i})$
9:         $impact \leftarrow \frac{\sum_{s \in V_i} \max\{0, \alpha c_s - \boldsymbol{A}_{is}\}}{\sum_{s \in V_i} \boldsymbol{A}_{is} - |\boldsymbol{A}_{is} - \alpha c_s|}$
10:        if $impact \leq \theta$ then
11:            $\boldsymbol{b}_i \leftarrow \alpha$
12:        end if
13:    end for
14:    return $\boldsymbol{b}$
15: end function
```

Since blocks often heavily overlap, we are susceptible to finding only their fragments – some parts of a block can be dominated by another block and subsequently not recognized. Hence, we need to expand found blocks to make them complete. This is done separately for rows and columns in the method called `AddRows` (Algorithm 6), which, given a starting block $\boldsymbol{X} = \boldsymbol{bc}$ and the original matrix $\boldsymbol{A}$, tries to add new nonzero elements to $\boldsymbol{b}$. It iterates through all rows of $\boldsymbol{A}$ and adds those that would make a positive impact on the objective without unnecessarily overcovering the data. In order to decide whether a given row should be added, it first extracts a set V_i of indices where this row is a multiple of the row vector $\boldsymbol{c}$ of the block (if they are not sufficiently correlated, then the row does not belong to the block) (line 4). A row is added if the evaluation of the function in line 9

$$\psi(\alpha) = \frac{\sum_{s \in V_i} \max\{0, \alpha c_s - \boldsymbol{A}_{is}\}}{\sum_{s \in V_i} \boldsymbol{A}_{is} - |\boldsymbol{A}_{is} - \alpha c_s|} \tag{3.4}$$

is below the threshold θ. In (3.4), the numerator measures by how much the new row would overcover the original matrix, and the denominator reflects the improvement in the objective compared to a zero row.

Parameters. Capricorn has four parameters in addition to the common parameters in the Equator framework: $bucketSize > 0$, $\delta > 0$, $\theta > 0$, and $\tau \in [0, 1]$. The first one, $bucketSize$, determines the minimum number of elements in two rows that must have "approximately" the same ratio for them to be considered for building a block. The parameter δ defines the bucket width when computing row correlations. When expanding a block, θ is used to decide whether to add a row (or column) to it – the decision is positive whenever the expression (3.4) is at most θ. Finally τ is used during the discovery of correlated rows. The value of τ belongs to the closed unit interval, and the higher it is, the more rows will be added.

Computational complexity. Capricorn is a particular case of the Equator algorithm when Algorithm 2 is used as UpdateBlock routine. In order to determine its complexity, first observe that UpdateBlock is called Mk times, where M is a constant parameter. The complexity of Equator is thus k times the complexity of UpdateBlock. It now suffices to estimate the complexity of the UpdateBlock function since it accounts for most of the computation in Equator. There are three main contributors to the Capricorn's version of UpdateBlock (Algorithm 2): CorrelationsWithRow, RecoverBlock, and AddRows. **CorrelationsWithRow** compares every row to the seed row, each time calling FindRowSet, which in turn has to process all m elements of both rows. This results in the total complexity of CorrelationsWithRow being $O(nm)$. To find the complexity of RecoverBlock, first observe that any "pure" block $\boldsymbol{X}$ can be represented as $\boldsymbol{X} = \boldsymbol{b}\boldsymbol{c}$, where $\boldsymbol{b} \in \mathbb{R}_+^{n' \times 1}$ and $\boldsymbol{c} \in \mathbb{R}_+^{1 \times m'}$ with $n' \leq n$ and $m' \leq m$. RecoverBlock selects $\boldsymbol{c}$ from the rows of $\boldsymbol{X}$ and then finds the corresponding column vector $\boldsymbol{b}$ that minimizes $\|\boldsymbol{X} - \boldsymbol{b}\boldsymbol{c}\|_F$. In order to select the best row, we have to try each of the n' candidates, and since finding the corresponding $\boldsymbol{b}$ for each of them takes time $O(n'm')$, this gives the runtime of RecoverBlock as $O(n')O(n'm') = O(n^2m)$. The most computationally expensive parts of AddRows are FindRowSet (line 4), finding the mean (line 8), and computing the impact (line 9), which all run in $O(m)$ time. All of these operations have to be repeated $O(n)$ times, and hence the runtime of AddRows is $O(nm)$. Thus, we can now estimate the complexity of UpdateBlock to be $O(nm) + O(n^2m) + O(nm) = O(n^2m)$, which leads to the total runtime of Capricorn to be $O(n^2mk)$.

3.3 Algorithm (Cancer)

We now present our second algorithm, Cancer, which is a counterpart of Capricorn specifically designed to work in the presence of high levels of continuous noise. The reason why Capricorn cannot deal with continuous noise is that it expects the rows in a block to have an "almost" constant elementwise ratio, which is not the case when too many entries in the data are disturbed. For example, even low levels of Gaussian noise would make the ratios vary enough to hinder Capricorn's ability to spot blocks. With Cancer we take a new approach, which is based on polynomial approximation of the objective. We also replace the L_1 matrix norm, which was used as an objective for Capricorn, with the Frobenius norm. The reason for that is that when the noise is continuous, its level is defined as the total deviation of the noisy data from the original, rather than a count of the altered elements. This makes the Frobenius norm a good estimator for the amount of noise. `Cancer` conforms to the general framework of `Equator` (Algorithm 1), and differs from `Capricorn` only in how it finds the blocks and in the objective function.

Observe that in order to solve the problem (3.2), we need to find a column vector $\boldsymbol{b} \in \mathbb{R}_+^{n\times 1}$ and a row vector $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$ such that they provide the best rank-1 approximation of the input matrix given the current factorization. The objective function is not convex in either $\boldsymbol{b}$ or $\boldsymbol{c}$ and is generally hard to optimize directly, so we have to simplify the problem, which we do in two steps. First, instead of doing full optimization of $\boldsymbol{b}$ and $\boldsymbol{c}$ simultaneously, we update only a single element of one of them at a time. This way the problem is reduced to single variable optimization. Even then the objective is hard to minimize, and we replace it with a polynomial approximation, which is easy to optimize directly.

The `Cancer` version of the `UpdateBlock` function is described in Algorithm 7. It alternatingly updates the vectors $\boldsymbol{b}$ and $\boldsymbol{c}$ using the `AdjustOneElement` routine. Both $\boldsymbol{b}$ and $\boldsymbol{c}$ will be updated $\lfloor f(n+m)/2 \rfloor$ times. `UpdateBlock` starts by finding the index of the block that has to be changed (line 2). Since the purpose of `UpdateBlock` is to find the best rank-1 matrix to replace the current block, we also need to compute the reconstructed matrix without it, which is done on line 3. We then find the number of times `AdjustOneElement` will be called (line 4) and change the degree of polynomials used for objective function approximation (line 5). This is needed because high degree polynomials are better at finalizing a solution that is already reasonably good, but tend to overfit the data and cause the algorithm to get stuck in local minima at the beginning. It is therefore beneficial to start with polynomials of lower degrees and then gradually increase it. The actual changes to $\boldsymbol{b}$ and $\boldsymbol{c}$ happen in the loop (lines 7–9), where we update them using `AdjustOneElement`.

The `AdjustOneElement` function (Algorithm 8) updates a single entry in either a column vector $\boldsymbol{b}$ or a row vector $\boldsymbol{c}$. Let us consider the case when $\boldsymbol{b}$ is fixed and $\boldsymbol{c}$ varies. In order to decide which element of $\boldsymbol{c}$ to change, we need to compare the best changes to all m entries and then choose the one that yields the most improvement to the objective. A single element $\boldsymbol{c}_l$ only has an effect on the error along the column l. Assume that we are currently updating block with index q and let $\boldsymbol{N}$ denote the reconstruction matrix without this block, that is $\boldsymbol{N} = \boldsymbol{B}^{-q} \boxtimes \boldsymbol{C}_{-q}$. Minimizing $E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ with respect to $\boldsymbol{c}_l$ is then equivalent to minimizing

$$\gamma(\boldsymbol{A}_l, \boldsymbol{N}_l, \boldsymbol{b}, \boldsymbol{c}_l) = \sum_{i=1}^{n} (\boldsymbol{A}_{il} - \max\{\boldsymbol{N}_{il}, \boldsymbol{b}_i \boldsymbol{c}_l\})^2 , \tag{3.5}$$

which is done in the `PolyMin` routine (line 4). `PolyMin` returns the (approximate) best error err and the value of $\boldsymbol{c}_l$ achieving it. While minimizing (3.5) is substantially easier than solving the original matrix factorization problem, it presents its own challenges as γ is neither convex nor differentiable with respect to $\boldsymbol{c}_l$. Therefore, instead of minimizing (3.5) directly, we replace γ with a polynomial approximation. The advantage of this method is that the number of extreme points of a polynomial is bounded by its degree, and hence we can

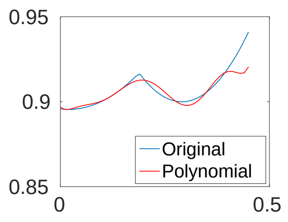


Figure 3.1: Original objective from (3.5) versus its polynomial approximation used in `PolyMin`.

Algorithm 7 `UpdateBlock` (`Cancer`)

```
Input: $\boldsymbol{A} \in \mathbb{R}_+^{n \times m}$, $\boldsymbol{B} \in \mathbb{R}_+^{n \times k}$, $\boldsymbol{C} \in \mathbb{R}_+^{k \times m}$, $count > 0$
Output: $\boldsymbol{b} \in \mathbb{R}_+^{n \times 1}$, $\boldsymbol{c} \in \mathbb{R}_+^{1 \times m}$
Parameters: $t > 2$, $0 < f < 1$
 1: function UpdateBlock($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, count$)
 2:     $l \leftarrow (count - 1) \pmod{k} + 1$        ▷ Index of the current block
 3:     $\boldsymbol{N} \leftarrow \boldsymbol{B}^{-l} \boxtimes \boldsymbol{C}_{-l}$        ▷ Reconstructed matrix without the $i$-th block
 4:     $niters \leftarrow \lfloor f(n+m)/2 \rfloor$
 5:     $deg \leftarrow 2 + \lfloor (count-1)/k \rfloor \pmod{t}$
 6:     $\boldsymbol{b} \leftarrow \boldsymbol{B}^l$, $\boldsymbol{c} \leftarrow \boldsymbol{C}_l$
 7:     for $iter \leftarrow 1$ to $niters$ do
 8:         $\boldsymbol{c} =$ AdjustOneElement($\boldsymbol{A}, \boldsymbol{N}, \boldsymbol{b}, \boldsymbol{c}, deg$)
 9:         $\boldsymbol{b} =$ AdjustOneElement($\boldsymbol{A}^T, \boldsymbol{N}^T, \boldsymbol{c}^T, \boldsymbol{b}^T, deg)^T$
10:     end for
11:     return $\boldsymbol{b}$, $\boldsymbol{c}$
12: end function
```

minimize it by examining each of these points. An example of function γ and its polynomial approximation is shown in Figure 3.1.

In order to build a degree deg approximation of γ, we first evaluate γ at $deg + 1$ points generated uniformly at random from the interval [0, 5] and then fit a polynomial to the obtained values. The upper bound of 5 does not have any special meaning, rather it was chosen by trial and error. `PolyMin` is a heuristic and does not necessarily find the global minimum of the objective function. Moreover, in rare cases it might even cause an increase in the objective value. In such cases it would, in theory, make sense to just keep the value prior to the update, as in that case the objective at least does not increase. However, in practice this phenomenon helps to get out of local minima. Since we are only interested in the improvement of the objective achieved by updating a single entry of $\boldsymbol{c}$, we compute the improvement of the objective after the change (line 5). After trying every column of $\boldsymbol{c}$, we update only the column that yields the largest improvement.

Parameters. `Cancer` has two parameters, $t > 2$ and $0 < f < 1$, that control its execution. The first one, t, is the maximum allowed degree of polynomials used for approximation of the objective, which we set to 16 in all our experiments. The second parameter, f, determines the number of single element updates we make to the row and column vectors of a block in `UpdateBlock`. To demonstrate

Algorithm 8 `AdjustOneElement`

```
Input: $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$, $\boldsymbol{N} \in \mathbb{R}_+^{n\times m}$, $\boldsymbol{b} \in \mathbb{R}_+^{n\times 1}$, $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$, $deg \geq 2$
Output: $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$
 1: function AdjustOneElement($\boldsymbol{A}, \boldsymbol{N}, \boldsymbol{b}, \boldsymbol{c}, deg$)
 2:     for $j \leftarrow 1$ to $m$ do
 3:         $baseError \leftarrow \sum_{i=1}^{n} (\boldsymbol{A}_{ij} - \max\{\boldsymbol{N}_{ij}, \boldsymbol{b}_i\boldsymbol{c}_j\})^2$
 4:         $[err, \boldsymbol{x}_i] \leftarrow$ PolyMin($\boldsymbol{A}^j, \boldsymbol{N}^j, \boldsymbol{b}, deg$)
 5:         $\boldsymbol{u}_i \leftarrow baseError - err$
 6:     end for
 7:     $i \leftarrow$ the index $i$ of largest value of $\boldsymbol{u}$
 8:     $\boldsymbol{c}_i \leftarrow \boldsymbol{x}_i$
 9:     return $\boldsymbol{c}$
10: end function
```

that the chosen values of the parameters are reasonable, we perform a grid search for various parameter values (Figure 3.2).

Computational complexity. Here `UpdateBlock` (Algorithm 7) is a loop that calls `AdjustOneElement` $\lfloor f(n+m) \rfloor$ times. In `AdjustOneElement` the contributors to the complexity are computing the base error (line 3) and a call to `PolyMin` (line 4). Both of them are performed n or m times depending on whether we supplied the column vector $\boldsymbol{b}$ or the row vector $\boldsymbol{c}$ to `AdjustOneElement`. Finding the base error takes time $O(m)$ for $\boldsymbol{b}$ and $O(n)$ for $\boldsymbol{c}$. The complexity of `PolyMin` boils down to that of evaluating the max-times objective at $deg+1$ points and then minimizing a degree deg polynomial. Hence, `PolyMin` runs in time $O(m)$ or $O(n)$ depending on whether we are optimizing $\boldsymbol{b}$ or $\boldsymbol{c}$, and the complexity of `AdjustOneElement` is $O(nm)$.

Since `AdjustOneElement` is called $\lfloor f(n+m)/2 \rfloor$ times and f is a fixed parameter, this gives the complexity $O\big((n+m)nm\big)$ for `UpdateBlock` and $O\big((n+m)nmk\big) = O(\max\{n, m\}nmk)$ for `Cancer`.

Empirical evaluation of the time complexity is reported in Section 3.4.3.

Generalized Cancer. The `Cancer` algorithm can be adapted to optimize other objective functions. Its general polynomial approximation framework allows for a wide variety of possible objectives, the only constraint being that they have to be additive (we call a function $E(\boldsymbol{A}, \boldsymbol{R})$ *additive* if there exists a mapping $\phi\colon \mathbb{R}_+ \times \mathbb{R}_+ \to \mathbb{R}_+$ such that for all $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$ and $\boldsymbol{R} \in \mathbb{R}_+^{n\times m}$ we have $E(\boldsymbol{A}, \boldsymbol{R}) = \sum_{ij} \phi(\boldsymbol{A}_{ij}, \boldsymbol{R}_{ij})$). Some examples of such functions are L_1 and Frobenius matrix norms, as well as Kullback–Leibler and Jensen–Shannon divergences. In order to use the generalized form of `Cancer` one simply has to replace the Frobenius norm with another cost function wherever the error is evaluated.

3.4 Experimentimental Evaluation

We test both `Capricorn` and `Cancer` on synthetic and real-world data. In addition, we also compare against a variation of `Cancer` that optimizes the

Jensen–Shannon divergence, which we call CancerJS. The purpose of the synthetic experiments is to evaluate the properties of the algorithm in a controlled environment, where we know that the data has the max-times structure. They also demonstrate on what kind of data each algorithm excels and what their limitations are. The purpose of the real-world experiments is to confirm that these observations also hold true in real-world data, and to study what kinds of data sets actually have max-times structure. A MATLAB implementation of Capricorn and Cancer as well as scripts that run the experiments in this section are freely available for academic use.[1]

Parameters of Cancer. Both variations of Cancer use the same set of parameters. For the synthetic experiments we used $M = 14$, $t = 16$, and $f = 0.1$. For the real world experiments we set $t = 16$, $f = 0.1$, and $M = 40$ (except for Eigenfaces, where we used $M = 50$ and Bas1LP, where we set $M = 8$). Increasing M, which controls the number of cycles of execution of Cancer, almost invariably improves the results. At some point though, the gains become marginal, and the value of $M = 40$ is chosen so as to reach the point where increasing M further would not yield much improvement. Sometimes though, this moment can be reached faster – for example the smaller choice of M for Bas1LP is motivated by the fact that Cancer quickly reached a point where it could no longer make significant progress, despite Bas1LP being the largest dataset. The relationship of the other two parameters and the quality of decomposition is more complex. In order to demonstrate it, we performed a grid search on a real-world dataset representing a land registry house price index (for more details see HPI dataset in Section 3.4.3) and a synthetic dataset. The results are shown in Figures 3.2(a) and 3.2(b), respectively. We see in Figure 3.2(a) that the dependence on f and t is not monotone, and it is hard to pinpoint the best combination exactly. Moreover, the optimal values can differ depending on the dataset; for example, Figure 3.2(b) features an almost monotone dependence on f that flattens out before f reaches 0.1. From our experience, however, the values of $t = 16$ and $f = 0.1$ seem to be a good choice.

Parameters of Capricorn. In both synthetic and real-world experiments we used the following default set of parameters: $M = 4$, $\mathit{bucketSize} = 3$, $\delta = 0.01$, $\theta = 0.5$, and $\tau = 0.5$. As with Cancer, there is a complex dependency of the results on the parameters, but the values chosen above seem to produce good results in most cases. We do not show comparison figures, as we did with Cancer, because of a large number of parameters.

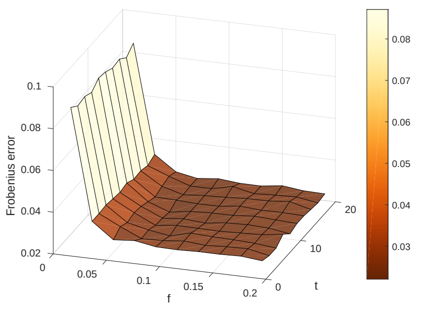
3.4.1 Other methods

We compared our algorithms against SVD and five versions of NMF. For SVD, we used Matlab's built-in implementation. The first form of NMF is a sparse NMF algorithm by Hoyer [2004],[2] which we call SNMF. It defines the sparsity of a vector $\boldsymbol{x} \in \mathbb{R}_+^n$ as

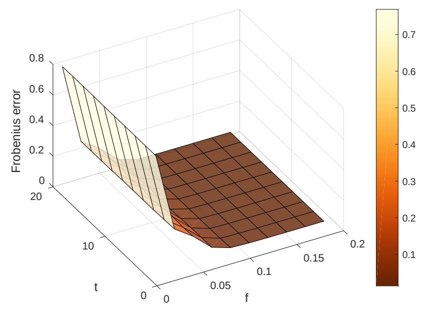
$$\text{sparsity}_H(\boldsymbol{x}) = \frac{\sqrt{n} - \left(\sum_i |\boldsymbol{x}_i|\right) / \sqrt{\sum_i \boldsymbol{x}_i^2}}{\sqrt{n} - 1} , \tag{3.6}$$

[1] http://cs.uef.fi/~pauli/tropical/

[2] https://github.com/aludnam/MATLAB/tree/master/nmfpack, accessed 18 July 2017

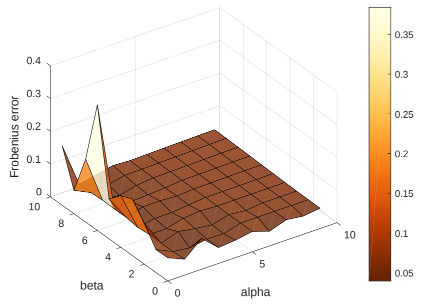


(a) HPI.

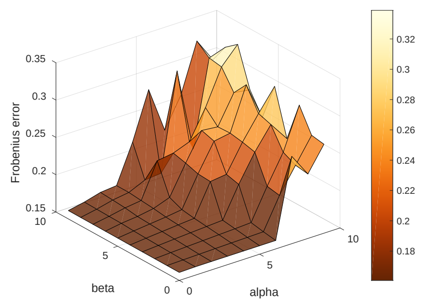


(b) Synthetic experiment with Gaussian noise.

Figure 3.2: Results of `Cancer` with different values of parameters t and f.



(a) HPI.



(b) Synthetic experiment with Gaussian noise.

Figure 3.3: Results of `ALSR` with different regularization parameters for the two factors matrices, which are denoted by α and β respectively.

and returns factorizations where the sparsity of the factor matrices is user-controllable. Note that the above definition of sparsity is different from the one we use elsewhere (see Equation (2.14)). In order to run `SNMF` we used the sparsity of `Cancer`'s factors (as defined by (3.6)) as its sparsity parameter. We also compare against a standard alternating least squares algorithm called `ALS` [Cichocki et al., 2009]. Next we have two versions of NMF that are essentially the same as `ALS`, but they use L_1 regularization for increased sparsity [Cichocki et al., 2009], that is, they aim at minimizing

$$\|\boldsymbol{A} - \boldsymbol{B}\boldsymbol{C}\|_F + \alpha \|\boldsymbol{B}\|_1 + \beta \|\boldsymbol{C}\|_1 \ .$$

The first method is called `ALSR` and uses regularizer coefficient $\alpha = \beta = 1$, and the other, called `ALSR5`, has regularizer coefficient $\alpha = \beta = 5$. It is natural to ask how `ALSR` would fare with different values of parameters. In Figure 3.3 we perform a grid search for the best parameter combination. While the experiment with HPI has a very uneven surface without much structure apart from a couple of spikes, the synthetic dataset demonstrates that high values of α and β can have serious adverse effects on the reconstruction error. It therefore seems safest to set $\alpha = \beta = 0$, which corresponds to the `ALS` method. It is worth mentioning that in many of our experiments larger values of α and β resulted in factors becoming close to zero, or some elements in the factors getting enormous values due to numeric instability. This was the case for some other real-world experiments, such as 4News, which is another indication to use the parameter values of $\alpha = \beta = 0$.

The last NMF algorithm, WNMF by Li and Ngom [2013], is resistant to high levels of missing data, which makes it effective for prediction tasks.

3.4.2 Synthetic experiments

The purpose of synthetic experiments is to prove the concept, that is that our algorithms are capable of identifying the max-times structure when it is there. In order to test this, we first generate the data with the pure max-times structure, then pollute it with some level of noise, and finally run the methods. The noise-free data is created by first generating random factors of some density with nonzero elements drawn from a uniform distribution on the $[0, 1]$ interval and then multiplying them using the max-times matrix product.

We distinguish two types of noise. The first one is the discrete (or tropical) noise, which is introduced in the following way. Assume that we are given an input matrix $\boldsymbol{A}$ of size n-by-m. We first generate an n-by-m noise matrix $\boldsymbol{N}$ with elements drawn from a uniform distribution on the $[0, 1]$ interval. Given a level of noise l, we then turn $\lfloor (1 - l)nm \rfloor$ random elements of $\boldsymbol{N}$ to 0, so that its resulting density is l. Finally, the noise is applied by taking elementwise maximum between the original data and the noise matrix $\boldsymbol{F} = \max\{\boldsymbol{A}, \boldsymbol{N}\}$. This is the kind of noise that Capricorn was designed to handle, so we expect it to be better than Cancer and other comparison algorithms.

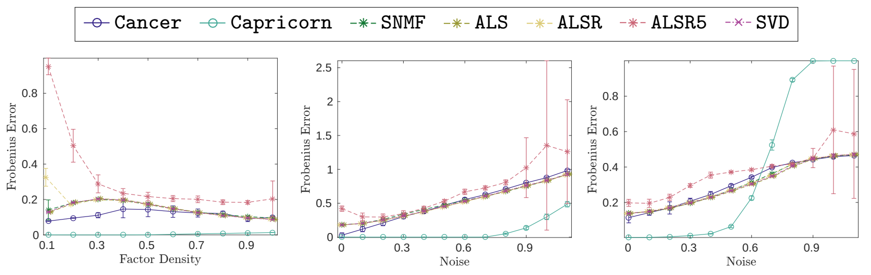
We also test against continuous noise, as it is arguably more common in the real world. For that we chose Gaussian noise with 0 mean, where the noise level is defined to be its standard deviation. Since adding this noise to the data might result in negative entries, we truncate all values in a resulting matrix that are below zero.

Unless specified otherwise, all matrices in the synthetic experiments are of size 1000-by-800 with true max-times rank 10. All results presented in this section are averaged over 10 instances. For reconstruction error tests, we compared our algorithms Capricorn, Cancer, and CancerJS against SVD, NMF, SNMF, ALS, ALSR, and ALSR5. The error is measured as the relative Frobenius norm $\|\tilde{\boldsymbol{A}} - \boldsymbol{A}\|_F / \|\boldsymbol{A}\|$, where $\boldsymbol{A}$ is the data and $\tilde{\boldsymbol{A}}$ its approximation, as that is the measure both SVD and NMF aim at minimizing. We also report the sparsity s of factor matrices obtained by algorithms, which is defined as a fraction of zero elements in the factor matrices, see (2.14). For the experiments with tropical noise, the reconstruction errors are reported in Figure 3.4 and factor sparsity in Figure 3.5. For the Gaussian noise experiments, the reconstruction errors and factor sparsity are shown in Figure 3.6 and Figure 3.7, respectively.

Varying density with tropical noise. In our first experiment we studied the effects of varying the density of the factor matrices in presence of the tropical noise. We changed the density of the factors from $10\,\%$ to $100\,\%$ with an increment of $10\,\%$, while keeping the noise level at $10\,\%$. Figure 3.4(a) shows the reconstruction error and Figure 3.5(a) the sparsity of the obtained factors. Capricorn is consistently the best method, obtaining almost perfect reconstruction; only when the density approaches $100\,\%$ does its reconstruction error deviate slightly from 0. This is expected since the data was generated with the tropical (flipping) noise, that `Capricorn` was designed to work with. Compared to `Capricorn` all other methods clearly underperform, with `Cancer` being the second best. With the exception of `ALSR5`, all NMF methods obtain results similar to those of `SVD`, while having a somewhat higher reconstruction error than `Cancer`. That `SVD` and NMF methods (except `ALSR5`) start behaving better at higher levels of density indicates that these matrices can be explained relatively well using standard algebra. `Capricorn` and `Cancer` also have the highest sparsity of factors, with `Capricorn` exhibiting a decrease in sparsity as the density of the input increases. This behaviour is desirable since ideally we would prefer to find factors that are as close to the original ones as possible. For NMF methods there is a trade-off between the reconstruction error and the sparsity of the factors – the algorithms that were worse at reconstruction tend to have sparser factors.

Varying tropical noise. The amount of noise is always with respect to the number of nonzero elements in a matrix, that is, for a matrix $\boldsymbol{A}$ with $\kappa(\boldsymbol{A})$ nonzero elements and noise level α, we flip $\alpha\kappa(\boldsymbol{A})$ elements to random values. There are two versions of this experiment – one with factor density $30\,\%$ and the other with $60\,\%$. In both cases we varied the noise level from $0\,\%$ to $110\,\%$ with increments of $10\,\%$. Figure 3.4(b) and Figure 3.4(c) show the respective reconstruction errors and Figure 3.5(b) and Figure 3.5(c) the corresponding sparsities of the obtained factors. In the low-density case, `Capricorn` is consistently the best method with essentially perfect reconstruction for up to $80\,\%$ of noise. In the high-density case, however, the noise has more severe effects, and in particular after $60\,\%$ of noise, `Cancer`, `SVD`, and all versions of NMF are better than `Capricorn`. The severity of the noise is, at least partially, explained by the fact that in the denser data we flip more elements than in sparser data: for example when the data matrices are full, at $50\,\%$ of noise, we have already replaced half of the values in the matrices with random values. Further, the quick increase of the reconstruction error for `Capricorn` hints strongly that the max-times structure of the data is mostly gone at these noise levels. `Capricorn` also produces clearly the sparsest factors for the low density case, and is mostly tied with `Cancer` and `ALSR5` when the density is high. It should be noted, however, that `ALSR5` generally has the highest reconstruction error among all the methods, which suggests that its sparse factors come at the cost of recovering little structure from the data.

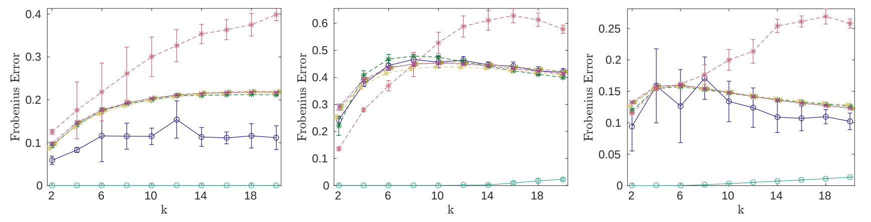
Varying rank with tropical noise. Here we test the effects of the (max-times) rank, with the assumption that higher-rank matrices are harder to reconstruct. The true max-times rank of the data varied from 2 to 20 with increments of 2. There are three variations of this experiment: with $30\,\%$ factor density and $10\,\%$ noise (Figure 3.4(d)), with $30\,\%$ factor density and $50\,\%$ noise (Figure 3.4(e)), and with $60\,\%$ factor density and $10\,\%$ noise (Figure 3.4(f)). The corresponding sparsities are shown in Figures 3.5(d)–(f). `Capricorn` has a clear advantage for all settings, obtaining nearly perfect reconstruction. `Cancer` is generally second best, except for the high noise case, where it is mostly tied with a bunch of NMF methods. It also has a relatively high variance. To see why this happens,



(a) Varying density test.

(b) Varying noise with 30 % density.

(c) Varying noise with 60 % density.



(d) Varying rank test with 10 % noise and 30 % factor density.

(e) Varying rank test with 50 % noise and 30 % factor density.

(f) Varying rank test with 10 % noise and 60 % factor density.

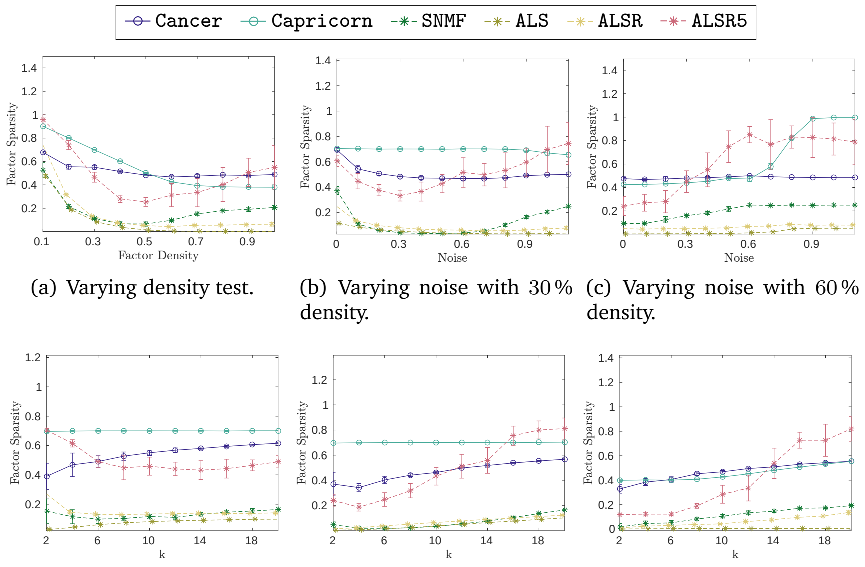
Figure 3.4: Reconstruction errors on synthetic data with tropical noise. x-axis is the parameter varied and y-axis is the relative Frobenius norm. All results are averages over 10 random matrices and the width of the error bars is twice the standard deviation.

consider that Cancer always updates one element in factor matrices at a time. This update is completely dependent on values on a single row (or column) and is sensitive to the spikes that tropical noise introduces to some elements. Interestingly, on the last two plots the reconstruction error actually drops for Cancer, SVD, and NMF-based methods. This is a strong indication that at this point they no longer can extract meaningful structure in the data, and the improvement of the reconstruction error is largely due to uniformization of the data caused by high density and high noise levels.

Varying Gaussian noise. Here we investigate how the algorithms respond to different levels of Gaussian noise, which was varied from 0 to 0.14 with increments of 0.01. A level of noise is a standard deviation of the Gaussian noise used to generate the noise matrix as described earlier. The factor density was kept at $50\,\%$. The results are given in Figure 3.6(a) (reconstruction error) and Figure 3.7(a) (sparsity of factors).

Here Cancer is generally the best method in reconstruction error, and second in sparsity only to Capricorn. The only time it loses to any method is when there is no noise, and Capricorn obtains a perfect decomposition. This is expected since Capricorn is by design better at spotting the pure subtropical structure.

Varying density with Gaussian noise. In this experiment we studied what effects the density of factor matrices used in data generation has on the algo-



(a) Varying density test.

(b) Varying noise with 30 % density.

(c) Varying noise with 60 % density.

(d) Varying rank test with 10 % noise and 30 % factor density.

(e) Varying rank test with 50 % noise and 30 % factor density.

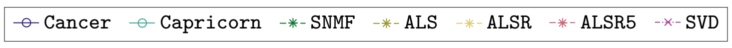
(f) Varying rank test with 10 % noise and 60 % factor density.

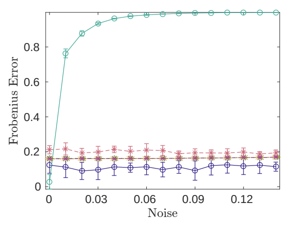
Figure 3.5: Sparsity (fraction of zeros) of the factor matrices for synthetic data with tropical noise. x-axis is the parameter varied and y-axis is the sparsity of the factors. The markers are averages of 10 random matrices and the width of the error bars is twice the standard deviation.

rithms' performance. For this purpose we varied the density from $10\,\%$ to $100\,\%$ with increments of $10\,\%$, while keeping the other parameters fixed. There are two versions of this experiment, one with low noise level of 0.01 (Figures 3.6(b) and 3.7(b)), and a more noisy case at 0.08 (Figures 3.6(c) and 3.7(c)).

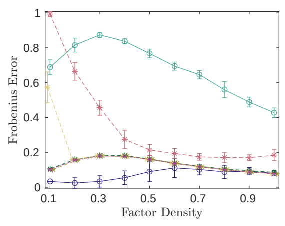
Cancer provides the least reconstruction error in this experiment, being clearly the best until the density is 0.7, from which point on it is tied with SVD and the NMF-based methods (the only exception being the least-dense high-noise case, where ALSR obtains a slightly better reconstruction error). Capricorn is the worst by a wide margin, but this is not surprising, as the data does not follow its assumptions. On the other hand, Capricorn does produce generally the sparsest factorization, but these are of little use given its bad reconstruction error. Cancer produces the sparsest factors from the remaining methods, except in the first few cases where ALSR5 is sparser (and worse in reconstruction error), meaning that Cancer produces factors that are both the most accurate and very sparse.

Varying rank with Gaussian noise. The purpose of this test is to study the performance of algorithms on data of different max-times ranks. We varied the true rank of the data from 2 to 20 with increments of 2. The factor density was fixed at $50\,\%$ and Gaussian noise at 0.01. The results are shown in Figure 3.6(d) (reconstruction error) and Figure 3.7(d) (sparsity of factors). The findings are similar to the ones shown above, with Cancer returning the most accurate and

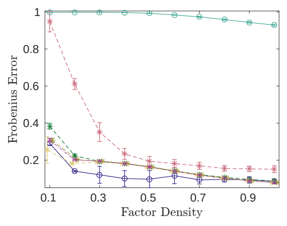




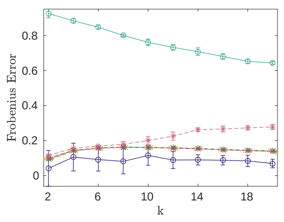
(a) Varying noise with density 50 %.



(b) Varying density with Gaussian noise of 0.01.



(c) Varying density with Gaussian noise of 0.08.



(d) Varying rank; 50 % density and Gaussian noise of 0.01.

Figure 3.6: Reconstruction error (Frobenius norm) for synthetic data with Gaussian noise. The markers are averages of 10 random matrices and the width of the error bars is twice the standard deviation.

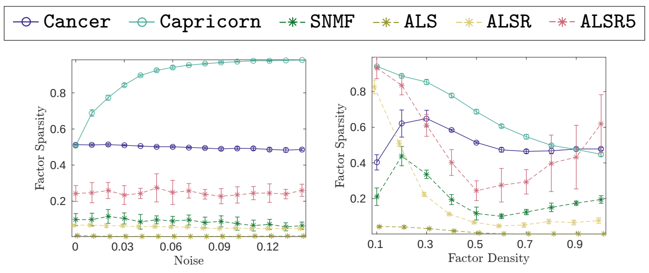
second sparsest factorizations.

Optimizing the Jensen–Shannon divergence. By default, `Cancer` optimizes the Frobenius reconstruction error, but it can be replaced by an arbitrary additive cost function. We performed experiments with Jensen–Shannon divergence, which is given by the formula

$$J(\boldsymbol{A},\boldsymbol{B}) = \sum_{ij} \boldsymbol{A}_{ij} \log\left(\frac{2\boldsymbol{A}_{ij}}{\boldsymbol{A}_{ij}+\boldsymbol{B}_{ij}}\right) + \boldsymbol{B}_{ij} \log\left(\frac{2\boldsymbol{B}_{ij}}{\boldsymbol{A}_{ij}+\boldsymbol{B}_{ij}}\right) . \tag{3.7}$$

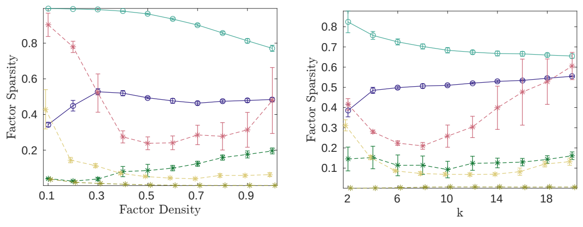
It is easy to see that (3.7) is an additive function, and hence can be plugged into `Cancer`. Figure 3.8 shows how this version of `Cancer` compares to other methods. The setup is the same as in the corresponding experiments in Figure 3.6. In all these experiments it is apparent that this version of `Cancer` is inferior to that optimizing the Frobenius error, but is generally on par with `SVD` and NMF-based methods. Also for the varying density test (Figure 3.8(b)) it produces better reconstruction errors than `SVD` and all the NMF methods, until the density reaches 50 %, after which they become tied.

Prediction. In this experiment we choose a random holdout set and remove it from the data (elements of this set are marked as missing values). We then try to learn the structure of the data from its remaining part using the algorithms, and finally test how well they predict the values inside the holdout set. The factors are drawn uniformly at random from the set of integers in an interval $[0, a]$ with



(a) Varying noise with density 50 %.

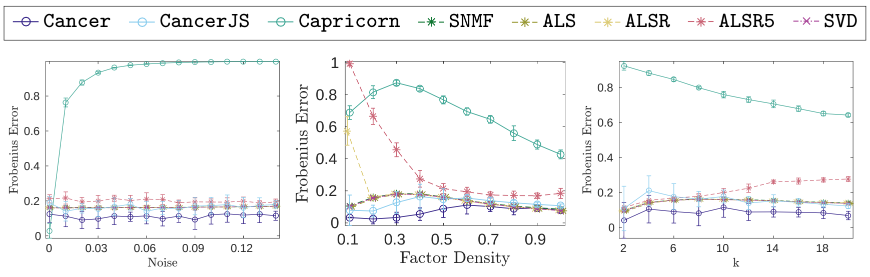
(b) Varying density with Gaussian noise of 0.01.



(c) Varying density with Gaussian noise of 0.08.

(d) Varying rank; 50 % density and Gaussian noise of 0.01.

Figure 3.7: Sparsity (fraction of zeroes) of the factor matrices for synthetic data with Gaussian noise. The markers are averages of 10 random matrices and the width of the error bars is twice the standard deviation.

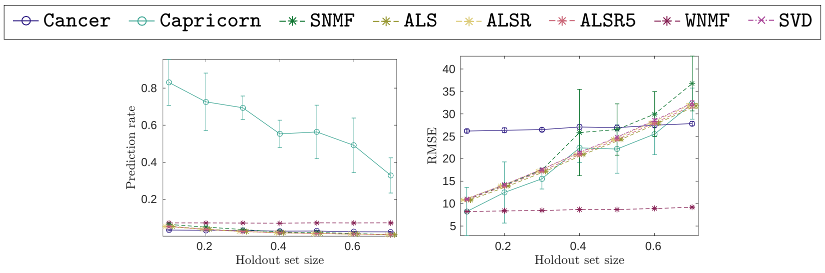


(a) Varying noise test.

(b) Varying density test.

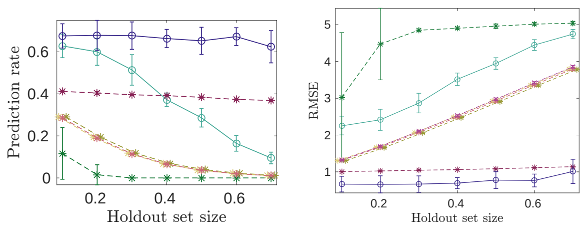
(c) Varying rank test with 10 % noise and 30 % factor density.

Figure 3.8: Comparison of `Cancer` with Jensen–Shannon objective and other methods on synthetic data with Gaussian noise. x-axis is the parameter varied and y-axis is the relative Frobenius error. All results are averages over 10 random matrices and the width of the error bars is twice the standard deviation.



(a) Prediction rate, range [0, 100].

(b) RMSE, range [0, 100].



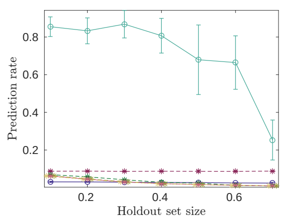
(c) Prediction rate, range [0, 9].

(d) RMSE, range [0, 9].

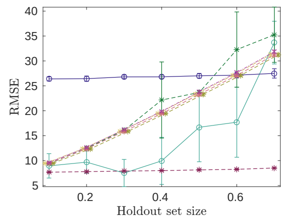
Figure 3.9: Prediction rate on synthetic data with tropical noise. The x-axis represents the size of the holdout set. The y-axis is the correct prediction rate in Figures 3.9(a) and 3.9(c), and RMSE in Figures 3.9(b) and 3.9(d). The range is the interval that the values in input matrices are restricted to. All results are averages over 10 random matrices and the width of the error bars is twice the standard deviation.

a predefined density of $30\,\%$, and then multiplied using the subtropical matrix product. We use two different values of a for each experiment, 10 and 3. With $a = 10$ input matrices have values in the range $[0, 100]$, and when $a = 3$, the range is $[0, 9]$. We then apply noise to the obtained matrices and feed them to the algorithms. Since all input matrices are integer-valued, and since the recovered data produced by the algorithms can be continuous-valued, we round it to the nearest integer. We report two measures of the prediction quality – prediction rate, which is defined as the fraction of correctly guessed values in the hold-out set, and root mean square error (RMSE). We tested this setup with both tropical noise (Figure 3.9) and Gaussian noise (Figure 3.10).

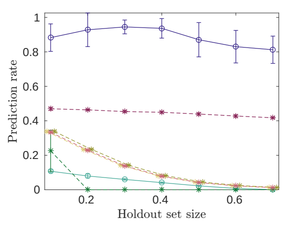
`Capricorn` gives by far the best prediction rate when using the higher $[0, 100]$ range of values in input matrices (Figures 3.9(a) and 3.10(a)). Especially interesting is that it also beats all other methods in the presence of Gaussian noise. In terms of RMSE it generally lands somewhere in the middle of the pack among various NMF methods. Such a large difference between these measures is caused by `Capricorn` not really being an approximation algorithm. It extracts subtropical patterns where they exist, while ignoring parts of the data where they cannot be found. This results in it either predicting the integer values exactly or missing by a wide margin. With the $[0, 9]$ range of values the results of `Capricorn` become worse, which is especially evident with Gaussian noise. Although this behaviour might seem counterintuitive, it is simply a consequence



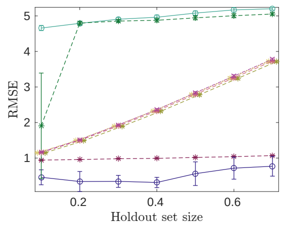
(a) Prediction rate, range [0, 100].



(b) RMSE, range [0, 100].



(c) Prediction rate, range [0, 9].



(d) RMSE, range [0, 9].

Figure 3.10: Prediction rate on synthetic data with Gaussian noise. The x-axis represents the size of the holdout set. The y-axis is the correct prediction rate in Figures 3.10(a) and 3.10(c), and RMSE in Figures 3.10(b) and 3.10(d). The range is the interval that the values in input matrices are restricted to. All results are averages over 10 random matrices and the width of the error bars is twice the standard deviation.

of noise having a larger effect when values in the data are smaller. Cancer shows the opposite behaviour to Capricorn in that it benefits from smaller value range and Gaussian noise, where it consistently outperforms all other methods. Unlike Capricorn, Cancer approximates values in input data, which allows it to get a high number of hits with the $[0, 9]$ range after the rounding. On the $[0, 100]$ interval though, it is liable to guessing many values incorrectly since a much higher level of precision is required. For many prediction tasks, like predicting user ratings, Cancer's approach seems more useful as input values are usually drawn from a relatively small range (for example, in Movielens, all ratings are from $[0, 5]$). Other competing methods generally do not perform well, with the exception of SVD winning the first place with RMSE measure for the high range experiments (Figures 3.9(b) and 3.10(b)). It illustrates once again that SVD is a good approximation method but does not help its prediction accuracy. In all other experiments the first place is held by either Capricorn or Cancer. As a general guideline, when choosing between Capricorn and Cancer for value prediction, one should consider that Cancer usually gives a superior performance, while Capricorn tends to be better for exact guessing of values having a wider range.

Discussion. The synthetic experiments confirm that both Capricorn and Cancer are able to recover the max-times structure. The main practical difference between them is that Capricorn is designed to handle the tropical (flipping) noise, while Cancer is meant for the data that is perturbed with white (e.g. Gaussian) noise. While Capricorn is clearly the best method when the data has only the flipping noise – and is capable of tolerating very high noise levels – its results deteriorate when we apply Gaussian noise. Hence, when the exact noise type is not known a priori, it is advisable to try both methods. It is also important to note that Cancer is not restricted to optimizing the Frobenius matrix norm and can optimize various objectives. To demonstrate that, we performed experiments with Jensen–Shannon divergence as objective and obtained results that, while inferior to Cancer that optimizes the Frobenius error, are still slightly better than the rest of the algorithms. Overall, we can conclude that SVD and NMF-based methods generally cannot recover the structure from subtropical data, that is, we cannot use existing methods as a substitute to find the max-times structure.

3.4.3 Real-world experiments

The main purpose of the real-world experiments is to study to which extent Capricorn and Cancer can find max-times structure from various real-world data sets. Having established with the synthetic experiments that both algorithms are capable of finding the structure when it is present, here we look at what kind of results they obtain in the real-world data.

It is probably unrealistic to expect real-world data sets to have the "pure" max-times structure, as in the synthetic experiments. Rather, we expect SVD to be the best method (in reconstruction error's sense), and our algorithms to obtain reconstruction error comparable to the NMF-based methods. We will also verify that the results from the real-world data sets are intuitive.

The datasets

Bas1LP represents a linear program.[3] It is available from the University of Florida Sparse Matrix Collection[4] [Davis and Hu, 2011].

Trec12 is a brute force disjoint product matrix in tree algebra on n nodes.[5] It can be obtained from the same repository as Bas1LP.

Worldclim contains weather records for Europe between 1960 and 1990 and was obtained from the global climate data repository.[6] The data has $2\,575$ rows that correspond to 50-by-50 kilometer squares of land where measurements were made and 48 columns corresponding to observations. More precisely, the first 12 columns represent the average low temperature for each month, the next 12 columns the average high temperature, and the next 12 columns the daily mean. The last 12 columns represent the mean monthly precipitation for each month. We preprocessed every column of the data by first subtracting its mean, dividing by the standard deviation, and then subtracting its minimum value, so that the smallest value becomes 0.

[3]Submitted to the matrix repository by Csaba Meszaros.
[4]http://www.cise.ufl.edu/research/sparse/matrices/, accessed 18 July 2017
[5]Submitted by Nicolas Thiery.
[6]The raw data is available at http://www.worldclim.org/, accessed 18 July 2017.

NPAS is a nerdiness personality test that uses different attributes to determine the level of nerdiness of a person.[7] It contains answers by 1418 respondents to a set of 36 questions that asked them to self-assess various statements about themselves on a scale of 1 to 7. We preprocessed the input matrix in a manner similar to Worldclim, only this time subtracting the smallest value of the entire matrix instead of doing it column-wise.

Eigenfaces is a subset of the Extended Yale Face collection of face images [Georghiades et al., 2000]. It consists of 32-by-32 pixel images under different lighting conditions. We used a preprocessed data by Xiaofei He et al.[8] We selected a subset of pictures with lighting from the left and then preprocessed the input matrix analogously to Worldclim, except without subtracting the standard deviation.

4News is a subset of the 20Newsgroups dataset,[9] containing the usage of 800 words over 400 posts for 4 newsgroups.[10] Before running the algorithms we represented the dataset as a TF-IDF matrix, and then scaled it by dividing each entry by the greatest entry in the matrix.

HPI is a land registry house price index.[11] Rows represent months, columns are locations, and entries are residential property price indices. This data set was prepared in the same way as Eigenfaces.

Movielens is a collection of user ratings for a set of movies. The original dataset[12] consists of 100 000 ratings from 1000 users on 1700 movies, with ratings ranging from 1 to 5. In order to be able to perform cross-validation on it, we had to preprocess Movielens by removing users that rated fewer than 10 movies and movies that were rated less than 5 times. After that we were left with 943 users, 1349 movies and 99 287 ratings.

Mammals is a matrix whose rows and columns correspond to locations in Europe, and for every column-row pair, the corresponding entry represents the degree to which the sets of mammals inhabiting them overlap. This dataset[13] was obtained from the original binary location-species matrix [see Mitchell-Jones et al., 1999] by multiplying it with its transpose and then normalizing by dividing each column by its maximal element. Due to the special nature of this dataset, we use it only to illustrate the effects of factor overlap, and do not report any other results.

The basic properties of these data sets are listed in Table 3.1.

Quantitative results: reconstruction error, sparsity, convergence, and runtime

The following experiments are meant to test `Cancer` and `Capricorn`, and how they compare to other methods, such as `SVD` and NMF. Table 3.2 provides the

[7]The dataset can be obtained on the online personality website `http://personality-testing.info/_rawdata/NPAS-data.zip`, accessed 18 July 2017.

[8]`http://www.cad.zju.edu.cn/home/dengcai/Data/FaceData.html`, accessed 18 July 2017

[9]`http://qwone.com/~jason/20Newsgroups/`, accessed 18 July 2017

[10]The authors are grateful to Ata Kabán for pre-processing the data, see Miettinen [2009].

[11]Available at https://data.gov.uk/dataset/land-registry-house-price-index-background-tables/, accessed 18 July 2017

[12]Available at `http://grouplens.org/datasets/movielens/100k/`, accessed 18 July 2017

[13]Available for research purposes from the Societas Europaea Mammalogica at `http://www.european-mammals.org`

Table 3.1: Real world datasets properties.

Dataset	Rows	Columns	Density
Bas1LP	9825	5411	1.1 %
Trec12	2726	551	10.0 %
Worldclim	2575	48	99.9 %
NPAS	1418	36	99.6 %
Eigenfaces	1024	222	97.0 %
4News	400	800	3.5 %
HPI	253	177	99.5 %
Movielens	943	1349	7.8 %
Mammals	2670	2670	91 %

relative Frobenius reconstruction errors for various real-world data sets, as well as ranks used for factorization.[14] Since there is no ground truth for these datasets, the ranks are chosen based mainly on the size of the data and our intuition on what the true rank should be. `SVD` is, as expected, consistently the best method, followed by `WNMF` and `SNMF`. `Cancer` generally lands in the middle of the pack of the NMF methods, which suggests that it is capable of finding max-times structure that is comparable to what NMF-based methods provide. Consequently, we can study the max-times structure found by `Cancer`, knowing that it is (relatively) accurate. On the other hand, `Capricorn` has a high reconstruction error. The discrepancy between `Cancer`'s and `Capricorn`'s results indicates that the datasets used cannot be represented using 'the 'pure" subtropical structure. Rather, they are either a mix of NMF and subtropical patterns or have relatively high levels of continuous noise.

The sparsity of the factors for real-world data is presented in Table 3.3 (we do not include the sparsities for `SVD` and `WNMF` as they were all 0). Here, `Cancer` often returns the second-sparsest factors (behind only `Capricorn`), but with 4News and HPI, `ALSR` and `ALSR5` obtains sparser decompositions.

We also studied the convergence behavior of `Cancer` using some of the real-world data sets. The results can be seen in Figure 3.11, where we plot the relative error with respect to the number of iterations over the main for-loop in `Cancer`. As we can see, in both cases `Cancer` has obtained a good reconstruction error already after a few full cycles, with the remaining runs only providing minor improvements. We can deduce that `Cancer` quickly reaches an acceptable solution.

To give some idea about the speed performance of the algorithms, we ran each of the competing methods on some of the real-world datasets. The runtime of each algorithm (in seconds) is shown in Table 3.4, where we report its mean value and the standard deviation averaged over 5 runs. All tests were performed on a Linux machine with Intel Xeon E5530 CPU with 16 2.40 GHz cores, although `Cancer` and `Capricorn` were only utilizing one core. As we can see, the simplest methods, such as `SVD` and `ALS`, are also the fastest, while

[14]The values are different than those presented by Karaev and Miettinen [2016b] because we used Frobenius error instead of L_1 and counted all elements towards the error, not just nonnegative ones.

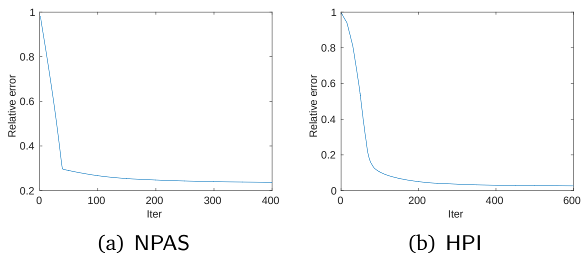


Figure 3.11: Convergence rate of Cancer for two real-world datasets. Each iteration is a single run of UpdateBlock, that is if a factorization has rank k, then one full cycle would correspond to k iterations.

more involved algorithms, such as Cancer or SNMF, take much longer to run. It is worth noting that Cancer and Capricorn are written in Matlab, and their performance can potentially be significantly improved by implementing time critical parts in C or another low-level programming language.

Prediction

Here we investigate how well both Capricorn and Cancer can predict missing values in the data. We used three real-world datasets, a user-movie rating matrix Movielens, a brute force disjoint product matrix in tree algebra Trec12, and Bas1LP, that represents a linear program. All these matrices are integer valued, and hence we will also round the results of all methods to the nearest integer. We compare the results of our methods against WNMF and SVD. The choice of WNMF is motivated by its ability to ignore missing elements in the input data and its generally good performance on the previous tests. There is only one caveat: WNMF sometimes produces very high spikes for some elements in the matrix. They do not cause too much problem with prediction, but they seriously deteriorate the results of WNMF with respect to various distance measures. For this reason we always ignore such elements when computing various measures of prediction quality. While this comparison method is obviously not completely fair towards other methods, it can serve as a rough upper bound for what performance is possible with NMF-based algorithms. Comparing against other methods is obviously not fair as they are not designed to deal with missing values, but we will still present the results of SVD for completeness.

On Movielens we perform standard cross-validation tests, where a random selection of elements is chosen as a holdout set and removed from the data. The data has 943 users, each having rated from 19 to 648 movies. A holdout set is chosen by sampling uniformly at random 5 ratings from each user. We run the algorithms, while treating the elements from the holdout set as missing values, and then compare the reconstructed matrices to the original data. This procedure is repeated 10 times.

To get a more complete view on how good the predictions are, we report various measures of quality: Frobenius error, root mean square error (RMSE), reciprocal rank, Spearman's ρ, mean absolute error (MAE), Jensen–Shannon divergence (JS), optimistic reciprocal rank, Kendall's τ, and prediction accuracy.

The prediction accuracy allows us to see if the methods are capable of recovering the missing user ratings. The remaining tests can be divided into two categories. The first one, which comprises Frobenius error, root mean square error, mean absolute error, and Jensen–Shannon divergence, aims to quantify the distance between the original data and the reconstructed matrix. The second group of tests finds the correlation between the rankings of movies for each user. It includes Spearman's ρ, Kendall's τ, reciprocal rank, and optimistic reciprocal rank. All these measures are well known, with perhaps only the reciprocal rank requiring some explanation. Let us first denote by U the set of all users. In the following, for each user $u \in U$ we only consider the set of movies $M(u)$ that this user has rated that belong to the holdout set. The ratings by user u induce a natural ranking on $M(u)$. On the other hand, the algorithms produce approximations $r'(u,m)$ to the true ratings $r(u,m)$, which also induce a corresponding ranking of the movies. The reciprocal rank is a convenient way of comparing the rankings obtained by the algorithms to the original one. For any user $u \in U$, denote by $H(u)$ a set of movies that this user ranked the highest (that is $H(u) = \{m \in M(u) : r(u,m) = \max_{m' \in M(u)} r(u,m')\}$). The reciprocal rank for user u is now defined as

$$RR(u) = \frac{1}{\min\limits_{m \in H} R(u,m)} , \tag{3.8}$$

where $R(u,m)$ is the rank of the movie m within $M(u)$ according to the rating approximations given by the algorithm in question. Now the mean reciprocal rank is defined as the average of the reciprocal ranks for each individual user $MRR = \frac{1}{|U|} \sum_{u \in U} RR(u)$. When computing the ranks $R(u,m)$, all tied elements receive the same rank, which is computed by averaging. That means that if, say, movies m_1 and m_2 have tied ranks of 2 and 3, then they both receive the rank of 2.5. An alternative way is to always assign the smallest possible rank. In the above example both m_1 and m_2 will receive rank 2. When ranks $R(u,m)$ are computed like this, the equation (3.8) defines the optimistic reciprocal rank.

For each test, Table 3.5 shows the mean and the standard deviation of the results of each algorithm. In addition we report the p-value based on the Wilcoxon signed-rank test. It shows if an advantage of one method over another is statistically significant. We say that a method A is significantly better than method B if the p-value is < 0.05. It is unreasonable to report the p-value for every method pair – instead we only show p-values involving the best method. For each method, the value given next to it is the p-value for this method and the best method.

`Cancer` is significantly better for the Frobenius error, root mean square error, mean absolute error, Jensen–Shannon divergence, and accuracy. For the remaining tests the results are less clear, with `Cancer` winning on the reciprocal rank, `Capricorn` taking the optimistic reciprocal rank, and `WNMF` being better on Spearman's ρ and Kendall's τ tests. It should be noted though, that the victories of `WNMF` on Spearman's ρ and Kendall's τ tests, as well as `Cancer`'s on the reciprocal rank, are not statistically significant as the p-values are quite high. In summary, our experiments show that `Cancer` is significantly better in tests that measure the direct distance between the original and the reconstructed matrices, as well as the prediction accuracy, whereas for the ranking experiments it is difficult to give any of the algorithms an edge.

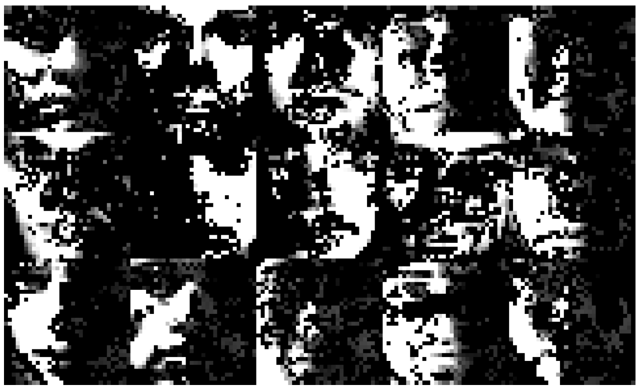
For Trec12 and Bas1LP we also perform cross-validation, where on each fold we take $10\,\%$ of the nonzero values in the data as a holdout set and then try to predict them. We repeat this process 5 times. Unlike Movielens, Trec12 and Bas1LP datasets are not so readily interpretable as they were generated from abstract mathematical data structures. Ranking, in particular, makes no sense, and hence we do not perform any ranking associated experiments. The results for Trec12 are shown in Table 3.6 and for Bas1LP in Table 3.7. It is apparent that on Trec12 WNMF performs significantly better than any other method, being better in all metrics. As discussed earlier, however, that should be taken with a grain of salt as we ignore the elements where WNMF produced unreasonably large values. Without this preprocessing its results are much worse than those of Cancer. This presents evidence, although not conclusive, that the Trec12 dataset has less subtropical structure than Movielens. The p-value is 0.004 for all metrics, which is the result of a particular number of folds (5) that we used. The fact that we have this number everywhere in the table simply indicates that WNMF was better than any other method on every fold with respect to all measures. With Bas1LP the roles reverse, and this time Cancer is clearly the best method, winning according to all metrics and on all folds, just as WNMF did on Trec12.

Interpretability of the results

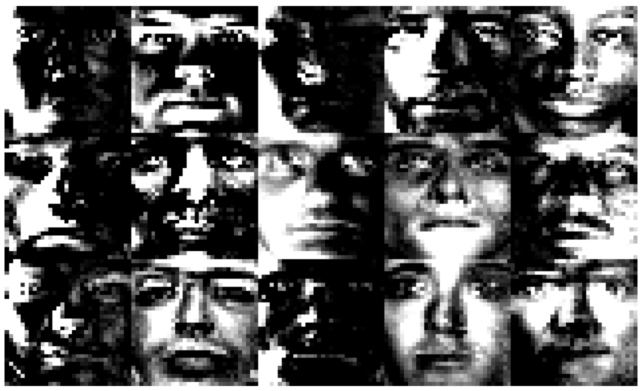
The crux of using max-times factorizations instead of standard (nonnegative) ones is that the factors (are supposed to) exhibit the "winner-takes-it-all" structure instead of the "parts-of-whole" structure. To demonstrate this, we analysed results in four different datasets: Eigenfaces, NPAS, Worldclim, and Mammals.

We plotted the left factor matrices for the Eigenfaces data for Cancer and ALS in Figure 3.12. At first, it might look like ALS provides more interpretable results, as most factors are easily identifiable as faces. This, however, is not a very interesting result: we already knew that the data has faces, and many factors in the ALS's result are simply some kind of "prototypical" faces. The results of Cancer are harder to interpret at first sight. Upon closer inspection, though, one can see that they identify areas that are lighter in different images, that is, have higher grayscale values. These factors tell us the variances in the lighting in different photos, and can reveal information we did not know a priori. In addition almost every one of Cancer's factors contains one or two main feature of the face (such as nose, left eye, right cheek, etc.). In other words, while NMF's patterns are for the most part close to fully formed faces, Cancer finds independent fragments that indicate the direction of the lighting and (or) contain some of the main features of a face.

In order to interpret NPAS, we first observe that each column represents a single personality attribute. Denote by $\boldsymbol{A}$ the obtained approximation of the original matrix. For each rank-1 factor $\boldsymbol{X}$ and each column $\boldsymbol{A}_i$ we define the score $\sigma(i)$ as the number of elements in $\boldsymbol{A}_i$ that are determined by $\boldsymbol{X}$. By sorting attributes in descending order of $\sigma(i)$, we obtain relative rankings of the attributes for a given factor. The results are shown in Table 3.8. The first factor clearly shows



(a) Cancer



(b) ALS

Figure 3.12: Cancer finds the dominant patterns from the Eigenfaces data. Pictured are the left factor matrices for the Eigenfaces data.

introverted tendencies, while the second one can be summarized as having interests in fiction and games.

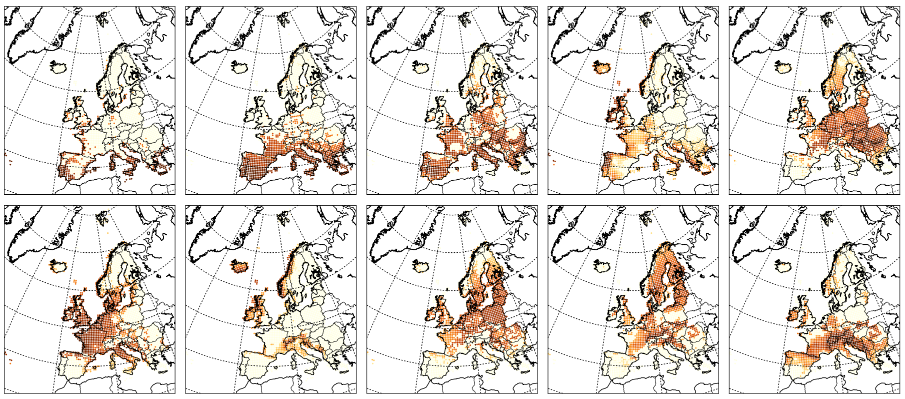
For the Worldclim dataset Figures 3.13(a) and 3.13(b) show left-hand sides of factors found by Cancer and WNMF (the best NMF-method in Table 3.2), plotted on the map. Darker colours indicate higher values, which can be interpreted as "more important". The right-hand side factors are presented in Figures 3.13(c) and 3.13(d), respectively. Here each row corresponds to a factor, and each column to a single observation column from the original data (that is columns 1–12 represent average low temperatures for each month, columns 13–24 average high temperatures, columns 25–36 daily means, and columns 37–48 average monthly precipitation). Again, higher values can be seen as having more importance. Recall that a pattern is formed by taking an outer product of a single left-hand factor and the corresponding right-hand factor. It is easy to see that largest (and thus the most important) values in a pattern are those that are products of high values in both right-hand side and left-hand side factors.

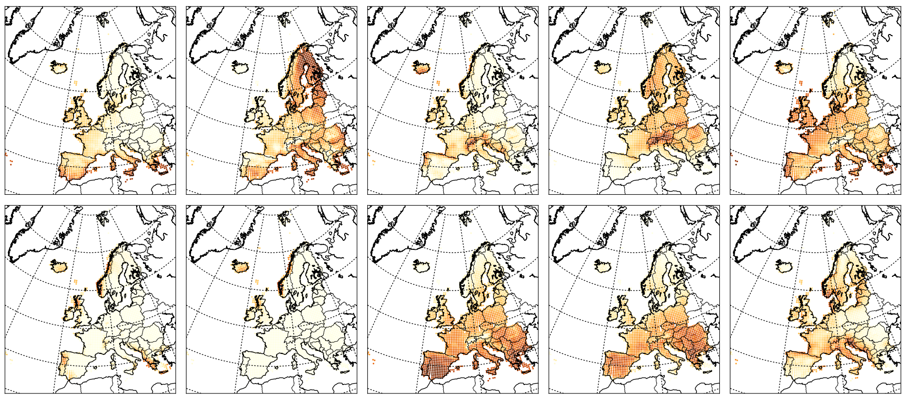
It is evident that Cancer's factors have more large values. This highlights the difference between the subtropical and the normal algebra: in the normal algebra, if you sum two large values, the result would be even greater, whereas with the subtropical algebra it is equal to the largest summand. In decompositions this means that `WNMF` cannot have high overlapping values in the factors; instead it has to split its factors to mostly non-overlapping parts. `Cancer`, on the other hand, can have overlap, and hence its factors can share some phenomena. We will illustrate this by a more detailed analysis of two weather aspects: precipitation and daily maximum temperatures in summer (i.e. June, July, and August). To be able to validate the results of the algorithms, we also include the average annual precipitation and average maximum summer temperature in Figures 3.17(a) and 3.17(b), respectively.

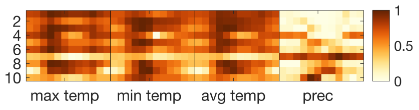
Both `Cancer` and `WNMF` identify the areas of high precipitation, and their corresponding left-hand factors are shown in Figures 3.16(a) and 3.16(c), respectively. There are, however, two significant differences between their interpretations. First, `Cancer` emphasizes the wettest areas, while `WNMF` shows a much smoother transition, similar to the original data. The second difference is that, unlike `Cancer`, `WNMF` does not identify either the UK or Norwegian coast as areas of (particularly) high precipitation. A potential explanation is that there are many overlaps with other factors (see Figure 3.13(b)), and hence having large values in any of them might lead to overcovering the original data. `Cancer`, as a subtropical method, does not suffer from this issue, as its results are entirely based on factors with highest values.

In order to make the above argument more concrete, let us see what happens when we try to combine `Cancer`'s factors using the standard algebra instead of the subtropical one. Recall that if $\boldsymbol{A} = \boldsymbol{BC}$ is a rank-k matrix decomposition of $\boldsymbol{A}$, then we have $(\boldsymbol{A})_{ij} = \sum_{s=1}^{k} (\boldsymbol{F}_s)_{ij}$, where each pattern $\boldsymbol{F}_s$ is an outer product of the sth column of $\boldsymbol{B}$ and the sth row of $\boldsymbol{C}$. If for some l and t we have $(\boldsymbol{F}_l)_{ij} + (\boldsymbol{F}_t)_{ij} > \boldsymbol{A}_{ij}$, then also $(\boldsymbol{BC})_{ij} > \boldsymbol{A}_{ij}$ since all values are nonnegative. It is therefore generally undesirable for any subset of the patterns to overcover values in the original data, as there would be no way of decreasing these values by adding more patterns. As an example, we will combine the patterns corresponding to `Cancer`'s factors from Figures 3.16(a)–(b). To obtain the actual rank-1 patterns we first need to compute the outer products of these factors with the corresponding rows of the right-hand side matrix. Now if we denote the obtained patterns by $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$, then the elements of the matrix $\max\{\boldsymbol{F}_1 + \boldsymbol{F}_2 - \boldsymbol{A}, 0\}$ show by how much the combination of $\boldsymbol{F}_1$ and $\boldsymbol{F}_2$ overcovers the original data $\boldsymbol{A}$. We now plot the average value of every row of this "overcover" matrix, scaled by the average value in the original data (Figure 3.18(b)). Since each row corresponds to a location on the map, it shows the average amount by which we would overcover the data, were we to use the standard algebra for combining the `Cancer`'s factors. It is evident that this method produces many values that are too high (mostly around Alps and other high precipitation areas). On the other hand, when we perform the same procedure using the subtropical algebra (Figure 3.18(a)), there is almost no overcovering.

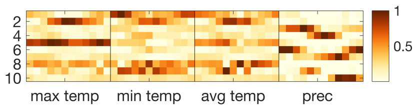
It is worth mentioning that, although the UK and the coastal regions of Norway are not prominent in the `WNMF`'s factor shown above, they actually belong to some of its other factors (see Figure 3.13(b)). In other words, the high


(a) `Cancer` left-hand factors.


(b) `WNMF` left-hand factors.



(c) `Cancer` right-hand factors.



(d) `WNMF` right-hand factors.

Figure 3.13: `Cancer` factors in the Worldclim data. The factor vectors are normalized to take values from the unit interval and darker shades indicate higher values.

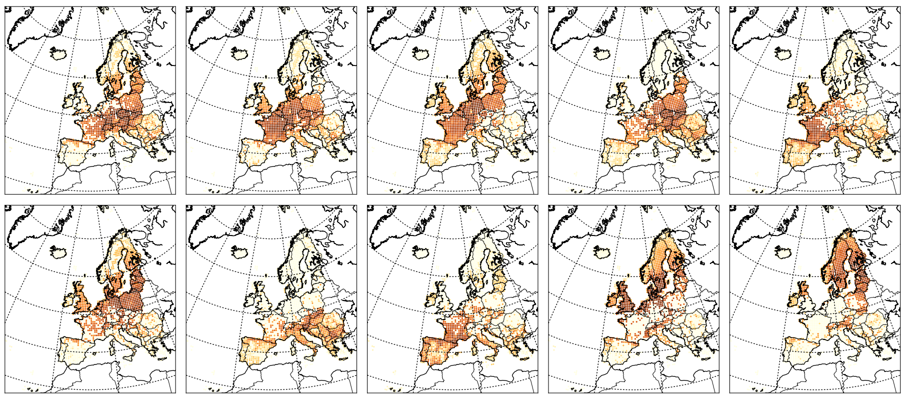


Figure 3.14: Factors by `Cancer` in the Mammals data plotted on a map. Every factor is normalized to take values from the unit interval and darker shades indicate higher values.

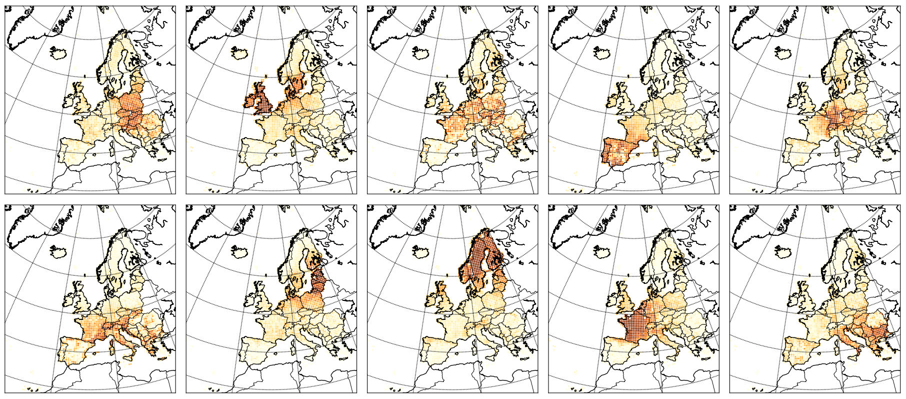
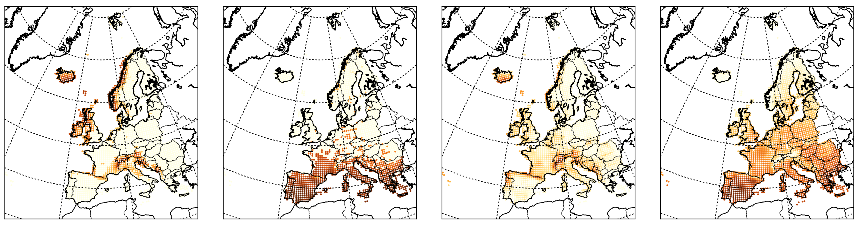


Figure 3.15: Factors by `WNMF` in the Mammals data plotted on a map. Every factor is normalized to take values from the unit interval and darker shades indicate higher values.



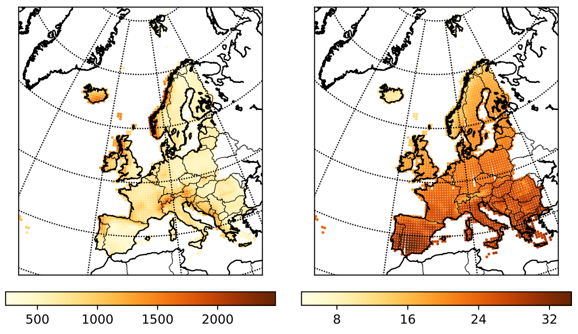
(a) `Cancer` high precipitation (factor 7)

(b) `Cancer` hot summer days (factor 2)

(c) `WNMF` high precipitation (factor 3)

(d) `WNMF` maximum daily temperature in summer (factor 8)

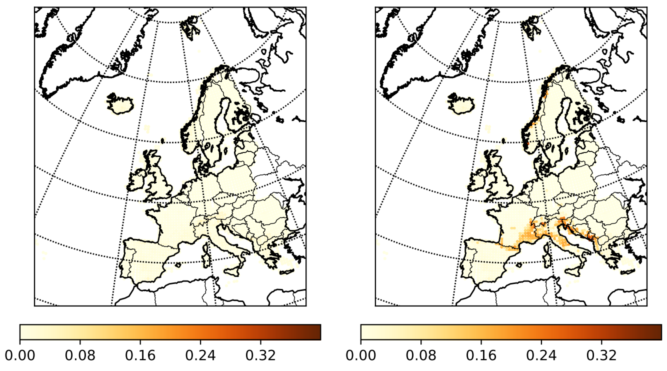
Figure 3.16: Example results by `Cancer` and `WNMF` on the Worldclim dataset. For each method, two selected columns from the left-hand factor matrix are shown on a map. The values are normalized to the unit interval, and darker shades indicate higher values.



(a) Average annual precipitation in mm.

(b) Average daily maximum temperature in summer in degrees Celsius.

Figure 3.17: Average climate data to be compared with the factors in Figure 3.16.



(a) Overcovering when using the subtropical algebra.

(b) Overcovering when using the standard algebra.

Figure 3.18: Comparison of the overcovering when combining the `Cancer` factors from Figures 3.16(a)–(b) using subtropical (3.18(a)) and the standard (3.18(b)) algebras. All values are divided by the average value in the original matrix, negative values are ignored. Darker shades indicate higher values.

precipitation pattern is split into several parts and partially merged with other factors.

A somewhat similar behaviour is seen with the maximal daily temperatures in summer. WNMF finds a factor that, with the exception of the Scandinavian peninsula, closely mimics the original data, and maintains a smooth gradient of decreasing temperatures when moving towards the north (Figure 3.16(d)). In contrast, Cancer identifies the areas where summer days are hot, while practically disregarding other parts (Figure 3.16(b)).

Many of the differences between the subtropical matrix factorization and NMF in the discussion above were a consequence of how they handle factor overlap. While overlapping factors are not always desired, in some applications they can be seen as almost necessary. Consider, for example, the factors obtained by Cancer with the Mammals data (Figure 3.14). We can see that many of them cover the central parts of the European Plain, extending a bit south to cover most of Germany. There are, naturally, many mammal species that inhabit the whole European Plain, and the east–west change is gradual. This gradual change is easier to model in the subtropical algebra, as we do not have to worry about the sums of the factors getting too large. Factors 1–6 model various aspects of the east–west change, emphasizing either the south–west, central, or eastern parts of the plain. Similarly, the ninth factor explains mammal species found in the UK and southern Scandinavia, while the tenth factor covers species found in Scotland, Scandinavia, and the Baltic countries, indicating that these areas have roughly the same biome. If we compare these results to those of WNMF (Figure 3.15), then it becomes evident that the latter tries to find relatively disjoint factors and avoids factor overlap whenever possible. This is because in NMF any feature that is nonzero at a given data point is always "active", in the sense that it contributes to the final value. That being said, WNMF does find some interesting patterns, such as rather distinct factors representing France and the Scandinavian peninsula.

While the above discussion shows that the subtropical model can be a useful complement to NMF, it is generally difficult to claim that either of them is superior. For example Cancer generally provided a more concise representation of patterns in the climate data, outlining its most prominent properties, while WNMF's strength was recovering the smooth transition between values.

Table 3.2: Reconstruction error for various real-world datasets.

	Worldclim	NPAS	Eigenfaces	4News	HPI	Movielens	Trec12	Bas1LP
$k =$	10	10	40	20	15	10	25	25
`Cancer`	0.071	0.240	0.204	0.556	0.027	0.756	0.864	0.813
`Capricorn`	0.392	0.395	0.972	0.987	0.217	1.003	0.998	0.912
`SNMF`	0.046	0.225	0.178	0.546	0.023	0.745	0.841	0.749
`ALS`	0.087	0.227	0.313	0.538	0.074	0.749	0.828	0.733
`ALSR`	0.122	0.226	0.294	1.000	0.045	0.748	0.827	0.733
`ALSR5`	0.081	0.233	0.291	1.000	0.063	0.748	0.826	0.733
`WNMF`	0.034	0.221	0.169	0.545	0.021	0.741	0.824	0.733
`SVD`	**0.025**	**0.209**	**0.140**	**0.533**	**0.015**	**0.728**	**0.802**	**0.722**

Table 3.3: Factor sparsity for various real-world datasets.

	Worldclim	NPAS	Eigenfaces	4News	HPI	Movielens	Trec12	Bas1LP
$k =$	10	10	40	20	15	10	25	25
Cancer	0.645	0.528	0.571	0.812	0.422	0.666	0.838	0.951
Capricorn	**0.795**	**0.733**	**0.949**	0.991	0.685	**0.957**	**0.988**	**0.978**
SNMF	0.383	0.330	0.403	0.499	0.226	0.543	0.758	0.738
ALS	0.226	0.120	0.434	0.513	0.331	0.420	0.573	0.634
ALSR	0.275	0.117	0.480	**1.000**	**0.729**	0.438	0.681	0.748
ALSR5	0.549	0.189	0.648	**1.000**	0.622	0.481	0.743	0.811

Table 3.4: Average runtime (in seconds) of the algorithms for various real-world datasets. The results were calculated based on 5 restarts of each method, and standard deviations are reported with the ± sign. Bold indicates the smallest average runtime.

	Worldclim	NPAS	4News	HPI
Cancer	20116.000 ± 15.14	6023.000 ± 25.00	25520.000 ± 60.44	924.000 ± 8.00
Capricorn	205.870 ± 1.39	87.000 ± 1.30	165.960 ± 7.12	41.000 ± 0.72
SNMF	115.100 ± 0.53	72.000 ± 1.50	195.570 ± 1.76	64.000 ± 0.51
ALS	0.194 ± 0.08	0.374 ± 0.14	3.649 ± 1.45	0.156 ± 0.12
ALSR	**0.187** ± 0.02	0.280 ± 0.04	4.684 ± 0.74	0.309 ± 0.13
WNMF	2.240 ± 0.20	1.201 ± 0.11	5.164 ± 0.87	1.288 ± 0.10
SVD	0.598 ± 0.04	**0.155** ± 0.04	**0.142** ± 0.04	**0.027** ± 0.01

Table 3.5: Comparison between the predictive power of different methods on the Movielens data. The arrow after the value indicates whether higher or lower values are preferable. The p-values are computed using the Wilcoxon signed-rank test.

	Frobenius		RMSE	
	value(↓)	p-value	value(↓)	p-value
`Cancer`	$\mathbf{0.2876} \pm 0.003$		$\mathbf{1.0802} \pm 0.011$	
`Capricorn`	0.6993 ± 0.024	0.0001	2.6267 ± 0.085	0.0001
`WNMF`	0.2989 ± 0.003	0.0001	1.1227 ± 0.012	0.0001
`SVD`	0.7336 ± 0.002	0.0001	2.7558 ± 0.014	0.0001
	Recip. rank		Spearman's ρ	
	value(↑)	p-value	value(↑)	p-value
`Cancer`	$\mathbf{0.7451} \pm 0.010$		0.3071 ± 0.015	0.5749
`Capricorn`	0.5601 ± 0.017	0.0001	0.2354 ± 0.017	0.0001
`WNMF`	0.7395 ± 0.004	0.0521	$\mathbf{0.3084} \pm 0.012$	
`SVD`	0.7217 ± 0.008	0.0004	0.2445 ± 0.013	0.0001
	MAE		JS	
	value(↓)	p-value	value(↓)	p-value
`Cancer`	$\mathbf{0.8203} \pm 0.008$		$\mathbf{0.0201} \pm 0.000$	
`Capricorn`	2.0518 ± 0.106	0.0001	0.2826 ± 0.026	0.0001
`WNMF`	0.8555 ± 0.008	0.0001	0.0209 ± 0.000	0.0057
`SVD`	2.4756 ± 0.014	0.0001	0.1153 ± 0.001	0.0001
	Recip. rank opt.		Kendall's τ	
	value(↑)	p-value	value(↑)	p-value
`Cancer`	0.7451 ± 0.010	0.0001	0.2659 ± 0.013	0.4251
`Capricorn`	$\mathbf{0.8547} \pm 0.010$		0.2127 ± 0.016	0.0001
`WNMF`	0.7395 ± 0.004	0.0001	$\mathbf{0.2679} \pm 0.010$	
`SVD`	0.7217 ± 0.008	0.0001	0.2111 ± 0.012	0.0001
	Accuracy			
	value(↑)	p-value		
`Cancer`	$\mathbf{0.3968} \pm 0.008$			
`Capricorn`	0.2053 ± 0.019	0.0001		
`WNMF`	0.3828 ± 0.006	0.0011		
`SVD`	0.0588 ± 0.003	0.0001		

Table 3.6: Comparison between the predictive power of different methods on the Trec12 data. The arrow after the value indicates whether higher or lower values are preferable. The p-values are computed using the Wilcoxon signed-rank test.

	Frobenius		RMSE	
	value($\downarrow$)	p-value	value($\downarrow$)	p-value
`Cancer`	0.4824 ± 0.016	0.0040	2.3124 ± 0.085	0.0040
`Capricorn`	0.7827 ± 0.023	0.0040	3.7521 ± 0.131	0.0040
`WNMF`	$\mathbf{0.4374} \pm 0.006$		$\mathbf{2.0925} \pm 0.041$	
`SVD`	0.6005 ± 0.003	0.0040	2.8784 ± 0.032	0.0040
	MAE		**JS**	
	value($\downarrow$)	p-value	value($\downarrow$)	p-value
`Cancer`	1.5852 ± 0.050	0.0040	0.0675 ± 0.005	0.0040
`Capricorn`	2.3871 ± 0.099	0.0040	0.2929 ± 0.028	0.0040
`WNMF`	$\mathbf{1.2138} \pm 0.011$		$\mathbf{0.0367} \pm 0.000$	
`SVD`	1.8413 ± 0.013	0.0040	0.0786 ± 0.001	0.0040
	Accuracy			
	value($\uparrow$)	p-value		
`Cancer`	0.2315 ± 0.010	0.0040		
`Capricorn`	0.1918 ± 0.019	0.0040		
`WNMF`	$\mathbf{0.3996} \pm 0.004$			
`SVD`	0.2061 ± 0.002	0.0040		

Table 3.7: Comparison between the predictive power of different methods on the Bas1LP data. The arrow after the value indicates whether higher or lower values are preferable. The p-values are computed using the Wilcoxon signed-rank test.

	Frobenius		RMSE	
	value(↓)	p-value	value(↓)	p-value
`Cancer`	$\mathbf{0.3690} \pm 0.018$		$\mathbf{1.1462} \pm 0.065$	
`Capricorn`	0.5741 ± 0.054	0.0040	1.7822 ± 0.161	0.0040
`WNMF`	0.4113 ± 0.014	0.0040	1.2748 ± 0.038	0.0040
`SVD`	0.5003 ± 0.002	0.0040	1.5534 ± 0.019	0.0040
	MAE		JS	
	value(↓)	p-value	value(↓)	p-value
`Cancer`	$\mathbf{0.3286} \pm 0.014$		$\mathbf{0.0228} \pm 0.001$	
`Capricorn`	0.6712 ± 0.094	0.0040	0.1208 ± 0.037	0.0040
`WNMF`	0.3932 ± 0.006	0.0040	0.0268 ± 0.000	0.0040
`SVD`	0.9391 ± 0.006	0.0040	0.0919 ± 0.000	0.0040
	Accuracy			
	value(↑)	p-value		
`Cancer`	$\mathbf{0.8841} \pm 0.002$			
`Capricorn`	0.7111 ± 0.050	0.0040		
`WNMF`	0.8562 ± 0.001	0.0040		
`SVD`	0.2837 ± 0.002	0.0040		

Table 3.8: Top three attributes for the first two factors of NPAS.

Factor 1	Factor 2
I am more comfortable with my hobbies than I am with other people	I have played a lot of video games
I gravitate towards introspection	I collect books
I sometimes prefer fictional people to real ones	I care about super heroes

CHAPTER 4

Mixed Model: Combining Nonnegative and Subtropical Matrix Factorization

As discussed in previous chapters, the subtropical matrix factorization model describes data as a union of rank-1 patterns. This leads to what we call the "winner takes it all" interpretation, as opposed to the standard "parts of a whole", that NMF is known for. This approach allowed us to find locally dominant features, such as climate zones, that are harder to discover using NMF. NMF, on the other hand, is better at recovering smooth transitions from one part of data to another.

It is well known that choosing a model can be hard as much of the real-world data does not follow any particular one exactly. For example, it is perfectly plausible that, even though parts of the data can be well approximated using the NMF's "parts of a whole" model, its other parts have distinct dominant features, that are better described with the "winner takes it all" approach. This last consideration leads to the idea that it may be worthwhile to combine different models, using each one to describe only some parts of the data. This chapter discusses a mixed model that integrates NMF and subtropical matrix factorization, and that automatically determines which model to use on each part of the data.

4.1 The Mixed Linear-Tropical Model

Rather than describing data using NMF or subtropical matrix factorization, here we propose a hybrid model that incorporates them both and allows for a smooth transition between the two. Ideally, given an input matrix $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$, we want to be able to determine what elements $\boldsymbol{A}_{ij}$ are better represented using the standard algebra, and which ones require the subtropical one. Namely, we seek factor matrices $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k\times m}$ and parameters $\boldsymbol{\alpha} \in \mathbb{R}^{n\times m}$ such that

$$\boldsymbol{A}_{ij} \approx f(\boldsymbol{\alpha}_{ij})(\boldsymbol{B} \boxtimes \boldsymbol{C}) + g(\boldsymbol{\alpha}_{ij})(\boldsymbol{B}\boldsymbol{C})_{ij} \ , \tag{4.1}$$

for some functions f and g that we will define later. By representing $\boldsymbol{A}$ as a "mixture" of the normal and subtropical products of the factor matrices, we allow for more flexibility in fitting the data. Since by altering the parameter matrix $\boldsymbol{\alpha}$ we can set different mixing coefficients for different elements of $\boldsymbol{A}$, it is possible to better explain data that has piecewise NMF and piecewise subtropical structure. Moreover, since the functions $f(\boldsymbol{\alpha}_{ij})$ and $g(\boldsymbol{\alpha}_{ij})$ do not have to be restricted to binary values, we can also express some elements $\boldsymbol{A}_{ij}$ as a weighted sum of $(\boldsymbol{B} \boxtimes \boldsymbol{C})_{ij}$ and $(\boldsymbol{BC})_{ij}$.

It is important to note that equation (4.1) is quite general, and unless we impose restrictions on functions f and g, as well as the matrix $\boldsymbol{\alpha}$, our model will overfit the data. When it comes to choosing the proper functions f and g, there is a trade-off between fitting the data and keeping the model simple. We will use the convex combination $f(\boldsymbol{\alpha}_{ij}) = \boldsymbol{\alpha}_{ij}, g(\boldsymbol{\alpha}_{ij}) = 1 - \boldsymbol{\alpha}_{ij}, \boldsymbol{\alpha}_{ij} \in [0,1]$, which is very simple, and at the same time provides an intuitive transition from the standard product at $\boldsymbol{\alpha}_{ij} = 0$ to the subtropical product at $\boldsymbol{\alpha}_{ij} = 1$. We obtain

$$\boldsymbol{A}_{ij} \approx \boldsymbol{\alpha}_{ij}(\boldsymbol{B} \boxtimes \boldsymbol{C}) + (1 - \boldsymbol{\alpha}_{ij})(\boldsymbol{BC})_{ij} \tag{4.2}$$

for $\boldsymbol{\alpha}_{ij} \in [0,1]$.

When choosing $\boldsymbol{\alpha}$ we are faced with a similar trade-off. Indeed, without additional constraints, we can fit arbitrarily complex matrices with *constant* factor matrices, as the following proposition illustrates.

Proposition 9. *Let $\boldsymbol{A} \in [1,2]^{n \times m}$ and let $k = 4$. There exists $\boldsymbol{\alpha} \in [0,1]^{n \times m}$, $\boldsymbol{B} \in \mathbb{R}_+^{n \times k}$, and $\boldsymbol{C} \in \mathbb{R}_+^{k \times m}$ such that all entries of $\boldsymbol{B}$ and $\boldsymbol{C}$ are the same and that $\boldsymbol{A}_{ij} = \boldsymbol{\alpha}_{ij}(\boldsymbol{B} \boxtimes \boldsymbol{C}) + (1 - \boldsymbol{\alpha}_{ij})(\boldsymbol{BC})_{ij}$ for all $i = 1, \ldots, n$ and $j = 1, \ldots, m$.*

Proof. Let all entries of $\boldsymbol{B}$ and $\boldsymbol{C}$ be $\sqrt{3}/2$. Then for any $1 \leq i \leq n$ and $1 \leq j \leq m$ we have $(\boldsymbol{B} \boxtimes \boldsymbol{C})_{ij} = 3/4$ and $(\boldsymbol{BC})_{ij} = 3$. Now if we set

$$\boldsymbol{\alpha}_{ij} = \frac{\boldsymbol{A}_{ij} - (\boldsymbol{BC})_{ij}}{(\boldsymbol{B} \boxtimes \boldsymbol{C})_{ij} - (\boldsymbol{BC})_{ij}} , \tag{4.3}$$

then $0 \leq \boldsymbol{\alpha}_{ij} \leq 1$ holds. By plugging (4.3) into (4.2), we obtain $\boldsymbol{\alpha}_{ij}(\boldsymbol{B} \boxtimes \boldsymbol{C})_{ij} + (1 - \boldsymbol{\alpha}_{ij})(\boldsymbol{BC})_{ij} = \boldsymbol{A}_{ij}$, concluding the proof. □

Being able to decompose essentially arbitrary matrices into constant factor matrices shows that unrestricted $\boldsymbol{\alpha}$ can have too much power. To constrain $\boldsymbol{\alpha}$, we force it to have essentially a tropical rank-1 structure:

$$\boldsymbol{\alpha}_{ij} = \sigma(\boldsymbol{\theta}_i + \boldsymbol{\phi}_j) , \tag{4.4}$$

where $\boldsymbol{\theta} \in \mathbb{R}^{n \times 1}$ and $\boldsymbol{\phi} \in \mathbb{R}^{1 \times m}$ are arbitrary vectors, and $\sigma(x) = 1/(1 + e^{-x})$ is the sigmoid function.

Now, given factors $\boldsymbol{B} \in \mathbb{R}_+^{n \times k}$, $\boldsymbol{C} \in \mathbb{R}_+^{k \times m}$ and parameter vectors $\boldsymbol{\theta} \in \mathbb{R}^{n \times 1}$ and $\boldsymbol{\phi} \in \mathbb{R}^{1 \times m}$, we can define their mixed linear–tropical product, $\boldsymbol{B} \boxtimes_{\boldsymbol{\theta},\boldsymbol{\phi}} \boldsymbol{C}$, elementwise as

$$(\boldsymbol{B} \boxtimes_{\boldsymbol{\theta},\boldsymbol{\phi}} \boldsymbol{C})_{ij} = \boldsymbol{\alpha}_{ij}(\boldsymbol{B} \boxtimes \boldsymbol{C})_{ij} + (1 - \boldsymbol{\alpha}_{ij})(\boldsymbol{BC})_{ij} , \tag{4.5}$$

where $\boldsymbol{\alpha}_{ij} = \sigma(\boldsymbol{\theta}_i + \boldsymbol{\phi}_j)$.

It is trivial to see that when elements in both $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ tend to $-\infty$, we have $\boldsymbol{B} \boxtimes_{\boldsymbol{\theta},\boldsymbol{\phi}} \boldsymbol{C} \to \boldsymbol{BC}$. Conversely, the greater the sum $\boldsymbol{\theta}_i + \boldsymbol{\phi}_j$ is, the closer the corresponding element in the mixed product is to the subtropical product. In the limit, when all $\boldsymbol{\theta}_i + \boldsymbol{\phi}_j$ tend to ∞, we have $\boldsymbol{B} \boxtimes_{\boldsymbol{\theta},\boldsymbol{\phi}} \boldsymbol{C} \to \boldsymbol{B} \boxtimes \boldsymbol{C}$.

We can interpret the values in parameter vectors $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ to give the "typical" level of NMF or subtropical structure associated with the corresponding rows and columns. If, for example, $\boldsymbol{\theta}_i \ll 0$, it means that row i has a strong NMF-type structure, while $\boldsymbol{\theta}_i \gg 0$ would mean a strongly subtropical structure. If $\boldsymbol{\theta}_i \approx 0$, then the structure is an even mixture of the two. Similarly, if $\boldsymbol{\theta}_i + \boldsymbol{\phi}_j \gg 0$, then the element $\boldsymbol{A}_{ij}$ has a subtropical structure, and vice versa for $\boldsymbol{\theta}_i + \boldsymbol{\phi}_j \ll 0$. This interpretation also explains why we use the tropical rank-1 model, that is, summation, instead of the standard rank-1 model $\boldsymbol{\theta}\boldsymbol{\phi}^T$: if we calculate the product, we cannot interpret negative values of $\boldsymbol{\theta}_i$ or $\boldsymbol{\phi}_j$ as indicative of a "typically NMF" structure, as if both $\boldsymbol{\theta}_i, \boldsymbol{\phi}_j < 0$, then $\boldsymbol{\theta}_i\boldsymbol{\phi}_j > 0$, indicating a subtropical structure.

Now we can define the main problem considered in this chapter.

Problem 8. Given an input matrix $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$ and an integer $k > 0$, find two factor matrices $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k\times m}$ and parameter vectors $\boldsymbol{\theta} \in \mathbb{R}^{n\times 1}$ and $\boldsymbol{\phi} \in \mathbb{R}^{1\times m}$ such that

$$E(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{\theta},\boldsymbol{\phi}) = \|\boldsymbol{A} - \boldsymbol{B} \boxtimes_{\boldsymbol{\theta},\boldsymbol{\phi}} \boldsymbol{C}\|_F \tag{4.6}$$

is minimized.

Unfortunately it seems that the optimization of the above problem is hard:

Proposition 10. *Given $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$, k, $\boldsymbol{\theta}$, and $\boldsymbol{\phi}$, finding $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k\times m}$ that minimize $E(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{\theta},\boldsymbol{\phi})$ is* NP*-hard. It is also* NP*-hard to find $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k\times m}$ that approximate $E(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C},\boldsymbol{\theta},\boldsymbol{\phi})$ to within any polynomially computable factor.*

The proposition is a direct consequence of the NP-hardness of computing or approximating NMF [Vavasis, 2009] or subtropical matrix factorization (see eg. Corollary 3).

4.2 Algorithm (`Latitude`)

We propose a new algorithm, `Latitude` (Algorithm 9), that finds a mixed linear–tropical matrix decomposition of the given input data. As input it accepts the data matrix $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$, the rank of the sought decomposition $k \in \mathbb{N}$, and an integer parameter $N \in \mathbb{N}$ that determines the number of iterations of the algorithm. It returns the computed factors $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k\times m}$, and parameter vectors $\boldsymbol{\theta} \in \mathbb{R}^{n\times 1}$ and $\boldsymbol{\phi} \in \mathbb{R}^{1\times m}$. `Latitude` has one additional parameter, $M \in \mathbb{R}_+$. Each element in $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ must belong to the $[-M, M]$ interval. In practice, very high values in the parameter vectors do not make sense due to the use of the sigmoid function (see (4.4)) – they would get "smoothed out" and make only marginal changes to the parameter matrix $\boldsymbol{\alpha}$. For this reason for all experiments in this chapter we used $M = 5$, at which

Algorithm 9 `Latitude`

```
    Input: $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$, $k \in \mathbb{N}$, $N \in \mathbb{N}$
    Output: $\boldsymbol{B}^* \in \mathbb{R}_+^{n\times k}$, $\boldsymbol{C}^* \in \mathbb{R}_+^{k\times m}$, $\boldsymbol{\theta}^* \in \mathbb{R}^{n\times 1}$, $\boldsymbol{\phi}^* \in \mathbb{R}^{1\times m}$
    Parameters: $M$ ▷ The maximum possible value of parameter vectors. In practice 5
    is a good choice
 1: function Latitude($\boldsymbol{A}, k, N$)
 2:     initialize $\boldsymbol{B}$ and $\boldsymbol{C}$
 3:     $\boldsymbol{D} \leftarrow \boldsymbol{BC} - \boldsymbol{A}$
 4:     $\boldsymbol{f}_i \leftarrow \sum_{j=1}^m \boldsymbol{D}_{ij}, \boldsymbol{g}_j \leftarrow \sum_{i=1}^n \boldsymbol{D}_{ij}$
 5:     $\boldsymbol{s}_i \leftarrow$ index of the $i$-th smallest element of $\boldsymbol{f}$
 6:     $\boldsymbol{t}_j \leftarrow$ index of the $j$-th smallest element of $\boldsymbol{g}$
 7:     $\boldsymbol{\theta}_i \leftarrow \frac{i-n}{n-1}M$
 8:     $\boldsymbol{\phi}_j \leftarrow \frac{j-m}{m-1}M$
 9:     $\boldsymbol{B}^* \leftarrow \boldsymbol{B}, \boldsymbol{C}^* \leftarrow \boldsymbol{C}$                ▷ Initialize best factors.
10:     $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}, \boldsymbol{\phi}^* \leftarrow \boldsymbol{\phi}$             ▷ Initialize best parameters.
11:     $bestError \leftarrow \|\boldsymbol{A} - \boldsymbol{B} \boxtimes_{\boldsymbol{\theta},\boldsymbol{\phi}} \boldsymbol{C}\|_F$
12:     for $iter \leftarrow 1$ to $N$ do
13:         for $j \leftarrow 1$ to $m$ do
14:             $[\boldsymbol{C}^j, \boldsymbol{\phi}_j] \leftarrow$ MixReg($\boldsymbol{A}^j, \boldsymbol{B}, \boldsymbol{C}^j, \boldsymbol{\theta}, \boldsymbol{\phi}_j, M$)
15:         end for
16:         for $i \leftarrow 1$ to $n$ do
17:             $[\boldsymbol{B}_i, \boldsymbol{\theta}_i] \leftarrow$ MixReg($\boldsymbol{A}_i^T, \boldsymbol{C}^T, \boldsymbol{B}_i^T, \boldsymbol{\phi}, \boldsymbol{\theta}_i, M$)
18:         end for
19:         if $\|\boldsymbol{A} - \boldsymbol{B} \boxtimes_{\boldsymbol{\theta},\boldsymbol{\phi}} \boldsymbol{C}\|_F < bestError$ then
20:             $\boldsymbol{B}^* \leftarrow \boldsymbol{B}, \boldsymbol{C}^* \leftarrow \boldsymbol{C}$
21:             $\boldsymbol{\theta}^* \leftarrow \boldsymbol{\theta}, \boldsymbol{\phi}^* \leftarrow \boldsymbol{\phi}$
22:             $bestError \leftarrow \|\boldsymbol{A} - \boldsymbol{B} \boxtimes_{\boldsymbol{\theta},\boldsymbol{\phi}} \boldsymbol{C}\|_F$
23:         end if
24:     end for
25:     return $\boldsymbol{B}^*, \boldsymbol{C}^*, \boldsymbol{\theta}^*, \boldsymbol{\phi}^*$
26: end function
```

point $\sigma(M) = 0.9933$, and there is almost nothing to be gained by increasing M further.

The main idea of `Latitude` is to repeatedly use a routine that solves the linear–tropical regression problem to alternatingly update the factor matrices and the parameter vectors. Namely, when the factor matrix $\boldsymbol{B}$ and the parameter vector $\boldsymbol{\theta}$ are fixed, finding the other factor matrix $\boldsymbol{C}$ and parameter vector $\boldsymbol{\phi}$ reduces to solving the problem

$$[\boldsymbol{C}^j, \boldsymbol{\phi}_j] \leftarrow \operatorname*{arg\,min}_{\boldsymbol{c}\in\mathbb{R}_+^{k\times 1},\, s\in[-M,M]} \|\boldsymbol{A}^j - \boldsymbol{B} \boxtimes_{\boldsymbol{\theta},s} \boldsymbol{c}\|_F \tag{4.7}$$

m times (once per each column of $\boldsymbol{C}$). Then we fix $\boldsymbol{C}$ and $\boldsymbol{\phi}$ and do the same for $\boldsymbol{B}$ and $\boldsymbol{\theta}$. This process is repeated N times. The algorithm starts by initializing factor matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ (line 2). This can be done by using random matrices, or, for example, by using some NMF algorithm. Starting with a "pure" NMF solution gives us a reasonable initial solution, and we use that initialization in our experiments. The updates to the factors and parameters are done inside the main loop (lines 12–24), where line 14 updates $\boldsymbol{C}$ and $\boldsymbol{\phi}$, and line 17 updates $\boldsymbol{B}$ and $\boldsymbol{\theta}$. On each iteration we check if the current solution $\boldsymbol{B}$, $\boldsymbol{C}$, $\boldsymbol{\theta}$, $\boldsymbol{\phi}$ improves

Algorithm 10 `MixReg`

```
   Input: $\boldsymbol{a} \in \mathbb{R}_+^{n\times 1}$, $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$, $\boldsymbol{c} \in \mathbb{R}_+^{k\times 1}$, $\boldsymbol{\theta} \in \mathbb{R}^{n\times 1}$, $t \in \mathbb{R}$, $M > 0$
   Output: $\boldsymbol{c} \in \mathbb{R}_+^{k\times 1}$, $t \in \mathbb{R}$
1: function MixReg($\boldsymbol{a}, \boldsymbol{B}, \boldsymbol{c}, \boldsymbol{\theta}, t, M$)
2:     $\boldsymbol{X}_i \leftarrow \boldsymbol{B}_i \,.*\, \boldsymbol{c}^T$
3:     $\boldsymbol{\alpha} \leftarrow \sigma(\boldsymbol{\theta} + t)$
4:     $\boldsymbol{T}_{ij} \leftarrow \begin{cases} 1 & j = \arg\max_{1\le s\le k} \boldsymbol{X}_{is} \\ 1 - \boldsymbol{\alpha}_i & \text{otherwise} \end{cases}$
5:     $\boldsymbol{Y} \leftarrow \boldsymbol{B} \,.*\, \boldsymbol{T}$
6:     $\boldsymbol{c} \leftarrow \arg\min_{\boldsymbol{p}\in\mathbb{R}_+^{k\times 1}} \|\boldsymbol{a} - \boldsymbol{Y}\boldsymbol{p}\|_F$
7:     $t \leftarrow \arg\min_{s\in[-M,M]} \|\boldsymbol{a} - \boldsymbol{B} \boxtimes_{\boldsymbol{\theta},s} \boldsymbol{c}\|_F$
8:     return $\boldsymbol{c}$, $t$
9: end function
```

on the best one found before that (line 19), and if it does, then we update the best solution and the best error (lines 20–22).

The function `MixReg` (Algorithm 10) solves problem (4.7), and is where the actual updates to the factors and parameters are performed. It takes as input vector $\boldsymbol{a} \in \mathbb{R}_+^{n\times 1}$, the first factor matrix $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$, an initial solution for the output vector $\boldsymbol{c} \in \mathbb{R}_+^{k\times 1}$, the column parameter vector $\boldsymbol{\theta}$, the starting value for the row parameter element t, and the number $M > 0$ that defines the range of values in the parameter vectors. It returns the updated versions of the vector $\boldsymbol{c}$ and the element t. Finding the global minimum of (4.7) with respect to both $\boldsymbol{c}$ and t is hard, and hence we update them separately. In fact, even when the parameter t is fixed, optimizing (4.7) with respect to $\boldsymbol{c}$ is problematic. To see that, let us first rewrite (4.7) for a fixed value of t. It becomes

$$\arg\min_{\boldsymbol{c}\in\mathbb{R}_+^{k\times 1}} \|\boldsymbol{a} - (\sigma(\boldsymbol{\theta}+t)\boldsymbol{B} \boxtimes \boldsymbol{C} + (1-\sigma(\boldsymbol{\theta}+t))\boldsymbol{B}\boldsymbol{c})\|_F \,. \tag{4.8}$$

For every $1 \le i \le n$ denote by $\varphi(i, \boldsymbol{c})$ the index of the largest element in the vector $\boldsymbol{B}_i \,.*\, \boldsymbol{c}^T$, where $.*$ is the element-wise (Hadamard) product. We have

$$\begin{aligned} &\boldsymbol{\alpha}_i \boldsymbol{B}_i \boxtimes \boldsymbol{c} + (1-\boldsymbol{\alpha}_i)\boldsymbol{B}_i\boldsymbol{c} \\ &= \boldsymbol{\alpha}_i \max_s\{\boldsymbol{B}_{is}\boldsymbol{c}_s\} + (1-\boldsymbol{\alpha}_i)\sum_s \boldsymbol{B}_{is}\boldsymbol{c}_s \\ &= \boldsymbol{B}_{i\varphi(i,\boldsymbol{c})}\boldsymbol{c}_{\varphi(i,\boldsymbol{c})} + (1-\boldsymbol{\alpha}_i)\sum_{s\neq\varphi(i,\boldsymbol{c})} \boldsymbol{B}_{is}\boldsymbol{c}_s \,, \end{aligned} \tag{4.9}$$

and hence problem (4.8) is transformed into

$$\arg\min_{\boldsymbol{c}\in\mathbb{R}_+^{k\times 1}} \|\boldsymbol{a} - \boldsymbol{Y}(\boldsymbol{c})\boldsymbol{c}\|_F,\ \boldsymbol{Y}(\boldsymbol{c})_{ij} = \begin{cases} 1 & j = \varphi(i,\boldsymbol{c}) \\ 1-\boldsymbol{\alpha}_i & \text{otherwise} \,. \end{cases} \tag{4.10}$$

If the coefficient matrix $\boldsymbol{Y}(\boldsymbol{c})$ did not depend on $\boldsymbol{c}$, (4.10) would become a standard nonnegative linear regression problem. Unfortunately, the dependence of $\varphi(i, \boldsymbol{c})$ on $\boldsymbol{c}$ is very complex, and hence it is hard to solve (4.10) directly. In order to overcome this obstacle, we use another heuristic, that is we fix the coefficient matrix $\boldsymbol{Y}(\boldsymbol{c})$, and assume it to be independent from $\boldsymbol{c}$. Under these assumptions $\boldsymbol{c}$ can be found using a standard nonnegative linear regression algorithm. We use MATLAB built-in `lsqnonneg` method. The matrix $\boldsymbol{Y}$ is built on lines 2–5, and the vector $\boldsymbol{c}$ is found on line 6. Finally, on line 7 we update the parameter t. This is done using the binary search on the interval $[-M, M]$ for the point where the derivative with respect to t is close to 0.

Computational complexity. Running `Latitude` comprises of executing `NMF` to initialize the factors, and then repeatedly updating them, as well as the parameter vectors, using the `MixReg` routine. For each $i = 1, \dots, n$ and $j = 1, \dots, m$, `MixReg` is called N times. In order to estimate the complexity of `MixReg`, it suffices to consider the case when it is called to update $\boldsymbol{C}$ and $\boldsymbol{\phi}$ as the alternate case is analogous (one just needs to replace n by m). Computing the matrix $\boldsymbol{Y}$ (lines 2–5) and finding t (line 7) take time $O(nk)$ each; the latter one because it is enough to make a finite number of steps of the binary search. Thus, if we denote by $\Gamma(n, k)$ the complexity of solving the nonnegative linear regression problem, then the running time of `MixReg` would be given by $O(nk) + \Gamma(n, k)$. Since we use `NMF` to initialize the factors, the runtime of `Latitude` depends on what NMF algorithm is called. If we denote the complexity of `NMF` by $\Pi(n, m, k)$, then the total complexity of `Latitude` is $Nm(O(nk) + \Gamma(n, k)) + Nn(O(mk) + \Gamma(m, k)) + \Pi(n, m, k) = O(Nnmk) + Nm\Gamma(n, k) + Nn\Gamma(m, k) + \Pi(n, m, k)$.

Using `lsqnonneg` for the nonnegative regression and denoting its average number of iterations by ℓ as above, we have that $\Gamma(n, k) = O(\ell nk^2)$. Using projected ALS algorithm [Cichocki et al., 2009] for the NMF, each iteration takes $O(nk^2 + mk^2 + nmk)$ time, and we denote the expected number of iterations of the `NMF` algorithm by t. With these choices, the total time complexity becomes $O\big(Nk(nm + lnmk) + tk(nk + mk + nm)\big)$. Importantly, this is linear in the dimensions of the input matrix.

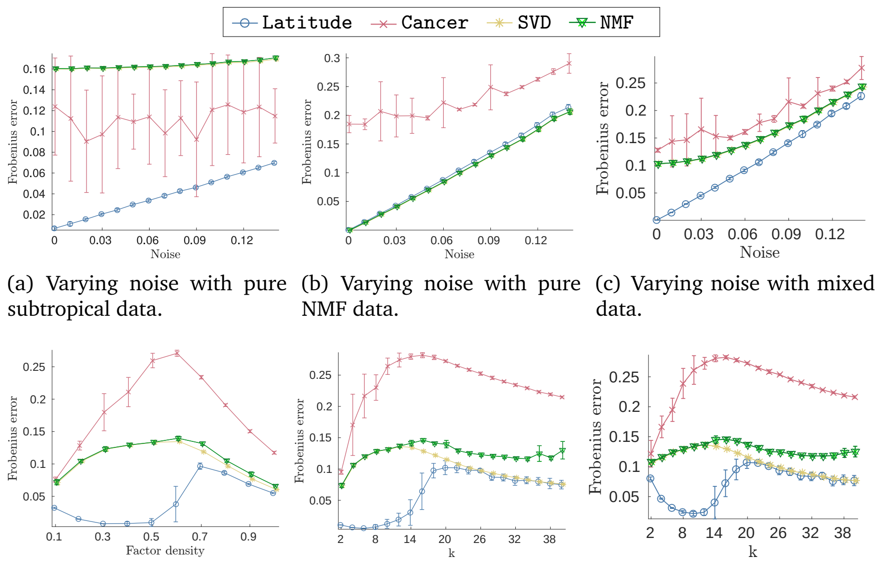
4.3 Experimentimental Evaluation

In this section we test `Latitude` on both synthetic and real-world data, in order to verify how well it can recover mixed tropical-linear structure. We also compare it against various benchmark matrix factorization methods. We make our code freely available for scientific purposes.[1]

4.3.1 Other methods

Since `Latitude` is designed to work with data that has a mixture of NMF and subtropical structures, it is important to compare against algorithms that target them both. There is a multitude of NMF algorithms, but here we will use MATLAB's default implementation, `nnmf`, to which we will refer simply as `NMF`. We will also compare against `SVD` since it is an important benchmark method and provides an optimal rank-k decomposition. Unlike `NMF` or `SVD`, the subtropical matrix factorization is a quite new direction of research, and to the best of our

[1] `https://cs.uef.fi/~pauli/linear-tropical/`



(a) Varying noise with pure subtropical data. (b) Varying noise with pure NMF data. (c) Varying noise with mixed data.

(d) Varying factor density with mixed data. (e) Varying rank with mixed data. (f) Varying rank with mixed data and high level of noise.

Figure 4.1: Reconstruction errors on synthetic data. The x-axis represents the varying parameter and the y-axis the Frobenius error. All results are averages over 10 random matrices and the width of the error bars is twice the standard deviation.

knowledge there are only two available algorithms: `Cancer` and `Capricorn` (see Chapter 3 for their in depth description). `Cancer` is more suitable for our purpose than `Capricorn` due to its ability to handle Gaussian noise, and hence we chose it over `Capricorn`.

4.3.2 Synthetic experiments

The purpose of the synthetic experiments is to verify that the proposed algorithms are actually capable of recovering the sought structure when the data conforms to the mixed tropical-linear model. First we generate the data using the mixed tropical-linear structure, then add some Gaussian noise, and finally run the methods to see how much structure they can recover. Unless stated otherwise, the matrices are of size 1000×800 with true rank 10 and values drawn uniformly at random from the $[0,1]$ interval. The factor density is by default 20%, and the standard deviation of the Gaussian noise is 0.01. In order to make sure that after applying the noise the data remains nonnegative, we truncate all values below 0. The parameter vectors $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are drawn uniformly at random from the $[-5,5]$ interval. For the pure subtropical and NMF structure experiments, we did not use parameters, but rather multiplied the factors directly. The reconstruction error is always measured against the original, noise-free matrix.

Varying noise with pure subtropical data. (Figure 4.1(a)) This experiment tests how well various methods can recover pure subtropical structure, that is, the extreme case of all parameters being set to ∞. The data is generated by multiplying the factors using the subtropical matrix product. We varied the standard deviation of the Gaussian noise from 0 to 0.14 with increments of 0.01. `Latitude` is clearly the best method, followed by `Cancer`, and `NMF` and `SVD` come close together in the last place. The reason why `Latitude` beats `Cancer` on its own kind of data is that it has more leeway in choosing what structure to use, thus being able to fit everywhere where `Cancer` approximates the data well, but also deviate from the pure subtropical model when needed. `NMF` and `SVD` do not seem to find much structure in this experiment. In this and some other experiments `SVD` and `NMF` produce similar reconstruction errors, which sometimes makes their lines hard to distinguish.

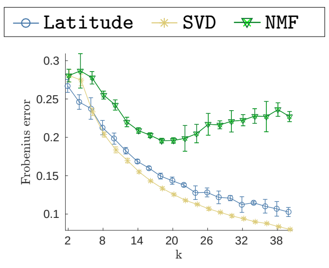
Varying noise with pure NMF data. (Fig. 4.1(b)) This setup is analogous to the previous one, except now the data was generated using the pure NMF structure. Here, `NMF` and `SVD` are performing very well, as is expected as the data is generated with the NMF structure. `Latitude`, although having been initialized by `NMF`, only achieves the same results for zero level of noise – then its results start to slowly deteriorate. The cause of this is that it overfits to the noise. Nevertheless, `Latitude`'s results are not much worse than `NMF` or `SVD`, and hence it is definitely applicable to datasets that exhibit the pure NMF structure. Meanwhile `Cancer` is the worst of the methods, which is expected given that the data has pure NMF rather than subtropical structure.

Varying noise with mixed data. (Fig. 4.1(c)) Here we test the actual mixed model by using parameters drawn uniformly at random from the $[-5, 5]$ interval. This means that the expected value of $\boldsymbol{\theta}_i + \boldsymbol{\phi}_j$ is 0, which corresponds to the midpoint between the NMF and subtropical structures. The randomness ensures that both structures are present in the data. Here `NMF` and `SVD` perform much better than for the pure subtropical case, but `Latitude` is nevertheless the best method by a wide margin, which demonstrates the advantage of combining both models.

Varying factor density with mixed data. (Fig. 5.1(e)) Here we varied the factor density from 10 % to 100 % with increments of 10 %. Again, we have `Latitude` as the best method. There is a peculiar bump on its curve at the very low density level. It can be explained by noise having more influence on sparse data, since then the data/noise ratio is worse.

Varying rank with mixed data. (Fig. 4.1(e)) Here we varied the tropical-linear rank of the data from 2 to 40 with increments of 2. The factor density was kept at 50 %. As in previous experiments, `Latitude` performs considerably better than other algorithms, being clearly the best method for lower ranks and tying with `SVD` when k gets large.

Varying rank with mixed data and a high level of noise. (Figure 4.1(f)) Same setup as above, but with a higher level of noise (standard deviation 0.07). `Latitude` again performs much better than other methods, albeit having a weird bump for lower ranks. Here again it is explained by lower rank data having also lower density, which exacerbates the effect of noise. As in the lower



(a) Varying rank with mixed data.

Figure 4.2: Reconstruction errors for varying k with the subtropical part of the data removed. The x-axis represents k and the y-axis the Frobenius error. All results are averages over 10 random matrices and the width of the error bars is twice the standard deviation.

noise version of this test, `Latitude` is tied with `SVD` for higher ranks. It is worth mentioning that, with the exception of the subtropical data test (Figure 4.1(a)), `Cancer` gives the highest reconstruction error. This is not surprising since it aims at recovering the subtropical structure, which is no more present in the data in its pure form.

Varying rank without the subtropical part. Earlier we have observed that in the varying rank experiments `Latitude` and `SVD` become very close for higher values of k. This inspired a hypothesis that as k grows, the mixed linear-tropical model becomes easier to describe using the standard algebra. Recall that for matrices $\boldsymbol{B} \in \mathbb{R}_+^{n\times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k\times m}$ and parameters $\boldsymbol{\alpha} \in [0,1]^{n\times m}$, the element i, j of the mixed linear-tropical matrix product is given by a convex combination of $(\boldsymbol{B} \boxtimes \boldsymbol{C})_{ij}$ and $(\boldsymbol{BC})_{ij}$

$$(\boldsymbol{B} \boxtimes_{\boldsymbol{\alpha}} \boldsymbol{C})_{ij} = \boldsymbol{\alpha}_{ij}(\boldsymbol{B} \boxtimes \boldsymbol{C})_{ij} + (1-\boldsymbol{\alpha}_{ij})(\boldsymbol{BC})_{ij} \ . \tag{4.11}$$

It is clear that if densities of $\boldsymbol{B}$ and $\boldsymbol{C}$ remain fixed, then as k grows, the second term of (4.11) becomes more and more dominant. This is because on expectation the sum of k elements grows much faster with k than does their maximum. As a result, when all other parameters are fixed, the influence of the tropical term diminishes as the dimensionality grows, and the data becomes more “classical”. That does not mean, however, that the structure becomes NMF-like since all the elements inside the NMF part are still scaled by $\boldsymbol{\alpha}$. To test our conjecture we once again generated data with varying k, but this time without the subtropical term in (4.11). The results are shown in Figure 4.2. It is apparent that `SVD` has improved compared to the normal mixed model, and for $k > 8$ it gives better reconstruction errors than `Latitude`. It is worth noting that this experiment was made to verify our hypothesis and does not follow the model that `Latitude` is designed to solve. As expected, `NMF` does not perform well – scaling by the parameter $\boldsymbol{\alpha}$ seems to destroy the structure it is looking for. We did not include `Cancer` in this experiment due to its generally poor results for non-tropical data and slow performance for higher ks.

Table 4.1: Reconstruction error for real-world datasets.

	Worldclim	NPAS	Eigenfaces	4News	HPI
$k =$	10	10	40	20	15
`Latitude`	**0.023**	**0.207**	0.157	0.536	0.016
`Lat.trunc.`	0.025	0.213	0.158	0.541	0.017
`SVD`	0.025	0.209	**0.140**	**0.533**	**0.015**
`NMF`	0.080	0.223	0.302	0.541	0.124
`Cancer`	0.066	0.237	0.205	0.554	0.026

4.3.3 Real-world experiments

Now that we have evidence that `Latitude` can extract mixed tropical-linear structure when it is present in the data, we want to see if this kind of structure is also present "in the wild". For that we ran all the competing algorithms on various real-world datasets. First we provide a numerical comparison of their results, such as the reconstruction error and runtime, and then show some example results. The description of the datasets used can be found in Section 3.4.3.

Numerical experiments. The reconstruction errors for all the real-world experiments are shown in Table 4.1. `Latitude` and `SVD` are competing for first place, with `Latitude` having the best reconstruction error in 2 datasets and `SVD` in 3. All other methods fall considerably behind. It is worth mentioning that `SVD` has an advantage in that it its factors are not restricted to nonnegative values. One can also argue that `Latitude` has more degrees of freedom due to having one additional dimension of parameters. For this reason we also test a truncated version, called `Lat.trunc.`, that was run with $k - 1$ dimensions. It is still the third best method (after `SVD` and `Latitude`), beating both `NMF` and `Cancer` by a wide margin. Given these results we can conclude that the mixed tropical-linear structure is present in the datasets that we tested, and that `Latitude` is an appropriate algorithm to extract this structure.

Table 4.2 shows the execution time of `Latitude` and the benchmark algorithms for all datasets used in this section. Although it is evident that both `SVD` and `NMF` are much faster, it is worth noting that `Latitude` is an iterative algorithm, and its objective improvements tend to become smaller over time. In many cases it produces reasonably high quality results after only a few iterations, which can be used to save execution time. Figure 4.3 demonstrates the convergence rate of `Latitude`.

Interpretation. In order to validate that our approach provides meaningful results, we study the results with Worldclim and Eigenfaces in more detail. We used the ranks from Table 4.1. Since NMF is used for climate models [see e.g. Paatero and Tapper, 1994], we expect this data to have mostly NMF structure, but certain phenomena, such as rainfall, and certain areas, such as mountains or coastal sites, could very well have a more subtropical structure. In order to check this intuition, we can study the parameter vectors $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ and the matrix $\boldsymbol{\theta} + \boldsymbol{\phi} = \left(\boldsymbol{\theta}_i + \boldsymbol{\phi}_j\right)_{ij}$. For the Worldclim data these are depicted in Figure 4.4.

Table 4.2: Runtime in seconds for real-world datasets.

	Worldclim	NPAS	Eigenfaces	4News	HPI
$k =$	10	10	40	20	15
`Latitude`	60.59	30.58	148.40	52.28	10.90
`Lat.trunc.`	57.67	28.98	143.89	49.20	11.58
`SVD`	1.43	**0.25**	**0.15**	**0.15**	**0.05**
`NMF`	**1.11**	0.45	2.49	1.62	0.13
`Cancer`	36574	11070	48476	10445	785

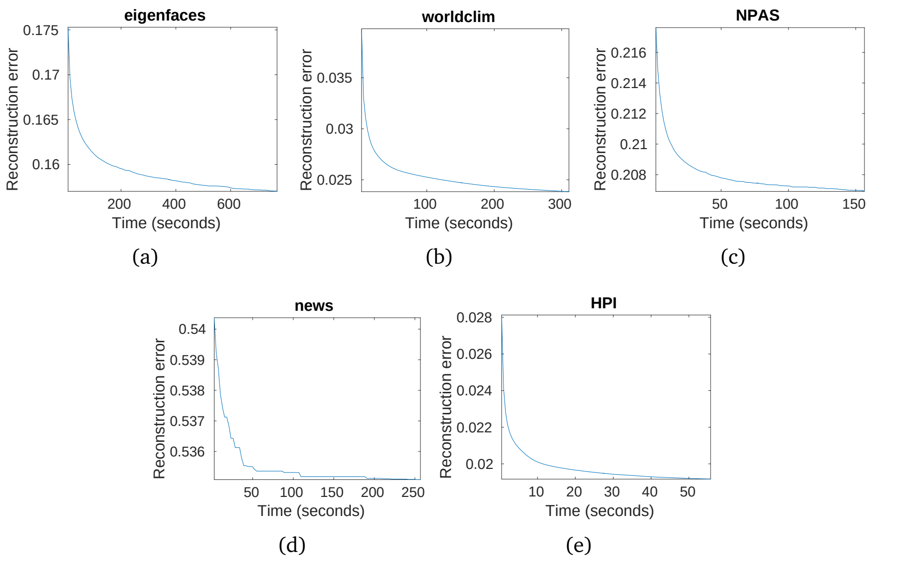
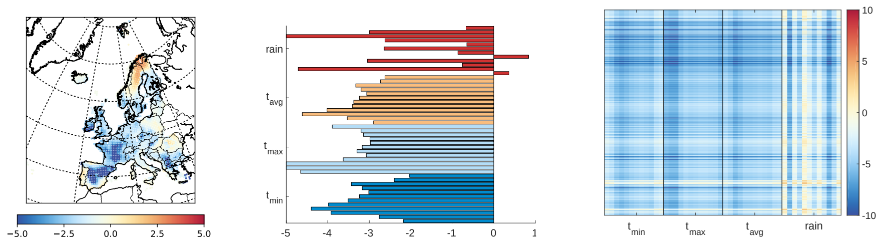


Figure 4.3: Reconstruction error of `Latitude` as a function of time for real-world datasets.

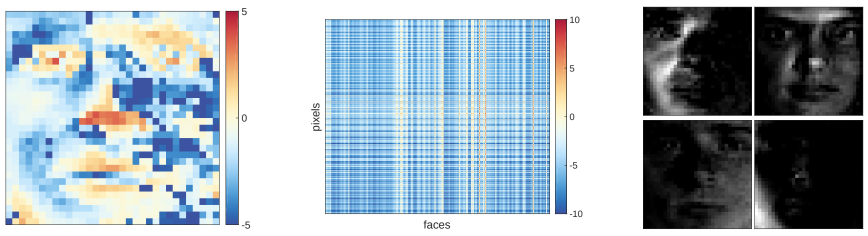
Recall that negative values of $\boldsymbol{\theta} + \phi$ indicate an NMF-type structure, while positive values indicate a subtropical-type structure. Vector $\boldsymbol{\theta}$ corresponds to geographical locations, and its values are plotted on a map in Figure 4.4(a). As we expected, most of the data has an NMF-type structure (depicted as blue), but especially Lapland, Portugal, and some mediterranean coastlines have a more subtropical-type structure. These areas probably have some dominating climate phenomena, for example, heavy rainfall or low temperatures, that is best explained using subtropical structure. Vector ϕ corresponds to the climate variables. The values in ϕ are shown in Figure 4.4(b), where we can see that most variables are negative, that is, they have NMF-type structure. Precipitation is an exception, as the precipitation variables for January and May are in fact positive, indicating a more subtropical-type structure. The complete parameter matrix $\boldsymbol{\theta} + \phi$ is shown in Figure 4.4(c). Most elements in the factorization have a medium to strong NMF-type structure, but there exist also elements with a more subtropical-type structure.

The vector $\boldsymbol{\theta}$ for the Eigenfaces data corresponds to pixels and is depicted in



(a) Vector $\boldsymbol{\theta}$ (b) Vector $\boldsymbol{\phi}$ (c) Matrix $\boldsymbol{\theta} + \boldsymbol{\phi}$

Figure 4.4: Visualizations for the parameters in the decomposition of Worldclim. (a) Values in vector $\boldsymbol{\theta}$ plotted in a map. (b) Values in vector $\boldsymbol{\phi}$ shown as a bar plot. The variables are divided in four groups of twelve months corresponding to minimum, maximum, and average temperature, and precipitation ($\mathrm{t_{min}}$, $\mathrm{t_{max}}$, $\mathrm{t_{avg}}$, and rain, respectively). January is always at the bottom. (c) The matrix $\boldsymbol{\theta} + \boldsymbol{\phi} = \left(\boldsymbol{\theta}_i + \boldsymbol{\phi}_j\right)_{i,j}$. Columns are divided in four groups of twelve months, as in (b). January is always at the left.



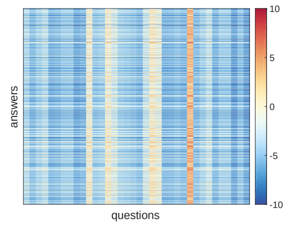
(a) Vector $\boldsymbol{\theta}$ (b) Matrix $\boldsymbol{\theta} + \boldsymbol{\phi}$ (c) Columns of $\boldsymbol{B}$

Figure 4.5: (a) Vector $\boldsymbol{\theta}$ for the Eigenfaces data as an image. (b) Matrix $\boldsymbol{\theta} + \boldsymbol{\phi}$ for the Eigenfaces data. (c) Four columns of $\boldsymbol{B}$ for the Eigenfaces data.

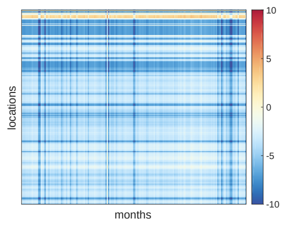
Figure 4.5(a). It is clear that the dominating features of faces – eyes, nose, and mouth – are best expressed using subtropical-type structure, while the other parts are better explained using NMF-type structure. This is to be expected, as the subtropical areas are those where lighting has the largest effects (either as bright areas, or areas in shadows, depending on the direction of the light). These extremes are often easiest to describe using the subtropical structure.

Similarly to Worldclim, we can plot the matrix $\boldsymbol{\theta} + \boldsymbol{\phi}$ (Figure 4.5(b)). There we notice that some faces have a strong subtropical structure, while most of the structure is still NMF-like. To validate that the factors are interpretable, we present examples from the left factor matrix $\boldsymbol{B}$ in Figure 4.5(c). We see that factors mostly depict facial features, except for the one at the bottom right, which can be used to add lighting effects to the bottom left part of the figures.

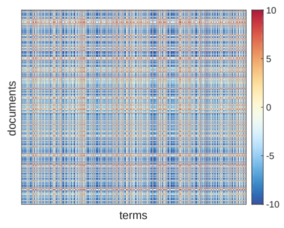
For the remaining real-world datasets the corresponding matrix $\boldsymbol{\theta} + \boldsymbol{\phi}$ is shown in Figure 4.6.



(a) NPAS



(b) HPI



(c) 4News

Figure 4.6: Visualizations for the parameter matrix $\boldsymbol{\theta} + \boldsymbol{\phi}$ with different real-world data sets.

CHAPTER 5

Nested Submatrix Identification Using Tropical Algebra

In the previous chapters we have explored how the tropical and subtropical algebras can be used for analysing continuous data. While applying them to real-valued matrices, which are their domain of definition, seems natural, here we will see that they can also be used for pattern mining in binary data. In this chapter we will obtain a tropical characterization of the nested subgraph mining problem, which is equivalent to identifying symmetric nested submatrices of the adjacency matrix of a graph.

5.1 Community Detection and Nested Matrices

Finding a concise set of dense subgraphs (quasi-cliques) that jointly explain most of the edges in an undirected graph is a fundamental problem in graph mining. Quasi-cliques are usually identified as the *communities* of the graph, and the problem of finding them is called *community detection*. Since it is equivalent to covering the adjacency matrix of a graph with symmetric rank-1 submatrices, it can be seen as a symmetric matrix factorization problem. Many variants of this problem exist, depending on whether the communities are allowed to overlap or not [see e.g. Yang and Leskovec, 2013, Galbrun et al., 2014], whether the graph has labels [Galbrun et al., 2014], and so forth.

Traditionally, much of the research focused on representing communities as quasi-cliques, and NMF [Yang and Leskovec, 2013] and Boolean matrix factorization [Galbrun et al., 2014] have been proposed for solving this problem. Quasi-cliques, however, are by no means the only way to view communities. Recent years saw an increased interest in moving beyond mining clique-like communities [see e.g. Lim et al., 2014, Araujo et al., 2014, Metzler et al., 2016, Koutra et al., 2014], and it was identified that real-world communities can have various shapes, including stars, hyperbolae, and cliques [Lim et al., 2014, Araujo et al., 2014, Metzler et al., 2016]. The so-called *core-periphery* model [Borgatti and Everett, 1999], known from social sciences, describes communities whose adjacency matrices are L-shaped, that is they have a densely connected center that is only loosely tied to the periphery.

In this chapter we study the problem of identifying *nested* submatrices (see Definition 14) of the adjacency matrix of a given undirected graph. Nested submatrices generalize many of the proposed shapes of communities, including stars, hyperbolic structures, core-periphery communities, and cliques. In addition, nested submatrices are important in their own right, and have been studied extensively, especially in ecology since the 1980's [Patterson and Atmar, 1986] [see also Mannila and Terzi, 2007].

While nested submatrices offer a generalized way of summarizing a binary matrix, existing work has mostly required or assumed these submatrices to be non-overlapping (with an exception of a paper by van Leeuwen et al. [2016], although it did not concentrate on nested matrices). This is, however, unrealistic in many applications. For instance, in ecology it would mean that no species can be a part of different ecosystems, and widely-spread species would have to be artificially assigned into one ecosystem only; in social networks, similarly, no person could be a member of multiple communities. To overcome this, we allow the nested submatrices to overlap. This brings us to the main problem studied in this chapter.

Problem 9 (Covering by nested submatrices)**.** Given a binary matrix $\boldsymbol{A}$ and an integer k, find k nested binary matrices $\boldsymbol{N}_1, \boldsymbol{N}_2, \ldots, \boldsymbol{N}_k$ such that their union (elementwise logical OR) $\tilde{\boldsymbol{A}} = \bigcup_{\ell=1}^{k} \boldsymbol{N}_\ell$ minimizes

$$\sum_i \sum_{j:j\neq i} \left|\boldsymbol{A}_{ij} - \tilde{\boldsymbol{A}}_{ij}\right| . \tag{5.1}$$

Notice that in (5.1), we do not consider the diagonal, ignoring any potential self-loops. We are mainly interested in the symmetric form of this problem, as it facilitates interpreting $\boldsymbol{A}$ as the adjacency matrix of a graph.

Computational complexity. To analyse the computational complexity of Problem 9, we will start by inspecting its parts. In the first problem we are given a binary matrix $\boldsymbol{A}$, and our task is to find the nested binary matrix $\boldsymbol{N}$ that is as close to $\boldsymbol{A}$ as possible (i.e. Problem 9 with $k = 1$). No polynomial-time algorithm for this problem is known, but the problem is also not known to be NP-hard [Junttila, 2011, Ch. 4.4]. On the other hand, the problem of finding the largest nested submatrix of $\boldsymbol{A}$ is NP-hard [Junttila, 2011, Thm. 4.13], where the size of the submatrix is counted as the total number of its rows and columns.

While generating the optimal matrices $\boldsymbol{N}_i$ is hard, we can generate *some* nested matrices, and choose from them. Unfortunately, a nested matrix is an example of a *generalized rank-1 matrix* [Miettinen, 2015], and hence, given a matrix $\boldsymbol{A}$ and a collection $\mathcal{N} = \{\boldsymbol{N}_i\}_{i=1}^{n}$, it is NP-hard to choose the smallest subcollection $\mathcal{S} \subseteq \mathcal{N}$ such that $\boldsymbol{A} = \bigcup_{\boldsymbol{N}\in\mathcal{S}} \boldsymbol{N}$ (assuming such collection exists). It is also NP-hard to choose the k matrices from $\mathcal{N}$ that minimize the distance between $\boldsymbol{A}$ and $\bigcup_{i=1}^{k} \boldsymbol{N}_i$ [Miettinen, 2015]. Even approximating the error to within a superpolylogarithmic factor is NP-hard [Miettinen, 2015].

Conclusively establishing the computational complexity of Problem 9 remains intriguing future work. As many of its subproblems are NP-hard, it seems reasonable to expect it to be NP-hard as well.

5.2 Tropical-Logistic Decompositions

Overlapping community detection is often formalized as a matrix factorization problem [Yang and Leskovec, 2013, Galbrun et al., 2014]. The key observation in these approaches is that rank-1 submatrices of the adjacency matrix correspond to cliques. Hence, representing the adjacency matrix as a union (or sum) of rank-1 matrices identifies the cliques. Re-writing the union (sum) of rank-1 matrices as a matrix product gives the standard matrix factorization formulation.[1]

In the same vein, we would like to reformulate Problem 9 as a matrix factorization problem. Unfortunately, the above approach does not work as such for nested submatrices, as they are not rank-1 matrices in the conventional sense. Instead, we base our approach on the recent work on *rounding rank* [Neumann et al., 2016] that provides us with a convenient characterization of nested matrices [see also Araujo et al., 2016].

Definition 22. [Neumann et al., 2016] A binary matrix $\boldsymbol{A} \in \{0,1\}^{n\times m}$ has *nonnegative rounding rank* of 1 if and only if there exist nonnegative vectors $\boldsymbol{x} \in \mathbb{R}^n_{\geq 0}$ and $\boldsymbol{y} \in \mathbb{R}^m_{\geq 0}$ such that $A = \tau_{1/2}(\boldsymbol{x}\boldsymbol{y}^T)$, where $\tau_\alpha \colon \mathbb{R} \to \{0,1\}$ is a *thresholding function*, which is defined for every $\alpha \in \mathbb{R}$ as follows

$$\tau_\alpha(x) = \begin{cases} 1 & \text{if} \quad x \geq \alpha \\ 0 & \text{otherwise} \end{cases} . \tag{5.2}$$

Notice that we apply the threshold function independently to each element of the matrix $\boldsymbol{x}\boldsymbol{y}^T$. Typical thresholds are $\alpha = 1/2$ in the binary domain, and $\alpha = 0$ when working with the tropical algebra. If it is implicitly clear, we will subsequently omit the subscript α.

The following proposition gives an alternative characterization of nested matrices.

Proposition 11. *[Neumann et al., 2016] Let $\boldsymbol{A}$ be an arbitrary binary matrix. $\boldsymbol{A}$ is nested if and only if it has nonnegative rounding rank of 1.*

Proposition 11 establishes a mapping from the set of nonnegative real rank-1 matrices to nested matrices, and we can now view nested matrices as being "rank-1" in this generalized way. Interestingly, prior to this characterization, the exact relation between nested and continuous matrices was not clear, although for example Junttila and Kaski [2013] used SVD to solve the segmented nestedness. Proposition 11 gives us the rounding rank formulation of Problem 9 for $k = 1$. It is tempting to think that we could extend this characterization to a union of multiple nested submatrices. Ideally, what we want is, given a symmetric binary matrix $\boldsymbol{A}$ of size n and a positive integer k, find an n-by-k nonnegative matrix $\boldsymbol{B}$ such that $\tau(\boldsymbol{B}\boldsymbol{B}^T) \approx \boldsymbol{A}$. Unfortunately, the characterization from rounding rank falls apart in higher-rank decompositions, and the nonnegative rounding rank-k decomposition is not equal to the union of k nested matrices, as the following example illustrates.

[1]The factorization is Boolean if we take the union [Galbrun et al., 2014], and standard or nonnegative if we take the sum of the rank-1 matrices [Yang and Leskovec, 2013].

Example 1. Consider the rank-1 matrices $\boldsymbol{A}$ and $\boldsymbol{B}$:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1/\sqrt{3} & 1/3 \\ 1/\sqrt{3} & 1/3 & 1/\sqrt{27} \\ 1/3 & 1/\sqrt{27} & 1/9 \end{pmatrix} \quad \boldsymbol{B} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1/9 & 1/3 \\ 0 & 1/3 & 1 \end{pmatrix} .$$

$\tau(\boldsymbol{A}) = \left(\begin{smallmatrix} 1&1&0 \\ 1&0&0 \\ 1&0&0 \end{smallmatrix}\right)$ and $\tau(\boldsymbol{B}) = \left(\begin{smallmatrix} 0&0&0 \\ 0&0&0 \\ 0&0&1 \end{smallmatrix}\right)$ are nested. But for $\boldsymbol{A}+\boldsymbol{B}$ the resulting matrix is not the union $\tau(\boldsymbol{A}) \cup \tau(\boldsymbol{B})$ as $\tau(\boldsymbol{A}+\boldsymbol{B})$ has 1s in the lower-right corner that are not present in either $\tau(\boldsymbol{A})$ or $\tau(\boldsymbol{B})$:

$$\boldsymbol{A}+\boldsymbol{B} = \begin{pmatrix} 1 & 1/\sqrt{3} & 1/3 \\ 1/\sqrt{3} & 4/9 & \frac{3+\sqrt{3}}{9} \\ 1/3 & \frac{3+\sqrt{3}}{9} & 10/9 \end{pmatrix} , \tau(\boldsymbol{A}+\boldsymbol{B}) = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} .$$

$\Diamond$

The crux of our approach is to replace the standard algebra with the *tropical* one: we will show that the characterization of nested matrices under rounding rank extends naturally to higher-rank tropical decompositions. Furthermore, as nested matrices do not allow any way of assessing the confidence of the algorithm, we relax the problem so that we can obtain the likelihood of the data under the model.

Tropical Matrix factorization formulation. With the tropical algebra we can preserve the original nested matrices. To see that, we will make use of the following lemma.

Lemma 12. *Let $f\colon \mathbb{R} \to \mathbb{R}$ be a monotonically increasing function. Then f and $\oplus$* distribute*, that is*

$$f(a \oplus b) = f(a) \oplus f(b) \quad \textit{for all } a, b \in \mathbb{R}. \tag{5.3}$$

Proof. Without loss of generality, let $a > b$. Then $f(a \oplus b) = f(a) = f(a) \oplus f(b)$, following from the monotonicity of f. $\square$

Elementwise exponentiation of a rank-1 tropical matrix produces a classic rank-1 matrix. This allows us to define nested matrices via the tropical algebra.

Proposition 13. *Let $\boldsymbol{A} \in \{0,1\}^{n\times m}$ be an arbitrary binary matrix. $\boldsymbol{A}$ is nested if and only if there exist vectors $\boldsymbol{x} \in \overline{\mathbb{R}}^{n\times 1}$ and $\boldsymbol{y} \in \overline{\mathbb{R}}^{1\times m}$ such that $\boldsymbol{A}_{ij} = \tau_0(\boldsymbol{x}_i + \boldsymbol{y}_j)$.*

Proof. First assume that $\boldsymbol{A}$ is nested. From Proposition 11 we know that $\boldsymbol{A}_{ij} = \tau(\boldsymbol{a}_i\boldsymbol{b}_j)$ for some vectors $\boldsymbol{a} \in \mathbb{R}^{n\times 1}$ and $\boldsymbol{b} \in \mathbb{R}^{1\times m}$. Define $\boldsymbol{x}_i = \log(\boldsymbol{a}_i) + \log(2)/2$ and $\boldsymbol{y}_j = \log(\boldsymbol{b}_j) + \log(2)/2$. We have

$$\begin{aligned} \tau_0(\boldsymbol{x}_i + \boldsymbol{y}_j) &= \tau_0(\log(\boldsymbol{a}_i) + \log(\boldsymbol{b}_j) + \log(2)) \\ &= \tau_0(\log(2\boldsymbol{a}_i\boldsymbol{b}_j)) . \end{aligned} \tag{5.4}$$

By exponentiating (5.4) and the monotonicity of the exponential function, we conclude that $\tau(\boldsymbol{x}_i+\boldsymbol{y}_j)=1$ if and only if $2\boldsymbol{a}_i\boldsymbol{b}_j \geq 1$, and hence $\tau(\boldsymbol{x}_i+\boldsymbol{y}_j)=\boldsymbol{A}_{ij}$. Conversely, let $\boldsymbol{A}_{ij}=\tau(\boldsymbol{x}_i+\boldsymbol{y}_j)$. Define $\boldsymbol{a}_i = e^{\boldsymbol{x}_i}/\sqrt{2}$ and $\boldsymbol{b}_j = e^{\boldsymbol{y}_j}/\sqrt{2}$. We have

$$\begin{aligned}\tau(\boldsymbol{a}_i\boldsymbol{b}_j) &= \tau\left(\frac{e^{\boldsymbol{x}_i}}{\sqrt{2}}\frac{e^{\boldsymbol{y}_j}}{\sqrt{2}}\right) \\ &= \tau(e^{\boldsymbol{x}_i+\boldsymbol{y}_j}/2)\ .\end{aligned} \tag{5.5}$$

Now, analogously to the previous part, by taking the logarithm of (5.5) and using the fact that the logarithmic function is monotonous, we conclude that $\tau(\boldsymbol{a}_i\boldsymbol{b}_j)=1$ if and only if $\boldsymbol{x}_i+\boldsymbol{y}_j \geq 0$. We thus obtain $\tau(\boldsymbol{a}_i\boldsymbol{b}_j)=\boldsymbol{A}_{ij}$, which concludes the proof. □

In the future we will omit the 0 index for the thresholding function when it is used with the tropical algebra and write τ instead of τ_0. The nested matrix preservation property now follows from Lemma 12.

Corollary 14. *Let $\boldsymbol{N}_i=\tau(\boldsymbol{a}_i+\boldsymbol{b}_i)$, $i=1,\ldots,k$, be a set of nested binary matrices, where $\boldsymbol{a}_i\in\overline{\mathbb{R}}^{n\times 1}$ and $\boldsymbol{b}_i\in\overline{\mathbb{R}}^{1\times m}$ for all i. Let $\boldsymbol{A}$ be the n-by-k matrix with the vectors $\boldsymbol{a}_i$ as its columns, and let $\boldsymbol{B}$ be the m-by-k matrix with $\boldsymbol{b}_i$ as its columns. Then the rounded tropical matrix product is the union of the nested matrices:*

$$\tau(\boldsymbol{A}\diamond\boldsymbol{B}^T)=\bigcup_{i=1}^{k}\boldsymbol{N}_i\ . \tag{5.6}$$

Proof. We have

$$\begin{aligned}\tau(\boldsymbol{A}\diamond\boldsymbol{B}^T) &= \tau((\boldsymbol{a}_1+\boldsymbol{b}_1)\oplus(\boldsymbol{a}_2+\boldsymbol{b}_2)\oplus\cdots\oplus(\boldsymbol{a}_k+\boldsymbol{b}_k)) \\ &= \tau(\boldsymbol{a}_1+\boldsymbol{b}_1)\oplus\tau(\boldsymbol{a}_2+\boldsymbol{b}_2)\oplus\cdots\oplus\tau(\boldsymbol{a}_k+\boldsymbol{b}_k) \\ &= \tau(\boldsymbol{a}_1+\boldsymbol{b}_1)\cup\tau(\boldsymbol{a}_2+\boldsymbol{b}_2)\cup\cdots\cup\tau(\boldsymbol{a}_k+\boldsymbol{b}_k)\ ,\end{aligned}$$

where we used (5.3) in the second equality, and the fact that if $a,b\in\{0,1\}$ then $a\oplus b=a\vee b$ in the last equality. □

Hence we can rewrite Problem 9 as follows.

Problem 10 (Rounded tropical factorization)**.** Given an n-by-m binary matrix $\boldsymbol{A}$ and an integer k, find matrices $\boldsymbol{B}\in\overline{\mathbb{R}}^{n\times k}$ and $\boldsymbol{C}\in\overline{\mathbb{R}}^{k\times m}$ that minimize

$$\sum_i\sum_{j:j\neq i}|\boldsymbol{A}_{ij}-\tau(\boldsymbol{B}\diamond\boldsymbol{C})_{ij}|\ . \tag{5.7}$$

For a symmetric decomposition we have $\boldsymbol{A}\approx\tau(\boldsymbol{B}\diamond\boldsymbol{B}^T)$.

Maximizing the likelihood of the data. Problem 10 has two issues: it is NP-hard to optimize, and, as (5.7) measures the binary reconstruction error, it does not give any indication of how *confident* we are that a particular entry should be 1 or 0. Therefore, we replace the threshold function τ with the *logistic* (or *sigmoid*) *function*

$$\sigma_t(x) = \big(1 + \exp(-tx)\big)^{-1} . \tag{5.8}$$

We will omit the subscripts when they are obvious and we will write $\sigma_t(\boldsymbol{A}) = \boldsymbol{B}$ for the matrix $\boldsymbol{B}$ that has $\boldsymbol{B}_{ij} = \sigma_t(\boldsymbol{A}_{ij})$.

Using the sigmoid function, we model the input matrix as a multivariate Bernoulli random variable with its odds given by $\sigma_t(\boldsymbol{B} \diamond \boldsymbol{C})$ [cf. Schein et al., 2003]. As the sigmoid function is monotonically increasing, the distributivity lemma (Lemma 12) holds. Thus, our goal becomes to maximize the likelihood of observing the data. This is equivalent to minimizing the negative log-likelihood of $\boldsymbol{A}$,

$$\begin{aligned} E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, t) = &- \sum_i \sum_{j \neq i} \boldsymbol{A}_{ij} \log(\sigma_t(\boldsymbol{B} \diamond \boldsymbol{C})_{ij}) \\ &- \sum_i \sum_{j \neq i} (1 - \boldsymbol{A}_{ij}) \log(1 - \sigma_t(\boldsymbol{B} \diamond \boldsymbol{C})_{ij}) , \end{aligned} \tag{5.9}$$

We can now formulate the maximum likelihood problem:

Problem 11 (Logistic-tropical factorization)**.** Given an n-by-m binary matrix $\boldsymbol{A}$, an integer k, and $t > 0$, find matrices $\boldsymbol{B} \in \overline{\mathbb{R}}^{n \times k}$ and $\boldsymbol{C} \in \overline{\mathbb{R}}^{k \times m}$ that minimize (5.9).

For a symmetric decomposition, the objective (5.9) changes to

$$\begin{aligned} E(\boldsymbol{A}, \boldsymbol{B}, t) = &- 2 \sum_i \sum_{j > i} \boldsymbol{A}_{ij} \log(\sigma_t(\boldsymbol{B} \diamond \boldsymbol{B}^T)_{ij}) \\ &- 2 \sum_i \sum_{j > i} (1 - \boldsymbol{A}_{ij}) \log(1 - \sigma_t(\boldsymbol{B} \diamond \boldsymbol{B}^T)_{ij}) , \end{aligned} \tag{5.10}$$

which yields the problem we use in the remainder of this chapter:

Problem 12 (Symmetric logistic-tropical factorization)**.** Given an n-by-n binary matrix $\boldsymbol{A}$, an integer k, and $t > 0$, find a matrix $\boldsymbol{B} \in \overline{\mathbb{R}}^{n \times k}$ minimizing (5.10).

A related approach was proposed by Araujo et al. [2016], who developed an algorithm called `FastStep` for doing thresholded nonnegative matrix factorization. Each rank-1 component of `FastStep` is nested after thresholding, but the full factorization is *not* a union of nested matrices.

Nested subgraph characterization. While we formulated Problem 9 in terms of nested submatrix covering, when the input matrix is symmetric, this is equivalent to covering a graph with nested subgraphs. An undirected graph $G = (V, E)$ is *nested* if we can order the vertices $v \in V$ in a sequence $(v_1, v_2, \ldots, v_n)$ such that $N(v_{i+1}) \subseteq N(v_i)$ for all $i = 1, \ldots, n-1$, where by $N(v)$ we denote the neighbourhood of the vertex $v \in V$ ($N(v) = \{u \in V : \{v, u\} \in E\}$).

5.3 Algorithm (SLTF)

Real-world networks tend to be very sparse, and hence we need an algorithm that can solve Problem 12 without having to process all $O(n^2)$ potential edges. To that end, we use stochastic gradient descent (SGD) with subsampling of the zeros. SGD is a common approach for decomposing sparse matrices, but usually – as in collaborative filtering – the zero entries are assumed to be unobserved. This is not the case here, as the zeros also carry information. Hence, we use subsampling.

The algorithm, called SLTF (Symmetric Logistic-Tropical Factorization) and presented in Algorithm 11, runs multiple epochs. On every epoch, we sample $O(|\boldsymbol{A}|)$ elements, where $|\boldsymbol{A}|$ is the number of non-zero elements in the matrix, and update the corresponding rows of the factor matrix. It takes three main parameters: k, the rank of the decomposition; t, the steepness of the sigmoid function; and μ that controls the behaviour of the soft max function.

The factor matrix $\boldsymbol{B}$ is initialized with random numbers from $[-0.1, 0]$ (Line 2). This guarantees that the initial solution is sparse. We use tiered sampling to sample the elements we update (Lines 6 and 7) so that we can ensure that we get enough 1s even for extremely sparse matrices. We sample the locations $\boldsymbol{A}_{ij} = 1$ uniformly, but to sample the locations of zeros, we use weighted sampling. We sample a location $\boldsymbol{A}_{i,j} = 0$ with a probability that is proportional to the number of 1s in rows i and j of $\boldsymbol{A}$.

We use separate step sizes for the elements that are 1 and that are 0 (*s1* and *s0*, respectively). These are updated (Line 11) using a bold driver heuristic: we increase them if the new error is smaller than the previous one, otherwise we decrease them. In addition, if the current error is greater on 1s than 0s, then we increase the step size for 1s relative to that for 0s, and vice versa. The actual gradient updates are explained in detail below.

Computing the objective function (Line 10) has complexity $O(n^2k)$ for n-by-k matrix $\boldsymbol{B}$. Instead of computing it completely, we approximate it on a sample of size $O(|\boldsymbol{A}|)$, again using tiered sampling.

UpdateFactors. The function UpdateFactors (Line 8) follows the SGD approach and updates the factor matrix $\boldsymbol{B}$ given a sequence of sampled data points by optimizing the objective locally. We will explain UpdateFactors using an asymmetric notation $\boldsymbol{A} \approx \boldsymbol{BC}$ as this simplifies the discussion. The symmetric variant will be explained at the end of this section.

In the standard matrix factorization setting the objective of SGD, $\|\boldsymbol{A} - \boldsymbol{BC}\|_F^2$, is represented as a sum of functions that each depend only on one row of $\boldsymbol{B}$ and one column of $\boldsymbol{C}$:

$$\|\boldsymbol{A} - \boldsymbol{BC}\|_F^2 = \sum_{ij}(\boldsymbol{A}_{ij} - (\boldsymbol{BC})_{ij})^2 = \sum_{ij}\varphi_{ij}(\boldsymbol{B}_i, \boldsymbol{C}^j)\ . \tag{5.11}$$

Then, given a sequence of index pairs $\{i(\alpha), j(\alpha)\}_{\alpha=1}^{l}$ that correspond to the elements of $\boldsymbol{A}$, SGD updates the individual rows of $\boldsymbol{B}$ and columns of $\boldsymbol{C}$ by each time taking a single step in the direction of the steepest descent of $\varphi_{i(\alpha)j(\alpha)}(\boldsymbol{B}_{i(\alpha)}, \boldsymbol{C}^{j(\alpha)})$.

Algorithm 11 SLTF

```
Input: $A \in \{0,1\}^{n\times m}$, $k \in \mathbb{N}$, $t \in \mathbb{R}_{>0}$, $\mu \in \mathbb{R}_{>0}$
Output: $B \in \overline{\mathbb{R}}^{n\times k}$
 1: function SLTF($A, k, t, \mu$)
 2:     Initialize $B$
 3:     $LL \leftarrow E(A, B, t)$
 4:     Initialize $s1$, $s0$
 5:     while not converged do
 6:         $idx_1 \leftarrow$ sample $O(|A|)$ 1s uniformly at random
 7:         $idx_0 \leftarrow$ sample $O(|A|)$ 0s weighted by degree
 8:         $B \leftarrow$ UpdateFactors($A, B, k, t, s1, s0, \mu, idx_1, idx_0$)
 9:         $LL_{\text{old}} \leftarrow LL$
10:         $LL \leftarrow E(A, B, t)$
11:         $[s1, s0] \leftarrow$ UpdateStepSizes($s1, s0, LL, LL_{\text{old}}$)
12:     end while
13:     return $B$ that had the smallest negative log-likelihood
14: end function
```

We adapt the SGD approach to the logistic-tropical factorization problem. First observe that (5.9) is additive and can be represented in a form identical to (5.11) by setting

$$\begin{aligned}\varphi_{ij}(\boldsymbol{B}_i, \boldsymbol{C}^j, t) = \big(&\boldsymbol{A}_{ij}\log(\sigma_t(\boldsymbol{B}_i \diamond \boldsymbol{C}_j))\\ &+ (1-\boldsymbol{A}_{ij})\log(1-\sigma_t(\boldsymbol{B}_i \diamond \boldsymbol{C}^j))\big)\ .\end{aligned} \tag{5.12}$$

Unfortunately, φ_{ij} are not differentiable, as they contain the $\max$ operator. In order to differentiate φ_{ij}, we replace the maximum with the *soft max* function

$$\max_{s=1..k}\{x_s\} \approx \sum_{s=1}^{k} \frac{e^{\mu x_s}}{\sum_{j=1}^{k} e^{\mu x_j}} x_s = \sum_{s=1}^{k} f(x_s, x, \mu) x_s\ , \tag{5.13}$$

where μ is a relaxation parameter.

Using the soft max in our original objective, we obtain

$$\begin{aligned}\varphi_{ij}(\boldsymbol{B}_i, \boldsymbol{C}^j, t) &\approx -\big(\boldsymbol{A}_{ij}\log(\sigma_t(\boldsymbol{B}_i \diamond_\mu \boldsymbol{C}_j))\\ &\qquad + (1-\boldsymbol{A}_{ij})\log(1-\sigma_t(\boldsymbol{B}_i \diamond_\mu \boldsymbol{C}^j))\big)\\ &= \tilde{\varphi}_{ij}(\boldsymbol{B}_i, \boldsymbol{C}^j, t, \mu)\ ,\end{aligned} \tag{5.14}$$

where $\diamond_\mu$ denotes the tropical matrix product with the $\max$ operation relaxed using parameter μ. For any matrices $\boldsymbol{B} \in \overline{\mathbb{R}}^{n\times k}$ and $\boldsymbol{C} \in \overline{\mathbb{R}}^{k\times m}$, we have $\boldsymbol{B} \diamond_\mu \boldsymbol{C} \to \boldsymbol{B} \diamond \boldsymbol{C}$ when $\mu \to \infty$. We can now use the gradient of the right side of (5.14) as a “relaxation” of the gradient of $\varphi_{ij}(\boldsymbol{B}_i, \boldsymbol{C}^j, t)$. Note that since the functions $\tilde{\varphi}_{ij}(\boldsymbol{b}, \boldsymbol{c}, t, \mu)$ depend only on elementwise sums of the vectors $\boldsymbol{b}$ and $\boldsymbol{c}$, we have $\nabla_{\boldsymbol{b}}\tilde{\varphi}_{ij}(\boldsymbol{b}, \boldsymbol{c}, t, \mu) = \nabla_{\boldsymbol{c}}\tilde{\varphi}_{ij}(\boldsymbol{b}, \boldsymbol{c}, t, \mu)$. If we denote $\boldsymbol{x} = \boldsymbol{b} + \boldsymbol{c}$, $a = \boldsymbol{A}_{ij}$, and $\max_\mu(x) = \sum_{s=1}^{k} f(x_s, x, \mu)x_s$, then the relaxed value of $\frac{\partial}{\partial \boldsymbol{b}_l}\tilde{\varphi}_{ij}(\boldsymbol{b}, \boldsymbol{c}, t, \mu)$

is given by

$$
\begin{aligned}
&- t\,[a(1-\sigma_t(\boldsymbol{b}\diamond_\mu \boldsymbol{c})) + (1-a)(1-\sigma_t(-\boldsymbol{b}\diamond_\mu \boldsymbol{c}))]\,\frac{\partial}{\partial \boldsymbol{b}_l}\boldsymbol{b}\diamond_\mu \boldsymbol{c}\,,\\
&\frac{\partial}{\partial \boldsymbol{b}_l}\boldsymbol{b}\diamond_\mu \boldsymbol{c} = f(x_l, x, \mu)\left[\mu(x_l - \max_\mu(x)) + 1\right]\,. \qquad (5.15)
\end{aligned}
$$

Finally, to adapt (5.15) to our symmetric objective, we need to set $\boldsymbol{b} = \boldsymbol{B}_i$ and $\boldsymbol{c} = \boldsymbol{B}_j^T$ and recall that the gradients with respect to $\boldsymbol{B}_i$ and $\boldsymbol{B}_j$ are the same.

Computational complexity. The major contributors to the complexity of SLTF are 1) the computation within UpdateFactors and 2) computing the likelihood of the data given the current factor matrix. On every iteration of SLTF, UpdateFactors performs updates to $\boldsymbol{B}$ for each of the $O(|\boldsymbol{A}|)$ sampled data points. Each time it has to compute the gradient of $\tilde{\varphi}_{ij}(\boldsymbol{B}_i, \boldsymbol{B}_j^T, t, \mu)$ and then update $\boldsymbol{B}_i$ and $\boldsymbol{B}_j$. Both of these procedures take time $O(k)$, and hence the complexity of UpdateFactors is $O(|\boldsymbol{A}|\,k)$. Since we use a fixed number of samplings for estimating the likelihood, and an evaluation of the likelihood at a single point takes $O(k)$ time, the complexity of approximating the objective is $O(|\boldsymbol{A}|\,k)$. Other parts of SLTF are not as computationally expensive – initializing $\boldsymbol{B}$ takes time $O(nk)$ and the complexity of SampleData is $O(|\boldsymbol{A}|)$. If M is the total number of cycles performed by SLTF, then its total complexity is $O(Mk(n + |\boldsymbol{A}|)$. Due to the complex objective, proofs of speed of convergence seem hard to obtain.

One of the benefits of SGD is that it allows us to parallelize the algorithm: we can use a partitioned approach, an asynchronous distributed approach, or other parallelization methods for SGD [see e.g. Teflioudi et al., 2012, and references therein]. For our implementation, we use shared-memory parallelization over the different elements we update, and SIMD vectorization over the individual gradient computations.

5.4 Experimentimental Evaluation

In this section we experimentally test SLTF and compare it to other methods. We first describe the competing algorithms, then report the performance for synthetically generated data, and finish with the results on various real-world data.

SLTF is implemented in Matlab and C and uses OpenMP for parallel processing. The source code for SLTF, the scripts to generate synthetic data and to execute the experiments, together with the parameters used in all experiments, are freely available.[2] For all of these experiments, we ran SLTF for 600 iterations without any early stopping criteria.

5.4.1 Methods and metrics

To evaluate SLTF, we compare it against existing approaches, that can be divided into three groups. The first group consists of basic matrix factorization methods, NMF and Asso, that aim at finding a good decomposition of

[2]http://cs.uef.fi/~pauli/tropical/logistic/

the input using nonnegative real values and binary values endowed with the Boolean algebra, respectively. In order to facilitate the comparison against SLTF, we also experimented with a symmetric version of NMF, $\mathtt{NMF}_{\mathtt{sym}}$, that has format $\boldsymbol{W}\boldsymbol{W}^T$ or $\boldsymbol{H}^T\boldsymbol{H}$, depending on which one gives less error. In addition, we used the factors found by NMF in a logistic tropical factorization, i.e. $\sigma(\boldsymbol{W}^1\boldsymbol{H}_1) \oplus \sigma(\boldsymbol{W}^2\boldsymbol{H}_2) \oplus \cdots \oplus \sigma(\boldsymbol{W}^k\boldsymbol{H}_k)$. We call this version $\mathtt{NMF}_{\boxplus}$. It solves Problem 11, but since NMF does not try to optimize for this, it is not expected to perform as well as SLTF.

The second group involves methods that directly find thresholded decompositions. The main methods here are LPCA and FastStep (see Section 5.2). We use the approximate version of FastStep with at least 10 internal iterations, as suggested by its authors.

The third group contains just one method, HyCoM-FIT [Araujo et al., 2014]. It fits power-law models to non-overlapping communities. Since HyCoM-FIT does not decompose the data in the matrix factorization sense, we obtain its reconstruction matrix by joining the top k largest communities predicted by its model.

We measure the quality using two metrics: negative log-likelihood (5.10) and relative binary reconstruction error. Assuming $\tilde{\boldsymbol{A}}$ is the reconstruction a method gave for input matrix $\boldsymbol{A}$, the relative binary error is $\|\boldsymbol{A} - \tau_\alpha(\tilde{\boldsymbol{A}})\|_F^2 / \|\boldsymbol{A}\|_F^2$. Here, α is selected depending on a method (usually $\alpha = 1/2$), and we set the diagonal of $\boldsymbol{A}$ and $\tilde{\boldsymbol{A}}$ to all-zeros to ignore self-loops. Notice that $\|\boldsymbol{A}\|_F^2$ simply counts the number of non-zeros in $\boldsymbol{A}$, as $\boldsymbol{A}$ is binary.

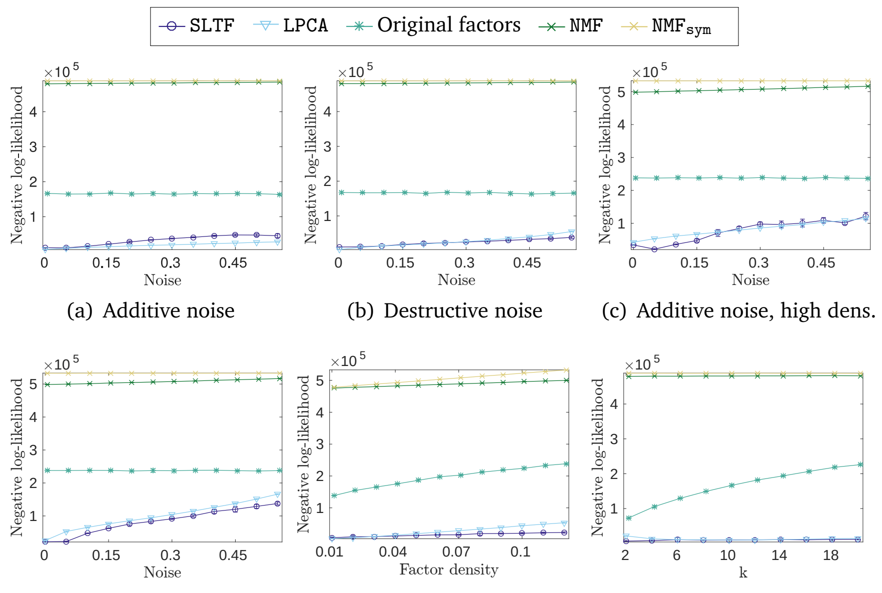
Most methods do not optimize for our likelihood model, and hence the binary error is a fairer measure for those. All methods are run 5 times on the synthetic data and 3 times on the real data to account for the random variance and select a good result. After the required number of runs, we return the best result.

5.4.2 Synthetic data

In the synthetic experiments we test whether SLTF can find the logistic tropical structure when it is present in the data. To generate the data, we first create a factor matrix $\boldsymbol{B} \in \mathbb{R}^{n \times k}$, and compute the thresholded product $\boldsymbol{A} = \tau(\sigma_t(\boldsymbol{B} \diamond \boldsymbol{B}^T))$.

Unless specified otherwise, we generated matrices of size 1000×1000 with matrix $\boldsymbol{B}$ having dimensions 1000×10 (i.e. rank 10). By default, the input matrix density is set to $3\,\%$, and the levels of both additive and destructive noise are $5\,\%$. In every experimental setup, we vary one of the above parameters, while keeping the rest fixed. The number of nonzeros in the synthetic data sets ranges from less than 1000 to roughly $120\,000$.

All algorithms were run 5 times on each matrix, with the best result being selected. We do not report the results for $\mathtt{LPCA}_{\mathtt{sym}}$ or $\mathtt{NMF}_{\boxplus}$, as their original versions rely heavily on the ability to use asymmetric factors, and hence their errors were orders of magnitude worse than those of other methods. For the negative log-likelihood, we also show the likelihood the original factors would give.



(a) Additive noise (b) Destructive noise (c) Additive noise, high dens.

(d) Destr. noise, high dens. (e) Factor density (f) Rank

Figure 5.1: Negative log-likelihoods on synthetic data. All results are averages over 10 random matrices and the width of the error bars is twice the standard deviation.

We will first present the experimental setups before discussing the results in detail.

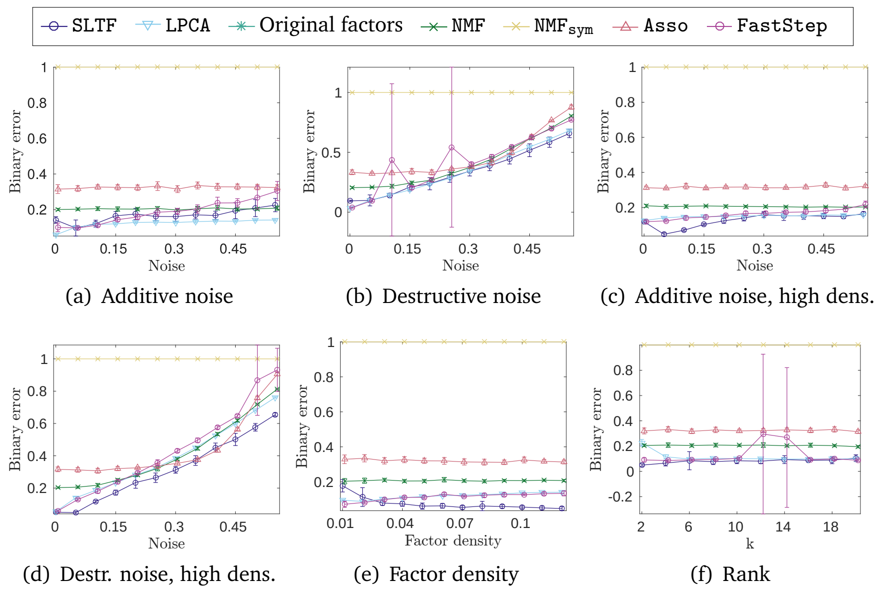
Varying additive noise. The level of additive noise (i.e. replacing 0s with 1s) is defined with respect to the number of 1s in the data. We varied the noise from $0\,\%$ to $55\,\%$ with increments of $5\,\%$. The results are in Figures 5.1(a) and 5.2(a).

Varying additive noise with $12\,\%$ density. This setup only differs from the one above in that the factor matrix density is $12\,\%$. The results are in Figures 5.1(c) and 5.2(c).

Varying destructive noise. The level of destructive noise (turning 1s to 0s) rises from $0\,\%$ to $55\,\%$ with increments of $5\,\%$. The results are shown in Figures 5.1(b) and 5.2(b), and the results of the dense version in Figures 5.1(d) and 5.2(d).

Varying density. We varied the density of the input matrix $\boldsymbol{A}$ from $1\,\%$ to $12\,\%$ with increments of $1\,\%$ and report the errors in Figures 5.1(e) and 5.2(e).

Varying rank. In this setup we investigate how the algorithms respond to varying the (tropical logistic) rank of the data. We varied the number of columns of the factor matrix $\boldsymbol{B}$ from 2 to 12 with increments of 2. The results are shown in Figures 5.1(f) and 5.2(f).



(a) Additive noise (b) Destructive noise (c) Additive noise, high dens.

(d) Destr. noise, high dens. (e) Factor density (f) Rank

Figure 5.2: Binary reconstruction errors on synthetic data. All results are averages over 10 random matrices and the width of the error bars is twice the standard deviation.

Scalability. Here we test how well SLTF scales with respect to the number of non-zeros (edges), rows (nodes), rank, and computing cores. The results are in Figure 5.3. These experiments were run on an Intel Core i7 CPU with 4 cores at 3.4 GHz. When not varied, these matrices had 2^{13} rows and columns and 2^{13} non-zeros. We used rank 50 and four cores.

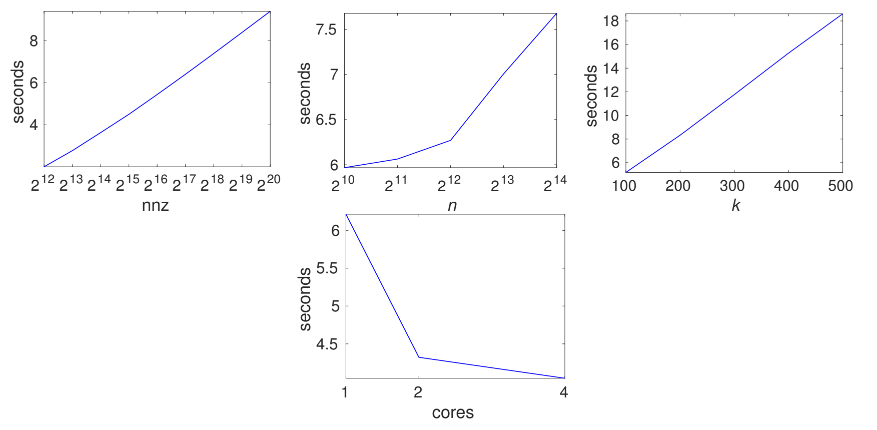


Figure 5.3: Scalability of SLTF with respect to (from left to right) number of non-zeros; dimensionality; rank; number of cores. All values are means over five restarts. Notice that the first two plots have a logarithmic x-scale.

Discussion. SLTF and LPCA give the best overall results. For the most part they are quite close, with SLTF producing somewhat better results on the varying density test and LPCA being slightly better on the additive noise test. The close results of SLTF and LPCA are not surprising because the data followed the structure that SLTF expects, while LPCA has more degrees of freedom for fitting the data. It is worth noting that both SLTF and LPCA produce likelihoods that are better than those of the original factors that were used to generate the data. While this might seem strange, it is easily explained by the fact that the original factors were merely used for generating binary matrices and not optimized for the likelihood. In contrast to SLTF and LPCA, the NMF-based methods do not show good results with either likelihood or binary error. FastStep gives quite good binary reconstruction errors, although it is on average inferior to SLTF and LPCA. Asso, on the other hand, has consistently high reconstruction errors, indicating that the structure of the data is very different from what it expects.

The behaviour of SLTF with respect to scalability is very good. Running time grows moderately with respect to the number of non-zeros; indeed, we can increase the number of non-zeros by 32-fold (from 2^{15} to 2^{19}) while only doubling the running time (from 4 to 8 seconds). Its behaviour with respect to the number of rows and factorization rank is also good, as expected by the runtime analysis. The algorithm also shows good speedups with increasing numbers of cores.

The synthetic experiments confirm that SLTF behaves as expected: high noise levels do have an effect, but otherwise it is quite robust against the characteristics of the data.

5.4.3 Real-world data

The experiments in this section are conducted to validate that our findings on synthetic data correlate with the real world. We will also use real-world data to study what kind of rank-1 matrices SLTF finds.

Data sets. We used two sets of real-world matrices: smaller, where we could compare all different methods, and larger ones, where only SLTF was able to run.

The smaller data sets are Mam_N, Mam_E, Jazz, Paleo, 4News, and Christ., and their properties are listed in Table 5.1.

The Paleo data is a genera-by-genera co-occurrence matrix based on fossil records from a set of locations.[3]

The 4News was first used in Section 3.4.3. This time, however, rather than computing a TF-IDF matrix, we use the terms-by-terms co-occurrence matrix for 800 terms.

The Mam_N data is a co-occurrence matrix of all terrestrial mammals species of the Northern hemisphere, and is based on the IUCN Red List [IUCN, 2014]. It has 3203 species, and a density of $8.4\,\%$. For the factorizations, we used rank $k = 40$.

[3]NOW 030717, http://www.helsinki.fi/science/now/ [Fortelius et al., 2003].

Table 5.1: Properties of small real-world data sets. All matrices are symmetric and k denotes the rank used for reconstruction.

	Mam$_N$	Mam$_E$	Jazz	Paleo	4News	Christ.
# rows	3203	194	198	139	800	1736
# nonzeros	864083	21844	5484	8995	263400	15010
density	0.084	0.580	0.139	0.465	0.411	0.005
k	40	10	5	6	10	30

Table 5.2: Properties of large real-world data sets. All matrices are symmetric.

	YouTube	DBLP	Amazon	LiveJournal
# rows	20329	212637	299902	775003
# communities	133	954	1675	9314
density	0.061	0.004	0.002	0.003

The Mam$_E$ data is analogous to Mam$_N$ but is restricted to European mammals. We have already used the same raw data in Section 3.4.3, but there the preprocessing was different.

The Jazz data [Gleiser and Danon, 2003] comes from the University of Florida sparse matrix collection[4] and describes a jazz musicians network.

Christ. is a dump of StackExchange forums[5] on the topic of Christianity over the last year. We built a graph of 1736 users by adding an edge between two users if they had asked, answered to, or commented on the same question.

The larger data sets come from the SNAP dataset collection.[6] We preprocessed the datasets by removing nodes that were not part of communities of size at least 100. The obtained datasets together with their properties are listed in Table 5.2.

Numerical results. We start by examining the performance of the algorithms on the small datasets. We report the binary errors for all methods in Table 5.3 and the log-likelihoods in Table 5.4.

It is clear that with the small real-world data, `LPCA` is the best, obtaining a very small negative log-likelihood and often perfect reconstructions (behaviour also observed by Neumann et al. [2016]). One should note, however, that `LPCA` is an asymmetric decomposition and clearly the slowest of the methods presented here. The symmetric version of `LPCA`, $\texttt{LPCA}_{\texttt{sym}}$, is among the worst methods. `FastStep` finds decompositions with reconstruction error comparable to or better than `SLTF`. It does this with a significantly slower running time, though. Hence, we also tested a variation, called $\texttt{FastStep}_5$, where we reduced the minimum number of iterations from 10 to 5. This improved the running time to be comparable to `SLTF`, but with a significant cost in quality. Finally, while `NMF` was generally close to `SLTF` in terms of the binary reconstruction

[4]`https://www.cise.ufl.edu/research/sparse/matrices/`, accessed 28 December 2018
[5]`https://archive.org/details/stackexchange`, accessed 20 May 2017
[6]`http://snap.stanford.edu/data/`, accessed 28 December 2018

Table 5.3: Binary error for small real-world data sets

	Mam$_N$	Mam$_E$	Jazz	Paleo	4News	Christ.
SLTF	1.84E−1	3.30E−2	5.08E−1	1.10E−1	2.64E−1	5.67E−1
LPCA	0.00E0	0.00E0	3.05E−1	0.00E0	2.22E−1	0.00E0
LPCA$_{sym}$	8.68E0	6.24E−1	3.89E0	3.94E−1	8.50E−1	2.03E2
NMF	1.89E−1	1.58E−1	4.81E−1	1.69E−1	3.25E−1	5.03E−1
NMF$_{sym}$	1.09E1	7.29E−1	5.53E0	1.13E0	1.43E0	2.01E2
NMF$_{\boxplus}$	4.77E−1	5.44E−1	5.77E−1	3.20E−1	5.25E−1	5.39E−1
Asso	3.36E−1	2.02E−1	5.62E−1	2.28E−1	4.23E−1	5.42E−1
HyCoM-FIT	8.69E−1	8.42E−1	8.29E−1	9.68E−1	9.55E−1	2.12E0
FastStep	2.40E−2	2.10E−2	4.05E−1	1.20E−1	2.36E−1	2.36E−1
FastStep$_5$	2.40E−2	4.44E−1	1.75E0	1.20E−1	1.10E0	1.10E0

Table 5.4: Negative log-likelihoods for real-world data sets.

	Mam$_N$	Mam$_E$	Jazz	Paleo	4News	Christ.
SLTF	3.99E5	2.44E3	6.95E3	2.68E3	1.58E5	3.52E4
LPCA	1.56E2	1.37E0	4.22E3	0.05E0	1.31E5	2.13E2
LPCA$_{sym}$	2.13E8	4.36E5	3.77E5	1.47E5	1.70E6	4.04E7
NMF	5.02E6	1.96E4	2.06E4	1.02E4	3.61E5	1.44E6
NMF$_{sym}$	7.11E6	2.57E4	2.70E4	1.32E4	4.43E5	2.08E6

Table 5.5: Rank, relative negative log-likelihood, and time for large real-world data sets and SLTF

	rank	$\log L/n^2$	time
YouTube	133	0.0091	22 min
DBLP	954	0.0283	263 min
Amazon	1675	0.0277	441 min
LiveJournal	5000	0.0440	2160 min

error, one should remember that it is an asymmetric method. In addition, its negative log-likelihood was quite high. Since FastStep and HyCoM-FIT do not give direct estimations of the likelihood of the data, their results are not shown in Table 5.4.

As only LPCA and FastStep were really comparable to our results, we tried to use them with the larger real-world data sets. For LPCA, this was clearly undoable, while FastStep$_5$ managed to run YouTube in a bit over 9 h, compared to the 21 min SLTF took, and it could not finish DBLP in a week. Hence, we will only report results from SLTF: Table 5.5 gives the negative log-likelihoods *relative* to the data size for the different data sets, together with the rank and the time SLTF took on a 16-core server.

As can be seen from Table 5.5, YouTube gives clearly the best result. This is probably a combination of the data being more amenable to the model and SLTF having a better initial solution or parameter configuration for the data (all results are best-of-3 restarts, but we tuned the parameters for the YouTube data). Nonetheless, SLTF was relatively fast with all data sets, including the large ones; for example, on LiveJournal, storing the 775003×5000 factor matrix in 64-bit floating point numbers takes approximately 31 GB.

Example results. To understand the kind of decompositions SLTF finds, and to evaluate our assumption that the datasets have nested communities, we looked at the factors in the real-world datasets. Example rank-1 matrices are shown in Figure 5.4. We see that the communities are nested, and that the area under the blue curve, which is where our factorization gives likelihoods above $1/2$, is also dense, while the area above it is much sparser. The shape of these communities also varies. The leftmost community (from Christ.) has a nearly star-like structure, while in the next one (from Jazz) the line is almost diagonal. The third and the last communities (from Mam_N and 4News, respectively) show a concave line, while the penultimate one (also from Mam_N) has a more convex structure. Notice that all these submatrices have essentially full rank, and are hence very hard to describe using the standard algebra.

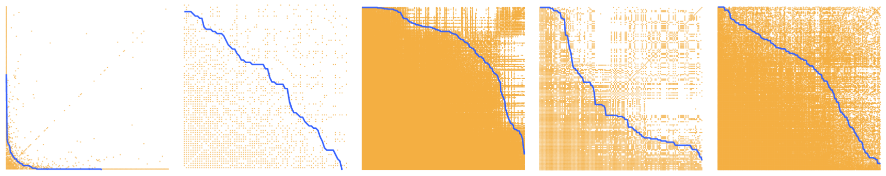


Figure 5.4: Example nested factors. Datasets from left: Christ., Jazz, Mam_N (twice), and 4News. The orange dots are the 1s in the matrix and the found community is the area left and down from the blue line.

These different communities provide an empirical verification for three of our hypotheses: First, the communities in the different real-world graphs are indeed nested. Second, the nested structure is not only 'hyperbolical' [Araujo et al., 2014, Metzler et al., 2016, cf.]. Third, SLTF can find these non-clique-like structures from different data sets.

CHAPTER **6**

Boolean Matrix Factorization with Minimum Description Length

While this chapter is the most far away from the other topics of this thesis, Boolean algebra is a dioid, and matrix factorizations over it exhibit the same "winner-takes-it-all" property, that was central to all the methods discussed so far. Boolean algebra is, in a way, at the other end of the continuity spectrum compared to the subtropical algebra since it only deals with the set $\{0, 1\}$. Here we explore ways of dealing with destructive noise in binary matrices – something that presents major problems to data miners handling occurrence data, and that was aptly dubbed *unknown unknowns* by the former U.S. Secretary of Defense:

> [A]s we know, there are known knowns; there are things we know we know. We also know there are known unknowns; that is to say we know there are some things we do not know. But there are also unknown unknowns – the ones we don't know we don't know. And . . . it is the latter category that tend to be the difficult ones.
>
> [U.S. Department of Defense]

While then-Secretary of Defense Donald Rumsfeld probably was not thinking about data mining when he made his famous comment about unknown unknowns, they are, arguably, the main problem data miners dealing with observation data have to face. A *known unknown*, in this setting, is an element for which we know we do not know its value. As an example, in a movie rating matrix we know which movies the users have not rated yet. In contrast, an *unknown unknown* is an element we have not observed, but we do not know if this is because it does not exist or because we have failed to observe it.

Evidence suggests that unknown unknowns are a common occurrence in real-world presence/absence (or *binary*) data, and that it is more common to fail to observe something that is there than it is to observe something that is not [see e.g. MacKenzie et al., 2002]. In other words, previous research seems to support the idea that there is more *destructive* noise than *additive* noise.

In order to find interesting patterns from data, we consider Boolean matrix factorization (BMF) [Miettinen et al., 2008], a technique that has been successfully applied to a variety of data mining problems involving binary data [see e.g. Miettinen et al., 2008, Lucchese et al., 2014, Lu et al., 2012, Miettinen and Vreeken, 2014]. In its standard form, BMF aims to find a low-rank Boolean factorization of the data that is as close to the original matrix as possible (i.e. minimizes the reconstruction error).

But here as well the unknown unknowns are the difficult ones. When we know which observations are unknown, we can ignore them, and fit the model (i.e. the Boolean matrix factorization) only to the observed parts of the data. But we cannot ignore all unobserved values, as that leaves only the observations (1s in the data) to work with. Still, we also have to be careful that we do not consider unobserved values as equally important to actual observations – as standard reconstruction error does – as this ignores our hypothesis that real data has more unobserved than spuriously observed values.

To address this problem, we use the Minimum Description Length principle (MDL) [Rissanen, 1978, Grünwald, 2007], which is very useful in tackling the problem of the trade-off between fitting the data and keeping the model simple. In general, the more complexity we allow in the model, the better we can fit the data. However, having a high model order comes at the cost of fitting the noise. MDL identifies the optimal balance.

In this chapter we introduce `Nassau`, a new BMF algorithm that is designed to directly minimize the description length. A key aspect is that – unlike the majority of previously proposed BMF algorithms – it can correct its previous mistakes. `Nassau` is quite robust to destructive noise, which is especially beneficial for real-world data as in many domains there are zeros simply due to the lack of observation.

6.1 MDL for BMF

If $\boldsymbol{A}$ is an n-by-m Boolean matrix, $|\boldsymbol{A}|$ denotes the number of 1s in it, i.e., $|\boldsymbol{A}| = \sum_{i,j} a_{ij}$. Given a matrix $\boldsymbol{A}$ of size n-by-m and a column vector $\boldsymbol{c}$ of length n, or a row vector $\boldsymbol{r}$ of length m, we denote by $[\boldsymbol{A}, \boldsymbol{c}]$ and $\left[\begin{smallmatrix}\boldsymbol{A}\\ \boldsymbol{r}\end{smallmatrix}\right]$ the matrices obtained by concatenating $\boldsymbol{A}$ with $\boldsymbol{c}$ and $\boldsymbol{r}$, respectively.

Let $\langle \boldsymbol{B}, \boldsymbol{C} \rangle$ be an (approximate) Boolean decomposition of $\boldsymbol{A}$, $\boldsymbol{A} \approx \boldsymbol{B} \circ \boldsymbol{C}$. We call $\boldsymbol{B}$ and $\boldsymbol{C}$ *factors* of this decomposition, and for any $1 \leq l \leq k$, we refer to the rank-1 matrix formed by the vector pair $\langle \boldsymbol{B}^l, \boldsymbol{C}_l \rangle$ as a *block*. If $\boldsymbol{X}$ and $\boldsymbol{Y}$ are n-by-m binary matrices, we use $\boldsymbol{X} \oplus \boldsymbol{Y}$ to denote their *element-wise exclusive or*. Finally, we denote by $L(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ the description length of $\boldsymbol{A}$ using factor matrices $\boldsymbol{B}$ and $\boldsymbol{C}$, which will be defined later in this section.

MDL: a brief introduction. The Minimum Description Length principle (MDL) [Grünwald, 2007] is a practical version of Kolmogorov Complexity. Both embrace the slogan *Induction by Compression*. This can be roughly described as follows: Given a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ is the one that minimizes $L(M) + L(D \mid M)$, in which $L(M)$ is the length in bits of the description of M, and $L(D \mid M)$ is the length of the data when encoded with model M.

This is called two-part, or *crude*, MDL – as opposed to *refined* MDL, where model and data are encoded together [Grünwald, 2007]. We use two-part MDL because we are specifically interested in the model: the factors that give the best description of the data. Note that MDL requires the compression to be *lossless* in order to allow for fair comparison between different $M \in \mathcal{M}$, and that we are only concerned with code lengths, not actual code words.

BMF: advantages and issues. In Boolean matrix factorization, the goal is to (approximately) represent a Boolean matrix as the Boolean product of two Boolean matrices. That is, given binary matrix $\boldsymbol{A}$, find binary matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ such that $\boldsymbol{A} \approx \boldsymbol{B} \circ \boldsymbol{C}$. In *minimum-error Boolean rank-k* decomposition one is given an n-by-m input matrix $\boldsymbol{A}$ and rank k, and the goal is to find n-by-k and k-by-m factor matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ that minimize $|\boldsymbol{A} \oplus (\boldsymbol{B} \circ \boldsymbol{C})|$.

Up until now we only considered continuous factor matrices (even when analysing binary data in Chapter 5, we used the tropical algebra). Despite that, using binary factors and Boolean decompositions in data mining can have advantages: binary factor matrices are often easier to interpret [Miettinen et al., 2008, Miettinen, 2009], binary factors might be required by the application [e.g. Lu et al., 2012], binary factors can provide better input for subsequent algorithms [e.g. Wicker et al., 2012, Cergani and Miettinen, 2013], and for sparse input data, the factors will be naturally sparse [Miettinen, 2010].

Unfortunately, computing the least-error BMF is NP-hard, and has strong inapproximability results [Miettinen, 2009]. But there is another, more fundamental problem: the formulation of the minimum-error Boolean rank-k problem requires *a priori* knowledge about k, the Boolean rank of the decomposition. And as was discussed earlier, using the Hamming distance essentially assumes a roughly equal distribution between destructive and additive noise – something that we hypothesize is often not true in practice. In order to tackle these two problems, we will use the description length instead of the reconstruction error to measure the quality of our decomposition.

Encoding BMF. To use MDL, we have to define what our models $\mathcal{M}$ are, how an $M \in \mathcal{M}$ describes a database, and how we encode these in bits. Miettinen and Vreeken [2014] proposed a number of MDL objective functions for BMF. Here we use their so-called *typed data-to-model* encoding, which was shown to be both the most efficient as well as providing the best empirical performance. Below we give the main ideas of this encoding scheme. For further details we refer the reader to [Miettinen and Vreeken, 2014].

The description length of a binary n-by-m matrix $\boldsymbol{A}$ factorized into $\boldsymbol{B}$ and $\boldsymbol{C}$, such that $\boldsymbol{A} \approx \boldsymbol{B} \circ \boldsymbol{C}$, is defined as

$$L(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) = \underbrace{L(\boldsymbol{B}) + L(\boldsymbol{C})}_{L(M)} + \underbrace{L(\boldsymbol{A} \mid \boldsymbol{B}, \boldsymbol{C})}_{L(D|M)} \quad , \tag{6.1}$$

where $L(\boldsymbol{B}) + L(\boldsymbol{C})$ are the description lengths of the factor matrices and $L(\boldsymbol{A} \mid \boldsymbol{B}, \boldsymbol{C})$ is the description length of the matrix $\boldsymbol{A}$ given this model. The columns of $\boldsymbol{B}$, and analogously the rows of $\boldsymbol{C}$, are encoded independently as binary vectors. One such vector can be identified by two integers: one encoding the number of nonzero elements in $\boldsymbol{B}^i$ (maximum n), and the other encoding the index of $\boldsymbol{B}^i$ among all binary strings having the same profile (maximum

$\binom{n}{|\boldsymbol{B}^i|}$). The number of bits for encoding the k columns of $\boldsymbol{B}$ is hence

$$L(\boldsymbol{B}) = k\log(n) + \sum_{i=1}^{k} \log \binom{n}{|\boldsymbol{B}^i|} \quad .$$

To reconstruct the data $\boldsymbol{A}$ given the factor matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ means we need to describe the error of this factorization. We hence have to encode the exclusive OR of the data with the model, $\boldsymbol{E} = \boldsymbol{A} \oplus (\boldsymbol{B} \circ \boldsymbol{C})$. We can split $\boldsymbol{E}$ into unmodelled 1s, $\boldsymbol{E}^+$, and superfluous 1s, $\boldsymbol{E}^-$, where $\boldsymbol{E}^+_{ij} = 1$ if and only if $\boldsymbol{A}_{ij} > (\boldsymbol{B} \circ \boldsymbol{C})_{ij}$ and $\boldsymbol{E}^-_{ij} = 1$ if and only if $\boldsymbol{A}_{ij} < (\boldsymbol{B} \circ \boldsymbol{C})_{ij}$. To avoid rewarding structure in the error, matrices $\boldsymbol{E}^+$ and $\boldsymbol{E}^-$ are encoded as binary strings in the same way as a column of $\boldsymbol{B}$. Combined, we have $L(\boldsymbol{A} \mid \boldsymbol{B}, \boldsymbol{C}) = L(\boldsymbol{E}^+) + L(\boldsymbol{E}^-)$, where

$$L(\boldsymbol{E}^+) = \log(mn - |\boldsymbol{B} \circ \boldsymbol{C}|) + \log \binom{mn - |\boldsymbol{B} \circ \boldsymbol{C}|}{|\boldsymbol{E}^+|} \quad ,$$

and

$$L(\boldsymbol{E}^-) = \log(|\boldsymbol{B} \circ \boldsymbol{C}|) + \log \binom{|\boldsymbol{B} \circ \boldsymbol{C}|}{|\boldsymbol{E}^-|} \quad .$$

This concludes the definition of our MDL objective function. We can now define the problem we aim to solve in this chapter.

Problem 13 (MDLBMF). Given a binary n-by-m matrix $\boldsymbol{A}$, the Minimum Description Length Boolean Matrix Factorization (MDLBMF) problem is to find binary factor matrices $\boldsymbol{B}$ and $\boldsymbol{C}$ that minimize the total description length (6.1).

Importantly, the rank of the decomposition does not need to be given; instead we automatically find the rank (and decomposition) that minimizes the description length. Although the computational complexity of the MDLBMF problem is unknown, there are strong reasons to believe it is NP-hard [Miettinen and Vreeken, 2014].

6.2 Algorithm (Nassau)

In this section we present Nassau (Algorithm 12), a new algorithm for heuristically solving the MDLBMF problem. The existing algorithms, Panda+ [Lucchese et al., 2014] and Asso [Miettinen et al., 2008], never change already found factors. While this simplifies the search for the (locally) MDL-minimizing factorization [Miettinen and Vreeken, 2014], it also causes earlier blocks to cover large parts of the data, and for higher-rank decompositions these initial blocks can become too coarse-grained. Nassau, on the other hand, iteratively refines previously discovered factors, which allows it to correct earlier made mistakes. It interleaves the addition of new blocks and updating of already-found ones. As new blocks add more fine-grained structure, the coarser older factors become obsolete and can be replaced.

The algorithm starts by finding a set of *seeds* that provide the starting point for finding the factorization (Line 3). After initialization, Nassau starts its

Algorithm 12 `Nassau`

```
Input: $\boldsymbol{A} \in \{0,1\}^{n\times m}$, $t, \tau, \theta \in (0,1)$, $M > 0$
Output: $\boldsymbol{B} \in \{0,1\}^{n\times k}$ and $\boldsymbol{C} \in \{0,1\}^{k\times m}$
 1: function Nassau($\boldsymbol{A}, t, \tau, \theta, M$)
 2:     $\boldsymbol{B}' \leftarrow 0^{n\times 0}$, $\boldsymbol{C}' \leftarrow 0^{0\times m}$
 3:     $Seeds \leftarrow$ GetSeeds($\boldsymbol{A}$)
 4:     repeat
 5:         $\boldsymbol{B} \leftarrow \boldsymbol{B}'$, $\boldsymbol{C} \leftarrow \boldsymbol{C}'$
 6:         $[\boldsymbol{b}, \boldsymbol{c}] \leftarrow$ FindBlock($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, Seeds$)
 7:         $\boldsymbol{B}' \leftarrow [\boldsymbol{B}, \boldsymbol{b}]$, $\boldsymbol{C}' \leftarrow \left[\begin{smallmatrix}\boldsymbol{C}\\ \boldsymbol{c}\end{smallmatrix}\right]$
 8:         if $M$ rounds since last update then
 9:             $[\boldsymbol{B}', \boldsymbol{C}'] \leftarrow$ CyclUpd($\boldsymbol{A}, \boldsymbol{B}', \boldsymbol{C}', Seeds, 0, \theta$)
10:         end if
11:     until $L(\boldsymbol{A}, \boldsymbol{B}', \boldsymbol{C}') \geq L(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$
12:     while not converged do
13:         $[\boldsymbol{B}, \boldsymbol{C}] \leftarrow$ CyclUpd($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, Seeds, t, \theta$)
14:         $t \leftarrow t \cdot \tau$
15:     end while
16:     return $\boldsymbol{B}, \boldsymbol{C}$
17: end function
```

first phase (Lines 4–11), where it repeatedly adds a new block to the existing factorization until the description length does not improve any more. The blocks are found using the `FindBlock` routine (Algorithm 13). To adjust the already-found factors, `Nassau` regularly calls `CyclUpd` (Algorithm 14).

When additional factors do not decrease the objective further, we enter the last phase of the algorithm (Lines 12–15), which entails refinement of the discovered blocks.

`Nassau` accepts several parameters that control its execution. Parameters t and τ control the simulated annealing by giving the initial temperature and update ratio, respectively, while M controls the frequency at which we update the found factors. Hence, at the expense of run time, higher t and lower τ and M potentially provide better results. Parameter θ controls the mining process and will be explained later in this section.

Finding seed columns. Finding a good block to add is a hard problem. Hence, we take an approach similar to [Miettinen et al., 2008, Lucchese et al., 2014], and start by finding a good collection of *seed columns*. The seed column vectors are collected in the n-by-s matrix $Seeds$. Later, the `FindBlock` algorithm will use these seeds to build the final blocks. In principle, we can use any method to create these seeds, including those used by Miettinen et al. [2008], Lucchese et al. [2014], but here we present our approach, which is based on restarted random walks.

A good seed captures the correlation between data rows. Consider an n-by-m binary matrix $\boldsymbol{A}$ and the corresponding bipartite graph $G = (V_{\text{rows}} \cup V_{\text{cols}}, E)$. The correlated rows of $\boldsymbol{A}$ correspond to the highly interconnected nodes in V_{rows}, and if two nodes are correlated, then a short random walk starting from one of them is likely to frequently visit the other [Shahaf and Guestrin, 2012].

Algorithm 13 `FindBlock`

```
Input: matrices $A \in \{0,1\}^{n \times m}$, $B \in \{0,1\}^{n \times k}$, $C \in \{0,1\}^{k \times m}$, $Seeds \in \{0,1\}^{n \times s}$,
    $\theta \in (0,1)$
Output: block $\langle b^*, c^* \rangle$
 1: function FindBlock($A, B, C, Seeds, \theta$)
 2:     $b^* \leftarrow 0^n, c^* \leftarrow 0^m$
 3:     for $i = 1$ to $s$ do
 4:         $b \leftarrow Seeds^i$                                   ▷ The $i$th seed column.
 5:         repeat
 6:             $c \leftarrow \arg\max_{c \in \{0,1\}^m}$ cover($A, [B, b], \left[\begin{smallmatrix} C \\ c \end{smallmatrix}\right], \theta$)
 7:             $b \leftarrow \arg\max_{b \in \{0,1\}^n}$ cover($A, [B, b], \left[\begin{smallmatrix} C \\ c \end{smallmatrix}\right], \theta$)
 8:         until stopping criteria are satisfied
 9:         if $L(A, [B, b], \left[\begin{smallmatrix} C \\ c \end{smallmatrix}\right]) < L(A, [B, b^*], \left[\begin{smallmatrix} C \\ c^* \end{smallmatrix}\right])$ then
10:             $b^* \leftarrow b, c^* \leftarrow c$
11:         end if
12:     end for
13:     return $b^*, c^*$
14: end function
```

If we restart each walk from the same node frequently, the fraction of time we spend on each node indicates how related it is to the origin of the walk.

Let $P(i \mid j)$ be the probability of reaching node i from node j. We do one random walk for each node on the left part of the graph, and on every step we have a fixed probability ϵ to return back to the starting node. Otherwise, we go to one of the neighbouring nodes, selecting them uniformly at random. The fraction of time we spend in node i in a random walk starting from node v is

$$\Pi_v(i) = \epsilon \cdot \mathbf{1}(v = i) + (1 - \epsilon) \sum_{(i,j) \in E} \Pi_v(j) P(i|j) \ .$$

The above equation can be solved using the dominant eigenvector, similar to [Shahaf and Guestrin, 2012]. We generate one seed for each data row v, and the seed has 1 on every row where the random walk starting from v spends sufficient time.

`FindBlock`. Given the input matrix and the current factorization, `FindBlock` (Algorithm 13) finds a new block to be added to the factorization.

The algorithm tries every seed one by one, and uses alternating updates. To compute the updates, we ignore all the locations where the existing factorization has a non-zero, as we cannot change that value. Then, given a fixed column factor $\boldsymbol{b}$, we build the corresponding row factor $\boldsymbol{c}$ by testing for every data column, if using $\boldsymbol{b}$ to cover that column would improve the `cover` function:

$$\begin{aligned} \texttt{cover}(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \theta) = \theta \left|\{(i,j) : a_{ij} = 1 \wedge (\boldsymbol{B} \circ \boldsymbol{C})_{ij} = 1\}\right| \\ - \left|\{(i,j) : a_{ij} = 0 \wedge (\boldsymbol{B} \circ \boldsymbol{C})_{ij} = 1\}\right| \ . \end{aligned}$$

We use the `cover` function as a surrogate to the full description length, partially to make this update step faster, and partially to avoid some local optima. Further local optima can be avoided by adjusting parameter θ; setting $\theta = 1$ returns locally optimal blocks, but we can introduce some locally nonoptimal decisions (that can yield to better global behaviour) by setting θ to a slightly larger (or smaller) value.

Updating the factors. As was mentioned above, earlier found blocks might become outdated – that is, they become less useful in covering data, or they can even become redundant. This happens if we have added new blocks that overlap with previous ones and cover the same data better. In these cases we can improve the factorization by updating the old blocks. `CyclUpd` (Algorithm 14) iteratively replaces blocks with better alternatives. This is performed by first removing a block, and then finding a replacement by calling `FindBlock` with the current factors (Line 4).

Note that the objective is not strictly decreasing: `FindBlock` is heuristic and is not guaranteed to improve the result. However, we might still want to keep the update to avoid getting stuck in a local minimum. In order to accomplish this, we apply a technique similar to simulated annealing. We use a temperature parameter t, which controls the probability of accepting a new block that does not improve the score. If a new block decreases the objective, we always accept it; otherwise we will only do so with a probability proportional to t. Notice that `Nassau` calls `CyclUpd` with non-zero t only in its last simulated annealing phase.

Computational complexity. `Nassau` does most of the work inside `CyclUpd`, which is called every M iterations before all blocks are found (Line 9), and then once again when the number of blocks is fixed (Line 13), making a total of $k/M + 1$ calls. `CyclUpd` in turn calls `FindBlock` $O(k)$ times. `FindBlock` tries every seed vector, and for each one builds (approximately) a corresponding cover in time $O(nm)$. Given that we have n seeds, this yields a complexity of $O(kn^2m)/M$ for `CyclUpd`. Finding the seed columns is done using restarted random walks [Shahaf and Guestrin, 2012], which takes time $O(n(n+m))$. Thus, the complexity of `Nassau` is $O(k/M)O(kn^2m) + O(n(n+m)) = O(k^2n^2m/M)$ (assuming $M < k$). It is worth noting that the number of blocks k is rather small and rarely exceeds 100, which, combined with a relatively small constant hidden behind the asymptotics, makes the algorithm a practical choice for most real-world applications.

6.3 Experimental Evaluation

In this section we empirically evaluate `Nassau` and compare its performance to that of two competing algorithms.

6.3.1 Algorithms and parameters

We used three algorithms: `Nassau`,[1] `Panda+` [Lucchese et al., 2014], and `Asso` [Miettinen et al., 2008]. For `Asso`, we followed the approach by Miettinen and Vreeken [2014] to compute the MDL-minimizing factorization. These algorithms take various parameters. For `Nassau` we used the same parameters in all experiments, and we set the initial annealing temperature $t = 0.8$,

[1]The `Nassau` code is available for research purposes at `http://cs.uef.fi/~pauli/nassau/`; for the other two algorithms, we used code provided by their respective authors.

Algorithm 14 `CyclUpd`

```
Input: matrices $\boldsymbol{A} \in \{0,1\}^{n\times m}$, $\boldsymbol{B} \in \{0,1\}^{n\times k}$, $\boldsymbol{C} \in \{0,1\}^{k\times m}$, $Seeds \in \{0,1\}^{n\times s}$,
    $t > 0, 0 < \theta < 1$
Output: Factors $\boldsymbol{B}^* \in \{0,1\}^{n\times k}$ and $\boldsymbol{C}^* \in \{0,1\}^{k\times m}$
 1: function CyclUpd($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, Seeds, t, \theta$)
 2:     $\boldsymbol{B}^* \leftarrow \boldsymbol{B}, \boldsymbol{C}^* \leftarrow \boldsymbol{C}$
 3:     for $l = 1$ to $k$ do
 4:         $[\boldsymbol{b}, \boldsymbol{c}] \leftarrow$ FindBlock($\boldsymbol{A}, \boldsymbol{B}^{-l}, \boldsymbol{C}_{-l}, Seeds, \theta$)
 5:         $\boldsymbol{B}' \leftarrow [\boldsymbol{B}^{-l}, \boldsymbol{b}], \boldsymbol{C}' \leftarrow \left[\begin{smallmatrix}\boldsymbol{C}_{-l}\\ \boldsymbol{c}\end{smallmatrix}\right]$        ▷ Replace block
 6:         if $L(\boldsymbol{A}, \boldsymbol{B}', \boldsymbol{C}') < L(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ then
 7:             $\boldsymbol{B} \leftarrow \boldsymbol{B}', \boldsymbol{C} \leftarrow \boldsymbol{C}'$
 8:         else
 9:             $\boldsymbol{B} \leftarrow \boldsymbol{B}', \boldsymbol{C} \leftarrow \boldsymbol{C}'$ with probability $t$
10:         end if
11:         if $L(\boldsymbol{A}, \boldsymbol{B}', \boldsymbol{C}') < L(\boldsymbol{A}, \boldsymbol{B}^*, \boldsymbol{C}^*)$ then
12:             $\boldsymbol{B}^* \leftarrow \boldsymbol{B}, \boldsymbol{C}^* \leftarrow \boldsymbol{C}$
13:         end if
14:     end for
15:     return $\boldsymbol{B}^*, \boldsymbol{C}^*$
16: end function
```

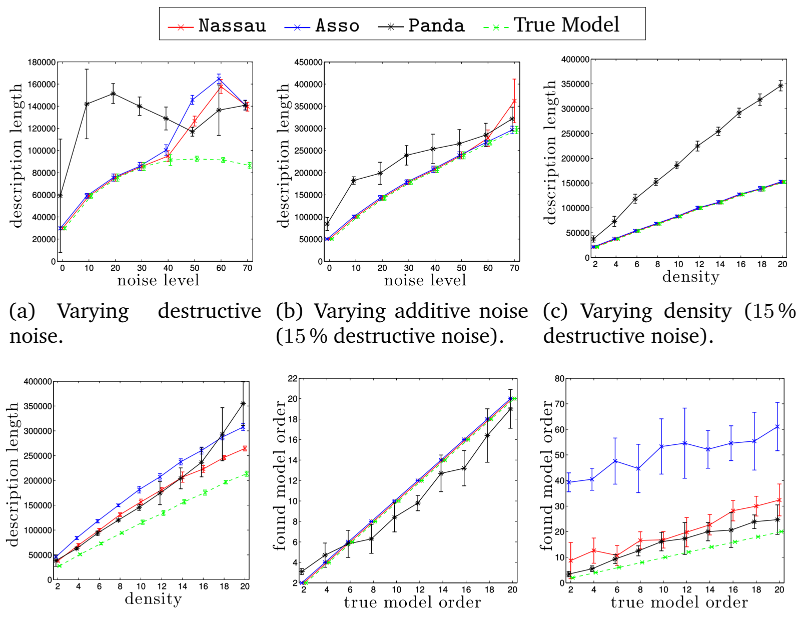
temperature scaling $\tau = 0.6$, weight $\theta = 1.1$, and launched the cyclic updates every $M = 5$ rounds. `Asso` takes one parameter, the rounding threshold τ. For synthetic experiments, we tried values from 0 to 1 at increments of 0.1; for real-world experiments, we followed the setup by Miettinen and Vreeken [2014], and tried the values from 0.1 to 0.95 with increments of 0.025. In all cases, we selected the value of τ that yields the minimal MDL cost factorization. For `Panda+`, we minimize the description length (i.e. the J_E error function) and use randomization with 10 re-starts. Other parameters were left at their default values.

`Panda+` optimizes an encoding that differs slightly to the one we focus on. For a fair comparison we thus ran a set of baseline experiments with both encodings. This, however, did not give results that were different in any significant way, and hence we do report these results separately – the fact that the encoding we use gives very similar results to the encoding used by `Panda+` was also noted by Miettinen and Vreeken [2014].

6.3.2 Synthetic experiments

We first use synthetic data to test the algorithms on data of known ground truth and characteristics. To that end, we generated 1000-by-800 matrices by first creating two binary factor matrices $\boldsymbol{B}^*$ and $\boldsymbol{C}^*$, computing their product $\boldsymbol{A} = \boldsymbol{B}^* \circ \boldsymbol{C}^*$, and applying noise to $\boldsymbol{A}$ in order to obtain the final input matrix $\tilde{\boldsymbol{A}}$. In addition to the results from the algorithms, we also report the results obtained with the original factor matrices $\boldsymbol{B}^*$ and $\boldsymbol{C}^*$. We call this the `True Model`. The amount of noise is measured in percentages of non-zeros in $\boldsymbol{A}$. That is, $p\%$ of destructive noise flips an expected $p\,|A|\,/100$ of the 1s of $\boldsymbol{A}$, while $p\%$ of additive noise flips an expected, $p\,|A|\,/100$ of the 0s.

When generating the matrices, we varied one parameter and kept the rest



(a) Varying destructive noise. (b) Varying additive noise (15 % destructive noise). (c) Varying density (15 % destructive noise).

(d) Varying density (50 % destructive noise). (e) True and estimated model order (15 % destructive noise). (f) True and estimated model order (50 % destructive noise).

Figure 6.1: Results on synthetic data. Markers are mean values over 10 repetitions, error bars are twice the standard deviation. The noise level (both additive and destructive) is relative to the number of 1s.

fixed. In particular, we evaluated different levels of additive and destructive noise, density of the matrix, and rank (model order). All results are averaged over 10 instances per configuration. When not varied, the rank was kept at 15, the density at 8 %, and additive noise at 3 %. To evaluate how well the algorithms perform with increasing amounts of destructive noise, we ran two sets of experiments, one with 15 % destructive noise and one with 50 %.

Destructive noise. We start by testing the effects of the destructive noise, shown in Figure 6.1(a). Initially Nassau and Asso are indistinguishable from the true model, while Panda+ gives a much higher description length. At around 40 % destructive noise, however, Nassau starts to perform better than Asso, and continues in that way until the end. Meanwhile, we observe that while Panda+ underperforms overall, it does relatively well when the data becomes very sparse (high levels of destructive noise). This may mean its search procedure is well-equipped to find blocks in sparse data.

Additive noise. The purpose of this test is to find out how robust the algorithms are to additive noise, that is when 0s are turned to 1s. Keeping the destructive noise constant at 15 %, we find that up to 60 % additive noise both Nassau and Asso still find models virtually indistinguishable from the true model, while Panda+ is consistently worse except at 70 % noise (Figure 6.1(b)).

Varying density. We varied the density of the noise-free matrices from 2 % to 20 %. At 15 % destructive noise (Fig. 6.1(c)), Nassau and Asso discover models on par with the true model, while Panda+ is clearly much worse. With 50 % destructive noise (Fig. 6.1(d)), Nassau and Panda+ are virtually indistinguishable until the density reaches 16 %, after which the performance of Panda+ starts to degrade.

Model order. Last, we varied the model order (rank) between 2 and 20. With 15 % destructive noise, Nassau and Asso obtained exact results, while Panda+ was mostly under-estimating (Fig. 6.1(e)). With 50 % destructive noise (Fig. 6.1(f)), Nassau overestimates slightly more than Panda+, although for both methods we see occasional high variance.

6.3.3 Real-world experiments

Now that we have verified that Nassau can recover structure from synthetically generated data, it is time to test it on real-world data. We start by describing the datasets used for the experiments, most of which are publicly available. Below we give a short description for each, and an overview of their properties is shown in Table 6.1.

Abstracts represents the words in all abstracts of the papers accepted at the ICDM conference up to 2007, where the words have been stemmed and stop words removed.[2]

DBLP conf. contains records of 6980 authors publishing in 19 conferences. The dataset is collected from the DBLP database and it is pre-processed as by Miettinen [2009].[3] DBLP co-auth. is a (symmetric) co-authorship matrix of a subset of the authors in the DBLP conf. data.

Dialect contains presence data of dialectical linguistic properties in 506 Finnish municipalities [Embleton and Wheeler, 1997, 2000].

DNA Amp. contains information on DNA copy number amplifications. Such copies activate oncogenes and are hallmarks of nearly all advanced tumours [Myllykangas et al., 2006]. Amplified genes represent attractive targets for therapy and prognosis.

Firewall 2 describes the reachability between two IP addresses [Ene et al., 2008].[4] It has an exact Boolean decomposition of 10 factors [Ene et al., 2008], and hence should be highly compressible.

Mushroom contains edibility records of mushrooms [Frank and Asuncion, 2010].

We will also use some of the datasets described in previous chapters. Mammals and 4News were first introduced in Section 3.4.3 and Paleo in Section 5.4.3.

Quantitative evaluation of real-world results. We start by studying the compression ratios of the real-world data sets. By compression ratio, we mean the description length L obtained by an algorithm divided by the description length of the data using an empty model ($L_\emptyset$), i.e. when no factors are used to represent it. The smaller this ratio is, the better the discovered factorization compresses the data. The results are presented in Table 6.2.

[2]Available upon request from the author [De Bie, 2011], accessed 30 October 2014
[3]http://www.informatik.uni-trier.de/~ley/db/, accessed 30 October 2014
[4]http://www.hpl.hp.com/personal/Robert_Schreiber/, accessed 30 October 2014

Table 6.1: Real-world dataset overview. $L_\emptyset$ is the description length in bits with the empty model (without any factors).

Dataset	$Rows$	$Columns$	$\%1s$	$L_\emptyset$
4News	400	800	3.5	70379
Abstracts	859	3 933	1.2	319468
DBLP co-auth.	2345	2 345	0.5	244754
DBLP conf.	6980	19	13.0	73785
Dialect	1334	506	16.1	430435
DNA Amp.	4590	392	1.5	199429
Firewall 2	325	590	19.0	134546
Mammals	2670	194	16.1	330302
Mushroom	8192	112	19.3	650373
Paleo	501	139	5.1	20223

Table 6.2: Compression ratio $L\% = 100 \times L/L_\emptyset$ (smaller is better) and model order k for the real-world datasets.

	Nassau		Panda+		Asso	
Dataset	$L\%$	k	$L\%$	k	$L\%$	k
4News	93.1	12	**92.7**	5	93.6	17
Abstracts	95.3	3	**86.7**	128	97.2	19
DBLP conf.	90.3	3	92.4	3	**90.0**	4
DBLP co-auth.	**94.1**	60	95.9	11	95.8	178
Dialect	**42.0**	30	57.3	17	48.8	37
DNA Amp.	**43.6**	100	63.4	20	49.8	58
Firewall 2	2.4	6	2.7	8	**1.7**	5
Mammals	**54.5**	29	66.8	8	64.6	17
Mushroom	72.6	4	63.6	15	**50.6**	59
Paleo	**89.7**	15	91.2	3	91.4	19

Out of the ten datasets in Table 6.2, Nassau obtains the best compression ratio in five – and in each of these cases it outperforms the second-best result by at least one percentage point. Panda+ is the best for the two sparse text datasets, 4News and Abstracts, although for 4News, Nassau is only 0.4 and Asso only 0.9 percentage points worse. For all practical purposes, we would consider 4News a tie. Similarly, Asso is better in DBLP conf., but only by 0.3 percentage points, more clearly in Firewall 2, and by a wide margin in Mushroom.

If we consider the two cases where Nassau is significantly worse than the best method, Abstracts (against Panda+) and Mushroom (against Asso), we notice that in both cases Nassau reports significantly smaller model order than the best method (3 vs. 128 and 4 vs. 59, respectively). It seems that in these cases the random walks were unable to find good seed vectors. Indeed, when we re-ran Nassau on Mushroom but generating the seeds as Asso does, we obtained a compression ratio of 56.4 with 44 factors – a significant improvement confirming that the random walks do not necessarily produce good seeds for every data set.

Scalability. The implementations of these methods are not fully comparable. Panda+ is a full-C implementation, Asso is a mixture of C and Matlab, while Nassau is written purely in Matlab. Also, the different methods use different

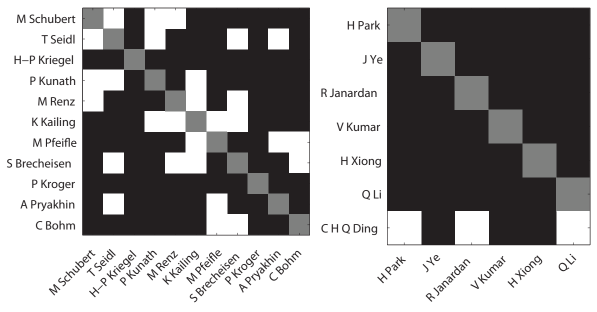


Figure 6.2: Example submatrices of DBLP co-auth. as selected by Nassau. Black cells indicate joint papers, grey cells stand for self-loops.

levels of parallelization, and the running time for all these algorithms is heavily influenced by the eventual model order. For Asso, the number of different rounding thresholds τ to try, and for Panda+, the number of random restarts, also have obvious effects. That said, in our experiments, Panda was generally the fastest, followed by Nassau and Asso, but the order could vary from dataset to dataset. All methods were able to finish all datasets within 24 hours, except for Asso with DBLP co-auth. , which took more than four days.

Qualitative evaluation of real-world results. We studied two datasets, DBLP co-auth. and Mammals, more carefully in order to understand the types of results we can obtain from Nassau (and its competitors). As Table 6.2 shows, DBLP co-auth. is not very compressible, while Mammals is reduced to almost half of its original size.

The DBLP co-auth. data is a graph, i.e. a symmetric matrix, and while none of the methods tested here assume symmetry, we would expect that a good factorization would be (approximately) symmetric. This, however, is not the case for Panda+ or Asso: in symmetric decompositions, the two factor matrices would be the same, but for Asso, $1.2\,\%$ of the values differ, and for Panda+, $1.6\,\%$ of the values in the factor matrices differ. For Nassau, only $0.3\,\%$ of the values are different.

Looking at the factors, we observe that in particular Asso finds very skewed blocks that have many rows but few columns, or vice versa, being clearly unable to capitalize on the symmetry of the input. Panda+ returns more square-like blocks, but they still have almost four times as many rows as columns (or vice versa) on average. The patterns returned by Nassau have an average rows/columns ratio of 1.7, thereby being the most square-like.

We show two exemplary factors found by Nassau in the DBLP co-auth. data in Figure 6.2. It shows the known collaborations between the authors in a factor. Here, Nassau has identified two (quasi-) cliques of famous data mining and machine learning researchers with strong collaboration patterns.

Both Panda+ and Asso perform so-called *hierarchical* factorization (rank-$(k-1)$ factorization is part of the rank-k factorization), while Nassau is designed to avoid that. We studied these different behaviours with the Mammals data and

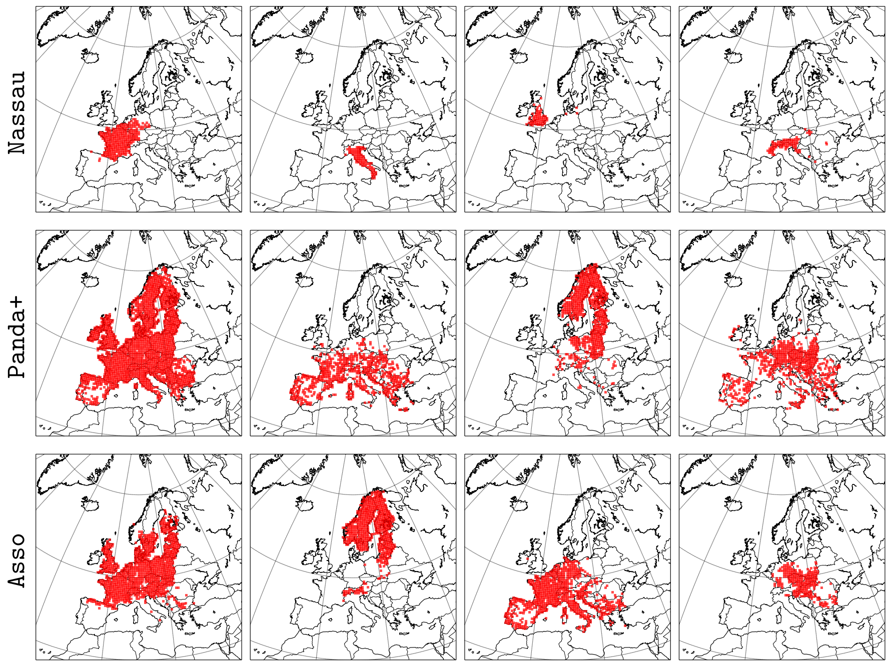


Figure 6.3: The first four factors discovered on the Mammals data using, from top to bottom, `Nassau`, `Panda+`, and `Asso`.

using the first four left-hand-side factors (i.e. columns of $\boldsymbol{B}$) for each method. As the rows of the data correspond to locations in Europe, we visualize the factors using maps in Figure 6.3. For `Nassau` the first four factors are very compact and correspond very closely to countries in Europe. For `Panda+`, the first factor covers almost the whole area of Europe that was part of the data, and the other three factors also cover large, not very well-defined areas. The same holds true for `Asso`, although to a slightly lesser extent. In other words, the results by `Nassau` in Figure 6.3 correspond to species that are very specific to certain areas, while both `Panda+` and `Asso` selected common species.

6.4 Related Work

In BMF one decomposes a binary matrix into the Boolean product of two matrices while minimizing some cost function. Perhaps the intuitively most straightforward objective is to minimize the number of errors, i.e. the Frobenius norm of the residual. The `Asso` algorithm to solve BMF was proposed by Miettinen et al. [2008]. Later, Lu et al. [2008] proposed a heuristic based on a mixed-integer-programming formulation.

Error minimization is prone to overfitting, however, as more factors always allow better reconstruction. In practice, users thus have to choose the number of factors in advance. Moreover, this objective builds on the assumption that noise is equally likely to flip true 1s to 0s as it is to flip true 0s to 1s, which we argue here is not realistic.

As discussed by Faloutsos and Megalooikonomou [2007], Kolmogorov Complexity and its practical implementation, the Minimum Description Length principle [Rissanen, 1978, Grünwald, 2007], are powerful, well-founded, and natural approaches to data mining, as they allow us to identify the most succinct and least redundant model for a dataset. MDL has been successfully employed for a wide range of data mining tasks, including discretization [Fayyad and Irani, 1993], outlier detection [Smets and Vreeken, 2011], classification [Quinlan, 1993], and clustering [Mampaey and Vreeken, 2013].

Miettinen and Vreeken [2011, 2014] formulated the BMF problem in terms of the Minimum Description Length (MDL) principle [Rissanen, 1978, Grünwald, 2007] in order to solve the model order selection problem. Loosely speaking, this objective identifies the best factorization as the one that provides the best lossless compression of the data – thus automatically balancing the complexity of the factorization with the reconstruction error. The authors applied their score as post-processing for `Asso` to identify the optimal model order.

Tiling [Geerts et al., 2004] is closely related to BMF, but aims at finding submatrices full of 1s. Xiang et al. [2011] proposed an algorithm to mine noisy tiles – i.e. allowing 0s in the area covered by a tile. Given a collection of noisy tiles, Kontonasios and De Bie [2010] iteratively discover the most interesting tile, defining interestingness through the maximum entropy. Tatti and Vreeken [2012] use MDL to hierarchically identify the most informative tile from data with ordered rows and columns.

Closer to this work is the `Panda` algorithm, proposed by Lucchese et al. [2010]. `Panda` performs Boolean matrix factorization, but instead of minimizing only the error, it minimizes the sum of Frobenius norms of the residual and the factor – by which the hidden assumption again is that false positives and false negatives are equally likely. Recently, the same authors proposed `Panda+` [Lucchese et al., 2014] in which they optimize the TypedXOR encoding of [Miettinen and Vreeken, 2011].

CHAPTER 7

Conclusions

In this thesis we discussed matrix factorizations over idempotent semirings, also known as dioids, and their applications in data mining. What sets arising structures apart from those based on the standard algebra, is that at any given point of the decomposition the feature that has the largest value always dominates the rest. This means that every element of the reconstructed matrix is determined by a single pattern that has the largest value at this point. We call this the “winner-takes-it-all” property, which can be contrasted with the “parts-of-whole” interpretation of NMF. This allows us to discover dominant features in the data in a more pure form than when using the standard algebra.

Using dioids for data analysis often yields patterns that are complementary to those found using classical methods, such as NMF. Depending on the data, the reconstruction error can be better or worse, but the very different kinds of structure that we are after makes this approach attractive from the interpretability point of view. Thus, dioids provide an alternative view on the data, which enables discovering patterns that are hard to identify using the normal algebra. This means that data analysts can use both the classical and subtropical factorizations to get a broader understanding of the kinds of patterns that are present in the data.

We discussed several types of dioids in detail, namely the subtropical, tropical, and Boolean algebras. In addition, two more associated data models were introduced: the mixed linear-tropical and logistic-tropical models. The former allows to represent the data as a mixture of the NMF and subtropical structures. Our experiments revealed that NMF and the subtropical matrix factorization are often complementary to each other, and combining them tends to yield better results than either of them on its own. The purpose of the latter model is to use the tropical algebra for community mining in graphs. Five novel matrix factorization algorithms were proposed: `Capricorn` and `Cancer` for the subtropical algebra, `Latitude` for the mixed linear-tropical structure, `SLTF` for the logistic-tropical factorization, and `Nassau` for the Boolean algebra.

Subtropical low-rank factorizations are a novel method for finding latent structure from nonnegative data, that allow interpreting it using the winner-takes-it-all approach mentioned above. Working in the subtropical algebra is harder than in the normal algebra, though. For example, alternative definitions of rank do not agree, and computing many of them – including the subtropical Schein rank, which is arguably the most useful one for data analysis – is computationally hard. That said, based on the synthetic experiments, our proposed algorithms, `Capricorn` and `Cancer`, can find the subtropical structure when it is present in the data, and therefore our hypothesis is that a failure to find a good factorization more probably indicates the lack of the subtropical structure rather than the algorithms' failure. Naturally, more experiments using data with known subtropical structure should improve our confidence in the correctness of the hypothesis.

Although both NMF and the subtropical model can produce useful patterns in the data, having to choose between the two is not always desirable. Indeed, real-world data rarely follows any mathematical model exactly, and trying to enforce it can make algorithms miss important patterns. We presented a new linear-tropical model that allows to mix the NMF and subtropical structures. By smoothly combining factorizations over two different algebras, it allows us to model complex data in an interpretable way. In the experiments, our method, `Latitude`, was able to consistently obtain better reconstruction errors than either NMF or the subtropical matrix factorization algorithms. Indeed, `Latitude` was often better than even SVD. And while SVD comes with well-known limitations to interpretability, `Latitude`'s factorizations are easier to interpret due to the nonnegative factor matrices and intuitive interpretation of the parameter vectors.

While `Latitude` generally showed superior performance compared to NMF or the subtropical matrix factorization, there were a few instances where it performed slightly worse, which was due to overfitting to the noise. This raises the question of the use of regularization in mixed linear-tropical factorization and is left for future studies.

When it comes to interpreting binary data, Boolean matrix factorization is a natural choice since it produces binary factors and allows for the elementwise "yes"/"no" interpretation. Traditionally, BMF methods focused on finding decompositions that minimize the Frobenius reconstruction error. This, however, does not take into account that the balance between additive and destructive noise in real-world data might be heavily skewed. We address this issue by using the so called Minimum Description Length principle, which postulates that a model yielding the shortest encoding of the data should be preferred. While this approach was known before, we propose a new algorithm, called `Nassau`, that directly optimizes the description length. Unlike most of the existing BMF algorithms, `Nassau` is non-hierarchical, meaning that an optimal rank-$(k-1)$ decomposition might not be a part of any optimal rank-k decomposition. This allows it to revisit earlier found parts of the factorization and correct previously made mistakes. Empirical evaluation on synthetic data shows that `Nassau` is very robust to high levels of destructive noise. Experiments on real-world data demonstrate that it finds more compact and intuitive factors than state-of-the-art methods, which are more inclined to "simply" cover large parts of the data.

Finally, we applied the dioid structures to community mining by using the rounding rank characterization of nested matrices together with the tropical algebra. Expressing the community mining as a continuous problem allowed us to use optimization methods, such as stochastic gradient descent, that would otherwise not be applicable to the combinatorial domain. The use of SGD also allows us to utilize the vast literature on parallel and distributed implementations and create a scalable algorithm, which we called SLTF. For this work, we have only studied a shared-memory parallel implementation, but a distributed approach is equally viable. It can also solve the biggest efficiency bottleneck of our approach, namely that the factor matrix $\boldsymbol{B}$ is dense and storing it is memory-intensive for larger matrices.

Being mainly interested in undirected graphs, we designed SLTF for symmetric matrices. An asymmetric version would not be a significant change, though, and could be used on directed or bipartite graphs (e.g. locations-by-species matrices).

All algorithms presented in this thesis are heuristics, which is a consequence of the hardness of the problems they are trying to solve. Developing algorithms that achieve better reconstruction error, as well as improving the scalability, is naturally an important direction of future work.

Bibliography

Marianne Akian. Densities of idempotent measures and large deviations. *Trans. Amer. Math. Soc.*, 351(11):4515–4543, 1999.

Marianne Akian, Ravindra Bapat, and Stéphane Gaubert. Max-Plus Algebra. In Leslie Hogben, editor, *Handbook of Linear Algebra*. Chapman & Hall/CRC, Boca Raton, 2007.

Marianne Akian, Stéphane Gaubert, and Alexander Guterman. Linear independence over tropical semirings and beyond. *Contemp. Math.*, 495:1–38, 2009.

Jesús Angulo and Santiago Velasco-Forero. Non-negative sparse mathematical morphology. *Adv. Imag. Elect. Phys.*, 202:1–37, 2017.

Miguel Araujo, Stephan Günnemann, Gonzalo Mateos, and Christos Faloutsos. Beyond Blocks: Hyperbolic Community Detection. In *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 50–65, 2014.

Miguel Araujo, Pedro Ribeiro, and Christos Faloutsos. Faststep: Scalable boolean matrix decomposition. In *The Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 461–473, 2016.

François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and Linearity: An Algebra for Discrete Event Systems*. John Wiley & Sons, Hoboken, New Jersey, 1992.

R.B. Bapat, David P. Stanford, and P. Van den Driessche. Pattern properties and spectral inequalities in max algebra. *SIAM J. Matrix Anal. Appl.*, 16(3): 964–976, 1995.

Michael W. Berry, Murray Browne, Amy N. Langville, V. Paul Pauca, and Robert J. Plemmons. Algorithms and applications for approximate nonnegative matrix factorization. *Comput. Stat. Data Anal.*, 52(1):155–173, 2007.

Vincent D. Blondel, Stéphane Gaubert, and John N. Tsitsiklis. Approximating the spectral radius of sets of matrices in the max-algebra is NP-hard. *IEEE Trans. Autom. Control*, 45(9):1762–1765, 2000.

Stephen P. Borgatti and Martin G. Everett. Models of core/periphery structures. *Soc. Networks*, 21:375–395, 1999.

Jean-Philippe Brunet, Pablo Tamayo, Todd R Golub, and Jill P Mesirov. Metagenes and molecular pattern discovery using matrix factorization. *Proc. Natl. Acad. Sci. U.S.A.*, 101(12):4164–4169, 2004.

Peter Butkovič. Max-algebra: The linear algebra of combinatorics? *Linear Algebra Appl.*, 367:313–335, 2003.

Peter Butkovič. *Max-linear systems: Theory and algorithms*. Springer Science & Business Media, New York, 2010.

Peter Butkovič and Gábor Hegedüs. An elimination method for finding all solutions of the system of linear equations over an extremal algebra. *Ekon.-Mat. Obzor*, 20(2):203–215, 1984.

Peter Butkovič and Ferdinand Hevery. A condition for the strong regularity of matrices in the minimax algebra. *Discrete Appl. Math.*, 11(3):209–222, 1985.

Christos G. Cassandras and Stéphane Lafortune. *Introduction to discrete event systems*. Springer, Berlin, second edition, 2008.

Ervina Cergani and Pauli Miettinen. Discovering relations using matrix factorization methods. In *CIKM*, pages 1549–1552, 2013.

Andrzej Cichocki, Rafal Zdunek, Anh Huy Phan, and Shun'ichi Amari. *Nonnegative matrix and tensor factorizations: Applications to exploratory multi-way data analysis and blind source separation*. John Wiley & Sons, Chichester, 2009.

Guy Cohen, Stéphane Gaubert, and Jean-Pierre Quadrat. Max-plus algebra and system theory: Where we are and where to go now. *Annu Rev Control*, 23: 207–219, January 1999.

Joel E. Cohen and Uriel G. Rothblum. Nonnegative ranks, decompositions, and factorizations of nonnegative matrices. *Linear Algebra Appl.*, 190:149–168, 1993.

Raymond A. Cuninghame-Green. *Minimax Algebra*. Springer, Berlin, 1979.

Timothy A. Davis and Yifan Hu. The university of Florida sparse matrix collection. *ACM Trans Math Soft*, 38(1):1–25, 2011.

Tijl De Bie. Maximum entropy models and subjective interestingness: an application to tiles in binary databases. *Data Mining and Knowledge Discovery*, 23(3):407–446, 2011.

Bart De Schutter and Bart De Moor. The QR decomposition and the singular value decomposition in the symmetrized max-plus algebra revisited. *SIAM Rev.*, 44(3):417–454, 2002.

Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *J. Am. Soc. Inf. Sci.*, 41:391–407, 1990.

Amir Dembo and Ofer Zeitouni. *Large deviations techniques and applications*. Springer, Berlin, 2010.

Sheila M. Embleton and Eric S. Wheeler. Finnish dialect atlas for quantitative studies. *J. Quant. Linguist.*, 4(1–3):99–102, 1997.

Sheila M. Embleton and Eric S. Wheeler. Computerized dialect atlas of Finnish: Dealing with ambiguity. *J. Quant. Linguist.*, 7(3):227–231, 2000.

Alina Ene, William Horne, Nikola Milosavljevic, Prasad Rao, Robert Schreiber, and Robert E Tarjan. Fast exact and heuristic methods for role minimization problems. In *ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 1–10, 2008.

Christos Faloutsos and Vasilis Megalooikonomou. On data mining, compression and Kolmogorov complexity. *Data Min. Knowl. Discov.*, 15(1):3–20, 2007.

Usama Fayyad and K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 1022–1027, 1993.

Mikael Fortelius et al. Neogene of the old world database of fossil mammals (NOW), 2003. `http://www.helsinki.fi/science/now/` Accessed Oct 30 2014.

A. Frank and A. Asuncion. UCI machine learning repository, 2010. `http://archive.ics.uci.edu/ml` Accessed Oct 30 2014.

Esther Galbrun, Aristides Gionis, and Nikolaj Tatti. Overlapping community detection in labeled graphs. *Data Min. Knowl. Discov.*, 28(5-6):1586–1610, September 2014.

Bernd Gärtner and Martin Jaggi. Tropical support vector machines. Technical Report ACS-TR-362502-01, 2008.

Stéphane Gaubert. *Théorie des systèmes linéaires dans les dioïdes*. PhD thesis, Ecole nationale supérieure des mines de Paris, 1992.

Stephane Gaubert. Methods and applications of (max,+) linear algebra. In *14th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 261–282. Springer, 1997.

Floris Geerts, Bart Goethals, and Taneli Mielikäinen. Tiling databases. In *International Conference on Discovery Science (DS)*, pages 278–289, 2004.

Athinodoros S Georghiades, Peter N Belhumeur, and David J Kriegman. From few to many: Generative models for recognition under variable pose and illumination. In *4th IEEE International Conference on Automatic Face and Gesture Recognition (FG)*, pages 277–284, 2000.

Nicolas Gillis. The why and how of nonnegative matrix factorization. *Regularization, Optimization, Kernels, and Support Vector Machines*, 12(257), 2014.

Nicolas Gillis and François Glineur. Using underapproximations for sparse nonnegative matrix factorization. *Pattern Recogn.*, 43(4):1676–1687, 2010.

Pablo M. Gleiser and Leon Danon. Community structure in jazz. *Adv. Complex Syst.*, 6(4):565–573, 2003.

Gene H. Golub and Charles F. Van Loan. *Matrix computations*. Johns Hopkins University Press, Baltimore, 3 edition, 2012.

Michel Gondran and Michel Minoux. *Graphs and algorithms*. John Wiley & Sons, New York, 1984a.

Michel Gondran and Michel Minoux. Linear algebra in dioids: A survey of recent results. *North-Holland Math. Stud.*, 95:147–163, 1984b.

Peter Grünwald. *The Minimum Description Length Principle*. MIT Press, Campridge, MA, 2007.

David Guillamet and Jordi Vitrià. Non-negative matrix factorization for face recognition. In *Topics in artificial intelligence*, pages 336–344. Springer, 2002.

Pierre Guillon, Zur Izhakian, Jean Mairesse, and Glenn Merlet. The ultimate rank of tropical matrices. *J. Algebra*, 437:222–248, 2015.

J. Hook. Linear regression over the max-plus semiring: algorithms and applications. *arXiv:1712.03499*, 2017.

Patrik O. Hoyer. Non-negative Matrix Factorization with Sparseness Constraints. *J. Mach. Learn. Res.*, 5:1457–1469, 2004.

IUCN. The IUCN red list of threatened species. version 2014.1, 2014. `http://www.iucnredlist.org` Accessed June 03 2017.

Sunil K. Jha and R.D.S. Yadava. Denoising by singular value decomposition and its application to electronic nose data processing. *IEEE Sens. J.*, 11(1):35–44, 2011.

Esa Junttila. *Patterns in permuted binary matrices*. PhD thesis, Helsinki University Press, Helsinki, August 2011.

Esa Junttila and Petteri Kaski. Segmented nestedness in binary data. In *SDM '11*, pages 235–246, 2013.

Sanjar Karaev and Pauli Miettinen. Cancer: Another algorithm for subtropical matrix factorization. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 576–592, 2016a.

Sanjar Karaev and Pauli Miettinen. Capricorn: An algorithm for subtropical matrix factorization. In *16th SIAM International Conference on Data Mining (SDM)*, pages 702–710, 2016b.

Sanjar Karaev and Pauli Miettinen. Algorithms for approximate subtropical matrix factorization. *Data Min. Knowl. Discov.*, 33(2):526–576, 2019. doi: https://doi.org/10.1007/s10618-018-0599-1.

Sanjar Karaev, Pauli Miettinen, and Jilles Vreeken. Getting to know the unknown unknowns: Destructive-noise resistant boolean matrix factorization. In *15th SIAM International Conference on Data Mining (SDM)*, pages 325–333, 2015.

Sanjar Karaev, James Hook, and Pauli Miettinen. Latitude: A model for mixed linear–tropical matrix factorization. In *18th SIAM International Conference on Data Mining (SDM)*, pages 360–368, 2018a.

Sanjar Karaev, Saskia Metzler, and Pauli Miettinen. Logistic-tropical decompositions and nested subgraphs. In *4th International Workshop on Mining and Learning with Graphs*, 2018b.

Nirmal Keshava. A survey of spectral unmixing algorithms. *Lincoln laboratory journal*, 14(1):55–78, 2003.

Ki Hang Kim and Fred W. Roush. Factorization of polynomials in one variable over the tropical semiring. Technical Report math/0501167, arXiv, 2005.

Tamara Kolda and Dianne O'Leary. Algorithm 805: Computation and uses of the semidiscrete matrix decomposition. *ACM Trans. Math. Softw.*, 26(3):415–435, 2000.

Kleanthis-Nikolaus Kontonasios and Tijl De Bie. An information-theoretic approach to finding noisy tiles in binary databases. In *SIAM International Conference on Data Mining (SDM)*, pages 153–164, 2010.

Danai Koutra, U Kang, Jilles Vreeken, and Christos Faloutsos. VoG: Summarizing and Understanding Large Graphs. In *SDM '14*, pages 91–99, 2014.

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

Yifeng Li and Alioune Ngom. The non-negative matrix factorization toolbox for biological data mining. *Source Code Biol. Med.*, 8(1):1–15, 2013.

Yongsub Lim, U Kang, and Christos Faloutsos. SlashBurn: Graph Compression and Mining beyond Caveman Communities. *IEEE Trans. Knowl. Data Eng.*, 26 (12):3077–3089, 2014.

Haibing Lu, Jaideep Vaidya, and Vijayalakshmi Atluri. Optimal Boolean Matrix Decomposition: Application to Role Engineering. In *ICDE*, pages 297–306, 2008.

Haibing Lu, Jaideep Vaidya, Vijayalakshmi Atluri, and Yuan Hong. Constraint-Aware Role Mining Via Extended Boolean Matrix Decomposition. *IEEE Trans. Depend. Secure*, 9(5):655–669, 2012.

Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. Mining Top-K Patterns from Binary Datasets in presence of Noise. In *SDM*, page 165–176, 2010.

Claudio Lucchese, Salvatore Orlando, and R. Perego. A Unifying Framework for Mining Approximate Top-k Binary Patterns. *IEEE Trans. Knowl. Data En.*, 26 (12):2900–2913, 2014.

Darryl I MacKenzie, James D Nichols, Gideon B Lachman, Sam Droege, J Andrew Royle, and Catherine A Langtimm. Estimating site occupancy rates when detection probabilities are less than one. *Ecology*, 83(8):2248–2255, 2002.

Michael Mampaey and Jilles Vreeken. Summarising categorical data by clustering attributes. *Data Min. Knowl. Discov.*, 26(1):130–173, 2013.

Heikki Mannila and Evimaria Terzi. Nestedness and segmented nestedness. In *KDD '07*, pages 480–489, 2007.

Viktor Maslov. *Idempotent analysis*. American Mathematical Society, Providence, 1992.

Saskia Metzler, Stephan Günnemann, and Pauli Miettinen. Hyperbolae Are No Hyperbole: Modelling Communities That Are Not Cliques. In *ICDM '16*, pages 330–339, 2016.

Pauli Miettinen. *Matrix decomposition methods for data mining: Computational complexity and algorithms*. PhD thesis, University of Helsinki, 2009.

Pauli Miettinen. Sparse Boolean Matrix Factorizations. In *ICDM*, pages 935–940, 2010.

Pauli Miettinen. Generalized Matrix Factorizations as a Unifying Framework for Pattern Set Mining: Complexity Beyond Blocks. In *ECMLPKDD '15*, pages 36–52, 2015.

Pauli Miettinen and Jilles Vreeken. Model Order Selection for Boolean Matrix Factorization. In *KDD*, page 51–59, 2011.

Pauli Miettinen and Jilles Vreeken. MDL4BMF: Minimum Description Length for Boolean Matrix Factorization. Technical Report MPI–I–2012–5-001, June 2012.

Pauli Miettinen and Jilles Vreeken. MDL4BMF: Minimum description length for Boolean matrix factorization. *ACM Trans. Knowl. Discov. Data*, 8(4):A18, 2014.

Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. The Discrete Basis Problem. *IEEE Trans. Knowl. Data Eng.*, 20(10): 1348–1362, 2008.

A.J. Mitchell-Jones, G. Amori, W. Bogdanowicz, B. Krystufek, P. J. H. Reijnders, F. Spitzenberger, M. Stubbe, J.B.M. Thissen, V. Vohralik, and J. Zima. *The Atlas of European Mammals*. Academic Press, 1999.

Ian Munro. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters*, 1(2):56–58, 1971.

Samuel Myllykangas, J. Himberg, T. Böhling, B. Nagy, Jaakko Hollmén, and S. Knuutila. DNA copy number amplification profiling of human neoplasms. *Oncogene*, 25(55):7324–7332, 2006. ISSN 0950-9232.

Stefan Neumann, Rainer Gemulla, and Pauli Miettinen. What You Will Gain By Rounding: Theory and Algorithms for Rounding Rank. In *ICDM '16*, pages 380–389, 2016.

Pentti Paatero. Least squares formulation of robust non-negative factor analysis. *Chemometr. Intell. Lab.*, 37(1):23–35, 1997.

Pentti Paatero. The multilinear engine—a table-driven, least squares program for solving multilinear problems, including the n-way parallel factor analysis model. *J. Comp. Graph. Stat.*, 8(4):854–888, 1999.

Pentti Paatero and Unto Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5:111–126, 1994.

Bruce D. Patterson and Wirt Atmar. Nested subsets and the structure of insular mammalian faunas and archipelagos. *Biol. J. Linnean Soc.*, 28(1-2):65–82, May 1986.

V Paul Pauca, Farial Shahnaz, Michael W. Berry, and Robert J. Plemmons. Text mining using nonnegative matrix factorizations. In *4th SIAM International Conference on Data Mining (SDM)*, pages 22–24, 2004.

J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan-Kaufmann, Los Altos, California, 1993.

Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(1): 465–471, 1978.

Ankan Saha and Vikas Sindhwani. Learning evolving and emerging topics in social media: a dynamic nmf approach with temporal regularization. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 693–702. ACM, 2012.

Arto Salomaa and Matti Soittola. *Automata-theoretic aspects of formal power series*. Springer Science & Business Media, New York, 2012.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Application of dimensionality reduction in recommender system – a case study. Technical report, GroupLens Research Group, 2000.

Andrew I. Schein, Lawrence K. Saul, and Lyle H. Ungar. A Generalized Linear Model for Principal Component Analysis of Binary Data. In *AISTATS '03*, 2003.

Dafna Shahaf and Carlos Guestrin. Connecting Two (or Less) Dots: Discovering Structure in News Articles. *ACM Trans. Knowl. Discov. Data*, 5(4):A24:1–31, 2012.

Yaroslav Shitov. The complexity of tropical matrix factorization. *Adv. Math.*, 254:138–156, 2014.

Imre Simon. Limited subsets of a free monoid. In *19th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 143–150, 1978.

Imre Simon. On semigroups of matrices over the tropical semiring. *Inform. Theor. Appl.*, 28(3-4):277–294, 1994.

David Skillicorn. *Understanding complex datasets: Data mining with matrix decompositions*. Data Mining and Knowledge Discovery. Chapman & Hall/CRC, Boca Raton, 2007.

Koen Smets and Jilles Vreeken. The odd one out: Identifying and characterising anomalies. In *11th SIAM International Conference on Data Mining (SDM)*, pages 804–815, 2011.

Nathan Srebro, Jason Rennie, and Tommi S Jaakkola. Maximum-margin matrix factorization. In *17th Advances in Neural Information Processing Systems (NIPS)*, pages 1329–1336, 2004.

Nikolaj Tatti and Jilles Vreeken. Discovering descriptive tile trees by fast mining of optimal geometric subtiles. In *The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, pages 9–24, 2012.

Christina Teflioudi, Faraz Makari, and Rainer Gemulla. Distributed Matrix Completion. In *ICDM '12*, pages 655–664, 2012.

U.S. Department of Defense. DoD News Briefing — Secretary Rumsfeld and Gen. Myers, 12 Feb 2002. `http://www.defense.gov/transcripts/transcript.aspx?transcriptid=2636` Accessed 2 Oct 2014.

Matthijs van Leeuwen, Tijl De Bie, Eirini Spyropoulou, and Cédric Mesnage. Subjective interestingness of subgraph patterns. *Machine Learning*, 105(1): 41–75, 2016.

Stephen A. Vavasis. On the complexity of nonnegative matrix factorization. *SIAM J. Optim.*, 20(3):1364–1377, 2009.

N.N. Vorobyev. Extremal algebra of positive matrices. *Elektron. Informationsverarbeitung und Kybernetik*, 3:39–71, 1967.

Elizabeth A Walkup and Gaetano Borriello. A general linear max-plus solution technique. In J. Gunawardena, editor, *Idempotency*, pages 406–415. Cambridge University Press, Cambridge, 1998.

Jörg Wicker, Bernhard Pfahringer, and Stefan Kramer. Multi-Label Classification Using Boolean Matrix Decomposition. In *SAC*, pages 179–186, 2012.

Yang Xiang, Ruoming Jin, David Fuhry, and Feodor Dragan. Summarizing transactional databases with overlapped hyperrectangles. *Data Mining and Knowledge Discovery*, 23:215–251, 2011.

Wei Xu, Xin Liu, and Yihong Gong. Document clustering based on non-negative matrix factorization. In *26th Annual International ACM SIGIR Conference (SIGIR)*, pages 267–273, 2003.

Jaewon Yang and Jure Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *WSDM '13*, pages 587–596, 2013.

Uwe Zimmermann. *Linear and combinatorial optimization in ordered algebraic structures*. Elsevier, Amsterdam, 2011.