Cancer: Another Algorithm for Subtropical Matrix Factorization

Sanjar Karaev and Pauli Miettinen

Max-Planck-Institut für Informatik
Saarbrücken, Germany
{skaraev, pmiettin}@mpi-inf.mpg.de

Abstract. Subtropical algebra is a semi-ring over the nonnegative real numbers with standard multiplication and the addition defined as the maximum operator. Factorizing a matrix over the subtropical algebra gives us a representation of the original matrix with element-wise maximum over a collection of nonnegative rank-1 matrices. Such structure can be compared to the well-known Nonnegative Matrix Factorization (NMF) that gives an element-wise sum over a collection of nonnegative rank-1 matrices. Using the maximum instead of sum changes the 'parts-of-whole' interpretation of NMF to 'winner-takes-it-all' interpretation. We recently introduced an algorithm for subtropical matrix factorization, called `Capricorn`, that was designed to work on discrete-valued data with discrete noise [Karaev & Miettinen, SDM '16]. In this paper we present another algorithm, called `Cancer`, that is designed to work over continuous-valued data with continuous noise – arguably, the more common case. We show that `Cancer` is capable of finding sparse factors with excellent reconstruction error, being better than either `Capricorn`, NMF, or SVD in continuous subtropical data. We also show that the winner-takes-it-all interpretation is usable in many real-world scenarios and lets us find structure that is different, and often easier to interpret, than what is found by NMF.

1 Introduction

Matrix factorizations such as Singular Value Decomposition (SVD) or Nonnegative Matrix Factorization (NMF) are among the most-used methods in data analysis. One way to interpret the factorization is the so-called 'components view' that considers the factorization as a sum of rank-1 matrices. The rank-1 matrices can be considered as patterns found from the data, and different constraints on the factorizations yield different types of patterns. The non-negativity constraint in NMF, for example, yields patterns that are 'parts-of-whole'.

Instead of – or in addition to – constraining the rank-1 matrices, we can also change how we aggregate them. For factorizations made under the standard algebra, the aggregation is always the standard sum, but if we change the algebra, we can have different kinds of aggregations. One possible algebra is the so-called *subtropical algebra*: a semi-ring over the non-negative real numbers with the

standard multiplication but with the addition defined as the maximum operation. A subtropical factorization gives us non-negative rank-1 matrices, just as NMF, but unlike NMF's parts-of-whole interpretation, the subtropical factors are best interpreted using the 'winner-takes-it-all' interpretation: for each element of the matrix, only the largest value in any of the rank-1 components matter.

The winner-takes-it-all interpretation means that each rank-1 component tries to present a dominant pattern: the elements should be as close to the original matrix's elements as possible (but without being much larger) to have any effect to the final outcome of the factorization. Consequently, the values of a component that do not contribute to the final result (i.e. are not the largest ones) can be made as small as possible without any adverse effects; often, many of them can simply be set to 0.

Recently, we introduced an algorithm for subtropical matrix factorization called `Capricorn` [9]. `Capricorn` aims at finding subtropical factorizations from discrete-valued (e.g. integer) data and consequently, it also assumes a discrete noise model where only some of the entries are perturbed. We also empirically validated that `Capricorn` is capable of finding the exact subtropical decomposition if it exists. Many real-world data, however, are better modelled using Gaussian noise, where every element is slightly perturbed, but `Capricorn` often fails finding good factorizations from such data sets. In this paper we present `Cancer`, another algorithm for subtropical factorizations. `Cancer` is complementary to `Capricorn` as it is designed to work well on data perturbed with Gaussian noise; conversely, `Cancer` does not do well if the noise follows the model `Capricorn` was designed for. One could say that if `Capricorn` is the south, `Cancer` is the north.

2 Related Work

Our recent work on the `Capricorn` algorithm [9] is, to the best of our knowledge, the only existing work using tropical or subtropical algebra in data analysis. It also provides a number of theoretical results regarding the subtropical algebra and its close cousin, the tropical algebra (see below). Another application of subtropical algebra is [12], where it is used as a part of a recommender system.

In general, though, matrix factorization methods are ubiquitous in data analysis. A popular example is the nonnegative matrix factorization (NMF) (see, e.g. [5]), where the factorization is restricted to the semi-ring of the nonnegative real numbers. Another example of a matrix factorization over a non-standard algebra is the Boolean matrix factorization (see [11]), where the factorization is restricted to binary matrices and the algebra is the Boolean one (i.e. the summation is defined as $1 + 1 = 1$).

The tropical, or max-plus, algebra [1] is another semi-ring over the extended set of reals $\mathbb{R} \cup \{-\infty\}$ with addition defined as the maximum operator and the multiplication defined as the standard plus operator. Tropical and subtropical algebras are isomorphic (take the logarithm of the latter to obtain the former), and as such, many results obtained for max-plus automatically hold for max-times, although this is not directly true in the case of approximate matrix factorizations

(see [9]). Despite the theory of max-plus algebra being relatively young, it has been thoroughly studied in recent years. The reason for this is an explosion of interest in so called discrete event systems (DES) [4], where max-plus algebra has become ubiquitously used for modeling (see e.g. [2] and [6]).

Yet another approach of computing the matrix factorization over non-standard algebras involves using the Łukasiewicz algebra. They have been recently applied to decompose matrices with grade values [3].

3 Notation and Definitions

Throughout this paper, we will denote a matrix by upper-case boldface letters ($\boldsymbol{A}$), and vectors by lower-case boldface letters ($\boldsymbol{a}$). The ith row of matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}_i$ and the jth column by $\boldsymbol{A}^j$. The 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. Most matrices and vectors in this paper are restricted to the nonnegative real numbers $\mathbb{R}_+$.

In this paper we consider matrix factorization over so called max-times algebra. It differs from the standard algebra of real numbers in that addition is replaced with the operation of taking the maximum. Also the domain is restricted to the set of nonnegative real numbers.

Definition 1. *The* max-times *(or* subtropical*) algebra is a 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 *max-times* and *subtropical* interchangeably. It is straightforward to see that the max-times algebra is a *dioid*, that is, a semiring with idempotent addition ($a \boxplus a = a$). It is important to note that subtropical algebra is anti-negative, that is, there is no subtraction operation.

The subtropical matrix algebra follows naturally:

Definition 2. *The* max-times 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*

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

The definition of a *rank-1* matrix over the max-times algebra is the same as over the standard algebra, i.e. a matrix that can be expressed as an outer product of two vectors. We will use the term *block* to mean a rank-1 matrix. The general *rank* of a matrix over the max-times algebra is defined analogously to the standard Schein rank:

Definition 3. *The* max-times 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 a max of* k *rank-1 matrices,* $\boldsymbol{A} = \boldsymbol{F}_1 \boxplus \boldsymbol{F}_2 \boxplus \cdots \boxplus \boldsymbol{F}_k$*, where all* $\boldsymbol{F}_i$ *are rank-1.*

Now that we have sufficient notation, we can formally introduce the main problem considered in the paper.

Problem 1. 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

$$E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) = \|\boldsymbol{A} - \boldsymbol{B} \boxtimes \boldsymbol{C}\|_F^2 = \sum_{i,j} (\boldsymbol{A}_{ij} - (\boldsymbol{B} \boxtimes \boldsymbol{C})_{ij})^2 \,. \tag{2}$$

4 Algorithm

As we work over the max-times algebra, the common approaches for finding matrix factorizations under normal algebra do not work as such. The main problem is the non-linear behavior of the maximum function, and our algorithm tries to alleviate the problems caused by it. The two main ideas we employ are updating the rank-1 factors one-by-one in an iterative fashion, and approximating the max-times reconstruction error with a low-degree polynomial. The first idea is similar to what we used in [9], except that here we only update parts of the rank-1 factors. The motivation behind this is to avoid building few factors that try to explain the whole data (badly), but instead build many factors that explain small parts of the data well.

4.1 The main algorithm

Our proposed algorithm, `Cancer`, is outlined in Algorithm 1. It accepts as input the data to be decomposed $\boldsymbol{A}$, the required rank k, and three additional scalar parameters M, t, and f. Integer M is the number of cycles that the algorithm will make, that is each one of k blocks will be visited M times. A reasonable value for M would be 15 since further rounds provide only marginal improvement, although to make sure that the algorithm has converged, a value as high as 40 might be required. The next parameter, $t \in \mathbb{N}$, represents the maximum allowed degree of polynomials: after each cycle the degree of polynomials used for approximation is incremented, but when it reaches t, it is reset to the value of 2. Typically, we can set $t = 16$. Finally, $f \in (0, 1)$ controls how much of each block (rank-1 matrix) is revealed on each iteration. Namely, each block $\boldsymbol{bc} \in \mathbb{R}_+^{n \times m}$ consists a total of $n + m$ variables, and the maximum number of variables we can change when a block is visited is $\lfloor f(n + m)/2 \rfloor$. The algorithm outputs two factor matrices $\boldsymbol{B} \in \mathbb{R}_+^{n \times k}$ and $\boldsymbol{C} \in \mathbb{R}_+^{k \times m}$ whose product is a rank-k max-times approximation of $\boldsymbol{A}$.

`Cancer` starts with empty blocks (line 2) and updates them iteratively (lines 6–14) using the `UpdateBlock` routine (line 9). `UpdateBlock` updates one block while keeping all others fixed. We then compare the current decomposition to the best one seen so far, and if it provides an improvement, then the best solution is replaced with the current one (lines 10–12). The final step of the loop is to increment the degree of polynomials used for approximation (lines 13–14). Intuitively, lower degrees polynomials give more latitude for varying the variables,

Algorithm 1 `Cancer`

Input: $\boldsymbol{A} \in \mathbb{R}_+^{n \times m}$, $k > 0$, $M > 0$, $t > 2$, $0 < f < 1$
Output: $\boldsymbol{B}^* \in \mathbb{R}_+^{n \times k}$, $\boldsymbol{C}^* \in \mathbb{R}_+^{k \times m}$

function `Cancer`$(\boldsymbol{A}, k, M, t, f)$
  $\boldsymbol{B} \leftarrow 0^{n \times k}$, $\boldsymbol{C} \leftarrow 0^{k \times m}$
  $\boldsymbol{B}^* \leftarrow \boldsymbol{B}, \boldsymbol{C}^* \leftarrow \boldsymbol{C}$
  $bestError \leftarrow E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$
  $deg \leftarrow 2$
  for $count \leftarrow 0$ **to** $k \times M - 1$ **do**
    $i \leftarrow count \pmod k + 1$ ▷ Index of the current block.
    $\boldsymbol{N} \leftarrow \boldsymbol{B}^{-i} \boxtimes \boldsymbol{C}_{-i}$ ▷ Reconstructed matrix without the i-th block.
    $[\boldsymbol{B}^i, \boldsymbol{C}_i] \leftarrow$ `UpdateBlock`$(\boldsymbol{A}, \boldsymbol{N}, \boldsymbol{B}^i, \boldsymbol{C}_i, deg, f)$
    if $E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) < bestError$ **then**
      $\boldsymbol{B}^* \leftarrow \boldsymbol{B}, \boldsymbol{C}^* \leftarrow \boldsymbol{C}$
      $bestError \leftarrow E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$
    if $count > k$ **and** $count \pmod k = 0$ **then**

$$deg \leftarrow \begin{cases} deg + 1 & \text{if } deg < t \\ 2 & \text{otherwise} \end{cases}$$

  return $\boldsymbol{B}^*$, $\boldsymbol{C}^*$

whereas polynomials of higher degrees are better suited for finalizing results since they provide better approximations. This is similar to an execution of a simulated annealing algorithm, where high temperatures are used to make big steps and get out of local minima, and lower temperatures are better suited for converging to a particular minimum. In our case low degrees correspond to high temperatures and vice versa.

Most of the time the reconstruction error decreases gradually with increased iterations of `Cancer`. There are however rare cases where it would remain almost constant for some time or even increase slightly, and then start dropping again. For this reason the algorithm is run until all cycles are complete and is not stopped using any sort of convergence criteria.

4.2 Updating a block

The `UpdateBlock` procedure (Algorithm 2) performs the work of updating a single block $\boldsymbol{bc} \in \mathbb{R}_+^{n \times m}$ on one iteration of `Cancer`. It takes a block $\boldsymbol{bc}$, where $\boldsymbol{b} \in \mathbb{R}_+^{n \times 1}$ and $\boldsymbol{c} \in \mathbb{R}_+^{1 \times m}$, and performs alternating updates of $\boldsymbol{b}$ and $\boldsymbol{c}$ one element at a time using the `AdjustOneElement` function. That function is called $\lfloor f(n+m)/2 \rfloor$ times to update only a part of the block, as explained above.

The `AdjustOneElement` function (Algorithm 3) 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

Algorithm 2 `UpdateBlock`

Input: $\boldsymbol{A} \in \mathbb{R}_+^{n\times m}$, $\boldsymbol{N} \in \mathbb{R}_+^{n\times m}$, $\boldsymbol{b} \in \mathbb{R}_+^{n\times 1}$, $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$, $deg \geq 2$, $0 < f < 1$
Output: $\boldsymbol{b} \in \mathbb{R}_+^{n\times 1}$, $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$

function `UpdateBlock`$(\boldsymbol{A}, \boldsymbol{N}, \boldsymbol{b}, \boldsymbol{c}, deg, f)$
  $niters \leftarrow \lfloor f(n+m)/2 \rfloor$
  for $count \leftarrow 1$ **to** $niters$ **do**
    $\boldsymbol{c} =$ `AdjustOneElement`$(\boldsymbol{A}, \boldsymbol{N}, \boldsymbol{b}, \boldsymbol{c}, deg)$
    $\boldsymbol{b} =$ `AdjustOneElement`$(\boldsymbol{A}^T, \boldsymbol{N}^T, \boldsymbol{c}^T, \boldsymbol{b}^T, deg)^T$
  return $\boldsymbol{b}$, $\boldsymbol{c}$

Algorithm 3 `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: $\boldsymbol{c} \in \mathbb{R}_+^{1\times m}$

function `AdjustOneElement`$(\boldsymbol{A}, \boldsymbol{N}, \boldsymbol{b}, \boldsymbol{c}, deg)$
  for $j \leftarrow 1$ **to** m **do**
    $baseError \leftarrow \sum_{i=1}^{n} (\boldsymbol{A}_{ij} - \max\{\boldsymbol{N}_{ij}, \boldsymbol{b}_i \boldsymbol{c}_j\})^2$
    $[err, \boldsymbol{x}_i] \leftarrow$ `PolyMin`$(\boldsymbol{A}^j, \boldsymbol{N}^j, \boldsymbol{b}, deg)$
    $\boldsymbol{u}_i \leftarrow baseError - err$
  $i \leftarrow$ the index i of largest value of $\boldsymbol{u}$
  $\boldsymbol{c}_i \leftarrow \boldsymbol{x}_i$
  return $\boldsymbol{c}$

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

$$\gamma(\boldsymbol{A}_l, \boldsymbol{N}_l, \boldsymbol{b}, \boldsymbol{c}_l) = \sum_{i=1}^{n} (\boldsymbol{A}_{il} - \max\{\boldsymbol{N}_{il}, \boldsymbol{b}_i \boldsymbol{c}_l\})^2 \,. \tag{3}$$

Instead of minimizing (3) directly, we use polynomial approximation in the `PolyMin` routine (line 4). The routine returns the (approximate) error err and the value x achieving that. 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 yield the largest improvement.

4.3 The `PolyMin` procedure

The function γ that we need to minimize in order to find the best change to the vector $\boldsymbol{c}$ in `AdjustOneElement` is hard to work with directly since it is not convex, and also not smooth because of the presence of the maximum operator. To alleviate this, we approximate the error function γ with a polynomial g of degree deg. Notice that when updating $\boldsymbol{c}_l$, other variables of γ are fixed and we only need to consider function $\gamma'(x) = \gamma(\boldsymbol{A}_l, \boldsymbol{N}_l, \boldsymbol{b}, x)$. To build g we sample

$deg + 1$ points from $(0, 1)$ and fit g to the values of γ' at these points. We then find the $x \in \mathbb{R}_+$ that minimizes $g(x)$ and return $g(x)$ (the approximate error) and x (the optimal value).

4.4 Computational complexity

We will express the complexity of the algorithm asymptotically in terms of the dimensions of the input data n and m and the rank k of the factorization. Most of the work in `Cancer` is performed in the `UpdateBlock` routine, which is called Mk times. `UpdateBlock` is in turn just 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 the parameters f and M are fixed, 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`.

5 Experiments

In this section we evaluate the performance of `Cancer` against other algorithms on various synthetic and real-world datasets. The purpose of the synthetic experiments is to verify that `Cancer` is capable of finding subtropical structure from data where we know it is present, and to evaluate its performance under different data characteristics in a controlled manner. Our tests demonstrate that `Cancer` not only provides better approximations than other methods, but also produces much sparser factors. The main purpose of experiments with real-world datasets is to see if they possess the max-times structure and whether `Cancer` is capable of extracting it.

Setting the parameters for ***`Cancer`****.* For all 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 for which we used $M = 50$).

5.1 Other methods

We compared `Cancer` against `Capricorn`, which is our previous tropical matrix factorization algorithm designed for discrete data [9],[1] `SVD`, and four different

[1] The source code for `Cancer` and `Capricorn` and the scripts to repeat the experiments are available at `http://people.mpi-inf.mpg.de/~pmiettin/tropical/`.

versions of NMF. The first form of NMF is a sparse NMF algorithm by Hoyer [8],[2] which we call `SNMF`. Hoyer's algorithm [8] defines the sparsity of a vector $\boldsymbol{x} \in \mathbb{R}^n_+$ as

$$\text{sparsity}(\boldsymbol{x}) = \frac{\sqrt{n} - \left(\sum_i |\boldsymbol{x}_i|\right) / \sqrt{\sum_i \boldsymbol{x}_i^2}}{\sqrt{n} - 1} \ , \tag{4}$$

and returns factorization where the sparsity of the factor matrices is user-controllable. In all of our experiments, we used the sparsity of `Cancer`'s factors as the sparsity parameter of `SNMF`.

The second form of NMF is a standard alternating least squares algorithm called `ALS` [5]. The remaining two versions of NMF are essentially the same as `ALS`, but they use L_1 regularization for increased sparsity [5], that is, they aim at minimizing

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

The first method is called `ALSR` and uses regularizer coefficient $\alpha = \beta = 1$, and the other, called `ALSR` 5, has regularizer coefficient $\alpha = \beta = 5$. All NMF algorithms were restarted 10 times, and the best result was selected.

5.2 Synthetic experiments

The general setup of synthetic experiments is as follows. First we create data that is guaranteed to have the subtropical structure by 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. Then we add noise and feed the obtained noisy matrices into algorithms to see how well they can approximate the original data. We distinguish two types of noise. One is the normal, or Gaussian, noise with 0 mean, for which we define the level of noise 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. We use two noise levels, 0.01 and 0.08, called low and high noise levels, respectively.

The other type of noise is a discrete (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 $\boldsymbol{F} = \max\{\boldsymbol{A}, \boldsymbol{N}\}$.

All synthetic experiments were performed on 1000-by-800 matrices. In all tests, except those with varying rank, the true max-times rank of the data was 10. For all experiments we report errors, which are measured as relative Frobenius errors between original and reconstructed matrices, that is, $E(\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) / \|\boldsymbol{A}\|_F^2$. We also report the sparsity s of factor matrices obtained by algorithms, which is

[2] `https://github.com/aludnam/MATLAB/tree/master/nmfpack`

defined as a fraction of zero elements in the factor matrices,

$$s(\boldsymbol{A}) = |\{(i,j) : \boldsymbol{A}_{ij} = 0\}| / (nm) \,, \tag{5}$$

for an n-by-m matrix $\boldsymbol{A}$. The results were averaged over 10 repetitions. The reconstruction errors are reported in Figure 1 and the sparsities in Figure 2.

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 noise matrix as described earlier. The factor density was kept at 50%. The results are given on Figure 1(a) (reconstruction error) and Figure 2(a) (sparsity of factors).

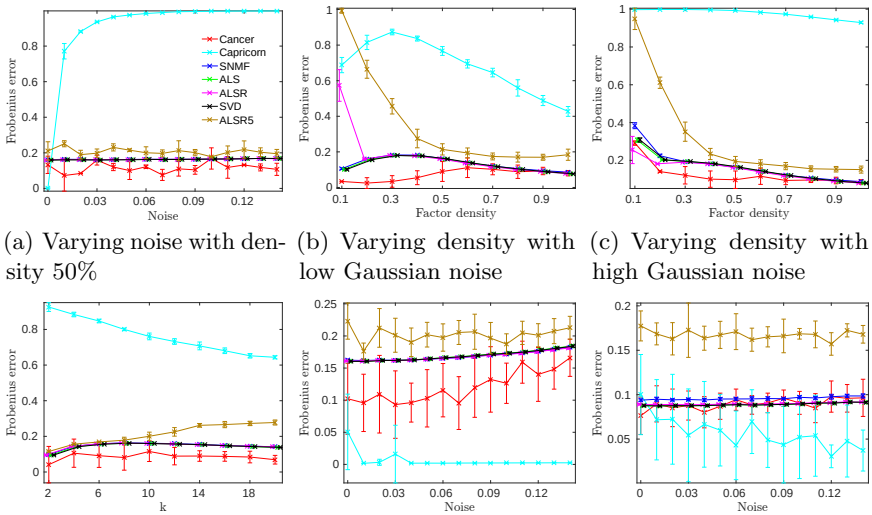
Here, `Cancer` is generally the best method in reconstruction error, and second in sparsity only to `Capricorn`. The sole exception to reconstruction error is the no-noise case, where `Capricorn` – as designed – obtains essentially a perfect decomposition, though its results deteriorate rapidly with increased noise levels.

Varying density. In this experiment we studied what effects the density of factor matrices used in data generation has on the algorithms' 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 1(b) and 2(b)), and a more noisy case at 0.08 (Figures 1(c) and 2(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 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 `ALSR` 5 is sparser (and worse in reconstruction error), meaning that `Cancer` produces factors that are both the most accurate and very sparse.

Varying rank. 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 on Figure 1(d) (reconstruction error) and Figure 2(d) (sparsity of factors). The results are similar to the two above ones, with `Cancer` returning the most accurate and second-most sparsest factorizations.

Varying tropical noise. In this setup we used the tropical noise instead of the Gaussian one. The level of noise represents the density of the noise matrix with which the original data is 'maxed'. We varied the noise from 0% to 14% with increments of 1%. There are two forms of this experiment, one with density 50% (Figure 1(e) shows the reconstruction error and Figure 2(e) shows the sparsity of factors) and with density 90% (Figures 1(f) and 2(f), respectively).



(a) Varying noise with density 50%

(b) Varying density with low Gaussian noise

(c) Varying density with high Gaussian noise

(d) Varying rank; 50% density and low Gaussian noise

(e) Varying tropical noise with density 50%

(f) Varying tropical noise with density 90%

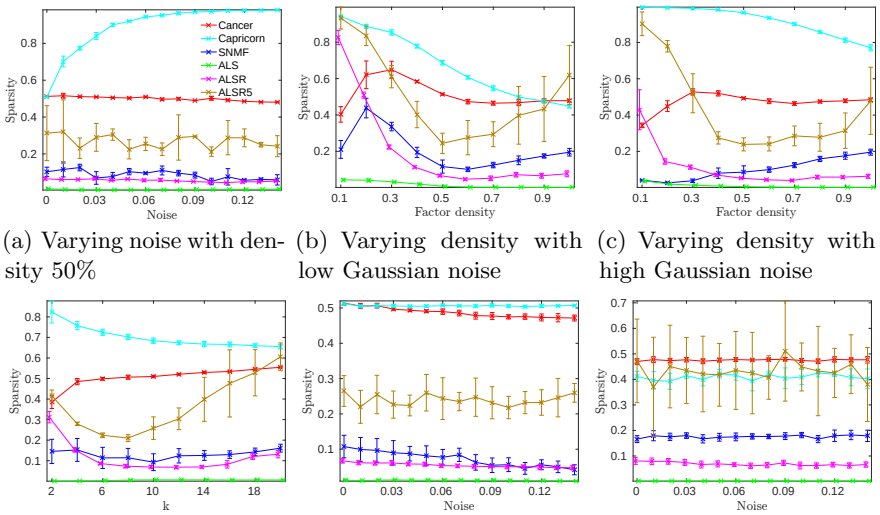
Fig. 1. Reconstruction error (Frobenius norm) for synthetic data. The markers are averages of 10 random matrices and the width of the error bars is twice the standard deviation.

As `Capricorn` was designed for tropical noise, unlike `Cancer` that was designed for standard 'white' noise, it obtains the least reconstruction error of all the methods (albeit with high deviation when the noise density is higher). `Cancer` is generally the second-best method, although with the high-density noise, it is mostly tied with `SVD`, `ALS` and `ALSR`. In the sparsity of the factors, `Cancer` and `Capricorn` are quite similar, with `Capricorn` having slightly sparser factors in the low-density noise case, but `Cancer` having an edge in the high-density noise case. In the latter case, `ALSR` 5 is also comparable on sparsity, but clearly the worst in reconstruction error.

Discussion. The synthetic experiments verify that `Cancer` can find the max-times structure from the data when it is present and potentially perturbed with Gaussian noise. It also shows strong invariance over the level of noise, rank, or density of the factors. The experiments also highlight the design differences between `Cancer` and `Capricorn`: the former is superior in Gaussian noise situation, while the latter excels with tropical noise. If the type of noise cannot be predetermined, it seems it is best to try both methods.

5.3 Real-world experiments

The main purpose of the real-world experiments is to study to which extend `Cancer` can find max-times structure from various real-world data sets. Having



(a) Varying noise with density 50%

(b) Varying density with low Gaussian noise

(c) Varying density with high Gaussian noise

(d) Varying rank; 50% density and low Gaussian noise

(e) Varying tropical noise with density 50%

(f) Varying tropical noise with density 90%

Fig. 2. Sparsity (fraction of zeroes) of the factor matrices for synthetic data. The markers are averages of 10 random matrices and the width of the error bars is twice the standard deviation.

established with the synthetic experiments that `Cancer` is indeed capable of finding the structure when it is present (and potentially perturbed with Gaussian noise), here we look at what kind of results it obtains in the real-world data.

It is probably unrealistic to expect real-world data sets to have 'pure' max-times structure, as in the synthetic experiments. Rather, we expect `SVD` to be the best method (in reconstruction error's sense), and `Cancer` 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. Worldclim was obtained from the global climate data repository.[3] It describes historical climate data across different geographical locations in Europe. Columns represent minimum, maximum and average temperatures and precipitation, and rows are 50-by-50 kilometer squares of land where measurements were made. 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.

NPAS is a nerdy personality test that uses different attributes to determine the level of nerdiness of a person.[4] It contains answers by 1418 respondents to

[3] The raw data is available at `http://www.worldclim.org/`.

[4] Tha dataset can be obtained on the online personality website `http://personality-testing.info/_rawdata/NPAS-data.zip`.

Table 1. Real world datasets specs.

Algorithm	Rows	Columns	Density
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%

a set of 36 questions that asked them to self assess various statements about themselves on a scale of 1 to 7. We preprocessed NPAS analogously to Worldclim.

Eigenfaces is a subset of the Extended Yale Face collection of face images [7]. It consists of 32-by-32 images under different lighting conditions. We used a preprocessed data by Xiaofei He et al.[5] We selected a subset of pictures with lighting from the left and then preprocessed the input matrix by first subtracting from every column its mean and then dividing it by its standard deviation.

4News is a subset of the 20Newsgroups dataset,[6] containing the usage of 800 words over 400 posts for 4 newsgroups.[7] 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.[8] Rows represent months, columns are locations, and entries are residential property price indices. We preprocessed the data by first dividing each column by its standard deviation and then subtracting its minimum, so that each column has minimum 0.

The basic properties of these data sets are listed in Table 1.

Reconstruction error, sparsity, and convergence. Table 2 provides the relative Frobenius reconstruction errors for the real-world data sets. We omitted `ALSR` 5 from these experiments due to its bad performance with the synthetic data. `SVD` is, as expected, consistently the best method. Somewhat surprisingly, Hoyer's `SNMF` is usually the second-best method, even though in the synthetic experiments it usually was the second-worst of the NMF-based methods. `Cancer` is usually the third-best method (with the exception of 4News and NPAS), and often very close to `SNMF` in reconstruction error. Overall, it seems `Cancer` 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.

The sparsity of the factors for real-world data is presented in Table 3, except for `SVD`. Here, `Cancer` often returns the second-sparsest factors (being second only to `Capricorn`), but with 4News and HPI, `ALSR` obtains sparser decompositions.

[5] `http://www.cad.zju.edu.cn/home/dengcai/Data/FaceData.html`

[6] `http://qwone.com/~jason/20Newsgroups/`

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

[8] Available at `https://data.gov.uk/dataset/land-registry-house-price-index-background-tables/`

Table 2. Reconstruction error for various real-world datasets.

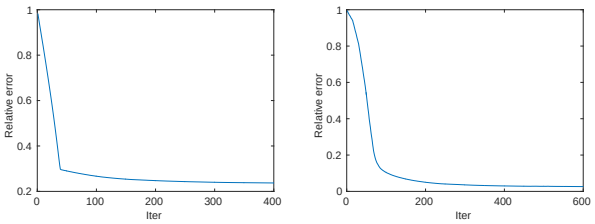
	Worldclim	NPAS	Eigenfaces	4News	HPI
$k =$	10	10	40	20	15
`Cancer`	0.071	0.240	0.204	0.556	0.027
`Capricorn`	0.392	0.395	0.972	0.987	0.217
`SNMF`	0.046	0.225	0.178	0.546	0.023
`ALS`	0.087	0.227	0.313	0.538	0.074
`ALSR`	0.122	0.226	0.294	1.000	0.045
`SVD`	0.025	0.209	0.140	0.533	0.015

Table 3. Factor sparsity for various real-world datasets.

	Worldclim	NPAS	Eigenfaces	4News	HPI
$k =$	10	10	40	20	15
`Cancer`	0.645	0.528	0.571	0.812	0.422
`Capricorn`	0.795	0.733	0.949	0.991	0.685
`SNMF`	0.383	0.330	0.403	0.499	0.226
`ALS`	0.226	0.120	0.434	0.513	0.331
`ALSR`	0.275	0.117	0.480	1.000	0.729

We also studied the convergence behavior of our algorithm using some of the real-world data sets. The results can be seen in Figure 3, where we plot the relative error with respect to the iterations over the main for-loop in `Cancer`. As we can see, in both cases `Cancer` has obtained a good reconstruction error already after few full cycles, with the remaining runs only providing minor improvements. We can deduce that `Cancer` reaches quickly an acceptable solution.

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 plotted the left factor matrices for the Eigenfaces data for `Cancer` and `ALS` in Figure 4. At first, it might look like `ALS` provides more interpretable results, as most factors are easily identifiable as faces. This, however, is not 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 identify on the first sight. Upon closer inspection, though, one can see that they identify areas that are lighter in the different images, that is, have higher grayscale values. These factors tell us the variances in the lightning in the different photos, and can reveal information we did not know a priori. Further, as seen in Table 2, `Cancer` obtains better reconstruction error than `ALS` with this data, confirming that these factors are indeed useful to recreate the data.



(a) NPAS (b) HPI

Fig. 3. 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.

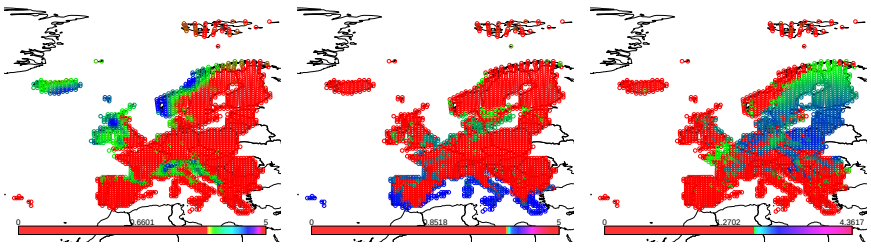
(a) `Cancer`

(b) `ALS`

Fig. 4. `Cancer` finds the dominant patterns from the Eigenfaces data. Pictured are the left factor matrices for the Eigenfaces data.

Table 4. 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



(a) (b) (c)

Fig. 5. `Cancer` can find interpretable factors from the Worldclim data. Shown are the values for three columns in the left-hand factor matrix $\boldsymbol{B}$ on a map. Red is zero.

In Figure 5, we show some factors from `Cancer` when applied to the Worldclim data. These factors clearly identify different bioclimatic areas from Europe: In Figure 5(a) we can identify the mountainous areas in Europe, including the Alps, the Pyrenees, the Scandes, and Scottish Highlands. In Figure 5(b) we can identify the mediterranean coastal regions, while in Figure 5(c) we see the temperate climate zone in blue, with the green color extending to the boreal zone. In all pictures, red corresponds to (near) zero values. As we can see, `Cancer` identifies these areas crisply, making it easy for the analyst to know which areas to look at.

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 4. The first factor clearly shows introvert tendencies, while the second one can be summarized as having interests in fiction and games.

6 Conclusions

Using max-times algebra instead of the standard (nonnegative) algebra, we can find factors that adhere to the 'winner-takes-it-all' interpretation instead of the 'parts-of-whole' interpretation of NMF. The winner-takes-it-all factors give us the most dominant features, building a sharper contrast between what is and is not important for that factor, making the factors potentially easier to interpret. As

we saw in our experimental evaluation, the factors are also sparse, emphasizing the winner-takes-it-all interpretation.

Finding a good max-times factorization, unfortunately, seems harder than – or at least as hard as – finding a good nonnegative factorization. Our earlier algorithm, `Capricorn`, was designed to work with discrete-valued data and what we call 'tropical' noise; in this paper we presented `Cancer` that is designed to work with Gaussian noise and matrices with continuous values. It seems that this latter case is more applicable to real-world data, as witnessed by `Cancer`'s good results with real-world data.

There are still questions that need to be addressed by future research. Could these two approaches be merged? That is, is it possible to design an algorithm that works well for both tropical and Gaussian noise? Can one achieve provable approximation ratios for max-times factorizations? In addition to data analysis, can max-times factorizations be used in other data mining and machine learning tasks (e.g. to do matrix completion or latent topic models)? We hope our initial work in this paper (and its predecessor [9]) helps to increase data mining and machine learning community's interest to max-times algebras so that the above question could be answered.

References

1. Akian, M., Bapat, R., Gaubert, S.: Max-Plus Algebra. In: Hogben, L. (ed.) Handbook of Linear Algebra. Chapman & Hall/CRC (2007)
2. Baccelli, F., Cohen, G., Olsder, G.J., Quadrat, J.P.: Synchronization and Linearity: An Algebra for Discrete Event Systems. John Wiley & Sons (1992)
3. Bělohlávek, R., Krmelova, M.: Factor analysis of ordinal data via decomposition of matrices with grades. Ann Math Artif Intell 72(1-2), 23–44 (Jan 2014)
4. Cassandras, C.G., Lafortune, S.: Introduction to Discrete Event Systems. Springer, second edn. (2008)
5. Cichocki, A., Zdunek, R., Phan, A.H., Amari, S.i.: Nonnegative Matrix and Tensor Factorizations: Applications to Exploratory Multi-way Data Analysis and Blind Source Separation. John Wiley & Sons, Chichester (2009)
6. Cohen, G., Gaubert, S., Quadrat, J.P.: Max-plus algebra and system theory: Where we are and where to go now. Annu Rev Control 23, 207–219 (Jan 1999)
7. Georghiades, A.S., Belhumeur, P.N., Kriegman, D.J.: From few to many: Generative models for recognition under variable pose and illumination. In: IEEE AFGR. pp. 277–284 (2000)
8. Hoyer, P.O.: Non-negative Matrix Factorization with Sparseness Constraints. J. Mach. Learn. Res. 5, 1457–1469 (2004)
9. Karaev, S., Miettinen, P.: Capricorn: An Algorithm for Subtropical Matrix Factorization. In: SDM '16 (2016)
10. Miettinen, P.: Matrix Decomposition Methods for Data Mining: Computational Complexity and Algorithms. Ph.D. thesis (2009)
11. Miettinen, P., Mielikäinen, T., Gionis, A., Das, G., Mannila, H.: The Discrete Basis Problem. IEEE Trans. Knowl. Data Eng. 20(10), 1348–1362 (Oct 2008)
12. Weston, J., Weiss, R.J., Yee, H.: Nonlinear latent factorization by embedding multiple user interests. In: RecSys '13. pp. 65–68 (2013)