UNIVERSITÀ DI PISA

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Triennale in Matematica

Tesi di Laurea

Nonnegative Matrix Factorization: Theory with an application to translations invariant image processing

Relatori:
Luca Gemignani
Francesco Romani

Candidato:
Barbarino Giovanni

ANNO ACCADEMICO 2015-2016

Intoduction

Nonnegative Matrix Factorization(NMF) is a common used technique in machine learning to extract features out of data such as text documents and images thanks to its natural clustering properties and the easy interpretation of the output data.

Many papers cite the article of Lee & Seung [25] as the first formalization of the method, but in the course of history it has been introduced by many authors in different contexts and applications. For example, the problem is known as the *self modeling curve resolution* in the field of chemometrics[24], and Paatero & Tapper [28] introduced it with the name of Positive Matrix Factorization. It is used also in biology and medicine[6], thanks to its application to vision research, in signal analysis[31], since it's able to separate different frequencies superimposed on the same track, in graph clustering, and in general in any application that needs a pattern recognition algorithm in order to analyze and give an interpretation to large amounts of data.

In the first chapter we review the original NMF problem, its common variants, and the main solving algorithms used nowadays. We'll also see how its particular framework makes it suitable for a lot of applications like clustering and text mining. In particular, in the first Section we state the NMF framework, along with some equivalent formulations, its common interpretation, and properties like uniqueness and sparseness of the solutions. In the second one, we'll take a look at the NMF applications to Clustering and Text Mining, discussing how its characteristics make this framework suitable for such problems, and in the last section of the chapter, we describe the state of art of the algorithms proposed to solve the problem.

One of the main applications of NMF is the analysis and decomposition of images, as shown by Lee & Seung[25], which processed a set of faces and recognized their principal features like eyebrows, lips, noses, etc. One of this method drawbacks is that NMF can't recognize the same objects or parts of them if they're located in different places on multiple images, or when they're rotated and stretched. In other words, NMF is not invariant under space transformations, so the input data must always be pre-calibrated and adjusted. Some authors have suggested to set some standard transformations of the images (such as translations or symmetries) and to look for the features we want to obtain, but this rises the number of the problem variables by a factor that's usually larger or equal to the number of pixels in a picture, like in [30] and [11], making the algorithm complexity and memory go up by at least the same factor.

In the second chapter, we present a way to fix the problem for translations, that keeps the interpretability property of the output to represent the wanted parts of images, doesn't change the original input data, and bounds the computational cost by the number of effective features we want to find. In the first section, we'll study the application of NMF to Image Processing, and state clearly why it lacks the ability to pinpoint the common features under space transformations. In the second chapter we'll describe a new domain for the variables in the matrices, and see how this change the NMF setting. In the third section we then devise a method to solve the new problem, and in the last section, we'll perform some experiments on handmade data.

The structure we built still lacks of a proper solid theory apparatus, so on the last chapter, various suggestions on further works are proposed.

Contents

Chapter 1

Nonnegative Matrix Factorization

In all the document, we'll call $\mathbb{R}_+$ the set of nonnegative real numbers, and use the notations $A.*B$ and $A./B$ to indicate the element-wise product and division between matrices.

Moreover, we'll refer to the i-th column and row of a matrix A respectively with $A_{:,i}$ and $A_{i,:}$, and we'll use the MATLAB syntax $A > 0$, $v > 0$ to define the matrix or vector containing ones in correspondence to the positive entries of A, v and zero otherwise.

1.1 Framework

In this section, we state the Nonnegative Matrix Factorization framework, and we'll explore some of its properties, like existence and uniqueness of the solution, equivalent formulations and alternative versions.

Problem 1.1.1 (NMF)**.** Given a data matrix $A \in \mathbb{R}_+^{n\times m}$ and a natural number k, the NMF problem statement is to find the matrices W, H that satisfy

$$\min_{W,H} F(W,H) = \min_{W,H} \frac{1}{2}\|A - WH^T\|_F^2 \qquad W \in \mathbb{R}_+^{n\times k} \quad H \in \mathbb{R}_+^{m\times k}. \tag{1.1}$$

We'll refer to the problem of finding an exact nonnegative factorization $A = WH^T$ as the Exact NMF problem, whereas we'll call the above statement the Approximate NMF or simply the NMF.

In the formulation above, we used the Frobenius norm, defined as

$$\|A\|_F^2 = \sum_{i,j} a_{ij}^2.$$

In some applications, various authors have explored different matrix distances $d(A, WH^T)$, like other matrix norms, or the symmetric Kullback-Leibler[36] divergence, defined on

real positive matrices A and B with sum of elements equal to 1, as

$$D(A||B) = \sum_{i,j} A_{ij} \log \frac{A_{ij}}{B_{ij}}.$$

The KL divergence is not a distance, but its usage in the NMF problem is important since it optimizes exactly the same objective function, known with the name of *Information Divergence*, as the *Probabilistic Latent Semantic Indexing*[7], that is a statistical technique for the analysis of co-occurrence data. In the rest of the document, for simplicity, we'll consider only the Frobenius norm.

Interpretations A natural interpretation of NMF derives from the observation that a solution to the problem gives the best approximation of the columns of A among the combinations of k nonnegative vectors, the columns of W, with nonnegative coefficients stored in the columns of H^T.

$$A \sim WH^T \implies A_{:,i} \sim H_{i,1}W_{:,1} + H_{i,2}W_{:,2} + \cdots + H_{i,k}W_{:,k} \quad \forall i.$$

This means that the problem is equivalent to find a nonnegative set of k vectors that approximately generate, through nonnegative coefficients, all the columns of A. In applications, we set k much smaller then the other dimensions n, m since the NMF is often used as a low-rank decomposition algorithm, and the resulting columns of W, called *features* or *components*, have a meaningful representation as characteristics or parts of the original data.

Since the problem is symmetrical in W and H, we can transpose the expression, obtaining $\|A^T - HW^T\|$, so we can repeat all the considerations done till now for the rows of A and the columns of H.

An other interpretation of the problem comes from the geometry. We call $\Gamma(M)$ the *simplicial cone* generated by the columns of M defined as

$$M \in \mathbb{R}_+^{n \times m} \qquad \Gamma(M) = \{ M\alpha \mid \alpha \in \mathbb{R}_+^m \} \subseteq \mathbb{R}_+^n,$$

that are all the linear combination of the columns with nonnegative coefficients. The Exact NMF problem applied on A is equivalent to find a nonnegative matrix $W \in \mathbb{R}_+^{n \times k}$ such that

$$\Gamma(A) \subseteq \Gamma(W) \subseteq \mathbb{R}_+^n,$$

since

$$\exists W \in \mathbb{R}_+^{n \times k}, H \in \mathbb{R}_+^{m \times k} \,:\, A = WH^T \iff \exists W \in \mathbb{R}_+^{n \times k} \,:\, A_{:,i} \in \Gamma(W) \quad \forall i$$

$$\iff \exists W \in \mathbb{R}_+^{n \times k} \,:\, \Gamma(A) \subseteq \Gamma(W).$$

We'll see how particular conditions on A, W, H (sparsity, separability, etc.) translate into this geometric interpretation.

KKT conditions NMF is a non-convex optimization problem, so finding a local minimum of (1.1) with the nonnegativity constraint is equivalent to solve the following system of Karush-Kuhn-Tucker conditions

$$W. * \nabla_W F(W,H) = W. * (WH^T H - AH) = 0,$$
$$H. * \nabla_H F(W,H) = H. * (HW^T W - A^T W) = 0,$$
$$\nabla_W F(W,H) = WH^T H - AH \geq 0, \qquad \nabla_H F(W,H) = HW^T W - A^T W \geq 0,$$
$$W,H \geq 0.$$

Notice that $(W,H) = (0,0)$ is a solution to the above system, so it is an useless local minimum of the NMF problem, and any solving algorithm has to deal with this inconvenience.

Since the above described system is too big, it is common use to consider two smaller convex problem, obtained fixing alternatively one of the two matrices W or H. In fact, if we consider H as a constant, then the problem reduces to

$$\min_{W \in \mathbb{R}_+^{n \times k}} F(W,H) = \min_{W \in \mathbb{R}_+^{n \times k}} \|A - WH^T\|_F^2, \tag{1.2}$$

that is a convex least square problem with convex constraints, so it can be attacked with the usual optimization methods. In particular its KKT conditions become a lot more simple

$$W. * \nabla_W F(W,H) = W. * (WH^T H - AH) = 0,$$
$$\nabla_W F(W,H) = WH^T H - AH \geq 0,$$
$$W \geq 0.$$

These equations are solved exactly by Active-Set Methods, and are approximated by algorithms like the Coordinate Descend, Hierarchical ALS and Projected Gradient.

1.1.1 Features

Nonnegative Rank The number k of the columns of W sometimes isn't fixed but can vary in a range of positive integers $[a,b]$, so that an algorithm solving NMF must also find automatically the optimal k. In any case, we know ([16],[8]) that there exists a minimum k, called *nonnegative rank*, such that A is exactly factorisable, indicated as $\mathrm{rk}_+(A)$.

We can notice that if the columns of A are generated by linear combinations of k vectors, then the rank of A must be less or equal to k, and moreover they are always generated as nonnegative linear combinations of the canonical base of $\mathbb{R}^n$. This leads to

$$\mathrm{rk}(A) \leq \mathrm{rk}_+(A) \leq \min\{n,m\}$$

since we can repeat the same reasoning for the rows of A and the columns of H.

In applications, usually A is a matrix of real measurements, distances or intensities, so it is often affected by random noise, that makes it a full-rank matrix. The nonnegative rank becomes thus equal to the rank, so if we fix $k = \mathrm{rk}_+(A)$ we obtain a trivial solution ($W = I$ and $H = A^T$ or viceversa), that does not contain any information.

The parameter k is usually tuned to be fairly low, since a large value of k implies a large set of solutions for the exact NMF problem, and it translates into a lot of local minima into the minimization problem, that leads to inaccuracy on the algorithmic part, and ambiguity in the interpretation of solutions. Another problem is that a big k may lead to the overfitting of the data, meaning that the error is lower then the expectations, and the solution isn't human-readable, but just an artificial one found by the method with no link to the real structure of the original data. When k is too large, we enter in the field of *overcomplete representation*[10], where the number of features is comparable with the dimension of the data set n or the number of the objects introduced m.

Since k is the number of columns of W, then the nonnegative rank is also the minimal number of extremal rays of the simplicial cone $\Gamma(W)$. After a normalization, the problem to find the minimum k translates into the problem of finding the polytope with the minimum number of vertices that contains the convex hull of the columns of A, called $conv(A)$, and that is contained into $\Delta^n = \left\{ x \in \mathbb{R}^n_+ \;\middle|\; \|x\|_1 = 1 \right\}$, that is the intersection between the unitary ball in norm 1 and the positive orthant. This problem is called the Nested Polytopes Problem (NPP) [19], and it is still equivalent to the original NMF.

Sparsity An other feature that is usually required to the input data or to the solution is the sparsity, since it is proved that can improve the quality and understandability of the solution, along with gaining uniqueness properties (for further studies, see Gillis[14], that proposes a preprocessing to improve the sparseness of A). This lead to several reformulations of the original problem, in order to add sparsity parameters, and their general framework, called Constrained NMF, is presented as

$$\min_{W,H} \|A - WH^T\|_F^2 + \alpha J_1(W) + \beta J_2(H) \qquad W \in \mathbb{R}^{n\times k}_+ \quad H \in \mathbb{R}^{m\times k}_+$$

where J_1 and J_2 are penalty functions used to ensure some additional condition on W, H, and α, β are regularizing parameter. This setting has been taken in consideration due to practical issues: in real applications, a normal algorithm may overfit the data (and the phenomenon goes under the name of *overcomplete representation*) and the regularization makes the problem numerically stable. If we want to find sparse matrices W, H, we can set J_1 and J_2 as the Frobenius norm squared [29], but other applications may want to minimize the L_1-norm of H or W, since it is the convex envelope of the cardinality function, leading to similar convex problems.

Back to the geometrical interpretation, we see that if a column of A or W have some zero element, then the corresponding nonnegative vector is on the border of $\mathbb{R}^n_+$. Considering that $\Gamma(A) \subseteq \Gamma(W) \subseteq \mathbb{R}^n_+$, then a column of A on the border gives more constrains to possible solutions W, and a formulation that ensures the sparsity of W has the same effect.

Separability The separability is a property introduced by [9], and later modified as follows:

Definition 1.1.1. A factorization $A = WH^T$ is separable if W has a permutation of a full-rank diagonal submatrix. Equivalently, for each column there is a positive entry that is the only nonzero element of its own row.

Under this condition, given A and W, then H is easily computable, it is unique, and moreover is composed by k rows of A rescaled with the k positive elements in W that makes the decomposition separable. Moreover [1] and [14] showed exact polynomial-time algorithm for computing separable solutions of the NMF under this hypothesis, against the NP-Hardness of the original problem[33]. Less restrictive conditions like the full-rank of W leads only to a computational time of $O((nm)^{r^2})$ for the Exact NMF (in this case, called *Simplicial Factorization*), but still can't escape the curse of NP-Hardness.

Geometrically, each column of W in a separable factorization belongs to a different $n-r+1$ vectorial space defined by its $n-1$ zero components, meaning that they are situated again on a border, with less degrees of freedom, as in the sparse case. Moreover, the decomposition $A = WH^T$ is separable if and only if $\Gamma(A^T) = \Gamma(H)$, since H is composed by rows of A.

Uniqueness Let's set k such that an exact factorization $A = WH^T$ exists, meaning that the rank of A is less or equal to k. Given any solution (W,H) of the exact or the approximation problem, we can take $S = PD \in \mathbb{R}_+^{k\times k}$ (called *monomial matrix*) where P is a permutation matrix, and D is a positive diagonal matrix, and obtain

$$WH^T = (WS^{-1})(SH^T) \qquad WS^{-1} \in \mathbb{R}_+^{n\times k} \quad HS^T \in \mathbb{R}_+^{m\times k}$$

with (WS^{-1}, HS^T) that is still a solution, since S^{-1} is nonnegative. The solution is thus never unique, but we can inquire the uniqueness without considering permutations and positive rescaling of the matrices columns.

The geometric intuition here is very useful: a permutation of the column of W or a rescaling doesn't change the cone $\Gamma(W)$, and moreover, if a column of W is not an extremal ray of the cone, then it isn't useful, since can be obtained as a combination of the others. These considerations tell us that the solution of NMF is unique if and only if k is set as the nonnegative rank, and there's only one simplicial cone with k extremal rays that contains $\Gamma(A)$.

Given a solution $A = WH^T$, we can now normalize the columns of A and W in norm L_1 through monomial matrices, obtaining $\widetilde{A} = \widetilde{W}\widetilde{H}^T$, where all the three matrices are column stochastics. This leads to the NPP problem already described above, or even to problems requiring to minimize[38] or maximize[34] the volume of the polytope $conv(\widetilde{W})$. Other formulations of NMF set the columns of W or H normalized with different norms, such that the norm L_2 or L_∞.

Even if we look for the cones or the convex hulls, we may still have non-unique solutions, for example when there's a solution with a column of W strictly positive. In this case, in fact, it's easy to see that there's always another nonnegative matrix W' such that $cone(A) \subseteq cone(W) \subset cone(W')$, so the results found on this topic always require an additional hypothesis like sparsity or separability. For example, Gillis showed that

Theorem 1.1.1. *Let $A \in \mathbb{R}_+^{n\times m}$ with $\mathrm{rk}(A) = \mathrm{rk}_+(A) = r$. If A has r non zero-columns, each having $r-1$ zero entries whose corresponding rows have different sparsity patterns, then the exact NMF of A is unique.*

The conditions set are similar to the separability of the factorization, and the request on the rank is also natural since the separation of WH^T tells us that W has full rank, so we expect even A to have the same rank, and it also leads to the full rank of H. Moreover, it has been proved that if A has rank 1 or 2, or one of n,m is less then 4, then this

condition is satisfied.

Eventually, under the same condition on the rank of A, then the uniqueness of the factorization $A = WH^T$ is equivalent to the non-existence of a non monomial matrix Q such that WQ and $Q^{-1}H^T$ are still nonnegative, and, as already said, it is even equivalent to the uniqueness of a polytope with k nonnegative vertices that contains the convex hull of the columns of A.

1.2 Applications

One of the most famous problem in data mining is the pattern recognition, namely the problem of finding similar characteristics in different objects. The areas of Text Mining and Image Processing require methods to solve these problems in order to categorize the data, or to find some simple common features that let us describe the objects in a compressed way.

One of the main tool used nowadays for compression and identification of common features, is the PCA (Principal Component Analysis), obtained from the Singular Values Decomposition of the data. In fact, given a set of objects (images, documents, signals, etc.), identified by real vectors stacked as columns of the matrix A, the PCA finds the best low rank approximation of the original data, and stores it in a space order of magnitudes smaller than the input. We can use the SVD to find the k most prominent features that are common to all the input objects as

$$A_k = U_k \Sigma_k V_k = S_k V_k$$

This formula suggests us to use the columns of S_k as features, and the columns of V_k of the coefficients, in fact each object (column of A) will be approximated as a linear combination of the features, with weights given by the V_k columns. The SVD gives us the best rank k approximation of A in L_2 and Frobenius norm, (and in general in any unitarily equivalent norm) and is relatively easy to compute, so it is widely used in a lot of applications.

A particularity of the PCA is that it produces negative entries in features and the coefficients, even when A is nonnegative. Recently, many applications introduced the nonnegative request to their pattern recognizing algorithm in order to gain the interpretability of the output. In this context, we can use the NMF framework: its decomposition, in fact, preserve the positiveness of the data, making the results readable. We'll see later how it is used in Image Processing, whereas we'll now list some of the other main applications of the NMF.

1.2.1 Clustering

The NMF is useful in a lot of practical problems, since it's strictly related to the clustering problem in the euclidean space. In fact, given m points in $\mathbb{R}^n_+$, we can consider them as columns of $A \in \mathbb{R}^{n\times m}_+$, and factorize them in

$$A \sim WH^T, \qquad W \in \mathbb{R}^{n\times k}_+, \quad H \in \mathbb{R}^{m\times k}_+.$$

In this case W is called *base matrix* and H is the *weight matrix*, since the columns of A can be approximated as linear combinations of W columns, with weights in H. The

columns of W are also called *centroids*, since we can partition the initially considered points into k (or less) clusters, looking for the centroid in W that approximates them better, through the weights in H: in fact, any column in A corresponds to a row in H, and taken the highest weight in that row, we can select the best-fit centroid. After a normalization of the elements in H, we can also see the columns of W like real points in the space, and the actual centers of their own cluster.

This partition suffers from the lack of independence of the columns of W, the non-uniqueness of the nonnegative factorization, and its instability, in particular his susceptibility to transformations of the space, such as rotations, translations and scaling. One of the possible solution to these problem is to use the variant SymNMF on a similarity matrix S, like

$$S_{ij} = \exp(-c\|x_i - x_j\|^2).$$

In fact, this formulation is invariant by translation and rotation, it solves the independence problem, and the scaling is dealt with by setting the right constant $c > 0$. We also know that this matrix is positive semidefinite; in fact, if we define a matrix *hollow* if all its diagonal elements are zeros, then the following theorem holds.

Theorem 1.2.1. *[32] Given an hollow symmetric matrix $A \in \mathbb{R}^{n\times n}$, then the following are equivalent:*

- *A is a matrix of square distances of n points in $\mathbb{R}^m$.*
- *$\exp(-\lambda a_{ij}) = S_{ij}$ is a positive semidefinite matrix.*

The dimension of the problem changes into $m \times m$, that is usually quite larger than $n \times m$, but the method tends to generate better solutions, and coupled with an hierarchical strategy, can handle even density-type clusters[12]. In this situation, it is also common to modify the classical structure of NMF in order to exploit the symmetry of S: more precisely, we can solve the SymNMF problem

$$\min_{W\in\mathbb{R}^{n\times k}_+} \|S - WW^T\|_F^2, \tag{1.3}$$

with k fixed as usual. One possibility is also to relax the symmetric condition adding a penalty factor, obtaining a penalized SymNMF

$$\min_{W,H\geq 0} \|A - WH^T\|_F^2 + \alpha\|W - H\|_F^2, \qquad W,H \in \mathbb{R}^{n\times k}_+,$$

where $\alpha > 0$ is a parameter that adjust the loss of symmetry in W and H. We notice that if the solution to this problem is symmetric, that is $W = H$, then the couple (W,H) is also a solution to (1.3).

Partial Clustering From the clustering problem, we can consider a slightly modified problem that arises when we work with incomplete informations. Let's suppose that our m points $\{x_1, x_2, \ldots, x_m\}$ in $\mathbb{R}^n_+$ are partially classified, that is, there exists $c_1, c_2, \ldots, c_r$ classes such that each one the first s points belongs to a class. We can then define the binary matrix

$$C \in \{0,1\}^{s\times r}, \qquad C_{ij} = \begin{cases} 1 & \text{if } x_i \text{ belongs to } c_j \\ 0 & \text{otherwise} \end{cases}$$

and the binary matrix

$$B = \begin{pmatrix} C & 0 \\ 0 & I \end{pmatrix}$$

so that, given Z any $(r+m-s)\times k$ matrix, and $H = BZ$, we have that if x_i and x_j belong to the same class, then the rows h_i and h_j are the same. We can now state the problem [27], similar to NMF,

$$\min_{W,H\geq 0} \|A - WH^T\|_F^2 = \min_{W,Z\geq 0} \|A - WZ^T B^T\|_F^2, \qquad W \in \mathbb{R}_+^{n\times k} \quad Z \in \mathbb{R}_+^{(r+m-s)\times k}.$$

We notice that a clustering obtained through this problem forcefully puts in the same cluster points within the same class.

1.2.2 Text Mining

Other examples of NMF usage are the Text Mining methods that try to categorize a lot of documents by topic. This type of application belongs to a generic pattern recognition category, along with the Signal Analysis, that wants to detect monophonic input in a complex audio track, and Image Processing, whose aim is to pinpoint common small shapes into a large set of images.

In Text Mining, if we have a lot of text files, we can build the dictionary of their words, and compute the relative frequencies of the words in respect with the documents; these will be nonnegative vectors, and texts with similar topics will have similar words frequencies. The NMF applied to the matrix of frequencies extracts the sets of words that refer to the same topics, putting them in W, and the topics of a single document can be read in the coefficients of H, since they'll be approximated by the combination of the topic words, each with a nonnegative coefficient that can tell us how much every single topic is relevant in the text.

Here each property of the NMF already listed have a special role:

- The uniqueness of the solution is important, since different outputs result into different classifications of the documents, meaning that the supposed topics change from one solution to the other, and it is not conceivable at all.
- The input is sparse, since if we have a lot of documents that refers to different topics, the overall dictionary will be larger than the words used in each text. In particular, even W will be sparse, since its columns represent the topics, and each one of them only takes a small fraction of the words used. Thus, in this application, sparsity is a natural request to be called upon the output.
- Even separability have a meaning in this context: a factorization will be separable if for each topic there exists at least one word that doesn't belong to any other topic. This is also a very reasonable assumption we can make on the data.

These are enough reasons for justifying the use of regularizing parameters in order to ensure the sparsity of the output [2].

1.3 Algorithms

The results on the Exact NMF factorization also gives us a bound on the computational time for the Approximate NMF:

Theorem 1.3.1. *Let $A \in \mathbb{R}_+^{n\times m}$ such that there exists a nonnegative couple (W,H) with inner dimension k satisfying $\|A - WH^T\|_F \leq \varepsilon\|A\|_F$. Then there is an algorithm that computes nonnegative matrices (W',H') satisfying*

$$\|A - W'(H')^T\|_F \leq O(\varepsilon^{1/2}k^{1/4})\|A\|_F$$

in time $2^{\text{poly}(k\log(1/\varepsilon))}\,\text{poly}(n,m)$.

Usually k is fixed and low, so we could consider it a constant, but n and m are really high parameter, so in practical uses, a time of $\text{poly}(n,m)$ is acceptable only when it is linear in both n and m.

Since NMF is a useful tool in a variety of applications, the scientific literature and software tools on the subject are rapidly expanding, but the majority of existing algorithms are typically iterative, and converge at local minima in order to keep the complexity low when dealing with huge amount of input data. In general, they follow the same scheme:

General NMF Algorithm
Inputs : $k \in \mathbb{N}, A \in \mathbb{R}_+^{n\times m}$

Initialize W (and H)
repeat
    update W,H
until a Stop Condition is satisfied

At each external iteration, the matrices W,H, or only one of them, are randomly initialized in order to converge at different local minima, or we use particular initializing techniques based on PCA or k-means clustering. The internal iterations, where we update W,H, are executed until a stopping condition is verified, for example when $\|A - WH^T\|$ is small enough, or when the number of iteration is too high, meaning that the convergence is really slow. In order to rise the algorithm probability of success and lower the computational cost, usually the stopping condition include a control on the gain between two consecutive internal iterations: when the gain is low, then W,H already reached a minimum, so it's useless to go ahead anymore.

In general, the solving algorithms use the fact that we can decouple the problem into

$$\min_X \|A - WX^T\|_F^2, \qquad \min_X \|A - XH^T\|_F^2.$$

The main differences between the algorithm are the methods used to solve these two subproblems, and the domain of the variable X. In all its formulation, though, the variable X varies in a convex set, making the two subproblem convex, so all the approximation methods exploit the fact that the local minima are actually absolute minima.

Even with alternative versions, the same idea is applied. For example, in the Penalized Symmetric NMF setting

$$\min_{W,H\geq 0} \|A - WH^T\|_F^2 + \alpha\|W - H\|_F^2, \qquad W,H \in \mathbb{R}_+^{n\times k},$$

we can write two associated convex problems fixing the matrices W, H one at a time

$$\min_{H\in\mathbb{R}_+^{n\times k}} \left\| \begin{pmatrix} A \\ \sqrt{\alpha}W^T \end{pmatrix} - \begin{pmatrix} W \\ \sqrt{\alpha}I \end{pmatrix} H^T \right\|_F^2, \qquad \min_{W\in\mathbb{R}_+^{n\times k}} \left\| \begin{pmatrix} A \\ \sqrt{\alpha}H^T \end{pmatrix} - \begin{pmatrix} H \\ \sqrt{\alpha}I \end{pmatrix} W^T \right\|_F^2$$

and similarly, in the CNMF we can encode the regularization parameters of

$$\min_{W,H} \|A - WH^T\|_F^2 + \alpha\|W\|_F^2 + \beta\|H\|_F^2, \qquad W \in \mathbb{R}_+^{n\times k}, \quad H \in \mathbb{R}_+^{m\times k}$$

into the two convex sub-problems

$$\min_{H\in\mathbb{R}_+^{n\times k}} \left\| \begin{pmatrix} A \\ 0 \end{pmatrix} - \begin{pmatrix} W \\ \sqrt{\beta}I \end{pmatrix} H^T \right\|_F^2, \qquad \min_{W\in\mathbb{R}_+^{n\times k}} \left\| \begin{pmatrix} A^T \\ 0 \end{pmatrix} - \begin{pmatrix} H \\ \sqrt{\alpha}I \end{pmatrix} W^T \right\|_F^2.$$

We'll thus focus mainly on how to solve the convex subproblem on the real field or on its nonnegative orthant.

1.3.1 Initializing Algorithms

NNDSVD The solution to the problem without positivity constraints obtained through the SVD, in general doesn't solve the NMF problem, because there may be some negative elements in the matrices W, H, but if we put the negative elements to zero, or we perform some other process in order to make them nonnegative matrices, we can use the SVD solution as a starting point for the iterative algorithms, in spite of initializing W and H randomly.

The Nonnegative Double Singular Value Decomposition (NNDSVD) try to adjust the SVD in a slightly different way, producing directly a nonnegative decomposition [4]. Given the SVD of A

$$A = \sum_{i=1}^{k} \sigma_i u_i v_i^T,$$

consider the positive and negative parts of the singular vectors

$$u_i^+ = u_i .* (u_i > 0) \in \mathbb{R}_+^n, \qquad u_i^- = -u_i .* (u_i < 0) \in \mathbb{R}_+^n, \qquad u_i = u_i^+ - u_i^-,$$
$$v_i^+ = v_i .* (v_i > 0) \in \mathbb{R}_+^m, \qquad v_i^- = -v_i .* (v_i < 0) \in \mathbb{R}_+^m, \qquad v_i = v_i^+ - v_i^-,$$

so we can write

$$A = \sum_{i=1}^{k} \sigma_i \left[u_i^+ (v_i^+)^T + u_i^- (v_i^-)^T\right] - \sum_{i=1}^{k} \sigma_i \left[u_i^+ (v_i^-)^T + u_i^- (v_i^+)^T\right]$$

and it's possible to prove that the SVD decomposition of $(u_i v_i^T)^+$ is

$$(u_i v_i^T)^+ = \|u_i^+\|\|v_i^+\| \frac{u_i^+}{\|u_i^+\|} \frac{(v_i^+)^T}{\|v_i^+\|} + \|u_i^-\|\|v_i^-\| \frac{u_i^-}{\|u_i^-\|} \frac{(v_i^-)^T}{\|v_i^-\|}.$$

The idea is to approximate each $u_i v_i^T$ with the leading singular component of $(u_i v_i^T)^+$, so we set the columns of W and H as follows:

$$\begin{cases} w_i = \sigma_i u_i^+ & h_i = v_i^+ \quad \text{if } \|u_i^+\|\|v_i^+\| > \|u_i^-\|\|v_i^-\|, \\ w_i = \sigma_i u_i^- & h_i = v_i^- \quad \text{if } \|u_i^-\|\|v_i^-\| \geq \|u_i^+\|\|v_i^+\|. \end{cases}$$

The variables W and H are both positive, but they are usually sparse, so they get modified by adding an ε smaller than the mean of elements to the zero entries.

In the symmetric case, this method can be adjusted in order to get $W = H$, and it gains in computational cost, since the SVD can be replaced by a diagonalization process.

The drawback of this initialization method it's that has no random part, so an NMF algorithm will output only one local minimum. It is nonetheless one of the optional initialization routine in the NMF algorithm of the Python package Sklearn, along with its variants.

K-means Another way to initialize the matrices employs a spherical k-means clustering [35]. As we saw before, NMF is strictly related to the clustering problem in the euclidean space, so it makes sense to use a clustering algorithm in order to boost the performance of the factorization.

In particular, given the nonnegative matrix $A \in \mathbb{R}_+^{n\times m}$, we can see its columns as points in $\mathbb{R}_+^n$, and apply a simple k-means clustering, or a spherical one to obtain the matrix of centroids W.

The spherical k-means guarantees the linear independence of centroids, improving the initialization obtained for the factorization. Also, choosing different starting centroids for the k means algorithms, we can obtain different initial W, so this process is not deterministic and the final output of NMF may vary.

1.3.2 Projected Methods

These methods solve iteratively the convex problems associated to NMF without positivity constraints

$$\min_{X\in\mathbb{R}^{m\times k}} \|A - WX^T\|_F^2, \qquad \min_{X\in\mathbb{R}^{n\times k}} \|A - XH^T\|_F^2,$$

and then project them on the positive orthant, usually putting negative entries of W and H to zero.

ALS The Alternating Least Square method solve the two subproblems with a QR decomposition. For example, given the update of H, we can decompose W into QR, where Q is an orthogonal matrix of size $n \times n$, and R a rectangular and upper triangular matrix of size $n \times r$. Since we usually impose $n >> r$, the R matrix has only r nonzero rows, so we can delete the other $n-r$ rows, and the last $n-r$ columns of Q to obtain $\widetilde{Q}\widetilde{R}$ with $\widetilde{Q}$ a rectangular matrix of size $n \times r$ with orthonormal columns, and $\widetilde{R}$ a square upper triangular matrix of size $r \times r$, so we can rewrite the problem as

$$\|A - WX^T\|_F^2 = \|A - \widetilde{Q}\widetilde{R}X^T\|_F^2 = \|\widetilde{Q}^T A - \widetilde{R}X^T\|_F^2,$$

and solve it columnwise through a $k \times k$ triangular system. Various author have also proposed quasi-Newton modifications to the algorithm in order to gain in computational time, like [37].

One of the most common way to modify W, H so that they become nonnegative, is to put the negative entries to zero, but it usually makes the error rise so much that at the end of a single update loop, the overall gain can be negative. One way to deal with this problem is to use the following OBS method.

OBS This is a second-order method used in origin to prune Neural Networks (hence the name Optimal Brain Surgeon)[18]. The operation to put the negative entries of W,H to zero makes the error $\|A - WH^T\|$ rise too much, so we can look for the best matrices $\widetilde{W},\widetilde{H}$ that have zeros on that positions. Namely, after we solve

$$\widetilde{W} = \underset{X\in\mathbb{R}^{n\times k}}{\arg\min} \|A - XH^T\|_F^2,$$

with ALS or modified algorithms, we look for

$$W = \underset{X\in\mathbb{R}^{n\times k}}{\arg\min} \left\{ \left\|A - XH^T\right\|_F^2 \;\middle|\; (\widetilde{W} < 0) .* X = 0 \right\},$$

and then we put the nonnegative entries of W to zero. A visual representation of this phenomenon is in the Figure 1: the dashed lines are contour lines for a function we want to minimize, and it can be noted that the ALS algorithm starting at any point P_1 always return in 1 step the projection P_2 of the correct solution for the unconstrained problem Q^*, that may have greater error value than the starting point, whereas the OBS modification produces in 1 step the correct solution P^*.

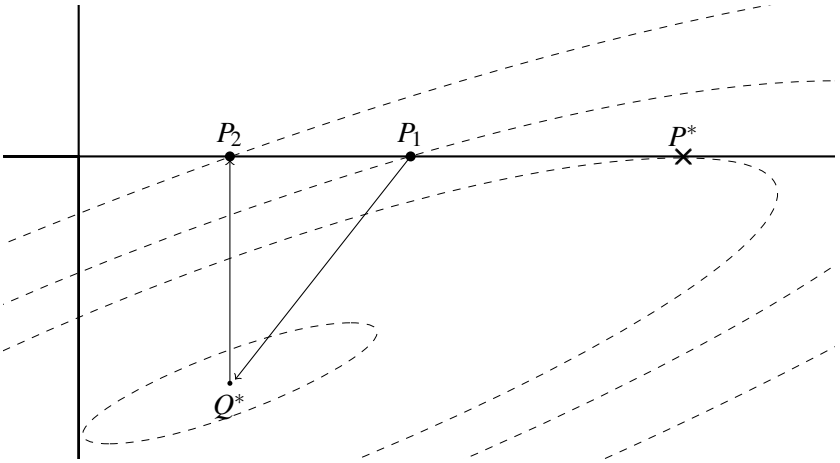

Figure 1

ALSOBS Update Method
Inputs : $A \in \mathbb{R}_+^{n\times m}, \quad W \in \mathbb{R}_+^{n\times k}$

$\widetilde{H} = \arg\min_{X\in\mathbb{R}^{m\times k}} \left\|A - WX^T\right\|_F^2$
$H = \arg\min_{X\in\mathbb{R}^{m\times k}} \left\{ \left\|A - WX^T\right\|_F^2 \;\middle|\; X_{ij} = 0 \;\forall (i,j) : \widetilde{H}_{ij} < 0 \right\}$
put negative entries of H to zero.
$\widetilde{W} = \arg\min_{X\in\mathbb{R}^{n\times k}} \left\|A - XH^T\right\|_F^2$
$W = \arg\min_{X\in\mathbb{R}^{m\times k}} \left\{ \left\|A - XH^T\right\|_F^2 \;\middle|\; X_{ij} = 0 \;\forall (i,j) : \widetilde{W}_{ij} < 0 \right\}$
put negative entries of W to zero.

The analogy with neural network is that we're literally "pruning" the negative entries of W and H. We observe that the operation to put negative entries to zero is common also to algorithms different from ALS, so we can use OBS wherever we need.

PG We can use a method of Projected Gradient to solve problem, where the updates of H, W are simultaneously updated through the rule

PG Update Method
Inputs : $A \in \mathbb{R}_+^{n\times m}, \quad W \in \mathbb{R}_+^{n\times k}, \quad H \in \mathbb{R}_+^{m\times k}$

$(W,H) = (W,H) - \alpha(\nabla_W \Phi(W,H), \nabla_H \Phi(W,H))$
Set negative entries of H, W to zero.

or alternated as

PG Update Method
Inputs : $A \in \mathbb{R}_+^{n\times m}, \quad W \in \mathbb{R}_+^{n\times k}, \quad H \in \mathbb{R}_+^{m\times k}$

$H = H - \alpha_H \nabla_H \Phi(W,H) = H - \alpha_H (HW^T W - A^T W)$
Set negative entries of H to zero.
$W = W - \alpha_W \nabla_W \Phi(W,H) = W - \alpha_W (WH^T H - AH)$
Set negative entries of W to zero.

Here α_W and α_H are the magnitude of the step along the decreasing direction given by the gradient, and usually it's dependent on W and H, or on the number of the iteration reached, and they generally tend to zero.

A drawback of this algorithm is that usually after the first iteration, (W,H) will converge to the useless stationary point $(0,0)$, but since the parameter α is chosen at each step in order to lower the error, it's sufficient to find an initializing algorithm ensuring the initial couple (W_0, H_0) to satisfy

$$\|A\|_F > \|A - W_0 H_0^T\|_F.$$

The descending property of the methods highly depends on the choice of the parameters, that have to be set through line search (usually using the Armijo rule or first order approximation).

It is actually one of the solving algorithm included in the NMF routine of Python package sklearn, but has been deprecated in the newest version.

1.3.3 Convex Nonnegative Problems

An other option is to use an Alternating Nonnegative Least Squares(ANLS) method, that considers the convex subproblems

$$H = \underset{X\in\mathbb{R}_+^{m\times k}}{\arg\min} \|A - WX^T\|_F^2, \qquad W = \underset{X\in\mathbb{R}_+^{n\times k}}{\arg\min} \|A - XH^T\|_F^2.$$

These two problems are usually difficult to solve exactly, and makes the computational cost rise substantially, but it also guarantees that the limit points of the (W,H) sequence in the algorithm is a local minimum of the error function thanks to the following theorem[3][17]:

Theorem 1.3.2. *Let f be a continuously differentiable function over $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_m$, where $\Omega_i \subseteq \mathbb{R}^{n_i}$ are closed convex sets, and $\sum_i n_i = n$. Let's partition the vector $x \in \mathbb{R}^n$ into $x = (x_1, \ldots, x_m)$ where $x_i \in \mathbb{R}^{n_i}$, and let's generate a succession of points $y^{(k)} \in \mathbb{R}^n$ where $y^{(0)}$ is random, and $\forall\, 0 \leq r$, $0 < i \leq m$*

$$\begin{cases} y_i^{(rm+i)} = \arg\min_{y_i \in \Omega_i} f(y_1^{rm+i-1}, \ldots, y_{i-1}^{rm+i-1}, y_i, y_{i+1}^{rm+i-1}, \ldots, y_m^{rm+i-1}) \\ y_j^{(rm+i)} = y_j^{(rm+i-1)} \quad \forall j \neq i \end{cases}$$

where the minimum is uniquely attained. Then every limit point of the succession $y^{(K)}$ is a local minimum of $f(x)$ over Ω. If $m = 2$, the uniqueness of the minimum is not required.

An algorithm that solves it completely is the Active Set Method, and variants like the Block NNLS that strife for lowering its computational cost. The most used methods are inexact ones thanks to their efficiency, and even if there's no theorem that binding their errors, or proving their convergence, it has been shown experimentally that the solutions generated are still pretty accurate, and the speed is considerably higher.

Active Set With the nonnegativity condition, we can adopt a dual-primal approach, embodied in the Active Set method. We notice that the problem can be rewritten in a row-wise fashion, in fact

$$\min_{Y \in \mathbb{R}_+^{n \times k}} \|A - YH^T\|_F^2 = \min_{X \in \mathbb{R}_+^{k \times n}} \|HX - A^T\|_F^2 = \sum_{i=1}^{n} \min_{x_i \in \mathbb{R}_+^k} \|Hx_i - a_i\|_2^2,$$

where x_i and a_i are the rows of Y and A. This means that we can solve n smaller problem with a Non-Negativity-Constrained Least Squares algorithm (NNLS), using the KKT conditions

$$y_i = \frac{1}{2}\nabla_{x_i}\|Hx_i - a_i\|_2^2 = H^T(Hx_i - a_i),$$
$$x_i .* y_i = 0, \qquad x_i, y_i \geq 0.$$

Since the problem is convex, a local minimum is also a global minimum, so we need to find only one solution of the KKT conditions. For any nonnegative vector, we can define their Active and Passive sets as a partition of the n indexes given by

$$\mathscr{A}(x) = \{\, i \mid x_i = 0 \,\}, \qquad \mathscr{P}(x) = \mathscr{A}(x)^c = \{\, i \mid x_i > 0 \,\},$$

and from the last two KKT conditions, we notice that $\mathscr{P}(x_i) \subseteq \mathscr{A}(y_i)$ and conversely $\mathscr{P}(y_i) \subseteq \mathscr{A}(x_i)$. Moreover, if $\bar{x}$ is an optimal solution, and we know $\mathscr{S} = \mathscr{A}(\bar{x})$, we can restrict the vectors and the matrix to these coordinates $H_{\mathscr{S}}, (x_i)_{\mathscr{S}}, (a_i)_{\mathscr{S}}$, solve the unconstrained problem

$$\min_{(x_i)_{\mathscr{S}} \in \mathbb{R}^{|S|}} \|H_{\mathscr{S}}(x_i)_{\mathscr{S}} - (a_i)_{\mathscr{S}}\|_2, \tag{1.4}$$

and the solution will be positive, thus optimal.

The Active-Set method aim is to find $\mathscr{A}(\bar{x})$ by iteratively generating various partitions of the n indexes through some update rule, that may vary from algorithm to algorithm, for example in [22] and [23]. In general, the updates are decided looking at the sets associated to x and y at each step, so we have to compute

$$y = H^T_{\mathscr{P}(y)}(H_{\mathscr{P}(x)}x - a_i),$$

and for (1.4), we need to solve the system

$$H^T_{\mathscr{F}}H_{\mathscr{F}}(x_i)_{\mathscr{F}} = H^T_{\mathscr{F}}(a_i)_{\mathscr{F}}.$$

This method is assured to finish and get the right solution in a finite number of steps, since the updates make the cost function decrease, and there are a finite number of possible partitions of the indexes, even if they can be potentially exponential.

Block NNLS Solving n distinct subproblem for each iteration is usually too expensive for a NMF algorithm, but using combinatorial rules or block pivoting, like in the Block Principal Pivoting ANLS [23], we are able to cut the computational cost.

First of all, we can compute only one time the matrices H^TH and H^TA^T, so that all the matrix used in the previous algorithm

$$H^T_{\mathscr{P}(y)}H_{\mathscr{P}(x)}, \qquad H^T_{\mathscr{P}(y)}a_i, \qquad H^T_{\mathscr{F}}H_{\mathscr{F}}, \qquad H^T_{\mathscr{F}}(a_i)_{\mathscr{F}}$$

are actually their submatrices, and we don't have to generate them at each step.

Moreover, we can update different columns with the same, or similar, active set altogether, and also the process of update all the active sets can be done contemporaneously. Like the NNLS, even this algorithm is bound to terminate in a finite number of steps and to return the correct solution. The Block NNLS is indeed one of the most performing algorithm known nowadays,

FNMA The FNMA (Fast Nonnegative Matrix Approximation) is a method that uses a gradient descend algorithm that updates at each step only some of the variables, chosen through a condition similar to the Active Set [21]. In particular, given the gradient of the error function and the current approximation W, It defines the set of fixed variables as

$$I_+ = \left\{ (i,j) \;\middle|\; W_{i,j} = 0 \quad \nabla_W F(W,H) > 0 \right\},$$

and the set of free variables as his complementary $G = I_+^C$. Ideally, we need to update only the free variables, since the fixed ones already satisfy the KKT conditions. So the updates will follow the rule

$$W_G = W_G - \alpha D \nabla_{W_G} F(H, W_G),$$

and then we put the negative entries of W_F to zero. Here we used a parameter $\alpha > 0$ and a positive definite scaling matrix D that are updated at each stage, the first with a linear search, the second with an approximation of the inverse of Hessian matrix. In particular, the update of D follows the second-order BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm.

This algorithm is bound to converge to a stationary point(and thus resolving completely the problem) whenever H has full rank. It has also an Inexact version (called

FNMA[I]) where the parameter α and the number of iteration is given as an input, and the matrix D is exactly the inverse of HH^T, so it doesn't vary along the iterations. This version has weaker convergence results, in particular it doesn't solve the problem exactly, but it's significantly faster than the exact version.

MU This algorithm tries to solve two problems correlated with the classical NMF, using its normal equation forms. In fact, supposing $A \sim WH^T$, we also have

$$W^T A \sim (W^T W)H^T, \qquad AH \sim W(H^T H). \tag{1.5}$$

The advantage is that both $W^T W$ and $H^T H$ are matrix in $\mathbb{R}_+^{k\times k}$, so very small compared to A. On the other hand, the condition number of the matrices gets squared, leading to instability of the methods.

The Multiplicative Update has been one of the first algorithm proposed for solving NMF, since it can be viewed as a Projected Gradient method, where the parameters α_H and α_W are the matrices

$$\alpha_H = H./(W^T WH), \qquad \alpha_W = W./(WHH^T).$$

The update operation can be written as

MU Update Method

Inputs : $A \in \mathbb{R}_+^{n\times m}, \quad W \in \mathbb{R}_+^{n\times k}, \quad H \in \mathbb{R}_+^{m\times k}$

$H = H .* \left[W^T A ./ (W^T WH^T + \varepsilon \mathbf{1})\right]$

$W = W .* \left[AH ./ (WH^T H + \varepsilon \mathbf{1})\right]$

where $\mathbf{1}$ is the matrix composed only by 1, and ε is a small constant that assure us the positivity of the matrix at the denominator (usually $\varepsilon \sim 10^{-9}$). As shown in [26], each update always makes the error function decrease, and it preserves the nonnegativity of the variables, so that we don't need the projection operator.

It's wrong that the method always converge to a stationary point, but it tends to produce sparse solutions, since each update is element-wise, so if H(or W) has a zero entry at any step, the algorithm will return H(or W) with a zero in that position.

CD The Coordinate Descend is an iterative convergent method designed to return a good solution in a short amount of time, but in general, the output is not optimal [20].

Given the convex problem with variable W, we try to approximate the optimal solution W one entry at a time. In fact, if at the r-th step we have an approximation W^r, we choose a couple of index (i, j), and solve

$$s^* = \arg\min_s \left\{ \left\| A - (W^r + s \cdot e_i e_j^T)H^T \right\|_F^2 \;\middle|\; (W^r)_{ij} + s \geq 0 \right\}.$$

We produce the next approximation as $W^{r+1} = W^r + s^* \cdot e_i e_j^T$, and we can define the relative gain as

$$d_{ij} = \|A - W^r H^T\|_F^2 - \|A - W^{r+1} H^T\|_F^2,$$

If we call

$$G(X) = (XH^T - A)H \qquad Q = H^T H,$$

then we know that

$$s^* = \begin{cases} -G(W^r)_{ij} \diagup Q_{jj} & \text{if } Q_{jj} \neq 0 \text{ and } W^{r+1} > 0, \\ -(W^r)_{ij} & \text{otherwise,} \end{cases}$$

$$d_{ij} = -2s^* G(W^r)_{ij} + (s^*)^2 Q_{jj},$$

so we can choose at each step the couple (i, j) by maximizing the gain d_{ij}, and even the stopping condition depends on the magnitude of the maximum gain of the step.

It is actually one of the solving algorithm included in the NMF routine of Python package sklearn, preferred with respect to the PG methods already discussed.

HALS A slight modification of this setting is called HALS [5], that decides the order of the update described in the CD setting preemptively, updating the columns of W and H in order. This method derives from the decomposition of the problem

$$\|A - WH^T\|_F^2 = \|A - \sum_i W_{:,i}(H_{:,i})^T\|_F^2.$$

In fact, if we fix the matrix H and the quantity

$$A^{(j)} := A - WH^T + W_{:,j}H_{:,j}^T = A - \sum_{i \neq j} W_{:,i}(H_{:,i})^T,$$

then what remains is an expression in the variables $W_{1j}, W_{2j}, \dots W_{nj}$ that we know how to solve exactly, since it can be decomposed into n disjoint problem we already solved in the CD section.

$$\|A^{(j)} - W_{:,j}H_{:,j}^T\|_F^2 = \sum_i \|A_{i,:}^{(j)} - W_{ij}(H_{:,j})^T\|_F^2.$$

We'll report only the update of W(it is analogous for H):

HALS Update Method for W
Inputs : $A \in \mathbb{R}_+^{n \times m}, \quad W \in \mathbb{R}_+^{n \times k}, \quad H \in \mathbb{R}_+^{m \times k}$

$B = AH$
$C = H^T H$
for $i = 1, 2, \dots, k$ **do**
  $D_{:,i} = \sum_{l=1}^{i-1} W_{:,l}C_{li} + \sum_{l=p+1}^{i} W_{:,l}C_{li}$
  $W_{:,i} = \max\left(0, \frac{B_{:,i} - D_{:,i}}{C_{ii}}\right)$
end for

Since the first operation of the update is the most expensive in terms of computational time, we can repeat the internal for cycle, obtaining an accelerated HALS algorithm[15] essentially modifying the stopping conditions for the single update. This acceleration can also be applied to other methods such as PG and MU through an accurate computation of the computational cost of each step.

Thanks to Theorem 1.3.2, we know that this method surely converge to a stationary point, since we're minimizing a function on $(n+m)k$ variables in order, and each variable is optimally updated on a convex domain. Moreover, HALS is, on par with the Block NNLS, one of the most performing algorithms known nowadays.

Chapter 2

Permutation NMF

2.1 Images and Permutations

One of the problem confronted by researchers in image processing is to decompose different images into common parts or features, both for identification purposes or for compression ones. For example, a common technique used in animation in order to contain the memory used is to not memorize into digital supports every pixel of each single frame, but to memorize only particular compressed or coded informations that lets a recorder to reproduce the film with little loss of quality.

In general, when confronted with a large set of images like the frames of a film, or a database of similar pictures, it can be convenient to memorize the common parts only one time, gaining space and also computational time for the recombining process. The problem is thus to find an efficient algorithm that automatically recognizes the common features and an intelligent way of storage of the informations.

Given a gray-scale image M expressed as a matrix of pixels, with values in the real range $[0,1]$, we can transform it into a real vector with as many coordinates as the pixels in the image. In particular, if $M \in \mathbb{R}_+^{r\times s}$, then we stack the columns of the matrix on top of one another, and obtain the vector $v \in \mathbb{R}_+^{rs}$ defined as

$$v_{i+(j-1)r} = M_{ij} \quad \forall i,j.$$

Given a set of pictures $\{M_i\}_{i=1:m}$ of the same shape, we can now vectorize them and stack the corresponding vectors as the columns of our data matrix A, and if we call $n = rs$ the number of pixels of a single picture, A becomes a nonnegative matrix in $\mathbb{R}_+^{n\times m}$, so, after having fixed the number k of common component we want to find, the NMF framework produces two matrices W,H such that $A \sim WH^T$.

As already noticed, each column of A is approximated by a linear combination of the columns of W, that are nonnegative vectors of length $n = rs$. After having normalized W by multiplication with a diagonal positive matrix (as discussed above), we can see its columns as images in the shape $r \times s$, so a generic column of A, that is one of the original images, is now approximated as the superimposition of the pictures represented by some of the columns of W.

Ideally, the images in W are parts of the pictures in A, like localized objects in the 2D space, so they're usually sparse and disjoint images, that translates into sparse and nearly orthogonal vectors, so the separability is a natural condition to impose, even in this case. In a famous experiment, Lee & Seung [25] processed a set of faces and the

NMF automatically recognized their principal features like eyebrows, lips, noses, eyes, and so on, so that they were immediately human-recognizable. This example shows the importance of NMF as a decomposition tool for graphical entities.

As already said, the sparseness and the choice of k are important factors. The sparseness is an index of the uniqueness of the solution, that is important on the side of interpretation of the output, since different solutions usually brings up set of pictures not human-recognizable as real objects and features. On the side of compression, we can see that the original nm pixels of A are now coded into kn pixels in W and km coefficients in H, so the compression is useful when the approximation is good with a low k. On terms of images, it means that there are few components that span the whole set of pictures.

Transformations Issues When we use NMF on a matrix A we usually expect the original images to have some predominant common features, so that the algorithm can find them with little noise. This may be true in the case of sets of static pictures, when calibrated and centered, but even in the case of facial recognition, there may be cases of misalignment, as already noticed by [13] and many others. In general, the NMF suffers in this cases since it is not invariant under a vast set of transformations, for example shifts, rotations, symmetry, stretches and so on, in fact the common features must be in the exact same positions on the different pictures in order to be pinpointed.

This is a common problem faced in the animations programs, since, even if the subjects in a scene of a footage are the same, they constantly move on the screen, so their detection must follow some temporal scheme, and can't be performed by a simple NMF.

Possible ways to deal with this problem are to change the data in one of the three matrices A, W or H. For example, if we add to A a transformed copy of each original picture for every transformation in a set we choose, then the common features get detected even if they're deformed, but this increases the size of the problem by the square of the number of alterations used, that's usually greater than the number of pixels in a single image. One possible solution is obviously to rise the parameter k, but this leads to instability in the solution, as we already discussed.

A good idea seems instead to rise the quantity of data contained in the matrix H, since we strife to maintain the graphical property of the columns of W to represent the common features of the original images. In the next chapters we'll define new notations and operators to deal with a matrix whose elements are capable to transmit more informations on pictures than simple real numbers.

2.1.1 Permutations

In this document, our focus is on the problems related to the lack of translation invariance of NMF, so we use shift permutations to modify the kind of elements contained in the matrix H. First of all, we review a bit of theory on permutations and their expansion to the permutation algebra, whose elements will be used in the matrix H to encode the space transformations of the picture. Then we define an operator between matrices with real elements or permutation ones, and exploit it to reformulate the NMF problem in order to gain the shift invariance.

Permutation Algebra Given an element $\tau \in \mathbb{R} \times S_n$, it is represented by a couple $\tau \equiv [r, \sigma]$, where r is a real number, and σ is an element of S_n, the group of permutation

of n indexes. It is well defined the action of τ on a real vector $v \in \mathbb{R}^n$ as

$$\tau(v) \in \mathbb{R}^n \ : \ \tau(v)_i = [r, \sigma](v)_i = r v_{\sigma(i)} \quad \forall i.$$

The action of τ on $\mathbb{R}^n$ makes it a linear operator, so it can be represented by a matrix, and in particular, since the action of each permutation $\sigma \in S_n$ is associated with a permutation matrix P_σ, it's easy to see that

$$\tau \equiv [r, \sigma] \implies \tau(v) = r P_\sigma(v).$$

The algebra generated by the permutation group over the real field is denoted as $\mathbb{R}S_n$, defined as

Definition 2.1.1. The algebra $\mathbb{R}S_n$ is composed by sums of $\mathbb{R} \times S_n$ elements

$$\alpha \in \mathbb{R}S_n \implies \alpha = \sum_{\sigma \in S_n} [r_\sigma, \sigma] \quad r_\sigma \in \mathbb{R}.$$

The sum is defined element-wise

$$\alpha = \sum_{\sigma \in S_n} [r_\sigma, \sigma] \qquad \beta = \sum_{\sigma \in S_n} [t_\sigma, \sigma] \implies \alpha + \beta = \sum_{\sigma \in S_n} [r_\sigma + t_\sigma, \sigma],$$

and the multiplication is Cauchy-like

$$\alpha = \sum_{\sigma \in S_n} [r_\sigma, \sigma] \qquad \beta = \sum_{\sigma \in S_n} [t_\sigma, \sigma] \implies \alpha \cdot \beta = \sum_{\sigma \in S_n} \left[\sum_{\tau, \mu \in S_n}^{\tau \circ \mu = \sigma} r_\tau t_\mu, \sigma \right].$$

As before, these elements have a natural action on $\mathbb{R}^n$, that is an extension of the action of $\mathbb{R} \times S_n$, given by

$$\alpha(v) = \sum_{i=1}^{s} [r_i, \sigma_i](v) = \sum_{i=1}^{s} r_i P_{\sigma_i} v = \left(\sum_{i=1}^{s} r_i P_{\sigma_i} \right) v,$$

so there exists an homomorphism of $\mathbb{R}$ algebras $\varphi : \mathbb{R}S_n \to \mathbb{R}^{n \times n}$ that associates to each element of the algebra a real matrix. It is not surjective if $n \geq 2$, since its image is the set of matrices with sum of rows equal to sum of columns, and it isn't injective if $n \geq 3$ since the dimension of $\mathbb{R}S_n$ as vectorial space is $n!$, whereas the dimension of the image is $(n-1)^2 + 1$. The kernel of φ are elements that act on all the vectors in $\mathbb{R}^n$ as the zero matrix, so we call them all "zero elements".

The homomorphism preserves the algebra operations, and in particular the composition of the permutations becomes the multiplication of matrices. We already said that the image of a permutation $\sigma \in S_n$ is a permutation matrix P_σ that is orthogonal, so the inverse coincides with the transposition, and given $\alpha \in \mathbb{R}S_n$, we can define its transposition as

Definition 2.1.2. Given $\alpha = \sum_{\sigma \in S_n} [r_\sigma, \sigma]$ an element of $\mathbb{R}S_n$, we define its *transpose* as

$$\alpha' := \sum_{\sigma \in S_n} [r_\sigma, \sigma^{-1}].$$

It is thus easy to prove that $\varphi(\alpha')$ coincides with the transpose of $\varphi(\alpha)$

$$\varphi(\alpha') = \varphi\left(\sum_{\sigma\in S_n}[r_\sigma, \sigma^{-1}]\right) = \sum_{\sigma\in S_n} r_\sigma \varphi(\sigma^{-1})$$

$$= \sum_{\sigma\in S_n} r_\sigma P_\sigma^{-1} = \left(\sum_{\sigma\in S_n} r_\sigma P_\sigma\right)^T = \varphi(\alpha)^T.$$

Shifts Given n a natural number, we can denote as T_n the cyclic subgroup of the permutation group S_n whose elements shift cyclically all the indexes of vectors in $\mathbb{R}^n$ by an integer constant. We'll call σ_p the shift by p position, where $p \in \mathbb{Z}/n\mathbb{Z}$:

$$\sigma_p \in T_n \quad v \in \mathbb{R}^n \quad p \in \mathbb{Z}/n\mathbb{Z} \implies \sigma_p(v) = w \quad : \quad w_i = v_{i+p} \quad \forall i,$$

where the indexes are to be considered modulus n.

The images of T_n through the above mentioned homomorphism φ are sparse circulant matrix. In particular, the element σ_1 is associated to the circulant matrix C that has 1 on the first cyclic superdiagonal and 0 anywhere else, and $\sigma_p = \sigma_1 \circ \cdots \circ \sigma_1$, so $\varphi(\sigma_p) = \varphi(\sigma_1)^p = C^p$ that has 1 on the p-th cyclic diagonal and zero otherwise.

$$\varphi(\sigma_1) = C = \begin{pmatrix} 0 & 1 & & & \\ & 0 & 1 & & \\ & & 0 & \ddots & \\ & & & \ddots & 1 \\ 1 & & & & 0 \end{pmatrix}, \qquad \varphi(\sigma_2) = C^2 = \begin{pmatrix} 0 & 0 & 1 & & \\ & 0 & 0 & \ddots & \\ & & 0 & \ddots & 1 \\ 1 & & & \ddots & 0 \\ 0 & 1 & & & 0 \end{pmatrix} \quad \dots$$

$$\sigma_p(v) = C^p v.$$

The elements of type $\alpha = [r, \sigma_p] \in \mathbb{R}_+ \times T_n$ are positive multiples of the circulant matrices described above, and since the shift σ_p is completely identified by the remainder class p, we'll refer to α from now on as the couple $[r, p]$. We'll use these elements to define a new problem with the same shape of a normal NMF, but we need to define beforehand an operator between matrices with entries in this group.

Diamond Operator Let's now suppose that N is a matrix with entries in the above described algebra $\mathbb{R}S_n$, and M is a real matrix. We need an operator to apply the elements of N to the columns of M , so we define the *diamond product* :

Definition 2.1.3 (Diamond Product)**.** The diamond operator between a real matrix $A \in \mathbb{R}^{n\times m}$ and a matrix $N \in (\mathbb{R}S_n)^{m\times k}$ is defined as

$$(A \diamond N)_{:,i} := \sum_j N_{ji}(A_{:,j}),$$

and returns a real matrix in $\mathbb{R}^{n\times k}$.

In other words, the i-th column of the diamond product is a linear combination of permutations of M columns, with coefficients and permutations described by the elements of the N's i-th column.

Let's also define the multiplication between two matrices with entries in the algebra of permutations. Remember that $\mathbb{R}S_n$ is an algebra, so sum and product are well defined, and the elements of $\mathbb{R}S_n$ can be viewed as well as matrices through the homomorphism φ, so the two operations correspond to the usual sum and composition of matrices.

Definition 2.1.4 (Diamond Product)**.** The diamond operator between two matrices $M \in (\mathbb{R}S_n)^{n\times m}$ and $N \in (\mathbb{R}S_n)^{m\times k}$ is defined as

$$(M \diamond N)_{ij} := \sum_k N_{kj} \cdot M_{ik},$$

and returns a matrix in $(\mathbb{R}S_n)^{n\times k}$.

This operation differs from the normal multiplication of matrices only because $\mathbb{R}S_n$ isn't a commutative algebra, so we need to specify the order of the multiplication between the elements. The inverted order is necessary to partially maintain the associativity of the operation: given a real matrix A, and two matrices N, M with elements in the algebra, it's easy to verify that

$$(A \diamond M) \diamond N = A \diamond (M \diamond N).$$

Ideally we need to invert the elements of N and M since M is the first to act on the columns of A, followed by N.

One downside of this operation is that it doesn't cope well with the normal matrix multiplication: given A, B real matrices, and M a matrix in the permutation algebra, then

$$A(B \diamond M) \neq (AB) \diamond M.$$

Some of this operator's properties are given and proved in the AppendixA.

Let's now return to image transformations, and focus on a particular subgroup of the permutation algebra.

2.1.2 PermNMF

Given a gray-scale image M, we've seen how to transform it into a vector $v \in \mathbb{R}_+^{rs}$. We want now to codify a shift on the image as a vectorial transformation: a shift of the original image M by r_1 position on the horizontal axis and s_1 position on the vertical one will be encoded as a circular shift on v of magnitude $p = r_1 r + s_1$, that is, we produce a vector w whose i-th coordinate is the $(i+p)$-th coordinate of v.

This shift is easily encoded by the already defined elements $[r, p] \in \mathbb{R}_+ \times T_n$, where $n = rs$ is the number of pixels in M, and the length of v. For example, if $\tau = [1, r_1 r + s_1]$, then $\tau(v)$ corresponds to the translation of the image described above, and we can vary the constant term to regulate the intensity of the image. With this tools, we are not able to formulate the problem we want to solve.

Now we reconsider the classic NMF, and widen the domain of the matrix H. Our aim here is to find a new method to decompose pictures into common components, even when they're shifted, so, like in the NMF, we stack the original images as columns of

the matrix A, and look for a matrix W whose columns are the wanted common features, and a matrix H with elements in $\mathbb{R}_+ \times T_n$, so that it can tell us both the intensity and the position of each component in W into each original picture in A.

In particular, we want to rewrite the NMF problem as

Problem 2.1.1 (PermNMF)**.** Given a matrix A is in $\mathbb{R}_+^{n\times m}$, we want to find a matrix H in $(\mathbb{R}_+ \times T_n)^{m\times k}$ and a matrix W in $\mathbb{R}_+^{n\times k}$ that minimize

$$F(W,H) = \|A - W \diamond H^T\|_F^2.$$

The diamond operator is defined on elements of $\mathbb{R}S_n$, but we restrict the entries of H to elements in $\mathbb{R}_+ \times T_n$, so that a single image (column of A) is a linear combination of the images represented by the columns of W, but shifted.

Working with this group also ensure a low computational cost: a shift of a vector can be performed in constant times with the right structures, so given a real matrix $A \in \mathbb{R}^{n\times m}$ and a matrix $M \in (\mathbb{R}_+ \times T_n)^{m\times k}$, the operation $A \diamond M$ costs the same as a matrix multiplication, so $O(nmk)$. In our case, the computation of $F(W,H)$ costs $O(nmk)$.

We notice that expanding further the domain of H usually leads to trivial and useless solutions; for example, if we let the elements of H be in $\mathbb{R}_+ T_n$, that are linear nonnegative combinations of permutations in T_n, then even with $k = 1$ there's a trivial solution that decomposes perfectly the matrix A:

$$A = W \diamond H^T, \qquad W = e_1, \qquad H_{i,1} = \sum_j [A_{ij}, j-1].$$

In fact,

$$(W \diamond H^T)_{:,i} = H_{i,1}(W) = \sum_j A_{ij}\sigma_{j-1}(e_1) = \sum_j A_{ij}e_j = A_{:,i}.$$

In other words, a linear combination of the translations of a single pixel can reconstruct any image, so it is an exact and completely useless solution. Moreover, expanding the group T_n usually leads to the dismembering of the images represented by the columns of W, so we stick to work with this framework for this document.

An other particularity of this formulation is that, if we impose each element of H to be of the type $[r,0]$, that is, we fix all the permutations to be the trivial identity, then the problem returns exactly the original NMF, and the diamond operator coincides with the normal matrix multiplication.

2.2 Algorithm

The PermNMF has the same structure of the normal NMF, so we can try to use similar solving algorithms, but we lost the symmetry of the expression, so it's necessary to resort to different methods for the two updates. We'll see that the MU and PG settings can be adapted for the update of W, whereas H will undergo an update similar to CD. In general, we follow the alternating framework already discussed, meaning we update W and H one at a time, and we iterate the updates until the error gets small enough, or the convergence becomes too slow.

Alternated Update Method
Inputs : $A \in \mathbb{R}_+^{n\times m}, \quad W \in \mathbb{R}_+^{n\times k}, \quad H \in (\mathbb{R}_+ \times T_n)^{m\times k}$

$newerr = \|A - W \diamond H^T\|_F$
repeat
  $err = newerr$
  $H = \arg\min_{X\in(\mathbb{R}_+\times T_n)^{m\times k}} \|A - W \diamond X^T\|_F^2$
  $W = \arg\min_{X\in\mathbb{R}_+^{n\times k}} \|A - X \diamond H^T\|_F^2$
  $newerr = \|A - W \diamond H^T\|_F$
until $err - newerr \leq \varepsilon$

Rounding and Threshold In order to choose the error threshold ε, we remember that the pixels into images can be represented by integers in the range $[0,255]$, so an error of ± 0.4 is still acceptable, since it would be rounded off. If we translate this to matrices with pixels in the real range $[0,1]$, then the acceptable error becomes $\pm 0.4/255$, that we round to 10^{-3}. An error matrix $A - W \diamond H^T$ with all elements in absolute value less than 10^{-3} is thus acceptable, and its Frobenius norm squared is $\varepsilon^2 = 10^{-6}nm$.

In the same way, we can show that a change of elements bound by the same constant is again not important, so we use ε to tell if the update does converge too slowly. In fact, given a matrix X, and called E the matrix with all entries equal to 1, then

$$|\|X\|_F - \|X - 10^{-3}E\|_F| \leq \|10^{-3}E\|_F = \varepsilon.$$

For the sake of speed, we'll give only approximated solution to the inner problems, as we'll see in the next two sections, and in the methods used, we will round off to zero the elements in the matrices W and H that are too small. In particular, we already saw how in a normal picture, a value of 10^{-3} for a pixel is negligible, but in general, it is proportional to the intensity of the entire figure. On average, each pixel has an intensity of 0.5, so the mean Frobenius squared norm for an $n \times m$ matrix X is $nm/4$, meaning that a pixel can be put to zero if

$$X_{ij}^2 \leq 4 \cdot 10^{-6}\|X\|_F^2/nm = \delta^2.$$

In the next sections, we'll use the notation $X = X_+$ to indicate that we are setting the entries of X less than δ to zero, including all the negative ones.

Eventually, we'll need also a stopping criterion based on the magnitude of the changes for every step, in order to detect slow convergences. As before, we can say that a step that brings the matrix X to the matrix X' is not relevant if $X - X'$ is element-wise small in comparison to X. In particular, we can stop if

$$\|X - X'\|_F^2 \leq \|\delta E\|_F^2 = \delta^2 nm = 4 \cdot 10^{-6}\|X\|_F^2.$$

2.2.1 Update of W

The update of W requires to solve a convex problem, so we can use a Projected Gradient, that is particularly good for this case, since we can't transpose the expression in order

to obtain the setting of the Active Set algorithms. The ALS method here is inapplicable, since we would need a *QR* decomposition of the matrix H, that's still not well defined, and adaptations of HALS/CD methods become too expensive.

DPG The most simple algorithm simply computes the deepest descend for the opposite direction with respect to the gradient, and then project the obtained element on the positive orthant. This means we have to find the best positive parameter α such that

$$W - \alpha \nabla_W F(W,H)$$

minimizes the error. The computation for the gradient in the algorithm are developed in Appendix A, and it shows that

$$\nabla_W \|A - W \diamond H^T\|_F^2 = -2(A - W \diamond H^T) \diamond H',$$

where the matrix H' entries are the transpose elements of H. If we call $DW = \nabla_W F(W,H)$, then we compute the optimal α by putting its gradient to zero.

$$\frac{\partial}{\partial \alpha}\|A - (W - \alpha DW) \diamond H^T\|_F^2 = 2\alpha \|DW \diamond H^T\|_F^2 + 2\,\text{sum}((A - W \diamond H^T) .* (DW \diamond H^T))$$

$$\implies \alpha^* = -\frac{\text{sum}((A - W \diamond H^T) .* (DW \diamond H^T))}{\|DW \diamond H^T\|_F^2}.$$

Where "sum(B)" indicates is the sum of all entries of the matrix B. Using the property of the Diamond Operator proved in Appendix A, we obtain

$$\begin{aligned}\alpha^* &= -\frac{\text{sum}((A - W \diamond H^T) .* (DW \diamond H^T))}{\|DW \diamond H^T\|_F^2} \\ &= -\frac{\text{sum}(DW .* ((A - W \diamond H^T) \diamond H'))}{\|DW \diamond H^T\|_F^2} \\ &= \frac{\text{sum}(DW .* DW)}{2\|DW \diamond H^T\|_F^2} = \frac{\|DW\|_F^2}{2\|DW \diamond H^T\|_F^2}.\end{aligned}$$

With this, the Deepest PG (DPG) algorithm is presented as

DPG Update Method
Inputs : $A \in \mathbb{R}^{n\times m}$, $W \in \mathbb{R}^{n\times k}$, $H \in (\mathbb{R}S_n)^{m\times k}$

$W' = W$
repeat
  $W = W'$
  $\alpha = \|\nabla_W F(W,H)\|_F^2 / 2\|\nabla_W F(W,H) \diamond H^T\|_F^2$
  $W' = (W - \alpha \nabla_W F(W,H))_+$
until $\|A - W' \diamond H^T\|_F < \varepsilon$ or $\|W' - W\|_F < 2 \cdot 10^{-3}\|W\|_F$
return W'

We use also a *maxiter* variable to bound the number of inner loops performed, that will usually be set as a low number. We'll refer to this function from now on as

$$W = DPG(A,W,H).$$

The operations performed in each cycle of the method have a computational cost of $O(mnk)$, and in particular the most expensive operations are the computation of the gradient and of the error matrix.

PPG Usually the DPG algorithm tends to converge slowly when the global minimum of $F(W,H)$ is not located in the positive orthant, because the negative gradient points outside the feasible region, and the projection has to bring the variable W back at every step. A way to correct this behavior is to not let the direction in each step of the method point outwards, meaning that if W is on a border of the orthant, then the direction shouldn't have a part perpendicular to the border and pointing towards negative coordinates.

After having corrected the step descent direction from $-DW$ to $-D$, we can recompute the deepest step α and obtain

$$\begin{aligned}\alpha &= -\frac{\text{sum}((A - W \diamond H^T) .* (D \diamond H^T))}{\|D \diamond H^T\|_F^2} \\ &= -\frac{\text{sum}(D .* ((A - W \diamond H^T) \diamond H'))}{\|D \diamond H^T\|_F^2} \\ &= \frac{\text{sum}(D .* DW)}{2\|D \diamond H^T\|_F^2}.\end{aligned}$$

We can then write the Positive PG (PPG) algorithm as

PPG Update Method
Inputs : $A \in \mathbb{R}^{n\times m}, \quad W \in \mathbb{R}^{n\times k}, \quad H \in (\mathbb{R}S_n)^{m\times k}$

$W' = W$
repeat
  $W = W'$
  $D = \nabla_W F(W,H)$
  $D(D > 0 \;\&\&\; W = 0) = 0$
  $\alpha = \text{sum}(D .* \nabla_W F(W,H))/2\|D \diamond H^T\|_F^2$
  $W' = (W - \alpha D)_+$
until $\|A - W' \diamond H^T\|_F < \varepsilon$ or $\|W' - W\|_F < 2 \cdot 10^{-3}\|W\|_F$
return W'

We again use a *maxiter* variable to bound the number of inner loops performed. We'll refer to this function from now on as

$$W = PPG(A,W,H).$$

The method has the same asymptotic computational cost of the DPG algorithm, but the operations performed in each cycle are strictly more.

SPG Even with the different Descent Direction in the PPG, the method generates points that don't belong to the positive orthant, so that a projection is always needed. A way to get rid of this operation is to determine the step α so that the variable never get negative values. In particular, after computing the deepest descend step α, we can ensure that $W - \alpha D$ by the update

$$\alpha = \min\{\alpha, \min_{i,j:D_{ij}\neq 0}\{W(i,j)/D(i,j)\}\}.$$

If we add this line to the PPG algorithm, we obtain the Strictly Positive Gradient (SPG).

We notice that when the minimum of the problem lies on the interior of the positive orthant, then the algorithms above mentioned are usually equivalent, whereas when the solution is located on the border, then the DPG algorithm has some problem, as can be seen in Figure 2, and if the solution has a lot of zero entries, then the SPG becomes slow as well, since it has to perform at least one step for each zero.

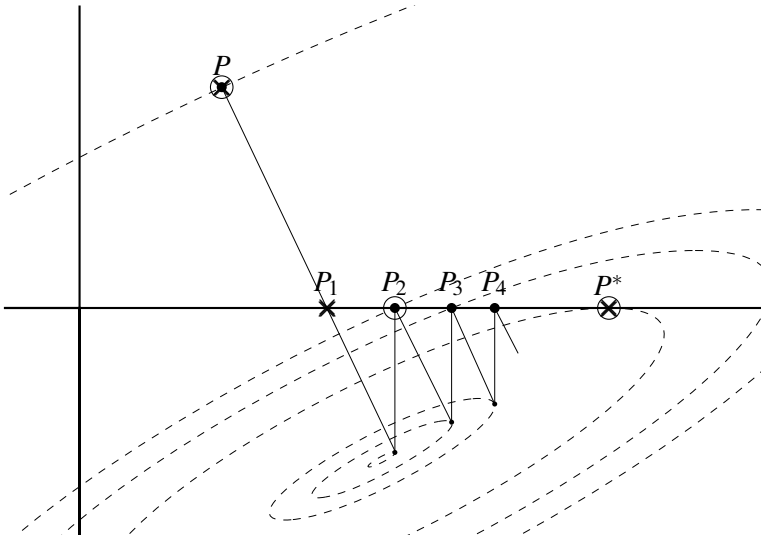

Figure 2: The methods DPG, PPG, and SPG compared on a bidimensional plane. The dashed ellipsoids are the contour lines of a function we want to minimize on the positive quadrant. If we start from the point P, both the algorithms PPG, marked with circles, and SPG, marked with crosses, converge to the optimum in 2 steps ($P \to P_2 \to P^*$ and $P \to P_1 \to P^*$), whereas the DPG method, whose iterations are drawn with dots and straight lines, needs a lot more steps to get near the optimum.

MU Like in the original NMF, the Multiplicative Update can be applied even in this case, substituting the normal matrix multiplication with the diamond operator, and applying few other adjustments.

$$W = W .* (A \diamond H') ./ (W \diamond H^T \diamond H').$$

We'll prove in Appendix C that

Theorem 2.2.1. *The MU update makes the error of PermNMF drop, as long as* $W \diamond H^T \diamond H'$ *has all elements positive.*

In the case that the condition isn't satisfied, the proof of the theorem suggests that the addition a small constant ε to each zero entry of the numerator doesn't change much the output, so the update is well defined in any case. The computational time in this case is still $O(nmk)$, so it is comparable with the PG update.

Let's now focus on the update of H, that requires to solve an optimization problem on the group $\mathbb{R} \times T_n$.

2.2.2 Update of H

Like in the W update, the QR decomposition of the ALS algorithm isn't useful, since the diamond operator behaves poorly with triangular matrices, and it lacks the associativity property $(QR) \diamond H^T \neq Q(R \diamond H^T)$. Even here, the HALS algorithm loses its efficiency, and, on the contrary of the W update, PG and MU methods aren't useful either, since the expression lacks a derivation for the entries in H, and in particular both algorithms don't know how to update the shifts.

We then devise an other method, that follows the idea of the CD to update one variable at a time. We need first to solve an analogous of the NNLS problem, called Single Permutation NNLS, that gives us an update rule for single variables in the space $\mathbb{R}_+ \times T_n$.

Single Permutation NNLS Let's suppose to have two vectors v, w in $\mathbb{R}^n$, and we want to find the best element $\tau = [r, p]$ of $\mathbb{R}_+ \times T_n$ that minimizes

$$E(\tau) = \|v - \tau(w)\|^2,$$

where the norm used is the euclidean one.

A natural assumption is that $w \neq 0$, otherwise every element τ gives the same value of $E(\tau) = \|v\|^2$. If we knew the optimal p, then we could find r without fail, because it becomes a simple Nonnegative Least Squares (NNLS) problem.

$$r_p := \operatorname*{arg\,min}_{r \in \mathbb{R}} \|v - r\sigma_p(w)\|^2 = {v^T \sigma_p(w)} \big/ {\sigma_p(w)^T \sigma_p(w)},$$

$$r_p^+ := \operatorname*{arg\,min}_{r \in \mathbb{R}^+} \|v - r\sigma_p(w)\|^2 = \begin{cases} 0 & v^T \sigma_p(w) < 0, \\ {v^T \sigma_p(w)} \big/ {\sigma_p(w)^T \sigma_p(w)} & v^T \sigma_p(w) \geq 0. \end{cases}$$

A simple solution consists into computing the optimal r_p^+ for every $\sigma_p \in T_n$, and check which couple $[r_p^+, p]$ gives us the minimal error. We know that $\sigma_p(w)^T \sigma_p(w) = \|w\|^2$, so we can compute the error as a function of p

$$\|v - r_p \sigma_p(w)\|^2 = \|v\|^2 - \frac{(v^T \sigma_p(w))^2}{\|w\|^2}.$$

The problem is thus equivalent to maximize $(v^T \sigma_p(w))^2$, but we're interested only in the positive case, so we focus on maximizing the scalar product $v^T \sigma_p(w)$, since if

$v^T \sigma_p(w) < 0$ then $r_p^+ = 0$ for every p, so $E([r_p^+, p]) = E([0,p]) = \|v\|^2$.

By definition, $\sigma_p(w)$ is the vector w shifted, so we can call $Circ(w)$ the real nonnegative matrix that has all the shifted versions of w as columns, and compute the maximal component of $v^T Circ(w)$. Since $Circ(w)$ is a circulant matrix, this operation costs $O(n \log n)$ if performed with Fast Fourier Transformations, so this method is fast and gives us the correct solution.

Single Permutation NNLS
Inputs : $v, w \in \mathbb{R}^n, \quad w \neq 0$

$p = 1 - \arg\max_i (v^T Circ(w))_i$
if $v^T \sigma_p(w) > 0$ **then**
  $r = v^T \sigma_p(w) / \|w\|^2$
else
  $r = 0$
end if
return $[r, p]$

From now on, we'll use this algorithm with the syntax

$$\tau = \text{SinglePermNNLS}(v, w).$$

This method will be used in the setting we're going to explain in the next paragraphs.

CD Given now a vector $v \in \mathbb{R}^n$, and a bunch of vectors $w_1, w_2, \ldots, w_k \in \mathbb{R}^n$ we can now try to find the best elements $\tau_1, \ldots, \tau_k \in (\mathbb{R}_+ \times T_n)$ that minimize the quantity

$$\|v - (\tau_1(w_1) + \tau_2(w_2) + \cdots + \tau_k(w_k))\|.$$

We're thus looking for the best linear combination with positive coefficients of the shifted vectors w_i that gives us the original vector v. If we call W the matrix with w_i as columns, and x the (column) vector of τ_i, then we can rewrite the problem in a compact way as

$$\min_{x \in (\mathbb{R} \times T_n)^k} \|v - W \diamond x\|^2, \qquad v \in \mathbb{R}_+^n \quad W \in \mathbb{R}_+^{n \times k}.$$

A way to solve this problem is using the precedent algorithm in an alternated fashion. In fact, if we fix $\tau_2, \tau_3, \ldots, \tau_k$, then it becomes a Singular Permutation NNLS problem on τ_1, and we know how to solve it exactly.

So we can solve the problem sequentially for each τ_i and repeat. The initial value of x is usually given as an input parameter, but it can also be generated casually at the beginning of the algorithm.

Multiple Permutations NNLS
Inputs : $v \in \mathbb{R}^n, \quad W \in \mathbb{R}^{n \times k}, \quad x \in (\mathbb{R}^+ \times T_n)^k$

$w = W \diamond x$
for $i = 1 : k$ **do**
  $w = w - x_i(W_{:,i})$
  $x_i = SinglePermNNLS(v - w, W_{:,i})$;
  $w = w + x_i(W_{:,i})$
end for
return x

The internal loop is usually repeated a given number *maxiter* of iterations, before returning the output x. From now on, we'll use this algorithm with the syntax

$$x = \text{MultPermNNLS}(v, W, x).$$

Its computational cost is the number of iterations multiplied k times the cost of the Single Permutation Problem, so it is $O(kn\log(n))$. In particular cases, it may be useful to randomize the choice of the index i, since it's important not to impose a preference order on the components in W.

This algorithm has an immediate application to the original problem

$$H = \underset{X \in (\mathbb{R}_+ \times T_n)^{m \times k}}{\arg\min} \|A - W \diamond X^T\|_F^2.$$

Like the normal NMF, it can be decomposed into smaller problems

$$\|A - W \diamond X^T\|_F^2 = \sum_{i=1}^{m} \|A_{:,i} - W \diamond (X^T)_{:,i}\|^2,$$

$$H_{i,:} = \underset{x \in (\mathbb{R}_+ \times T_n)^k}{\arg\min} \|A_{:,i} - W \diamond x\|^2,$$

that can be solved with the Multiple Permutation NNLS algorithm. If we put everything together, we obtain the final method

CD Update Method
Inputs : $A \in \mathbb{R}_+^{n \times m}$, $W \in \mathbb{R}_+^{n \times k}$, $H \in (\mathbb{R}_+ \times T_n)^{m \times k}$

for $i = 1 : m$ **do**
  $H_{i,:} = MultPermNNLS(A_{:,i}, W, (H_{i,:})^T)$
end for

Every step of this Update Method costs $O(kmn\log(n))$ if we consider the number of iterations in the internal methods as constants. We will stop the updates when the convergence is too slow, when we loop on the same matrices, or when we reach a number of iterations too high.

2.2.3 Extension to Multiple Images

Given a set a pictures, now we're able to perform a PermNMF and obtain a set of k common features that can reconstruct the original data once combined through coefficients and permutations codified in H. Given one of the images in W, the algorithm tells us if it is present in the original images, but it doesn't detect if it appears multiple times. One example of such instance may be a set of radar images, in which different objects intercepted by the wave signals have distinct shapes, but each one can appear multiple time in the same picture.

One possible solution is to perform an initial PermNMF with a parameter k proportional to the effective number of distinct objects with multiplicity that can appear on a single image, discard the found components with low coefficients, and repeat the PermNMF on the output components with a low k corresponding to the number of distinct shapes without multiplicity. Let's call K the first larger parameter, and $A \in \mathbb{R}_+^{n\times m}$ the set of pictures to analyze. We obtain

$$A \sim \widetilde{W} \diamond H_1^T \sim (W \diamond H_2^T) \diamond H_1^T = W \diamond (H_2^T \diamond H_1^T),$$

where $\widetilde{W} \in \mathbb{R}_+^{n\times K}$, $W \in \mathbb{R}_+^{n\times k}$ and $H_1 \in (\mathbb{R}_+ \times T_n)^{m\times K}$, $H_2 \in (\mathbb{R}_+ \times T_n)^{K\times k}$, so the final decomposition will be again a real matrix with k components, and a matrix $H_2^T \diamond H_1^T \in (\mathbb{R}_+ T_n)^{k\times m}$. This last matrix is able to tell, for each component, even if there are multiple instances in every original image.

The computational cost of such method (for each cycle, till convergence) is

$$O(nmK\log(n) + nKk\log(n)) = O(nK\log(n)(m+k)),$$

that, under the assumption $k << m$, is equivalent to $O(nmK\log(n))$, meaning that the second step has a negligible computational cost compared to the first. If K is still on the order of magnitude of k, the asymptotic cost doesn't change, but if that's not the case, it is better to look for other ways.

On this topic, Potluru, Plis and Calhourn in [30] offer an algorithm that uses Fast Fourier Transformations and circulant matrices in order to compute and codify permutations of the components, called ssiNMF (sparse-shift invariant NMF). As in the PermNMF, the basic idea is to find k components and a set of permutations that could reconstruct the original images, but the ssiNMF sets as target the permutations in the group $\mathbb{R}_+ T_n$, corresponding through φ with all the circulant nonnegative matrices, so that all the operations can be performed through FFTs. Thanks to this, their algorithm is able to directly construct an approximation

$$A \sim W \diamond H^T, \qquad W \in \mathbb{R}_+^{n\times k}, \quad H \in (\mathbb{R}_+ T_n)^{m\times k}.$$

Eggert, Wersing and Korner in [11] took a more general approach to the problem: as we set a subgroup of S_n, they chose a general set of transformations of the plane, seen as operators on the columns of W, and multiplied the number of parameter of H by the cardinality of the chosen set, so that for each transformation of the components there would be coefficients in H stating their intensity in the original images.

Both the approaches suffer by the presence of the trivial and exact solution described in section 2.3: a single pixel can generate any image if we allow too many transformations of the space. They propose to perform a common modification on the NMF framework, that is adding a penalty factor to ensure the sparseness of the output, since the

presence of a single pixel in the component output corresponds to a lot of positive coefficients in H, and it leads to the presence of an additional parameter λ to set manually or through validations techniques.

An other common characteristic of both the algorithms is the rise in memory used and asymptotic computational cost by at least a factor on par with the number of pixels on a single image, leading to a cost by iteration at least of $O(n^2mk)$. When compared with the PermNMF algorithm, we see that they're comparable when $K \sim nk/\log(n)$, meaning that a component have to appear in the original image on average $n/\log(n)$ times.

2.3 Experiments

First, let's give some immediate visual representation of the output of such algorithms. In these experiments, we use the PermNMF algorithms seen in the previous chapter, with the different updates for W. We'll use as initial parameters W and H some randomly generated matrices, and the *maxiter* variable set to 10 in both the MultPermNNLS and the PG methods.

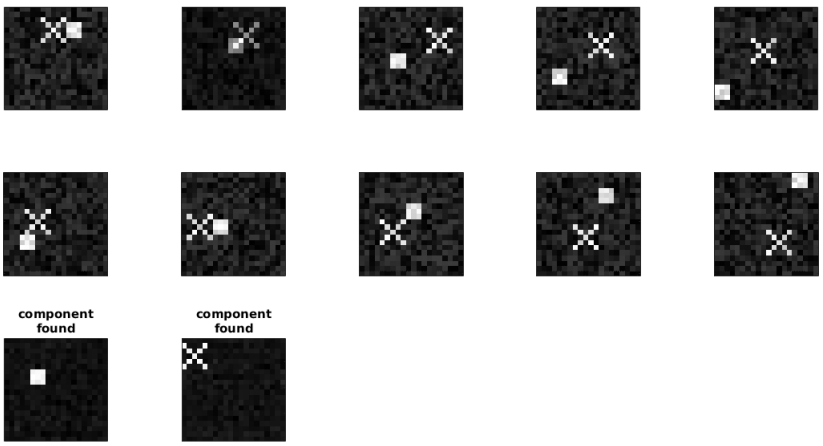

Figure 3: On the first 2 rows, there are the original 10 images, that are the columns of A. The other 2 rows are the components found as columns of W.

In the first set of experiments (Figure 3) we use 2 simple shapes (a square and a cross) of 9 pixels that move into a frame of dimensions 20×20, and add a casual error. We'll compare the PPG, DPG, SPG and MU updates for W, specifying for each one the running time and showing the graph of the error at each iteration of the algorithm.

All the methods usually manage to find the right components after less than 10 repetitions on average, but it may happen that the convergence leads to bad local minima, so for each algorithm we specify also the rate of success, computed as the percentage of different random initial points that leads to a solution with error less or equal to the noise introduced in the original images in 20 steps. The images shown on the bottom row are the column of W, and they're distinguishable as a cross and a square, with little noise given by the imperfections on the original images.

μ	Success Rate / Time			
	DPG	PPG	SPG	MU
0	91/1.756	88/1.693	89/2.690	0
0.01	88/1.748	88/1.713	94/2.609	94/1.678
0.1	89/1.750	90/1.734	86/2.133	88/1.609
0.3	1.857	1.827	1.961	1.616
0.5	1.901	1.901	2.026	1.636
0.7	2.061	2.054	2.122	1.623
1	2.034	2.023	2.0093	1.5815
random	1.965	1.909	1.862	1.571

Figure 4: All these values refer to the experiments with the images in Figure 3, with the exception for the last row, that refers to 10 randomly generated images of sizes 20×20. The intensity of the noise introduced is indicated by μ, and each row contains success rate (if it is below 100%) and computational time in seconds per iteration.

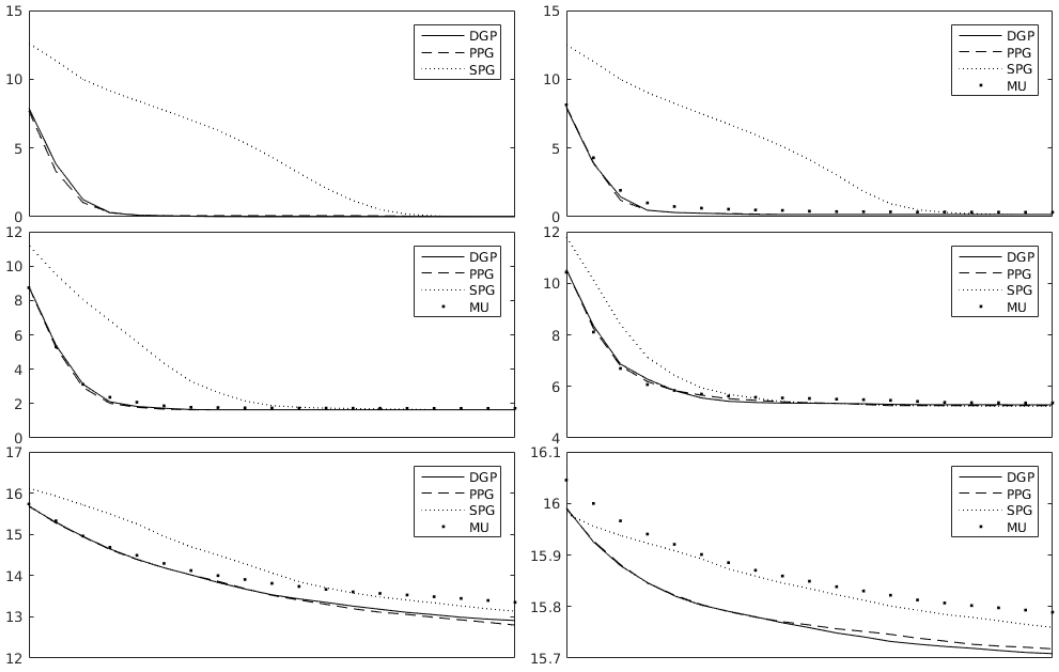

Figure 5: In order, from left to right and from top to bottom there are the graphs of the errors for the 4 methods with $\mu = 0, 0.01, 0.1, 0.3, 0.7$ and the random case.

After having generated the original images with squares and crosses whose pixels have intensity 1 (2 if they intersect), we introduce a random noise, where each pixel has intensity between 0 and 1, and multiply it by a parameter μ. In Figure 4, we can see how higher μ correspond with higher computational time, since the internal PG and MultPermNNLS methods reach more frequently the *maxiter* number of iterations.

From $\mu = 0.3$ the Success Rate is always 100%, meaning that the methods usually find better solutions in term of error than the one initially generated. In general the success rate is high enough, with the only exception of the MU method when there's not noise, since it always converge to not optimal local minima. When we add even a little noise of 0.01, though, the MU update is successful, and we can notice from Figure 5 that it is competitive with the other methods.

As we can see from the error graphs, the DGP and PPG algorithm are practically indistinguishable in all cases, but the computational time estimated suggest that PPG is slightly better. The SPG fail both in computational time and in reduction of error, whereas the MU behaves well enough with little noise, but in general tend to produce poor solutions, as we can see from the random case and in the noiseless case.

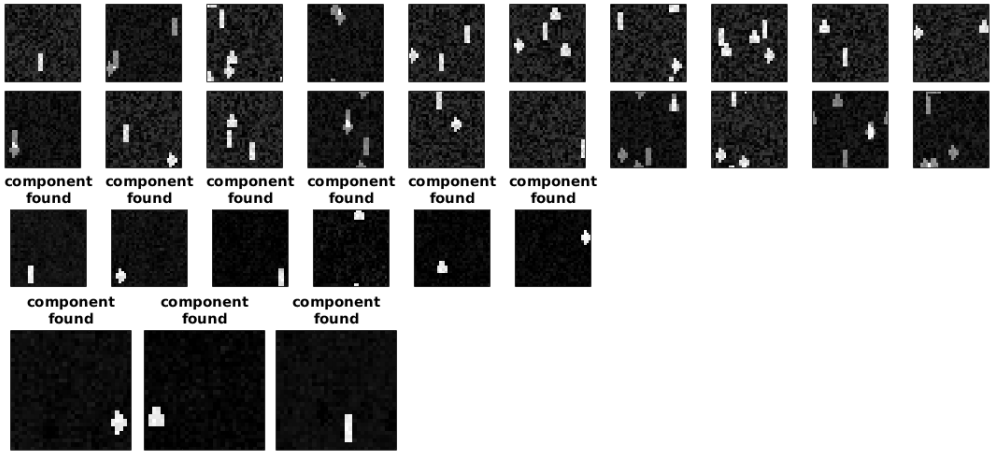

Figure 6: On the first 2 rows, there are the original 20 images, composed by three base pictures translated and superimposed. on the third row there are the components found by the first PermNMF, and in the last row there is the final output of the second PermNMF, that coincide with the base pictures.

In the second experiment, we generate 20 images of shape 30x30 from three simple figures (a plane, a tank and a ship), with a noise of mean 0.15. Each image can include up to two copies of the same figure, so we need to perform a first PermNMF with $k = 6$, and then a second time with $k = 3$ to extract the original ones. The first application of the algorithm is slowed down by the presence of the same shapes multiple times in the images, but the second application is really fast. As said, we managed to extract first the common features with their multiplicity, and then the actual features. Multiplying the two H matrices we obtained in the two steps of the algorithm, we can deduce the actual position with multiplicity of the shape found in all the 20 original images.

Chapter 3

Future Works

3.1 Further Research on PermNMF

The PermNMF has not been throughly studied and analyzed. First of all, it lacks a convergence result, both because the usual arguments used for the classical ANLS algorithms vastly use the fact that the two subproblems in the NMF are convex, and because we switched the framework to non-continuous spaces such as $\mathbb{R}_+ \times {}^{\mathbb{Z}}\!/_{n\mathbb{Z}}$, where it is still not even well defined a canonical concept of "local minimum" (the usual topological embedding of this space in $\mathbb{R}^3$ gives a notion of stationary points that doesn't cope well with the nature of permutations).

One suggestion is to define, for each one of the n^{km} possible combinations Ω of shifts in the elements of H, an error function $F^{(\Omega)}(W,H) = \|A - W \diamond (H^{(\Omega)})^T\|$, where H has Ω as shifts and can vary only in its real variables, and then use as global error the minimum $F(W,H) = \min_\Omega F^{(\Omega)}(W,H)$. This error function is continuous, but requires an efficient algorithm that, given W and the real constants in H, can find the best shifts among the possible n^{km} ones.

On the point of view of the PermNMF problem, there's a lot to say, for example, on whether there exists an exact algorithm, or if there are bounds on the minimum k, or even if the solution is unique (up to trivial transformations). In [14], Gillis find a preprocessing for the input data A that gives a more well-posed problem then the normal NMF, so such a transformation could be beneficial even to the PermNMF. In [1], the authors found precise conditions for A under which there exists a polynomial time algorithm for the exact NMF problem, and stated that in general the approximation problem is NP-hard, so it's highly possible that even the PermNMF problem is a NP-hard problem, and that a the polynomial time algorithm could be adapted for this case.

On the side of the algorithm itself, an efficient adaptation for HALS or CD method for the update of W has still to be found, altogether with an exact algorithm that would follow the ideas of the Active Set Method. A feature we'd like to obtain is also the sparseness of the solutions, so it has to be searched if it's possible to adapt a Constrained NMF for this case. The H update, moreover, is naturally inclined to a parallel computation, and like in the classical CD algorithm, it's also possible to devise a method to choose preemptively which element to update in every cycle, in order to make the error drop faster.

Eventually, we studied the problem when the elements of H are restricted to $\mathbb{R}_+ \times T_n$, but it's possible also to consider other subgroups and subalgebras of $\mathbb{R}S_n$ in order to encode different transformations of the plan, or just to make the NMF invariant with respect to particular linear operators. For example, the horizontal symmetry of the images is easy to encode as an element of S_n, and added to T_n produces the dihedral group D_n.

Sparsity Conditions for Extensions As already discussed in the last chapter, the algorithms devised for PermNMF work with the group $\mathbb{R}^+ \times T_n$, so they are not able to detect if a component in W appears more than once in a single original image in A. If we extend the group to $\mathbb{R}^+ T_n$, then we can differentiate the error function even on H, since its entries can now vary in the whole set of circulant matrices.

If we associate to each element H_{ij} the first row of its correspondent circulant matrix, then some computations show that

$$DH_{ij} := \frac{\partial}{\partial H_{ij}} F(W,H) = -2 * circ((A - W \diamond H^T)_{:,i}) W_{:,j},$$

where $circ(v)$ stands for the circulant matrix with first row the vector v. We can now perform a PG method even for H, like

$$H \leftarrow H - \alpha * D,$$

where D and α are chosen in the same way as the PPG method: D is the opposite of the above-computed gradient DH where we set to zero its entries if they lead to negative values for H; the parameter α is the deepest step for D, computed as

$$\alpha = -\frac{sum((A - W \diamond H^T) .* (W \diamond D^T))}{W \diamond D^T}.$$

The update of W is the same, since the computations for the gradient made in the Appendix B hold for the general algebra $\mathbb{R}S_n$. The main difference with the previous case is that now the diamond operator requires multiplications between circulant matrices and vectors, that are performed usually in $O(nmk \log(n))$. The good news is that the diamond operator is now the most expensive operation performed along all the algorithm, so that the final asymptotic computational cost is again $O(nmk \log(n))$, like the previous algorithm.

As already anticipated, this algorithm has a big flaw: it always converges to the trivial solution previously discussed, that is the one-pixel image. Moreover, there are a lot of other exact solutions, usually containing few pixels, and as proposed in the other papers, a way to deal with this issue is to adopt a CNMF setting, adding a sparsity constraint

$$\min_{W,H} \|A - W \diamond H^T\|_F^2 + \lambda \|H\|_*^2,$$

where λ has to be set manually, and $\|\cdot\|_*$ is an appropriate norm. The norm 1 is the most used, since it is the convex envelope of the "0-norm"[1] on the norm-1 unitary ball, but it can be substituted with the Frobenius norm for simple computations.

Even this algorithm is in testing phase, since it's hard to choose the proper parameters, being dependent on the specific original data in A.

[1] Defined as the cardinality of the non zero entries in a vector

3.2 Real Shifts

In the previous section, we've seen an extension of PermMNF that let us encode the possibility of multiple images in the same original picture, and we tried to approach it with a PG method. the derivation became possible thanks to the particular structure of $\mathbb{R}T_n$, but a similar operation is still not defined for the H update when using elements in the group $\mathbb{R}_+ \times T_n$.

As already stated, we can see the elements of H as a couple of numbers $[r,s]$, with r real and s natural less or equal to n. We can extend the second component to all the integer set $\mathbb{Z}$ simply adding a periodic condition, that is

$$(r,s) = (r,n+s) \qquad \forall\, r \in \mathbb{R},\ s \in \mathbb{Z}.$$

Since we want to do actual derivations on the second component, we furthermore to extend it to $\mathbb{R}$. Given a vector $v \in \mathbb{R}^n$ and an interpolation function $f : \mathbb{R}^n \to \mathbb{R}$ such that it is periodic of period n and $f(i) = v_i$ for all the components i, we can define a real shift through f as

Definition 3.2.1. The *real shift* of a vector in $\mathbb{R}^n$ by a real number t through a function f described above, is

$$\sigma_t(v)_i = f(i+t) \quad \forall\, i.$$

In particular, if t is a natural number, then the real shift σ_t coincides with the old shift of t places. It follows that the regularity of the real shifts as operators with respect to t is the same regularity of f, so we look for interpolating functions at least C^2, that is, with the second derivative continuous.

One possible solution is to use a Cyclic Spline interpolation, meaning that each segment $f(x)$ in the range $x \in [m, m+1]$ with m integer, is actually a cubic polynomial function. Spline functions are really useful in this case, since it has a nice control over its curvature (the natural splines have a Smoothness Theorem that assures the minimality of the function mean curvature), and the coefficients of the cubic polynomial are obtained through circulant systems from the entries of v, so that their computation is fast through FFT.

It's easy now to compute the gradient of the error function, in both the constant and shift variable of each entry in H, and thus a PG method is now applicable, with the same complexity of the CD algorithm we have already seen, that is $O(nmk\log(n))$. An other good news is that each entry in H actually varies in the space $\mathbb{R}_+ \times \mathbb{R}/n\mathbb{Z}$, where the elements with zero constant are all identified. The described space is homeomorphic equivalent to $\mathbb{R}^2$ through classical polar operations

$$[r,t] \to (r\cos(2\pi t/n), r\sin(2\pi t/n)).$$

We can thus apply the PG method directly to the image space $\mathbb{R}^2$, and this would output a matrix H with elements still in $\mathbb{R}_+ \times \mathbb{R}/n\mathbb{Z}$, with no need to take the positive part, opposed to the classical PG.

In the case of images, the cyclic splines interpolate the pixel to create a continuous picture. An inconvenient is that the resulting interpolating formula may have negative values, but we round them off to zero, and it doesn't change much the overall function, since the spline tends to not generate such elements, opposed to a parabolic interpolation

that can oscillate too much between positive and negative values, and linear interpolation, whose interpolating function is nonnegative in every point, but it is not derivable.

An other problem is the loss of focus in the image through interpolation. For example, a linear interpolation is proved to always mix the pixels in the image in order to make them more homogeneous, like in the case of a chessboard with alternating pixels of magnitude 1 and 0, that after a real shift of 0.5, becomes totally monochromatic. The splines seem to solve even this problems, as experimental tests shows that even after a thousand of real shift with cubic interpolation, the images don't lose much of their clearness.

A feature of all the algorithms that works on T_n is that they don't recognize adjacent pixels as actually near to each other, but totally independent entities. Any interpolation mixes together the pixels, but the vectorization of the images doesn't let the shifts to mix vertically near pixels, so even here the whole picture is dismembered. A way to contrast this phenomenon is to perform a double cyclic spline interpolation on the images, that means to interpolate periodically on $\mathbb{R}^2$ the values of the pixels as if they were on the lattice of natural points (i, j) with $1 \leq i \leq n$ and $1 \leq j \leq m$. These splines have still good concavity properties, coincide on unitary squares with double cubic polynomials, and the coefficients can still be calculated easily through circulant systems, so that the final cost remains the same. The downside is that we lose the polar representation of the shifts, and we have to work with double shifts for any entry in H, that is, we have to work in $\mathbb{R}_+ \times T_r \times T_s$ if the original sizes of the image were (r, s).

The method has not been used in this document, since it still needs proper tuning and studies.

Appendices

Appendix A

Properties of the Diamond Operator

The Diamond Operator is defined so that it could maintain particular properties.

Lemma A.1. *Given N, M, P matrices with elements in the algebra $\mathbb{R}S_n$, and A a real matrix, then the following associativity rules hold.*

1. $(A \diamond M) \diamond N = A \diamond (M \diamond N)$.
2. $(M \diamond N) \diamond P = M \diamond (N \diamond P)$.

Proof.

1.

$$\begin{aligned}[(A \diamond M) \diamond N]_{:,i} &= \sum_k N_{ki} (A \diamond M)_{:,k} \\ &= \sum_{k,s} N_{ki} \cdot M_{sk} (A_{:,s}) \\ &= \sum_s (M \diamond N)_{si} (A_{:,s}) = [A \diamond (M \diamond N)]_{:,i}.\end{aligned}$$

2.

$$\begin{aligned}[(M \diamond N) \diamond P]_{:,i} &= \sum_k P_{ki} \cdot (M \diamond N)_{:,k} \\ &= \sum_{k,s} P_{ki} \cdot N_{sk} \cdot M_{:,s} \\ &= \sum_s (N \diamond P)_{si} \cdot M_{:,s} = [M \diamond (N \diamond P)]_{:,i}.\end{aligned}$$

□

One important remark, is that if $A \in \mathbb{R}^{n\times m}$, $B \in \mathbb{R}^{m\times k}$, and M has elements in $\mathbb{R}S_n$, then $(AB)\diamond M$ is computable, but $A(B\diamond M)$ isn't defined when $n \neq m$. However, even when A is a square matrix, and we restrict to $\mathbb{R}\times T_n$ we have almost never the associativity property:

Lemma A.2. *Given $A \in \mathbb{R}^{n\times n}$ real matrix, then*

$$(AB)\diamond M = A(B\diamond M) \quad \forall\, B \in \mathbb{R}^{n\times m},\, M \in (\mathbb{R}\times T_n)^{m\times k}$$

if and only if A is a circulant matrix.

Proof. We'll treat the elements of M, and in general any element of $\mathbb{R}\times T_n$, as circulant matrices.

$$((AB)\diamond M)_{:,i} = \sum_k M_{ki} AB_{:,k}, \qquad (A(B\diamond M))_{:,i} = \sum_k AM_{ki} B_{:,k},$$

and we want them to be equal for all M and B, so in particular we have that given any $D \in \mathbb{R}\times T_n$ and any $v \in \mathbb{R}^n$ we have $DAv = ADv$, or equivalently $AD = DA$. D is a power of the circulant matrix C with only 1 in the upper diagonal multiplied by a scalar, so

$$((AB)\diamond M)_{:,i} = (A(B\diamond M))_{:,i} \quad \forall\, B,M,i \iff AC^k = C^k A \quad \forall\, k \iff AC = CA.$$

Since C is an orthogonal matrix, we have

$$A_{ij} = (CAC^T)_{ij} = A_{i+1,j+1},$$

so A is constant on the diagonals, meaning it is circulant. □

In the next properties, we'll use the operator on $\mathbb{R}S_n$ valued matrices defined in Appendix B that transposes each entry, and we'll identify the entries with real matrices, so that this operator coincides with the matrix transposition.

Lemma A.3. *Given N, M matrices with elements in the algebra $\mathbb{R}S_n$, then*

$$(N'\diamond M')^T = (M^T \diamond N^T)'.$$

Proof.

$$\begin{aligned}
(N'\diamond M')^T_{ij} &= (N'\diamond M')_{ji} \\
&= \sum_k M'_{ki}\cdot N'_{jk} \\
&= \sum_k (N_{jk}\cdot M_{ki})' \\
&= \left(\sum_k N^T_{kj}\cdot M^T_{ik}\right)' = (M^T\diamond N^T)'_{ij}.
\end{aligned}$$

□

Eventually, we report a result we'll use to simplify the computations in the algorithms. As before, we consider the elements of $\mathbb{R}S_n$ as matrices, and given $h \in \mathbb{R}S_n$ we'll use h_{ij} to denote $\varphi(h)_{ij}$.

Lemma A.4. *Given A, B real matrices, and M a matrix with elements in the algebra $\mathbb{R}S_n$, then the sum of elements of $A .* (B \diamond M^T)$ and of $B .* (A \diamond M')$ are equal.*

Proof.

$$\begin{aligned}
\sum_{i,j}(A .* (B \diamond M^T))_{ij} &= \sum_{i,j} A_{ij}(B \diamond M^T)_{ij} \\
&= \sum_{i,j,k} A_{ij}[M_{jk}(B_{:,k})]_i \\
&= \sum_{i,j,k,s} A_{ij}(M_{jk})_{is}B_{sk} \\
&= \sum_{j,k,s} [(M_{jk})'(A_{:,j})]_s B_{sk} \\
&= \sum_{k,s}(A \diamond M')_{sk}B_{sk} = \sum_{s,k}(B .* (A \diamond M'))_{sk}.
\end{aligned}$$

□

Appendix B

Computation of W Gradient and Hessian

B.1 Gradient of the error

Let's derive the gradient of the error function with respect to W, when H has entries in $\mathbb{R}S_n$, so that it still holds when we restrict to $\mathbb{R}_+ \times T_n$.

In the following steps, we consider the general element of H as a (circulant) matrix, using implicitly the homomorphism φ, so we use the notation

$$(H_{ij})_{uv} := \varphi(H_{ij})_{uv},$$

and we can write explicitly the error function as

$$\begin{aligned} F(W) =& \|A - W \diamond H^T\|_F^2 \\ =& \sum_{i,j} (A - W \diamond H^T)_{ij}^2 \\ =& \sum_{i,j} \left[A_{ij} - \left(\sum_s H_{js}(W_{:,s}) \right)_i \right]^2 \\ =& \sum_{i,j} \left[A_{ij} - \sum_{k,s} (H_{js})_{ik} W_{ks} \right]^2. \end{aligned}$$

Deriving with respect to the entry (u,v) of the variables matrix, we obtain

$$\begin{aligned}\frac{\partial}{\partial X_{uv}}F(X) =& \frac{\partial}{\partial X_{uv}}\sum_{i,j}(A - X\diamond H^T)_{ij}^2 \\ =& 2\sum_{i,j}(A - X\diamond H^T)_{ij}\frac{\partial}{\partial X_{uv}}(A - X\diamond H^T)_{ij} \\ =& 2\sum_{i,j}(A - X\diamond H^T)_{ij}\frac{\partial}{\partial X_{uv}}\left(A_{ij} - \sum_{k,s}(H_{js})_{ik}X_{ks}\right) \\ =& -2\sum_{i,j}(A - W\diamond H^T)_{ij}\frac{\partial}{\partial X_{uv}}(H_{jv})_{iu}X_{uv} \\ =& -2\sum_{i,j}(H_{jv})_{iu}(A - X\diamond H^T)_{ij}.\end{aligned}$$

Let's call H' the matrix with the same dimension of H and $H'_{ij} = (H_{ij})'$ for each entry. We've already showed that the transposition of elements in $\mathbb{R}S_n$ translate into the transposition of the image matrices, so given any $h \in \mathbb{R}S_n$, with our notation, we have

$$h_{uv} = h'_{vu},$$

and given $h = [r,t]$ an element of the group $R_+ \times T_n$, it's easy to see that the transpose h' is the element $[r, n-t]$.

The computation continues as

$$\begin{aligned}& -2\sum_{i,j}(H_{jv})_{iu}(A - X\diamond H^T)_{ij} \\ =& -2\sum_{i,j}(H'_{jv})_{ui}(A - X\diamond H^T)_{ij} \\ =& -2\sum_{j}(H'_{jv}(A - X\diamond H^T)_j)_u \\ =& -2((A - X\diamond H^T)\diamond H')_{uv}.\end{aligned}$$

So we can write in a compact form the gradient

$$\nabla_X F(W) = -2(A - W\diamond H^T)\diamond H'.$$

B.2 Hessian of the error

The derivate of the gradient function is a square matrix with size $nk \times nk$, so each coordinate has two indexes. We'll write

$$H_X F_{ij,uv}(X) = \frac{\partial^2}{\partial X_{ij}\partial X_{uv}}F(X)$$

to denote the element in position ij,kr of the Hessian matrix. We'll use the gradient computed in the section above.

$$\begin{aligned}
\frac{\partial^2}{\partial X_{kr}\partial X_{uv}}F(X) =& \frac{\partial}{\partial X_{kr}}\left(\frac{\partial}{\partial X_{uv}}F(X)\right) \\
=& -2\frac{\partial}{\partial X_{kr}}\sum_{i,j}(H'_{jv})_{ui}(A - X\diamond H^T)_{ij} \\
=& -2\sum_{i,j}(H'_{jv})_{ui}\frac{\partial}{\partial X_{kr}}(A - X\diamond H^T)_{ij} \\
=& 2\sum_{i,j}(H'_{jv})_{ui}(H_{jr})_{ik} \\
=& 2\sum_{j}(H'_{jv}\cdot H_{jr})_{uk} = 2((H^T\diamond H')_{rv})_{uk}.
\end{aligned}$$

In the last row, we used the multiplication of elements in $\mathbb{R}S_n$, that is equivalent to the multiplication of the corresponding matrix through φ.

Appendix C

Proof of the Descent of MU

Let's prove that the MU update for W makes the error drop, mimicking the proof given in [26] for the classical NMF. With the same notation of the last appendix, the update is

$$W \leftarrow W .* (A \diamond H') ./ (W \diamond H^T \diamond H'),$$

and the error function is still

$$F(X) = \|A - X \diamond H^T\|_F^2.$$

The matrix H in the PermNMF problem is composed by elements in $\mathbb{R}_+ \times T_n$, but the computation for the gradient and the Hessian done in Appendix B still holds for elements in $\mathbb{R}S_n$, so we can suppose that the entries of H are elements of $\mathbb{R}_+S_n$, that are $\alpha = \sum_{\sigma \in S_n} [r_\sigma, \sigma] \in \mathbb{R}S_n$ with $r_\sigma \geq 0$.

The expression is quadratic in the entries of X, so it is equivalent to its second order expansion. In the equation, we'll suppose that the matrices X, W and $\nabla_X F$ are vectorized so that the dot product makes sense.

$$F(X) = F(W) + (X - W)^T \nabla F(W) + \frac{1}{2}(X - W)^T HF(W)(X - W).$$

Let's also define a similar auxiliary function

$$G(X,Y) = F(Y) + (X - Y)^T \nabla F(Y) + \frac{1}{2}(X - Y)^T K(Y)(X - Y),$$

where $K(Y)$ is a diagonal matrix, whose (ij,ij) element is

$$K(Y)_{ij,ij} = 2(Y \diamond H^T \diamond H')_{ij}/Y_{ij},$$

and Y has positive entries.

Lemma C.1. *For every real matrix X and for every real positive matrix Y, we have $G(X,Y) \geq F(X)$ and $G(X,X) = F(X)$.*

Proof. It's obvious that $G(X,X) = F(X)$. For the other point, we use the second order expansion of $F(X)$ with center Y, and notice that the first two orders coincide, so that there remains only

$$G(X,Y) - F(X) = \frac{1}{2}(X - Y)^T (K(Y) - HF(Y))(X - Y).$$

We want to prove that this quantity is nonnegative for every X, and it is equivalent to prove that $K(Y) - HF(Y)$ is positive semidefinite for every positive matrix Y. Given a generic vector Z, we can compute

$$\begin{aligned} Z^T(K(Y) - H_X F(Y))Z &= \sum_{ij,kr} Z_{ij} Z_{kr} (K(Y) - HF(Y))_{ij,kr} \\ &= \sum_{ij} 2Z_{ij}^2 \frac{(Y \diamond H^T \diamond H')_{ij}}{Y_{ij}} - \sum_{ij,kr} Z_{ij} Z_{kr} HF(Y)_{ij,kr} \\ &= \sum_{ij,kr} 2Z_{ij}^2 \frac{((H^T \diamond H')_{rj})_{ik} Y_{kr}}{Y_{ij}} - 2Z_{ij} Z_{kr} ((H^T \diamond H')_{rj})_{ik}. \end{aligned}$$

The indexes ij and kr vary in the same range, so we can exchange them without changing the result. Moreover, the Hessian is a symmetric matrix, so

$$HF(Y)_{ij,kr} = ((H^T \diamond H')_{rj})_{ik} = ((H^T \diamond H')_{jr})_{ki} = HF(Y)_{kr,ij},$$

and this leads to

$$\begin{aligned} Z^T(K(Y) - HF(Y))Z &= \frac{1}{2} Z^T(K(Y) - HF(Y))Z + \frac{1}{2} Z^T(K(Y) - HF(Y))Z \\ &= \sum_{ij,kr} ((H^T \diamond H')_{rj})_{ik} \left[Z_{ij}^2 \frac{Y_{kr}}{Y_{ij}} + Z_{kr}^2 \frac{Y_{ij}}{Y_{kr}} - 2Z_{ij} Z_{kr} \right] \\ &= \sum_{ij,kr} \frac{((H^T \diamond H')_{rj})_{ik}}{Y_{kr} Y_{ij}} \left[Z_{ij}^2 Y_{kr}^2 + Z_{kr}^2 Y_{ij}^2 - 2Z_{ij} Z_{kr} Y_{kr} Y_{ij} \right] \\ &= \sum_{ij,kr} \frac{((H^T \diamond H')_{rj})_{ik}}{Y_{kr} Y_{ij}} \left[Z_{ij} Y_{kr} - Z_{kr} Y_{ij} \right]^2 \geq 0. \end{aligned}$$

This proves that $K(Y) - HF(Y)$ is positive semidefinite, concluding the proof. □

The function $G(X,Y)$ is quadratic in X, and if H hasn't any zero rows or columns, then $K(Y)$ is a positive diagonal matrix, so the function is convex and there exists a unique minimum at a fixed Y.

$$\nabla_X G(X,Y) = \nabla F(Y) + K(Y)X - K(Y)Y = 0 \implies X = Y - K(Y)^{-1} \nabla F(Y).$$

If we continue the computation, we discover that the minimum coincides with the MU

update formula

$$\begin{aligned} X_{ij} =& Y_{ij} - \sum_{kr} K(Y)^{-1}_{ij,kr} \nabla F(Y)_{kr} \\ =& Y_{ij} - K(Y)^{-1}_{ij,ij} \nabla F(Y)_{ij} \\ =& Y_{ij} + \frac{Y_{ij}}{(Y \diamond H^T \diamond H')_{ij}} ((A \diamond H')_{ij} - (Y \diamond H^T \diamond H')_{ij}) \\ =& \frac{Y_{ij}}{(Y \diamond H^T \diamond H')_{ij}} (A \diamond H')_{ij} \\ X =& Y .* (A \diamond H') ./ (Y \diamond H^T \diamond H'). \end{aligned}$$

So it's now easy to prove that the MU method is descending:

Lemma C.2. *If W is a positive matrix, and H is a matrix with entries in $\mathbb{R}_+ S_n$ with no zero rows or columns, then the error function $F(W)$ is nonincreasing with the MU update.*

Proof. Let $\widetilde{W} = W .* (A \diamond H') ./ (W \diamond H^T \diamond H')$. From the above observations, we know that

$$\widetilde{W} = \arg\min_X G(X, W),$$

and from the Lemma C.1, we conclude

$$F(\widetilde{W}) \leq G(\widetilde{W}, W) \leq G(W, W) = F(W).$$

□

The error function $F(W)$ is continuous, in both W and the real coefficients of H, so the conditions on H and W can be further simplified, since we only need that the update is well defined. The matrix W can thus be nonnegative (instead of positive), and H can have zero rows and columns, as long as the real matrix $W \diamond H^T \diamond H'$ has all elements positive.

Bibliography

[1] S. Arora, R. Ge, R. Kannan, and A. Moitra. Computing a Nonnegative Matrix Factorization – Provably. *STOC '12, to Appear*, page 29, 2011.

[2] P. C. Barman, N. Iqbal, and S. Y. Lee. Non-negative matrix factorization based text mining: Feature extraction and classification. *Neural Inf. Process. Pt 2, Proc.*, 4233:703–712, 2006.

[3] D. P. Bertsekas. *Nonlinear programming*. 1999.

[4] C. Boutsidis and E. Gallopoulos. SVD based initialization: A head start for non-negative matrix factorization. *Pattern Recognit.*, 41(4):1350–1362, 2008.

[5] A. Cichocki and A. H. Phan. Fast local algorithms for large scale nonnegative matrix and tensor factorizations. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E92-A(3):708–721, 2009.

[6] K. Devarajan. Nonnegative matrix factorization: An analytical and interpretive tool in computational biology, 2008.

[7] C. Ding, T. Li, and W. Peng. On the equivalence between Non-negative Matrix Factorization and Probabilistic Latent Semantic Indexing. *Comput. Stat. Data Anal.*, 52(8):3913–3927, 2008.

[8] B. Dong, M. M. Lin, and M. T. Chu. Nonnegative rank factorization-a heuristic approach via rank reduction, 2013.

[9] D. L. Donoho and V. C. Stodden. When does non-negative matrix factorization give a correct decomposition into parts? *Proc. Adv. Neural Inf. Process. Syst. 16*, pages 1141–1148, 2004.

[10] J. Eggert and E. Körner. Sparse coding and NMF. In *IEEE Int. Conf. Neural Networks - Conf. Proc.*, volume 4, pages 2529–2533, 2004.

[11] J. Eggert, H. Wersing, and E. Körner. Transformation-invariant representation and NMF. In *IEEE Int. Conf. Neural Networks - Conf. Proc.*, volume 4, pages 2535–2539, 2004.

[12] P. Favati, G. Lotti, O. Menchi, and F. Romani. Adaptive symmetric NMF for graph clustering. Technical report, Consiglio Nazionale delle Ricerche, IIT, 2016.

[13] N. Frey, B. J., Jojic. Transformation-invariant clustering and dimensionality reduction using em. In *IEEE Trans. Pattern Anal. Mach.*, volume 25, pages 1000–1014, 2000.

[14] N. Gillis. Sparse and Unique Nonnegative Matrix Factorization Through Data Preprocessing. *arXiv*, 13:34, 2012.

[15] N. Gillis and F. Glineur. Accelerated multiplicative updates and hierarchical ALS algorithms for nonnegative matrix factorization. *Neural Comput.*, 24(4):1085–105, 2012.

[16] D. A. Gregory and N. J. Pullman. Semiring rank: Boolean rank and nonnegative rank factorizations. *J. Comb. Inf. Syst. Sci.*, 8:223–233, 1983.

[17] L. Grippo and M. Sciandrone. On the convergence of the block nonlinear Gauss-Seidel method under convex constraints. *Oper. Res. Lett.*, 26(3):127–136, 2000.

[18] B. Hassibi, D. G. Stork, and G. J. Wolff. Optimal brain surgeon and general network pruning. In *IEEE Int. Conf. Neural Networks - Conf. Proc.*, volume 1993-Janua, pages 293–299, 1993.

[19] M. Hazewinkel. On positive vectors, positive matrices and the specialization order. *Dep. Pure Math.*, (R8407):1–11, jan 1984.

[20] C. J. Hsieh and I. S. Dhillon. Fast coordinate descent methods with variable selection for non-negative matrix factorization. In *Proc. 17th ACM SIGKDD*, pages 1064–1072, New York, New York, USA, 2011. ACM Press.

[21] D. Kim, S. Sra, and I. S. Dhillon. Fast projection-based methods for the least squares nonnegative matrix approximation problem. *Stat. Anal. Data Min.*, 1(1):38–51, 2008.

[22] H. Kim and H. Park. Nonnegative Matrix Factorization Based on Alternating Nonnegativity Constrained Least Squares and Active Set Method. *SIAM J. Matrix Anal. Appl.*, 30:713–730, 2008.

[23] J. Kim and H. Park. Fast nonnegative tensor factorization with an active-set-like method. In *High-Performance Sci. Comput. Algorithms Appl.*, volume 9781447124, pages 311–326. 2012.

[24] W. H. Lawton and E. A. Sylvestre. Self Modeling Curve Resolution. *Technometrics*, 13(3):617–633, 1971.

[25] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–91, 1999.

[26] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. *Adv. neural Inf. Process.*, (1):556–562, 2001.

[27] H. Liu. Non-Negative Matrix Factorization with Constraints. *Proc. 24th AAAI Conf. Artif. Intell.*, pages 506–511, 2010.

[28] P. Paatero and U. Tapper. Positive Matrix Factorization - A Nonnegative Factor Model with Optimal Utilization of Error Estimates of Data Values. *Environmetrics*, 5(2):111–126, 1994.

[29] V. P. Pauca, J. Piper, and R. J. Plemmons. Nonnegative matrix factorization for spectral data analysis. *Linear Algebra Appl.*, 416(1):29–47, 2006.

[30] V. K. Potluru, S. M. Plis, and V. D. Calhoun. Sparse shift-invariant NMF. In *Proc. IEEE Southwest Symp. Image Anal. Interpret.*, pages 69–72, 2008.

[31] H. Sawada, H. Kameoka, S. Araki, and N. Ueda. Multichannel extensions of non-negative matrix factorization with complex-valued data. *IEEE Trans. Audio, Speech Lang. Process.*, 21(5):971–982, 2013.

[32] I. J. Schoenberg. Metric spaces and positive definite functions. *Trans. Am. Math. Soc.*, 44(3):522, mar 1938.

[33] S. A. Vavasis. On the Complexity of Nonnegative Matrix Factorization. *SIAM J. Optim.*, 20(3):1364, 2010.

[34] F. Y. Wang, C. Y. Chi, T. H. Chan, and Y. Wang. Nonnegative least-correlated component analysis for separation of dependent sources by volume maximization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(5):875–888, 2010.

[35] S. Wild, J. Curry, and A. Dougherty. Improving non-negative matrix factorizations through structured initialization. *Pattern Recognit.*, 37(11):2217–2232, 2004.

[36] Z. Yang, H. Zhang, Z. Yuan, and E. Oja. Kullback-Leibler divergence for non-negative matrix factorization. In *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, volume 6791 LNCS, pages 250–257, 2011.

[37] R. Zdunek and A. Cichocki. Non-negative matrix factorization with quasi-newton optimization. In *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, volume 4029 LNAI, pages 870–879, 2006.

[38] G. Zhou, S. Xie, Z. Yang, J. M. Yang, and Z. He. Minimum-volume-constrained nonnegative matrix factorization: Enhanced ability of learning parts. *IEEE Trans. Neural Networks*, 22(10):1626–1637, 2011.