The Tucker tensor decomposition for data analysis: capabilities and advantages

Annabelle Gillet, Eric Leclercq, Lucile Sautot

The Tucker tensor decomposition for data analysis: capabilities and advantages

Annabelle Gillet
LIB Univ. Bourgogne Franche Comté
Dijon, France
annabelle.gillet@u-bourgogne.fr

Éric Leclercq
LIB Univ. Bourgogne Franche Comté
Dijon, France
eric.leclercq@u-bourgogne.fr

Lucile Sautot
UMR TETIS, AgroParisTech
Montpellier, France
lucile.sautot@agroparistech.fr

ABSTRACT

Tensors are powerful multi-dimensional mathematical objects, that easily embed various data models such as relational, graph, time series, etc. Furthermore, tensor decomposition operators are of great utility to reveal hidden patterns and complex relationships in data. In this article, we propose to study the analytical capabilities of the Tucker decomposition, as well as the differences brought by its major algorithms. We demonstrate these differences through practical examples on several datasets having a ground truth. It is a preliminary work to add the Tucker decomposition to the Tensor Data Model, a model aiming to make tensors data-centric, and to optimize operators in order to enable the manipulation of large tensors.

CCS CONCEPTS

• **Information systems → Information extraction**; **Clustering and classification**.

KEYWORDS

data analysis, tensor, tensor decomposition, Tucker

ACM Reference Format:
Annabelle Gillet, Éric Leclercq, and Lucile Sautot. 2022. The Tucker tensor decomposition for data analysis: capabilities and advantages. In *Proceedings of Gestion de Données – Principes, Technologies et Applications (BDA '22).* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

From a general point of view, data analysis is a complex process that requires the selection and retrieval of data as well as the application of one or several algorithms. This leads to two major issues: data must be transformed to fit the selected algorithms, and the output results must be interpretable. Furthermore, data analysis often requires to take into account multiple characteristics or features in algorithms to fully reveal and understand hidden value in data. For example, the study of community structures in social networks needs the possibility to discover overlapping communities as well as adding a temporal context for each community, i.e., over which period of time the community is active. Unfortunately, most algorithms do not take into consideration this kind of needs, and integrating them might require heavy modifications to the algorithm and/or produce a limited solution. To continue with the previous example, when using snapshots to detect communities their fixed size does not necessary correspond to the temporal activity of all the communities, and thus this can hide some information.

In this context, tensors are a valuable solution [2]. Indeed, their multi-dimensional nature allows to embed easily different data models. Graphs can be represented as adjacency matrices, that are 2-order tensors. In addition, when dealing with labelled graphs, the 2-order tensor can be extended to a $(2+n)$-order tensor, with n the number of types of labels. This includes temporal graphs, in which the temporality can be seen as a specific label, and they can be used for example to represent social network data [1]. Time series can be modeled as vectors in which each entry is the value at the time t. Vectors are 1-order tensors, so time series are directly embedded into tensors. Dimensions can be added to include more contextual information, as for example, in an IoT application, the sensor that took the measures and its localization. Other models such as relational, OLAP data cubes or key-value can also be represented by tensors.

Furthermore, tensors have powerful analytical operators: the decompositions. Tensor decompositions are used for various purposes such as dimensionality reduction, noise elimination, identification of latent factors, pattern discovery, ranking, recommendation or data completion. They are applied in a wide range of applications, including genomics [6], analysis of health records [21], graph mining [19] and identification and evolution of communities in social networks [1, 10]. Papalexakis et al. in [11] review major usages of tensor decompositions in data mining applications. One of these decomposition is Tucker, that factorizes a tensor with N dimensions into a smaller core tensor and a set of N factor matrices, i.e., one for each dimension. However, the results of the Tucker decompositon can be tricky to interpret, especially compared to more straightforward decompositions such as the CANDECOMP/PARAFAC [16]. Nevertheless, the core tensor, missing from the CANDECOMP/PARAFAC decomposition, provides additional insights concerning the relationships among the dimensions.

Our contributions are the following. We propose to study the Tucker decomposition, its two major algorithms and its uses in data analysis. We also identify the key points that can impact or reduce the processing of massive data, and conduct an experimentation of the algorithms on two real datasets with ground truth to show their capabilities and interpretability. It is a preliminary work aiming to incorporate the Tucker decomposition into the Tensor Data Model [5, 9]. TDM adds the notion of schema and data manipulation operators to tensors, in order to make them data-centric and to

BDA '22, October 24–27, 2022, Aubière, France

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

avoid technical and functional errors brought by the manipulation of dimensions and elements of dimensions solely through integer indexes [14]. It also uses optimization techniques to allow the execution of operators, including the decompositions, on large-scale data [4].

The remaining of this article is organized as follow: section 2 gives an overview of tensors and of some main operators, section 3 presents the Tucker decomposition and two major algorithms to compute it, section 4 relates of experiments that illustrate the different uses of the Tucker decomposition and section 5 concludes the article and presents perspectives of future works.

2 BACKGROUND OF TENSORS

Tensors are general abstract mathematical objects which can be considered according to various points of view such as a multilinear application, or as the generalization of matrices to multiple dimensions. We will use the definition of a tensor as an element of the set of the functions from the product of N sets I_j, $j = 1, \ldots, N$ to $\mathbb{R}$: $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, where N is the number of dimensions of the tensor or its order or its mode. Table 1 summarizes the notations used in this article.

Tensor operators, by analogy with operations on matrices and vectors, are multiplications, transpositions, matricizations (or unfolding) and decompositions (also named factorizations). We only highlight the most significant operators on tensors and matrices which are used in Tucker decomposition algorithms. The reader can consult [2, 8] for an overview of the major operators.

The **outer product** between a tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and another tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_M}$ noted $\boldsymbol{\mathcal{Y}} \circ \boldsymbol{\mathcal{X}}$ produces a tensor $\boldsymbol{\mathcal{Z}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N \times J_1 \times J_2 \times \cdots \times J_M}$ in which the elements are equal to:

$$z_{i_1,i_2,\ldots,i_N,j_1,j_2,\ldots,j_M} = y_{i_1,i_2,\ldots,i_N} x_{j_1,j_2,\ldots,j_M}$$

The **mode-n product** allows to multiply a tensor by a matrix or a vector. For a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times J_n \times \cdots \times J_N}$ and a matrix $\mathbf{M} \in \mathbb{R}^{I_n \times J_n}$, the result of the mode-n product noted $\boldsymbol{\mathcal{X}} \times_n \mathbf{M}$ is a new tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{J_1 \times J_2 \times \cdots \times I_n \times \cdots \times J_N}$ where:

$$y_{j_1,\ldots,j_{n-1},i_n,j_{n+1},\ldots,j_N} = \sum_{j_n=1}^{J_n} x_{j_1,\ldots,j_{n-1},j_n,j_{n+1},\ldots,j_N} m_{i_n,j_n}$$

The mode-n product between a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_n \times I_{n+1} \times \cdots \times I_N}$ and a vector $\mathbf{v} \in \mathbb{R}^{I_n}$ noted $\boldsymbol{\mathcal{X}} \times_n \mathbf{v}$ produces a tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{I_1 \times \cdots \times I_{n-1} \times I_{n+1} \times \cdots \times I_N}$ where:

$$y_{i_1,\ldots,i_{n-1},i_{n+1},\ldots,i_N} = \sum_{i_n=1}^{I_n} x_{i_1,\ldots,i_{n-1},i_n,i_{n+1},\ldots,i_N} \mathbf{v}_{i_n}$$

The **mode-n matricization** of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ noted $\boldsymbol{\mathcal{X}}_{(n)}$ produces a matrix $\mathbf{M} \in \mathbb{R}^{I_n \times \Pi_{j \neq n} I_j}$, where:

$$m_{i_n,j} = x_{i_1,\ldots,i_n,\ldots,i_N} \text{ with } j = 1 + \sum_{\substack{k=1 \\ k \neq n}}^{N} (i_k - 1) \prod_{\substack{m=1 \\ m \neq n}}^{k-1} I_m$$

The **Kronecker product** between two matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$ noted $\mathbf{A} \otimes \mathbf{B}$ produces a matrix $\mathbf{C} \in \mathbb{R}^{(IK) \times (JL)}$, in which every elements of $\mathbf{A}$ are multiplied by the matrix $\mathbf{B}$:

$$c_{m,n} = a_{i,j} b_{k,l} \text{ where } m = k + (i-1)K \text{ and } n = l + (j-1)L$$

3 TUCKER DECOMPOSITION

The Tucker decomposition [20] factorizes a N-order tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ into a core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{R_1 \times \cdots \times R_N}$ and N factor matrices $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ (see for example figure 1 for a representation of the Tucker decomposition on a 3-order tensor). The ranks $R_1, \ldots, R_N$ are input parameters that determine the number of vectors (that can be seen as different features) for each factor matrix. The input tensor can be approximated with:

$$\boldsymbol{\mathcal{X}} \simeq \boldsymbol{\mathcal{G}} \times_1 \mathbf{A}^{(1)} \cdots \times_N \mathbf{A}^{(N)}$$

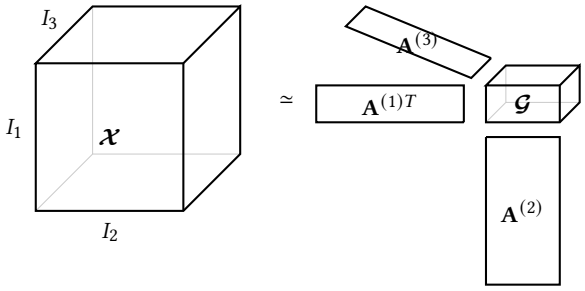


Figure 1: The Tucker decomposition, with $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times I_3}$ the input tensor, $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{R_1 \times R_2 \times R_3}$ the core tensor, and $\mathbf{A}^{(1)} \in \mathbb{R}^{I_1 \times R_1}$, $\mathbf{A}^{(2)} \in \mathbb{R}^{I_2 \times R_2}$ and $\mathbf{A}^{(3)} \in \mathbb{R}^{I_3 \times R_3}$ the factor matrices

The order of the elements of the dimensions of the input tensor does not impact the result of the decomposition. Indeed, changing it would only reorder the line vectors of the factor matrices, as each line vector stores the result of the decomposition for a given element on the dimension corresponding to the factor matrix.

To compute the Tucker decomposition, several algorithms have been proposed. Each has some advantages, as for example imposing more easily the orthogonality constraint (that allows a good clusterization of elements) or the non-negativity constraint (that provides more interpretable results). Two major algorithms are presented in this section: the Higher-Order Orthogonal Iteration (HOOI) and the Hierarchical Alternating Least Squares Non-negative Tucker Decomposition (HALS-NTD).

3.1 Higher-Order Orthogonal Iteration algorithm

The HOOI algorithm [3] is the most famous one to compute the Tucker decomposition (algorithm 1). It depends primarily on the Singular Value Decomposition (SVD), that it extends to cope with multiple dimensions.

The HOOI starts by initializing the factor matrices, by matricizing the original tensor on each dimension in order to apply the SVD and to use the R_n left singular vectors (matrix $\mathbf{U}$ of the result of the SVD truncated at the R_n^{th} column) as factor matrices. During the iterative phase (lines 2 to 7), each factor matrix is improved. To do so, a partial core tensor $\boldsymbol{\mathcal{Y}} \in \mathbb{R}^{R_1 \times \cdots \times I_n \times \cdots \times R_N}$ is computed by performing the mode-n product on the original tensor and all the factor matrices except the one being improved. This partial core tensor is then matricized on the mode corresponding to the concerned dimension, and the SVD is executed on it. As for the

Symbol	Definition	Symbol	Definition
$\boldsymbol{\mathcal{X}}$	A tensor	$\circ$	Outer product
$\mathbf{X}_{(n)}$	Matricization of a tensor $\boldsymbol{\mathcal{X}}$ on mode-n	$\otimes$	Kronecker product
a	A scalar	$\mathbf{A}^{\otimes_{-n}}$	$\mathbf{A}^{(N)} \otimes \cdots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \cdots \otimes \mathbf{A}^{(1)}$
$\mathbf{v}$	A column vector	$[\mathbf{M}]_+$	Replace negative elements by 0 or small positive value
$\mathbf{M}$	A matrix	$\times_n$	Mode-n product
		$\boldsymbol{\mathcal{X}} \times_{-n} \{\mathbf{A}\}$	$\boldsymbol{\mathcal{X}} \times_1 \mathbf{A}^{(1)} \cdots \times_{n-1} \mathbf{A}^{(n-1)} \times_{n+1} \mathbf{A}^{(n+1)} \cdots \times_N \mathbf{A}^{(N)}$

Table 1: Symbols and operators used

initialization step, the R_n left singular vectors are used as the new factor matrix. The iterative phase allows to refine the result, as the partial core tensor takes into consideration the other factor matrices. So, each factor matrix is improved depending on the other factor matrices, thus reinforcing the discovering of relationships among elements of dimensions. When a convergence criteria is met, the final core tensor is computed from the original core tensor and all the factor matrices (line 8).

Algorithm 1 Higher-Order Orthogonal Iteration (HOOI)

Require: Tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and target ranks $R_1, \ldots, R_N$
Ensure: Core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ and factor matrices $\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}$ with $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$
1: Initialize $\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(N)}$, with $\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times R_n}$, $\mathbf{U}^{(n)} \leftarrow SVD(\mathbf{X}_{(n)}).\mathbf{U}(:, 1 : R_n)$
2: **repeat**
3: **for** $n = 1, \ldots, N$ **do**
4: $\boldsymbol{\mathcal{Y}} \leftarrow \boldsymbol{\mathcal{X}} \times_N \mathbf{U}^{(N)T} \times_{n+1} \mathbf{U}^{(n+1)T} \times_{n-1} \mathbf{U}^{(n-1)T} \ldots \times_1 \mathbf{U}^{(1)T}$
5: $\mathbf{U}_{(n)} \leftarrow SVD(\mathbf{Y}_{(n)}).\mathbf{U}(:, 1 : R_n)$
6: **end for**
7: **until** < convergence >
8: $\boldsymbol{\mathcal{G}} \leftarrow \boldsymbol{\mathcal{X}} \times_N \mathbf{U}^{(N)T} \cdots \times_1 \mathbf{U}^{(1)T}$

A simpler version of the HOOI algorithm exists, the Higher-Order Singular Value Decomposition (HOSVD), that removes the iterative part of the HOOI algorithm (lines 2 to 7). It is less precise, as the iterative part allows to refine the result until a convergence is met.

The HOOI algorithm inherits from the orthogonality constraint of the SVD for the computation of the factor matrices. Thus, it works pretty well to cluster elements of a dimension depending on their behavior on the other dimensions. However, as the SVD produces matrices with positive and negative values, the HOOI is not well suited to impose the non-negativity constraint on factor matrices, as some negative values will be found (and must be removed) at each iteration.

An advantage of this algorithm is that it can easily be implemented on large tensors. The most costly operation is the computation of the SVD, that is found at the initialization (line 1) and during the iteration phase (line 5). During the iteration, as the SVD is executed on the mode-n matricized partial core tensor, that is relatively small compared to the matricized original tensor, the time and space complexity is reduced. At the initialization of the algorithm, it can be replaced with a random one to avoid the computation of the SVD on a too large matrix.

3.2 Hierarchical Alternating Least Squares algorithm

The HALS-NTD algorithm [2] uses a different approach than the HOOI algorithm (algorithm 2), even if the initialization step (line 1) can be done by using the HOSVD. An alternative version of the HALS-NTD was later proposed [12].

Algorithm 2 Hierarchical Alternating Least Squares (HALS-NTD)

Require: Tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$ and target ranks $R_1, \ldots, R_N$
Ensure: Core tensor $\boldsymbol{\mathcal{G}} \in \mathbb{R}^{R_1 \times R_2 \times \cdots \times R_N}$ and factor matrices $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$ with $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$
1: Initialize $\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(N)}$ with non-negativity constraint
2: $\boldsymbol{\mathcal{E}} \leftarrow \boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{G}} \times_1 \mathbf{A}^{(1)} \cdots \times_N \mathbf{A}^{(N)}$
3: **repeat**
4: **for** $n = 1, \ldots, N$ **do**
5: **for** $r = 1, \ldots, R_n$ **do**
6: $\mathbf{X}_{(n)}^{(r)} = \mathbf{E}_{(n)} + \mathbf{a}_r^{(n)} \left[\mathbf{G}_{(n)}\right]_r \mathbf{A}^{\otimes_{-n}T}$
7: $\mathbf{a}_r^{(n)} \leftarrow \left[\mathbf{X}_{(n)}^{(r)} \left[(\boldsymbol{\mathcal{G}} \times_{-n} \{\mathbf{A}\})_{(n)}\right]_r^T\right]_+$
8: $\mathbf{a}_r^{(n)} \leftarrow \mathbf{a}_r^{(n)} / \left\|\mathbf{a}_r^{(n)}\right\|_2$
9: $\mathbf{E}_{(n)} \leftarrow \mathbf{X}_{(n)}^{(r)} - \mathbf{a}_r^{(n)} \left[\mathbf{G}_{(n)}\right]_r \mathbf{A}^{\otimes_{-n}T}$
10: **end for**
11: **end for**
12: **for** $r_1 = 1, \ldots, R_1, \ldots, r_N = 1, \ldots, R_N$ **do**
13: $g_{r_1,\ldots,r_N} \leftarrow g_{r_1,\ldots,r_N} + \boldsymbol{\mathcal{E}} \times_1 \mathbf{a}_{r_1}^{(1)} \cdots \times_N \mathbf{a}_{r_N}^{(N)}$
14: $\boldsymbol{\mathcal{E}} \leftarrow \boldsymbol{\mathcal{E}} + \Delta_{g_{r_1,\ldots,r_N}} \mathbf{a}_{r_1}^{(1)} \circ \cdots \circ \mathbf{a}_{r_N}^{(N)}$
15: **end for**
16: **until** < convergence >

The HALS-NTD starts also by initializing the factor matrices (line 1), but adds a non-negativity constraint to manipulate only positive values in the remaining of the algorithm. An error tensor $\boldsymbol{\mathcal{E}} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ (noted $\mathbf{E}_{(n)}$ when it is matricized on dimension n), that stores the difference between the original tensor and the reconstructed tensor from the core tensor and the factor matrices, is computed (line 2). The iteration phase (lines 3 to 16) is more complex than the one of the HOOI algorithm. Rather than improving a whole factor matrix at a time, it improves a vector of a factor matrix at a time. To do so, at the line 6, the current vector (the one being improved) is put in relation with all the other factor matrices associated with the part of the core tensor representing the strength of the relationships of the current vector regarding the vectors of the other factor matrices. This result is added to the error tensor, and stored in $\mathbf{X}_{(n)}^{(r)}$, that can be seen as a matricized tensor

representing the contribution of the current vector to the global result combined to the error tensor. At line 7, the current vector is improved by multiplying $\mathbf{X}_{(n)}^{(r)}$ with the part of the reconstructed tensor corresponding to the current vector. The current vector is then normalized with a l_2 norm (line 8), and the error tensor is updated (line 9). Once all the vectors of the factor matrices have been improved, the core tensor is updated from the previous core tensor, the error tensor and the new vectors of the factor matrices (line 13), and finally the error tensor is updated to integrate the changes in the core tensor (line 14).

The major advantage of the HALS-NTD is that it enforces the non-negativity constraint by imposing it during the initialization step, and then by improving the result without obtaining (and without having to eliminate) negative values during the iterative part (line 3 to 12). Thus, it eases the direct interpretation of the factor matrices as well as the core tensor.

However, as this algorithm computes the decomposition column vector by column vector for each factor matrices, it is computationally demanding, and harder to optimize than the HOOI one. Furthermore, there is almost no implementation of the HALS-NTD alogrithm. To the best of our knowledge, only Cichocki and Phan have provided a Matlab implementation in [2].

3.3 Related work

The Tucker decomposition has been used in several applications. Due to the lack of implementation for the HALS-NTD algorithm, the following articles are related mainly to the HOOI algorithm.

Cichocki in his book [2] shows several application of various decompositions on small tensors, mainly for image analysis. Romeo et al. [13] used the Tucker decomposition to cluster documents. Thanks to this decomposition, they were able to process documents in several languages in the same tensor, in order to find similarities in the whole dataset. Sun et al. [18] used Tucker on social network data in order to find clusters. They applied it on the Enron dataset. They also propose visualization techniques based on graph to display the result of the decompositon. Huang et al. [7] compare the Tucker decomposition to the PCA and SVD associated to k-means. They run experiments on three datasets of images to show the similarities among these algorithms. Zhou et al. [22] took a different approach and used the Tucker decomposition as a supervised learning algorithm. They obtained promising results to cluster images.

As these works only use the HOOI algorithm, they only benefit from a part of the Tucker decomposition capabilities. They aim to cluster data, and do not rely on the direct interpretability of the factor matrices and the core tensor even if it brings valuable insights.

4 EXPERIMENTS

To better understand the capabilities of the Tucker decomposition regarding data analysis, we perform some experiments on two datasets: the database of faces (AT&T) [15] and the primary school dataset [17]. The full code of the experiments is available online[1].

[1]https://github.com/AnnabelleGillet/Tucker-experiments

4.1 Database of faces experiment

This dataset contains greyscale images of the faces of 40 persons. Each person has 10 images, so the dataset is composed of 400 images. The dimensions of the images are 92×112. The ground truth is the person corresponding to an image.

The goal of this experiment is to evaluate the capabilities of the Tucker decomposition to find clusters in data. To do so, we create a 3-order tensor of size $92 \times 112 \times 400$, with one dimension for the vertical pixels, one dimension for the horizontal pixels and one dimension for the images. The values of the tensor are these of the corresponding pixel in the images. We then run the Tucker decomposition with the ranks 3, 4 and 7, with both the HOOI and the HALS-NTD algorithms.

To cluster the images, we apply a k-means on the factor matrix corresponding to the dimension of the images. As we know that there are 40 subjects, we choose $k = 40$. To evaluate the result, we compute its accuracy with:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

with TP = True Positive, TN = True Negative, FP = False Positive and FN = False Negative.

As the k-means selects random seeds to perform its computations, we repeat 100 times its execution and summarize in table 2 the results obtained. The confusion matrices of the results can be found on the Github of the experiments.

Measure	HOOI	HALS-NTD
Minimum	0.5425	0.2425
Maximum	0.7175	0.38
Mean	0.6122	0.3184
Median	0.61	0.32
Standard deviation	0.0346	0.0209
Variance	0.0012	0.0004

Table 2: Accuracy of the result of the clustering of the images of the database of faces for each algorithm of the Tucker decomposition

The HOOI algorithm presents an accuracy score almost twice as good as the HALS-NTD one. With a mean of 61% as accuracy score, the HOOI algorithm is promising as an unsupervised classification tool. Indeed, this kind of task is often realized with supervised methods in which the algorithm is first trained on known images, but it comes at the cost of the necessity of having ground truth data to train the algorithm that also induces a risk of overfitting. With no training step, the Tucker decomposition avoids this risk.

This experiment highlights the power of the Tucker decomposition principle, which captures the relationships among dimensions in factor matrices. Here we used only one factor matrix. Thus, compared to the size of the original tensor, only few information have to be used to obtain satisfying results without any preprocessing nor using a image-specialized algorithm.

4.2 Primary school experiment

This dataset contains the interactions among students in a primary school, along with their teachers. There are 242 persons (including 10 teachers) split into 10 classes. An interaction is recorded if it lasts

at least 20 seconds. The recording takes the form (person1, person2, timestamp in second). The class of each student is the ground truth of this dataset.

In this experiment, we seek to highlight the capabilities of the Tucker decomposition to provide meaningful factor matrices. To do so, we build a 3-order tensor of size $242 \times 242 \times 208$, with two symmetric dimensions used to represent the persons, and the third dimension to represent the time, with a granularity of 5 minutes. If a person has been in contact with another person at a time t, then the value in the tensor indexed by the corresponding dimension values is 1.

As for the previous experiment, we run the Tucker decomposition with both algorithms in order to compare the results, with the empirically chosen ranks 13 (for the first person dimension), 13 (for the second person dimension) and 4 (for the time dimension).

The factor matrices for the first dimension are shown in figure 2. Each line represents a rank, and the columns are the persons. The students are ordered by their class: the first columns are the students of the class 1A, then 1B, and so on until 5B, and the 10 teachers are the last 10 columns. The natural non-negativity constraint of the HALS-NTD is of great help to understand the result. Indeed, we can distinguish 10 ranks in which each class appears distinctly, and three heterogeneous ranks. Conversely, the results of the HOOI algorithm are harder to interpret, and the classes do not appear as precisely as with the HALS-NTD algorithm.

In this experiment, the role of the core tensor is important: it gives insights regarding the strength of the relationships of the ranks among dimensions. For example, the $g_{1,1,1}$ value indicates if the vectors $\mathbf{a}_1^{(1)}$, $\mathbf{a}_1^{(2)}$ and $\mathbf{a}_1^{(3)}$ are strongly related or not.

To illustrate the usefulness of the core tensor, we can focus on a particular rank of the first dimension and see how it is related to the ranks of the other dimensions. The figure 3 represents this mechanism when fixing the rank of the first dimension to the one corresponding to the class 1A in the result of the Tucker decomposition performed with the HALS-NTD algorithm.

This figure shows some interesting results. The 1^{st} strongest value of the core tensor indicates that the class 1A has strong ties with itself, mainly at the break times and before and after the lunch break (figure 3a). It makes sense because at the breaks the students move from their classroom and go outside, so it creates more interactions among students. The 2^{nd} strongest value of the core tensor shows again a relationship of the class 1A with itself, but this time during the class hours (figure 3b). The 3^{rd} strongest value indicates a relationship between the class 1A and 1B during the breaks, including the lunch one (figure 3c). As the students of these two classes are of the same age, it is logical that they have more ties. Finally, the 4^{th} value of the core tensor shows a relationship between the class 1A and a heterogeneous cluster that gathers students from grades 1, 2 and 3, during the breaks (figure 3d).

The advantages of the high interpretable capability of the HALS-NTD algorithm are twofold: 1) the vectors of the factor matrices give insights regarding the elements that contribute to the rank; and 2) the core tensor allows to link the ranks of one factor matrix to the ranks of the other factor matrices, and thus it gives more context to the result, as for example in figure 3 where we have the temporal activity of the different communities.

5 CONCLUSION

To conclude, the Tucker decomposition is a useful data analysis technique. Its two main algorithms, the HOOI and the HALS-NTD, have both advantages over the other: the HOOI algorithm shows promising results to cluster elements of a dimension, while the HALS-NTD provides interpretable insights about the relationships of the elements among dimensions.

However, the HALS-NTD algorithm is less known than the HOOI one, and in consequence it has almost never been implemented. We plan to integrate these Tucker algorithms to the Tensor Data Model, and to optimize them in order to allow their execution on large tensors, as we did for the CANDECOMP/PARAFAC decomposition [4]. Indeed, real data can create such tensors, that emphasis the need for optimized algorithms regarding the space and the execution time.

From a more applied point of view, we plan to use the Tucker decomposition to find patterns among birds migration, in order to prevent the spread of the avian influenza, in the context of the MOOD project[2].

ACKNOWLEDGMENTS

This study was partially funded by EU grant 874850 MOOD and by ISITE-BFC (ANR-15-IDEX-0003).

REFERENCES

[1] Miguel Araujo, Spiros Papadimitriou, Stephan Günnemann, Christos Faloutsos, Prithwish Basu, Ananthram Swami, Evangelos E Papalexakis, and Danai Koutra. 2014. Com2: fast automatic discovery of temporal ('comet') communities. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 271–283.

[2] Andrzej Cichocki, Rafal Zdunek, Anh Huy Phan, and Shunichi Amari. 2009. *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation.* John Wiley & Sons.

[3] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. 2000. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications* 21, 4 (2000), 1253–1278.

[4] Annabelle Gillet, Éric Leclercq, and Nadine Cullot. 2021. MuLOT: Multi-level Optimization of the Canonical Polyadic Tensor Decomposition at Large-Scale. In *European Conference on Advances in Databases and Information Systems*. Springer, 198–212.

[5] Annabelle Gillet, Éric Leclercq, Marinette Savonnet, and Nadine Cullot. 2020. Empowering big data analytics with polystore and strongly typed functional queries. In *Symposium on International Database Engineering & Applications*. 1–10.

[6] Victoria Hore, Ana Viñuela, Alfonso Buil, Julian Knight, Mark I McCarthy, Kerrin Small, and Jonathan Marchini. 2016. Tensor decomposition for multiple-tissue gene expression experiments. *Nature genetics* 48, 9 (2016), 1094–1100.

[7] Heng Huang, Chris Ding, Dijun Luo, and Tao Li. 2008. Simultaneous tensor subspace selection and clustering: the equivalence of high order svd and k-means clustering. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge Discovery and Data mining*. 327–335.

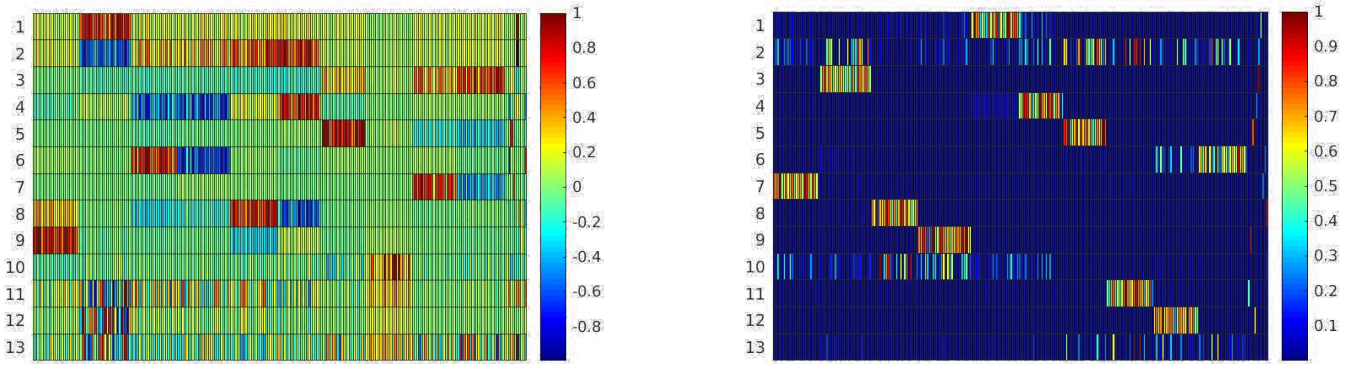
[8] Tamara G Kolda and Brett W Bader. 2009. Tensor decompositions and applications. *SIAM review* 51, 3 (2009), 455–500.

[9] Éric Leclercq, Annabelle Gillet, Thierry Grison, and Marinette Savonnet. 2019. Polystore and Tensor Data Model for Logical Data Independence and Impedance Mismatch in Big Data Analytics. In *Transactions on Large-Scale Data-and Knowledge-Centered Systems XLII*. Springer, 51–90.

[10] Evangelos E Papalexakis, Leman Akoglu, and Dino Ience. 2013. Do more views of a graph help? community detection and clustering in multi-graphs. In *International Conference on Information Fusion*. IEEE, 899–905.

[11] Evangelos E Papalexakis, Christos Faloutsos, and Nicholas D Sidiropoulos. 2016. Tensors for data mining and data fusion: Models, applications, and scalable algorithms. *Transactions on Intelligent Systems and Technology (TIST)* 8, 2 (2016), 16.

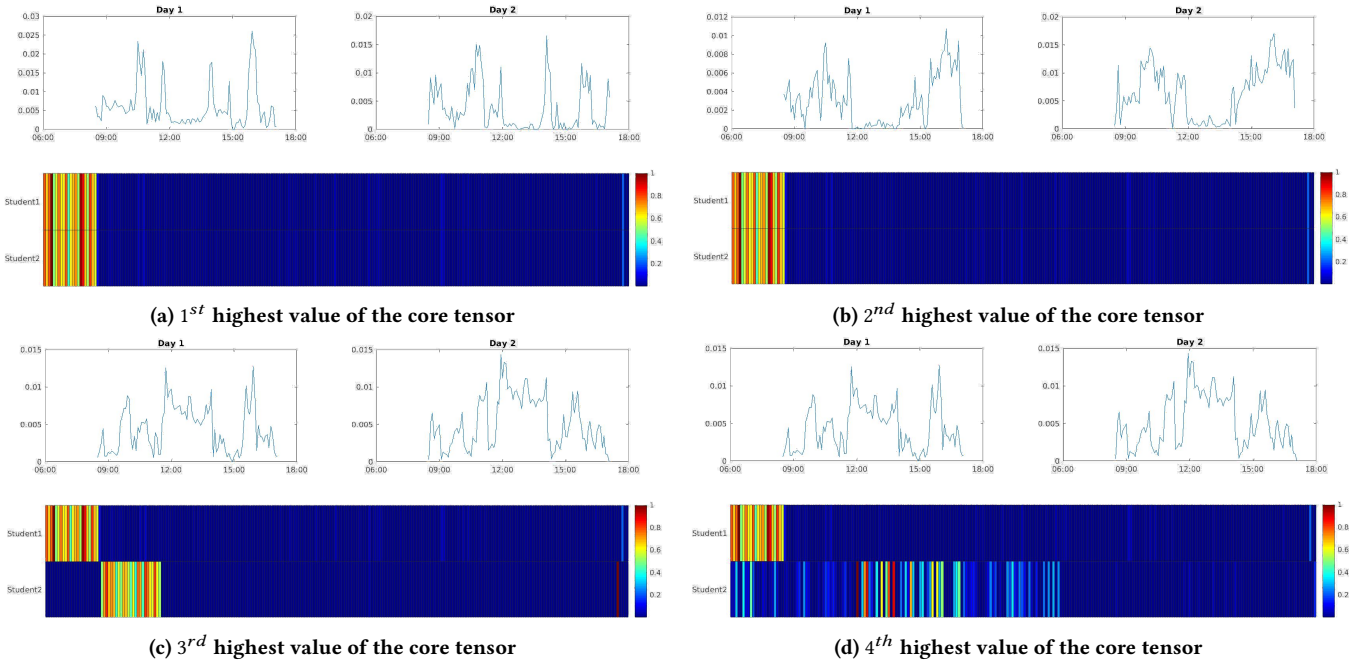
[2] https://mood-h2020.eu/



(a) HOOI algorithm

(b) HALS-NTD algorithm

Figure 2: Factor matrices for the dimension representing the persons in the primary school experiment



(a) 1^{st} highest value of the core tensor

(b) 2^{nd} highest value of the core tensor

(c) 3^{rd} highest value of the core tensor

(d) 4^{th} highest value of the core tensor

Figure 3: Ranks from each factor matrix that are strongly related to each other when fixing the rank of the first dimension to the one representing the class 1A

[12] Anh Huy Phan and Andrzej Cichocki. 2011. Extended HALS algorithm for nonnegative Tucker decomposition and its applications for multiway analysis and classification. *Neurocomputing* 74, 11 (2011), 1956–1969.

[13] Salvatore Romeo, Andrea Tagarelli, and Dino Ienco. 2014. Semantic-based multilingual document clustering via tensor modeling. In *EMNLP: Empirical Methods in Natural Language Processing*. 600–609.

[14] Alexander Rush. 2010. *Tensor Considered Harmful*. Technical Report. Harvard NLP. http://nlp.seas.harvard.edu/NamedTensor

[15] Ferdinando S Samaria and Andy C Harter. 1994. Parameterisation of a stochastic model for human face identification. In *Proceedings of 1994 IEEE workshop on applications of computer vision*. IEEE, 138–142.

[16] Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos. 2017. Tensor decomposition for signal processing and machine learning. *Transactions on Signal Processing* 65, 13 (2017), 3551–3582.

[17] Juliette Stehlé, Nicolas Voirin, Alain Barrat, Ciro Cattuto, Lorenzo Isella, Jean-François Pinton, Marco Quaggiotto, Wouter Van den Broeck, Corinne Régis, Bruno Lina, et al. 2011. High-resolution measurements of face-to-face contact patterns in a primary school. *PloS one* 6, 8 (2011), e23176.

[18] Jimeng Sun, Spiros Papadimitriou, Ching-Yung Lin, Nan Cao, Shixia Liu, and Weihong Qian. 2009. Multivis: Content-based social network exploration through multi-way visual analysis. In *Proceedings of the 2009 SIAM International Conference on Data Mining*. SIAM, 1064–1075.

[19] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. 2006. Beyond streams and graphs: dynamic tensor analysis. In *ACM SIGKDD International Conference on Knowledge Discovery and Data mining*. ACM, 374–383.

[20] Ledyard R Tucker. 1966. Some mathematical notes on three-mode factor analysis. *Psychometrika* 31, 3 (1966), 279–311.

[21] Kai Yang, Xiang Li, Haifeng Liu, Jing Mei, Guotong Xie, Junfeng Zhao, Bing Xie, and Fei Wang. 2017. TaGiTeD: Predictive task guided tensor decomposition for representation learning from electronic health records. In *Proc. of the Thirty-First AAAI Conference on Artificial Intelligence*.

[22] Guoxu Zhou, Andrzej Cichocki, Qibin Zhao, and Shengli Xie. 2015. Efficient nonnegative tucker decompositions: Algorithms and uniqueness. *IEEE Transactions on Image Processing* 24, 12 (2015), 4990–5003.