View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repositorio da Producao Cientifica e Intelectual da Unicamp

Universidade Estadual de Campinas
Instituto de Computação

Eric Velten de Melo

Semi e Weighted Semi-Nonnegative Matrix Factorization: Estudo Comparativo

Semi and Weighted Semi-Nonnegative Matrix Factorization: Comparative Study

CAMPINAS
2015

Universidade Estadual de Campinas
Instituto de Computação

Eric Velten de Melo

Semi and Weighted Semi-Nonnegative Matrix Factorization: Comparative Study

Semi e Weighted Semi-Nonnegative Matrix Factorization: Estudo Comparativo

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requerements for the degree of Master in Computer Science.

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Supervisor/Orientador: Prof. Dr. Jacques Wainer

Este exemplar corresponde à versão final da Dissertação defendida por Eric Velten de Melo e orientada pelo Prof. Dr. Jacques Wainer.

CAMPINAS
2015

Agência(s) de fomento e nº(s) de processo(s): CAPES

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Maria Fabiana Bezerra Muller - CRB 8/6162

M491s

Melo, Eric Velten de, 1987-
Semi and weighted semi-nonnegative matrix factorization : comparative study / Eric Velten de Melo. – Campinas, SP : [s.n.], 2015.

Orientador: Jacques Wainer.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Otimização com restrições. 2. Matrizes (Matemática). 3. Matrizes não-negativas. I. Wainer, Jacques,1958-. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Semi e weighted wemi-nonnegative matrix factorization : estudo comparativo
Palavras-chave em inglês:
Constrained optimization
Matrices
Nonnegative matrices
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Jacques Wainer [Orientador]
Siome Klein Goldenstein
Julio Michael Stern
Data de defesa: 16-11-2015
Programa de Pós-Graduação: Ciência da Computação

Universidade Estadual de Campinas
Instituto de Computação

Eric Velten de Melo

Semi and Weighted Semi-Nonnegative Matrix Factorization: Comparative Study

Semi e Weighted Semi-Nonnegative Matrix Factorization: Estudo Comparativo

Banca Examinadora:

- Prof. Dr. Jacques Wainer
  IC/Unicamp
- Prof. Dr. Julio Michael Stern
  IME/USP
- Prof. Dr. Siome Klein Goldenstein
  IC/Unicamp

A ata da defesa, onde constam as assinaturas dos membros da banca, está arquivada pela Universidade Estadual de Campinas.

Agradecimentos

Gostaria de agradecer primeiramente à Universidade Estadual de Campinas, à Fundação CAPES e ao meu orientador, que tornaram possível a realização dessa dissertação.

Essas foram as causas principais, mas inúmeros foram os fatores que me levaram até aqui e que me permitiram chegar até o fim. Agradeço à minha mãe por ter me dado condições e apoio para tudo isso, desde os meus primeiros passos até agora. Agradeço à minha querida Ethel, por ter me dado apoio e motivação nos momentos difíceis.

Agradeço aos meus amigos pelos momentos de descontração, aos moradores e agregados da República Traz o Martelo onde morei durante a maior parte desse processo. Agradeço ao pessoal do LBIC por terem me dado a oportunidade de trabalhar junto a eles de vez em quando. Agradeço ao pessoal do Eldorado, que me deram apoio e incentivo para concluir as etapas finais. Preferi não citar nomes para não correr o risco de me esquecer de alguém, mas sintam-se todos citados!

Enfim, para todos aqueles que fizeram parte da minha vida e contribuíram direta ou indiretamente para esse resultado final, meus sinceros agradecimentos!

Resumo

Algoritmos que envolvem fatoração de matrizes tem sido objeto de intensos estudos nos anos recentes, gerando uma ampla variedade de técnicas e aplicações para diversos tipos de problemas.

Dada uma matriz de dados de entrada X, a forma mais simples do problema de fatoração de matrizes pode ser definido como a tarefa de encontrar as matrizes F e G, usualmente com posto baixo, tal que $X \approx FG$.

São consideradas duas variações principais do problema de fatoração de matrizes: a fatoração de matrizes semi-não-negativa (*Semi Nonnegative Matrix Factorization* (**SNMF**)), que requer que a matriz G seja não-negativa, e a fatoração de matrizes semi-não-negativa com pesos (*Weighted Nonnegative Matriz Factorization* (**WSNMF**)), que lida adicionalmente com casos onde há dados de entrada faltantes ou incertos.

Essa dissertação tem como principal objetivo comparar diferentes algoritmos e estratégias para resolver esses problemas, focando em duas estratégias principais: Mínimos Quadrados Alternado com Restrição *Constrained Alternating Least Squares* e Atualização Multiplicativa *Multiplicative Updates.*

Abstract

Algorithms that involve matrix factorization have been the object of intense study in the recent years, generating a wide range of techniques and applications for many different problems.

Given an input data matrix X, the simplest matrix factorization problem can be defined as the task to find matrices F and G, usually of low rank, such that $X \approx FG$.

I consider two different variations of the matrix factorization problem, the *Semi-Nonnegative Matrix Factorization*, which requires the matrix G to be nonnegative, and the *Weighted Semi-Nonnegative Matrix Factorization*, which deals additionally with cases where the input data has missing or uncertain values.

This dissertation aims to compare different algorithms and strategies to solve these problems, focusing on two main strategies: *Constrained Alternating Least Squares* and *Multiplicative Updates.*

List of Figures

List of Tables

Contents

Chapter 1

Introduction

This dissertation is divided as follows: In this chapter I present a brief introduction for matrix factorization problems, their most relevant applications and the formulations which are the topic of this study: Semi Non-negative Matrix Factorization (SNMF) and Weighted Semi Non-Negative Factorization (WSNMF). Chapter 2 contains details of the implementation of the algorithms and related research on the topic. Chapter 3 details the methodology, results and discussion of the experiments with the algorithms implementations. Chapter 4 deals with the methodology, results and discussion of the clustering experiment using selected algorithms from the previous experiment. In the last chapter, I elaborate on the conclusion and future work.

Nonnegative Matrix Factorization (NMF) (sometimes also called NNMA *Nonnegative Matrix Approximation*) is a much discussed topic in current research. It has been used in many different applications and many alternative formulations and implementations were studied. From the applications which employ NMF algorithms or any of its variants, it can be cited: DNA gene expression analysis[40][46][3], spectra recovery[31], feature extraction and pattern recognition[21][22], multimedia data analysis[6], text mining[30][43], document summarization[28], financial data analysis[10], social network analysis[39], rating prediction, recommendation[45] and many others.

The most common formulation for the Nonnegative Matrix Factorization problem is as follows: Given a non-negative input data matrix X of dimensions $m \times n$, find non-negative matrix factors F and G of lower rank dimensions $m \times k$ and $n \times k$ respectively, such that it minimizes the Eq. 1.1 (The symbols X, F and G are used throughout the text to represent the input data Matrix, the left-hand and right-hand matrix factors, respectively). There are many variations developed for the basic NMF problem formulation. NMF itself is a constrained version of the general *Low Rank Approximation* problem, as seen in Eq. 1.2. One of the optimal solutions of the Low Rank Approximation can be obtained through *Truncated Singular Value Decomposition* [36].

NMF, however, is a non-convex problem with many local optima. It has been shown to be NP-hard[38].

$$\begin{aligned} &\min_{F,G} \|X - FG^T\|^1 \\ &\text{subject to} \qquad\qquad X \geq 0, F \geq 0, G \geq 0 \end{aligned} \tag{1.1}$$

$$\min_{F,G} \|(X - FG^T)\| \tag{1.2}$$

1.1 NMF algorithms classification

The survey [8] divides NMF variations into four broad categories: **Basic NMF**, **Constrained NMF**, **Structured NMF** and **Generalized NMF**. No classification is perfect and is able to fit to all problems, but it can be used as a loose reference to the many different approaches that stem from NMF.

1.1.1 Basic NMF

Basic NMF concerns solely with the non-negativity constraint formulation (Eq. 1.1) and their associated optimisation techniques and algorithms. To avoid ambiguity, I will take "basic NMF" to mean this specific NMF formulation. Otherwise, NMF refers generally to all algorithms and formulations that stem from NMF.

1.1.2 Constrained NMF

Constrained NMF applies additional constraints besides the non-negativity constraints on factor matrices F and G. We could extend an objective function $J(F, G)$ to include some general constraints that depend on F and G as in Eq. 1.3. $J_1(F)$ and $J_2(G)$ are penalty terms that enforce a certain constraint. α and β are regularization parameters that balance how strongly these constraints are enforced.

$$\min_{F,G} J(F, G) + \alpha J_1(F) + \beta J_2(F) \tag{1.3}$$

Varying the formulas for $J_1(F)$ and $J_2(G)$, one can come up with different Constrained NMF problems, such as Sparse NMF, Orthogonal NMF, discriminant NMF and NMF on manifold.

1.1.3 Structured NMF

Structured NMF algorithms modify the original structure of the objective function directly, rather than adding constraints to penalize the objective function as in Constrained NMF. It can be generalized by an application of an arbitrary function $M(F, G)$ as in Eq. 1.4. This general formula can vary slightly, but the basic idea remains the same. Examples of algorithms classified as Structured NMF are Weighted NMF, Convolutive NMF and Non-negative Marix Trifactorization.

$$X \approx M(F, G) \tag{1.4}$$

[1] $\|.\|$ denotes the Frobenius norm

1.1.4 Generalized NMF

We can consider the Generalized NMF as extensions of NMF or NMF variations. It might breach one or more characteristics of Basic NMF, such as the non-negativity constraints, data type or factorization pattern, therefore including a broad set of algorithms related to NMF but not strictly NMF. Examples include Semi-NMF, Non-negative Tensor Factorization, Non-negative Matrix-set Factorization and Kernel NMF.

1.2 NMF Applications

1.2.1 NMF for Clustering

Ding et al. [7] have shown that some NMF formulations are equivalent to some k-Means[7] problems. Specifically, that Symmetric NMF (Approximating $X \approx HH^T$) is equivalent to kernel k-means and basic NMF is equivalent to spectral clustering when the orthogonality constraints are relaxed. These demonstrations suggest that there is a strong relationship between matrix factorization and clustering. Given the appropriate set of conditions and restrictions, they can be shown to solve equivalent problems.

K-means clustering aims to minimize the distance between each data point x_i from $X = [x_1, x_2, ..., x_n]$ and its assigned cluster f_j. The objective function J_κ, κ being the number of clusters, is shown in Eq. 1.5.

$$J_k = \sum_{k=1}^{\kappa} \sum_{j \in c_j} \|x_i - f_j\|^2 \tag{1.5}$$

If we take the input data $X_{m \times n}$ composed of n feature vectors of size m, a matrix $F_{m \times k}$ where each column f_i corresponds to a centroid and a matrix $G_{n \times k}$ associating each of the n data points to one of the k centroids ($G_{ij} = 1$ when x_i is assigned to cluster j), then the objective function for K-means can be rewritten as Eq. 1.6. This is similar to a NMF problem, with G restricted to vectors with exactly one element set to 1 and the other elements set to zero. This restricts the feasible space of solutions considerably. NMF algorithms are much more general.

$$J_k = \|X - FG^T\| \tag{1.6}$$

NMF has several advantages over traditional clustering algorithms such as K-Means[8].

- NMF is flexible. It can model widely varying data distributions, as it does not assume much about the data, as compared to regular K-Means clustering, which assumes rigid spherical clusters. Also, many data mining and machine learning problems can be modelled as an NMF problem, given an appropriate set of constraints and structure.
- NMF can do both hard and soft clustering simultaneously.
- NMF is able to simultaneously cluster the rows (data points) and columns (features).

- It has been shown[17] that NMF with sparseness constraints applied can outperform K-Means.

Relationship between performance and objective function

When using NMF for clustering, it is useful to note that minimizing the objective function is not a guarantee of good clustering performance. This is due to the fact that in clustering problems, there is no unique way to determine the quality of a cluster, although many such metrics exist, like purity, normalized mutual information or accuracy. In practice, the clusters of a given dataset could have many different shapes and sizes. Also, these clusters could overlap with each other and they can not be easily identified and separated. As a result, it is difficult to effectively capture the cluster structures using a single clustering objective function[8]. Therefore, the ultimate clustering quality measure is always the suitability to solving a given problem.

1.2.2 NMF for Product Recommendation and Rating Prediction

Product Recommendation and Rating Prediction is an open interesting problem which have received a lot of attention recently. The Netflix Prize Competition has demonstrated of matrix factorization models over classic nearest-neighbour techniques[19]. A basic algorithm for recommendation using matrix factorization can be modelled in this way:

Let $X_{m\times n}$ be an input data matrix consisting of m users and n item ratings. Each user rates only a few of the items, resulting often in an incomplete data matrix with many missing values. We'd like to know, for each user/item pair $X(i,j)$, the predicted rating of the unrated items. A dimensionality reduction/matrix factorization approach can be applied by finding low dimensional matrices F and G such that $X_{mxn} \approx F_{mxk}{G^T}_{kxn}$. One can then compute the prediction matrix $\widetilde{X} = FG^T$ with $\widetilde{X}(i,j)$ denoting the rating prediction from user i to the item j.

The challenge in this approach is how to deal with the missing values. Earlier approaches to solve this problem relied on imputation to fill the missing values[32]. However, imputation can be very expensive and inaccurate imputation might distort the data considerably[19]. Other works suggested modelling directly only the observed ratings, while avoiding over-fitting through a regularized model[29]. These approaches minimize the regularized squared error on the set of known ratings, as in Eq.1.7. Here, κ is the subset of user/item pairs (i,j) which have been rated. The constant λ is a regularization parameter and is usually determined by cross-validation.

$$\min_{F,G} \sum_{i,j\in\kappa} (X_{ij} - F_i G_j)^2 + \lambda(\|F_i\|^2 + \|G_j\|^2) \tag{1.7}$$

Another approach which was shown to produce good results is modelling the problem as a Weighted Nonnegative Matrix Factorization (WNMF)[18] problem. In this approach, the problem becomes just as the WSNMF approach described here, except that nonnegativity is enforced on the input data and matrix factors.

Inputs with Varying Confidence Levels

A major advantage of using a Weight matrix is the ability to represent confidence levels for the input data. An example of this is a recommender system based on implicit feedback. On those systems, the user might not always give ratings directly, but instead provide hints or cues that he might like or dislike a product. Those hints can increase or decrease the confidence level of the recommender system for that particular rating. Confidence can stem from available numerical values that describe the frequency of actions, for example, how much time the user watched a certain show or how frequently a user bought a certain item. These numerical values indicate the confidence in each observation[19].

1.2.3 Weighted NMF Applications

Weighted NMF has been applied successfully to some problems, such as Face Feature Extraction[14], product recommendation[18], Data-driven simulation and control, matrix completion and system identification with missing data[25].

A multiplicative algorithm for WSNMF was applied for solving the problem of motion segmentation with missing data[26]. The formulation proposed was shown to outperform current state-of-the-art algorithm (which was based on spectral clustering) both in execution time and accuracy. They used the weighting matrix to account for object occlusions and missing tracked points. Using SNMF instead of NMF allowed them to use velocity information directly to build a more natural motion component representation. The implementation of WSNMF using multiplicative updates used in this work is the same used in my experimental tests (Chapter 3 and 4).

The work [24] uses a Weighted Low Rank Approximation algorithm which is an extension of SVD low-rank approximation using a weight matrix to design 2-D digital filters. The weights are used to give emphasis to important parts of the entries of the sampled frequency response matrix.

1.3 Studied Formulations

In this study I am concerned with Semi Non-negative Matrix Factorization (SNMF) and an extension of it for dealing with missing data, Weighted Semi Non-negative Matrix Factorization (WSNMF). The Semi-NMF problem formulation was first proposed in [9]. In this formulation, the input data matrix X as well as the left-hand side matrix F are unconstrained, but the right-hand side matrix G must be non-negative, as shown in Eq.1.8.

$$\begin{aligned} &\min_{F,G} \|X - FG^T\| \\ &\text{subject to} \qquad\quad G \geq 0 \end{aligned} \tag{1.8}$$

Therefore, Semi-NMF removes some of the constraints from the basic NMF formulation, while still retaining the non-negativity constraint on the right-hand side matrix G. One of the motivations for the Semi-NMF formulation lies on the fact that a clustering problem can be described in the form of a matrix factorization $X = FG^T$ in which X is

the data matrix, F contains the cluster centroids and G contains the cluster membership indicators. Despite the fact that F may typically contain positive and negative data values, G is non-negative [9]. Even though Semi-NMF is a lighter version of NMF in terms of constraints it is still a NP-Hard problem, as shown by [12].

When one has to deal with situations involving some missing or uncertain input data, one can reformulate the minimization in Eq. **??**, adding a weight matrix W_{mxn} whose values are 1 when the corresponding element in X is present and 0 when it is missing. The resulting minimization problem can be visualized in the Eq. 1.10. The symbol '$\odot$' represents an element-wise multiplication. In this way, the values of $(X - FG^T)$ in which the respective element of W is 0 is not considered in the calculation of the error. It is also possible to introduce weights relative to some noise estimate of the input data measured, making the weights inversely proportional to this presumed noise, thus being able to deal with not only missing data but also uncertain data. This approach can lead to a better reconstruction of the internal data structure [35]. The introduction of weights however, makes the problem considerably more complex to solve. [11] has shown that adding weights to the unconstrained Low Rank Approximation problem is already NP-Hard.

A variation of the matrix factorization problem which has been so far scarcely studied is the combination of the non-negativity constraint on the rightmost matrix G *Semi-NMF*, with the addition of a weight matrix W to account for missing or uncertain data, as shown in Eq. 1.9, from now on referred as *Weighted Semi-Nonnegative Matrix Factorization*, or *WSNMF*. While SNMF is related to clustering, WSNMF is related to clustering with missing or uncertain data.

$$\begin{aligned} &\min_{F,G} \|W \odot (X - FG^T)\| \\ &\text{subject to} \qquad\qquad G \geq 0 \end{aligned} \tag{1.9}$$

$$\min_{F,G} \|W \odot (X - FG^T)\| \tag{1.10}$$

Chapter 2

Algorithms

This chapter presents a discussion related to the implementation of the NMF algorithms in general and the proposed implementations to SNMF and WSNMF. Initialization, stopping criteria, solution uniqueness and complexity analysis are briefly discussed. The two main approaches for solving NMF, Multiplicative Updates and Constrained Alternating Least Squares are discussed along with implementation details for SNMF and WSNMF.

2.1 Initialization

Initialization is an important factor for NMF convergence speed and minimization of the objective function. Since it is a non-convex problem, the solution falls often into local minima. A good initialization can improve the algorithm performance, leading to rapid error reduction and faster convergence. One possible approach to overcome this is running the NMF algorithm several times with a random initialization. This approach, however, can be very time consuming.

Many initialization methods have been tested in the literature. Spherical K-means clustering[42], SVD[2], relaxed K-means clustering[44], PCA, fuzzy clustering, Gabor wavelet [47], population based[16] and many others. According to [8], factorization-based initialization methods and clustering-based initialization are able to lead to rapid error reduction and faster convergence. In [20], a comparison was made between six different initialization methods: Random, Centroid, SVD-Centroid, Random col, Random C and Co-ocurrence. From these, the authors conclude that SVD and Random col showed the best results.

2.2 Solution Uniqueness

Uniqueness is a common concern for the NMF problem. There are different ways one can structure decomposition matrices F and G to produce the same output $\widetilde{X} = FG^T$. In other words, if there exists a solution $X \approx F_0 G_0^T$, let $F = F_0 D$, $G^T = D^{-1} V_0$, then $X \approx FG^T$. Therefore, one can assign any invertible matrix to D such as to produce a range of equivalent solutions $X \approx F_0 D D^{-1} G_0$.

In practice, incorporating additional constraints such as sparseness (see section 2.4)

in the factor matrices or normalizing the columns of F (respectively rows of G^T) to unit length is helpful in alleviating this indeterminacy[5]. This is because adding constraints reduce the possible configurations the matrices can assume.

The work [26] attempts another solution for the uniqueness problem. To obtain rank-k factorizations, first k rank-1 factorizations are performed, resulting in $F_{m\times 1}$ and $G_{n\times 1}$. For each step, the vector F is normalized and the normalization constant is multiplied back to G. The residual error $E = X - FG^T$ becomes the input data for the next step.

2.3 Algorithm Implementation

2.3.1 Stopping Criteria

There are usually three different stopping criteria mostly adopted in NMF algorithms.[8]

- The objective function is reduced to below a given threshold.
- The change on the resulting matrices are not significant between iterations.
- The objective function decreases less than a given threshold between iterations.

2.3.2 Multiplicative Updates

Generally speaking, there are many methods for the resolution of matrix factorization problems. The most popular of them is the multiplicative update, proposed by Lee and Seung [33] for the NMF problem formulated in Eq. 1.1.

An additive update algorithm, such as the well-known gradient descent, would have the update rule for G shown in Eq. 2.1, where δ is the size of the step for each update, and $J(F, G)$ is the objective function. The method proposed by Lee and Seung is such that the step δ is rescaled in each iteration, as shown in Eq. 2.2. Thus, the update rule for G can be described as a multiplicative update as shown in Eq. 2.3. The same done for F results in a multiplicative update rule for F (Eq. 2.4). **Algorithm 1** shows the general form of a multiplicative update algorithm.

Algorithm 1 Multiplicative Updates

1: Initialize F and G.
2: Apply update rule for F, such as Eq. 2.3.
3: Apply update rule for G, such as Eq. 2.4.
4: Repeat steps 1 and 2 a fixed number of times or until the variation of the error norm is less than a tolerance value.

$$G^T \leftarrow G^T + \delta[\frac{\partial J}{\partial G}] \tag{2.1}$$

$$\delta = \frac{G^T}{F^T F G^T} \tag{2.2}$$

$$G^T \leftarrow G^T \frac{F^T X}{F^T F G^T} \tag{2.3}$$

$$F \leftarrow F \frac{XG}{FG^T G} \tag{2.4}$$

The main advantage of the multiplicative update rule is its simplicity of implementation, but there may be some more efficient algorithms. In [23], the author studies in detail the utilization of projected gradients, a bound-constrained gradient descent method for *Alternating Nonnegative Least Squares* and concludes that the multiplicative update method has a low cost per iteration, but the convergence is often slow, demanding a large number of iterations. The projected gradients technique, in contrast, converges faster, but the cost per iteration is higher.

2.3.3 Constrained Alternating Least Squares

The multiplicative update can be considered a special case of a more general approach called *block coordinate descent*, which consists of alternately fixing one block (matrix) and improving the other [23]. Instead of alternately improving each matrix, one can on each iteration find the best point, such as in the *Alternating Nonnegative Least Squares (ANLS)* [27] for NMF.

When the objective function is constrained (such as in NMF where both F and G must be non-negative and SNMF and WSNMF where G must be non-negative), one alternative is to use the *Alternating Least Squares (ALS)* scheme with bound-constrained gradient descent methods. Such a method is summarized as follows: Given a vector $\mathbf{x}$, an objective function $f(\mathbf{x})$ and a vector of lower and upper bounds $\mathbf{l}$ and $\mathbf{u}$, minimize $f(\mathbf{x})$ subject to $l_i \leq x_i \leq u_i$. For each step of the gradient descent, it guarantees that the solution remains within the lower and upper bounds. **Algorithm 2** shows the general form of a Constrained Alternating Least Squares algorithm.

Algorithm 2 Constrained Alternating Least Squares

1: Initialize F and G.
2: Fix G and find F that minimizes $J(F, G)$ subject to constraints.
3: Fix F and find G that minimizes $J(F, G)$ subject to constraints.
4: Repeat steps 1 and 2 a fixed number of times or until the variation of the error norm is less than a tolerance value.

Since SNMF and WSNMF have constraints only in matrix G, the following variation on **Algorithm 3** can be had.

Algorithm 3 Constrained Alternating Least Squares for SNMF and WSNMF

1: Initialize F and G.
2: Fix G and find F that minimizes $J(F, G)$.
3: Fix F and find G that minimizes $J(F, G)$ subject to $G \geq 0$.
4: Repeat steps 1 and 2 a fixed number of times or until the variation of the error norm is less than a tolerance value.

SNMF and WSNMF differ only by the objective function $J(F, G)$. SNMF objective function is Eq. 2.5 and WSNMF's is Eq. 2.6.

$$J(F, G) = \|X - FG^T\| \tag{2.5}$$

$$J(F, G) = \|W \odot (X - FG^T)\| \tag{2.6}$$

2.4 Sparseness

Sparseness is an often desired attribute of the result of a matrix factorization. NMF algorithms have been known to produce sparse low-dimensional representations of the input data. However, sometimes a higher control of the sparseness produced can result in a better representation. The sparseness constraint is helpful in improving the uniqueness of the decomposition and enforcing a local-based representation (See section 2.2). Sparse NMF is the most widely employed Constrained NMF problem, and is often a necessity to achieve better results [41].

If one interprets the left-hand-side matrix F as a set of k features and G as a matrix of coefficients, it is then desired that the input data be represented as a linear combination of just a few set of features. To achieve this, a sparseness constraint must be applied to the coefficients matrix G. Hoyer et al.[15] defined a metric for measuring the degree of sparseness of a vector, as shown in Eq. 2.8. Using this metric, a vector has maximum sparsity if and only if $\mathbf{x}$ contains only one non-empty element and minimum when all the components are the same. This measure is based on the relationship between the L_1 and L_2 norm of a vector.

This sparseness measure can be used to provide a sparseness constraint to the coefficient matrix G. Its sparseness measure would then be the mean of the sparseness of all its column vectors. If one wants to ensure a certain degree of sparseness to the matrix G, another variation of both multiplicative update and Constrained Alternating Least Squares algorithms can be produced. Such a technique is presented in [15], and involves the application of a projection operator to each column vector of the matrix G. The projection operator works by projecting the vector to L_2 unit norm and then adjusting L_1 norm to achieve the desired sparsity, according to the sparseness measure in Eq. 2.7.

$$sparseness(\mathbf{x}) = \frac{\sqrt{n} - (\sum |x_i|)/\sqrt{\sum x_i^2}}{\sqrt{n} - 1} \tag{2.7}$$

One can use this approach to implement a projected variant of both Multiplicative Updates and Constrained Alternating Least Squares algorithms. The resulting algorithms are shown on **Algorithm 4** and **Algorithm 5**.

$$sparseness(\mathbf{x}) = \frac{\sqrt{n} - (\sum |x_i|)/\sqrt{\sum x_i^2}}{\sqrt{n} - 1} \tag{2.8}$$

Hoyer et al. also developed a vector projection algorithm that projects a vector in $\mathbf{R_n}$ to the closest point in $\mathbf{R_n}$ satisfying a desired sparseness constraint, as shown in algorithm

Algorithm 4 Constrained Alternating Least Squares with Projection

1: Initialize F and G.
2: Fix G and find F that minimizes $J(F, G)$.
3: Fix F and find G that minimizes $J(F, G)$ subject to $G \geq 0$.
4: Project G such that it has the desired sparsity.
5: Repeat steps 1 and 2 a fixed number of times or until the variation of the error norm is less than a tolerance value.

Algorithm 5 Multiplicative Updates with Projection

1: Initialize F and G.
2: Apply update rule for F, such as Eq. 2.3.
3: Apply update rule for G, such as Eq. 2.4.
4: Project G such that it has the desired sparsity.
5: Repeat steps 1 and 2 a fixed number of times or until the variation of the error norm is less than a tolerance value.

6. Usually, L_2 remains the same and L_1 is defined such as to achieve the desired sparseness.

2.5 Dealing with Missing data

Missing data is a problem present in many real applications. For a variety of reasons, including error in the measuring process, non available data or invalid, missing values may appear in the input. One example of a field of research with many missing value problems is DNA microarray data analysis[37]. There are two main approaches that are commonly adopted: imputation and marginalization.

2.5.1 Marginalization

Marginalization is the simplest approach of them all, consisting of simply abandoning or ignoring the whole features of the data containing missing values. This approach is safe and conservative, but may be undesirable in some cases or even infeasible depending on the rate of missing values present in the data. Also, existing observed data in a feature is entirely discarded when one or more elements in the feature are missing, leading possibly to wasted relevant data.

2.5.2 Imputation

Imputation is also a common strategy to approach missing data, consisting of replacing the missing data with some reasonable estimate produced from the available data and known facts and properties of the data. This approach however, if not carefully executed, can generate biased and degraded data[19]. In the case of DNA microarray analysis, but also extensible to any missing data problem, [37] alerts that it is important to be cautious when drawing critical biological conclusions from data that is partially imputed.

Many ways to estimate values for imputation exist, many relying on previous assumptions about the data. One simple and often used imputation method is to take the mean

Algorithm 6 Given a vector $\mathbf{x}$, find the closest (in the euclidean sense) non-negative vector $\mathbf{s}$ with a given L_1 norm and a given L_2 norm

1: $s_i = x_i + (L_1 - \sum x_i)/dim(\mathbf{x}), \forall i$
2: $Z = \{\}$
3: **loop**
4: **if** $i \notin Z$ **then**
5: $m_i = L_1/dim(\mathbf{x}) - size(Z)$
6: **else**
7: $m_i = 0$
8: **end if**
9: $\mathbf{s} = \mathbf{m} + \alpha(\mathbf{s} - \mathbf{m})$, where α is such that the resulting $\mathbf{s}$ satisfies the L_2 norm constraint. This requires solving a quadratic equation.
10: **if** all the components of $\mathbf{s}$ are non-negative **then**
11: **return** $\mathbf{s}$
12: **end if**
13: $Z = Z \cup \{i; s_i < 0\}$
14: $s_i = 0, \forall i \in Z$
15: $c = (\sum s_i - L_1)/dim(\mathbf{x}) - size(Z))$
16: $s_i = s_i - c, \forall i \notin Z$
17: **end loop**
18: Repeat steps 1 and 2 a fixed number of times or until the variation of the error norm is less than a tolerance value.

or median of the elements in the same column.

2.5.3 Weights

In the case of matrix factorization, the introduction of a weight matrix to deal with the missing data is a much more robust approach towards missing data, presenting many advantages. We can set the missing elements in the weight matrix to zero, causing the missing elements to be ignored in the objective function minimization. This effectively ignores only the missing values, without discarding the whole feature vector as in the marginalization approach. Also, it avoids having to estimate the missing values and consequently biasing the results with artificially generated data and assumptions, as is the case for imputation methods. In fact, this approach can be used to actually produce estimations of the missing data, as is the case of some product recommendation algorithms[18].

Beyond missing data, one other major advantage of using a weight matrix is the ability to weight each entry according to some reliability measure. For instance, [35] notes that for gene expression analysis the error model provides entry-specific noise estimates. Setting the weights inversely proportional to the assumed noise variance can lead to a better reconstruction of the underlying structure.

Chapter 3

Minimization Error, Time and Sparseness Experiment

3.1 Experimental Methodology

A range of variations of algorithms that use constrained Alternating Least Squares technique are studied and compared with the multiplicative update algorithms, both for Semi-NMF (Eq. 1.8) problem and Weighted Semi-NMF (Eq. 1.9) problem. Variations of these algorithms which include projection to achieve a certain degree of sparseness are also implemented.

3.1.1 Semi-NMF Algorithms

$J(F,G) = \|X - FG^T\|$ is the objective function of the SNMF problem. Deriving partially for F gives Eq. 3.1. For the first step, one must find F such that $\frac{\delta J}{\delta F} = 0$. Solving this equation gives the analytical solution in Eq. 3.2. Either the analytical solution or or a gradient descent approach using Eq. 3.1 as the gradient function can be used for step 1. Preliminary studies showed that when the analytical function is too costly to compute, it may not perform so well as the gradient descent, therefore justifying the comparison between them.

$$\frac{\delta J}{\delta F} = FG^TG - XG \tag{3.1}$$

$$F = XG(G^TG)^{-1} \tag{3.2}$$

For the second step, $\frac{\delta J}{\delta G} = F^TFG^T - F^TX$. One can find an analytical solution when solving $\frac{\delta J}{\delta G} = 0$ for G, but the constraints of G is also a concern. Because of this, and also to keep the symmetry between SNMF and WSNMF comparisons, it was decided not to use the analytical solution for G in the experiments.

Three different algorithms for solving SNMF are implemented using constrained Alternating Least Squares. They use different bound-constrained gradient descent approaches from the Python Scipy optimization library. They are compared against each other and the multiplicative update algorithm from the Python Matrix Factorization Module

(PyMF), which I call M-SNMF (Multiplicative Update SNMF). The algorithms implemented for SNMF are the following:

- **TNC:** uses the Truncated Newton Conjugated Gradient (TNC) for both sides of the minimization problem (**step 1** and **step 2**). It is a constrained optimization method. On **step 1**, TNC is run without any constraints and on **step 2** it is run with non-negativity constraint. The Truncated Newton Conjugated Gradient is similar to the unconstrained Newton Conjugated Gradient, but never takes a step size large enough to leave the space of possible values.
- **L-BFGS-B:** is a variant of the L-BFGS (Limited Memory Broyden-Fletcher-Goldfarb-Shanno) that handles simple box constraints [4]. L-BFGS-B is applied to both sides of the minimization problem. As with the TNC case, unconstrained L-BFGS-B is applied on **step 1** and constrained L-BFGS-B on **step 2**.
- **Analytical:** takes advantage of the fact that the problem of minimizing F with G fixed has an analytical solution. This solution can the be used to solve the **step 1** of the alternating least squares algorithm. For **step 2**, the L-BFGS-B gradient descent algorithm is used.

3.1.2 Weighted Semi-NMF Algorithms

The objective function for WSNMF is $J(F,G) = \|W \odot (X - FG^T)\|$. Deriving it partially for F gives Eq. 3.3. For the first step, one must find F such as $\frac{\partial J}{\partial F} = 0$. As pointed in [34], this equation has an analytical solution, given by Eq. 3.4, where W_i is the diagonal matrix formed by the i_{th} row of W and X_i and F_i are column vectors formed by the i_{th} row of the matrix X and F, respectively.

$$\frac{\partial J}{\partial F} = 2(W \odot (FG^T - X)) \tag{3.3}$$

$$F_i = (G^T W_i G)^{-1} G^T W_i X_i \tag{3.4}$$

For step 1, the analytical solution given in Eq.3.4 or gradient descent can be used to minimize F. For the second step, deriving partially for G gives $\frac{\partial J}{\partial G} = 2(W^T \odot (GF^T - X^T))$. However, this equation does not have an analytical solution. Well-known techniques of gradient descent can be used to find a local optimal solution.

As in the SNMF problem, **TNC**, **L-BFGS-B** and **Analytical** solutions are implemented for SNMF. They are then compared against each other and the multiplicative update solution for WSNMF (M-WSNMF) proposed in [26]. M-WSNMF is shown in Algorithm 7.

3.1.3 Initialization

It is well known that most matrix factorization algorithms are very sensitive to initialization [1]. Since the problem is often non-convex, the starting point influences whether

Algorithm 7 Multiplicative Updates solving WSNMF (M-WSNMF)

1: Initialize F and G.
2: $R = ((W \otimes X)G) \oslash ((W \otimes FG^T)G)$
3: Apply update rule for F: $F = F \otimes R$.
4: $R_1 = ((W^T \otimes X^T)F)^+ + ((W^T \otimes GF^T)F)^-$
5: $R_2 = ((W^T \otimes X^T)F)^- + ((W^T \otimes GF^T)F)^+$
6: Apply update rule for G: $G \otimes R_1 \oslash R_2$.
7: Repeat steps 2 through 6 a fixed number of times or until the variation of the error norm is less than a tolerance value.

the algorithm will reach better or worse local minima. Therefore, several different initialization methods are tested: Random, Random Col, SVD Centroid, K-Means and Fuzzy C-Means. Except for the random initialization, all the other methods tested take into account the input data matrix. In the case where there are missing data, I impute the missing values using the mean of the values not missing in the column.

Random Initialization

Random initialization is a very common and simple form of initialization. It consists of setting the matrices elements to random values from a continuous uniform distribution from 0 to 1. It is low-cost, but tend to produce poorer results than more robust initialization methods.

Random col

Random Col method initializes the left-side matrix F to columns sampled randomly from the input data matrix [1]. The matrix G is initialized randomly as in the random initialization method. The presumption is that the input data matrix points give a good estimation for the cluster centroids, if we consider the analogy between matrix factorization and clustering. It can also be justified by assuming it is very likely that the input data and the solution will have a similar distribution.

SVD Centroid

SVD Centroid uses the output from a Truncated SVD Decomposition initializing F and G with the respective rank-k decompositions of the input data matrix. Since truncated SVD decomposition is one optimal solution for the unconstrained matrix factorization problem, it can be expected to yield a good initialization for the constrained problem.

K-Means

K-Means consists of running K-Means clustering on the input data matrix. The centroids found by K-Means are assigned to matrix F and the cluster membership matrix to the matrix G. The motivation behind this approach relies on the similarity between K-Means and matrix factorization problems. It is not uncommon to use K-Means as initialization for matrix factorization problems[13][42][44].

Fuzzy C-Means

Fuzzy C-Means is analogous to K-Means, the difference being that Fuzzy C-Means produces soft clusters while K-Means produces hard clusters.

3.1.4 Test cases

I run several tests varying the input data size, the proportion of missing data and the rank of the resulting matrices. The variables and their possible values are the following: Input Width (50, 100, 200, 500, 1000), Input Height (50, 100, 200, 500, 1000), Rank (2, 5, 10, 20, 30, 40), Weight (1, 0.75, 0.5, 0.3, 0.1). The permutation of the variables' values gives a total of 135 test cases for the weighted cases and 27 for the non-weighted cases. For each test case, I measure for each algorithm the overall time, the value of the error and the sparsity of the matrix G (when applicable).

Given a rank k, I generate the input matrix $X_{m\times n}$ by multiplying two random matrix F_{mxk} and $G_{k\times n}$. The matrix F is sampled from an uniform distribution from -1 to 1 and the matrix G is sampled from an uniform distribution from 0 to 1. This ensures that the produced input matrix X is clearly separable with zero error for the given rank k.

I execute the tests with different batches of algorithms variations, as follows:

1. SNMF without sparseness projection
2. SNMF with 0.5 sparseness projection
3. SNMF with 0.9 sparseness projection
4. WSNMF without sparseness projection
5. WSNMF with 0.5 sparseness projection
6. WSNMF with 0.9 sparseness projection

3.2 Experiment Results

3.2.1 Initialization Results

On each test case, the algorithms were ranked from 0 to $n-1$ integer values; 0 for the algorithm showing the best result and $n-1$ for the worst, n being the number of different algorithms on the test. This was done individually for each measure (Error, Time and Sparsity). On table 3.1 through 3.3 I present the results from the initialization tests. The time measured was the time of execution of each algorithm without initialization. That is relevant because sometimes initialization provides a better starting point and reduces the amount of work necessary to reach a local minima. I have also found that taking initialization time into account does not significantly change the time rankings of each algorithm. It should also be noted that in the case of sparsity, a higher ranking means higher sparsity measure. Since higher sparsity is desired, the higher the sparsity ranking, the better.

The initialization tests show that Random col method produced smaller errors than the other methods. It performs not so well in the time and sparsity comparison, though. SVD Centroid produced the worse results in terms of error reduction, but this is because applying SVD Centroid initialization in some cases resulted in a crash of the M-SNMF algorithm, when attempting to calculate a negative square root.

For SNMF, Fuzzy C-Means produced the best results for the error ranking. In both cases, random and random col performed very similar to actual random initialization. This might be explained by the fact that the distribution of the input data is also random. Therefore, generating random initialization and taking random columns from the randomly generated data did not produce much difference. Experiments with real applications suggest, however, that random col is generally better than simple random initialization[20].

For WSNMF, K-Means was clearly the best performing for both error and sparseness. It should be noted, however, that since M-WSNMF is a multiplicative update algorithm, when K-Means initializes many elements in the G matrix initialized to zero, these values remain zero. This is why the sparsity of M-SNMF and M-WSNMF with K-Means is always maximum.

Initialization	**Error Rank**	**Time Rank**	**Sparsity Rank**
Fuzzy C-Means	2.80	1.62	1.12
Random col	**0.53**	2.65	2.30
SVD Centroid	3.96	1.7	0.24
Random	0.55	2.84	2.34
K-Means	2.57	**0.0**	**4.0**

Table 3.1: Comparison between different initialization methods for M-SNMF

Initialization	**Error Rank**	**Time Rank**	**Sparsity Rank**
Fuzzy C-Means	**0.57**	3.47	0.19
Random col	2.43	1.92	1.77
SVD Centroid	1.39	1.6	**3.38**
Random	2.40	1.9	1.77
K-Means	3.2	**1.1**	2.9

Table 3.2: Comparison between different initialization methods for SNMF

Initialization	Error Rank	Time Rank	Sparsity Rank
Fuzzy C-Means	1.95	**0.82**	0.43
Random col	2.29	2.91	2.24
SVD Centroid	2.96	2.47	1.1
Random	2.2	2.94	2.23
K-Means	**0.59**	0.85	**4.0**

Table 3.3: Comparison between different initialization methods for M-WSNMF

Initialization	Error Rank	Time Rank	Sparsity Rank
Fuzzy C-Means	2.73	2.55	2.03
Random col	1.37	2.17	1.85
SVD Centroid	**0.87**	**1.28**	1.6
Random	3.74	1.84	0.96
K-Means	**0.69**	2.56	**3.55**

Table 3.4: Comparison between different initialization methods for WSNMF

3.2.2 Algorithm Comparison Results

Based on the initialization test results, I decided the best performing initialization for each algorithm group. I took the error rank as the decisive factor when choosing an initialization method over another. For WSNMF, **K-Means** was clearly the best initialization option, but the choice for SNMF was not so clear. I chose **Random col** for SNMF since it was the best option for M-SNMF and performed adequately well also for SNMF.

As in the initialization tests, I measure the mean error and mean time rankings from all the test runs and show the mean of the rankings obtained by each algorithm. I also measure mean sparsity rankings for the cases without sparseness projection. For the other cases, the projection enforces a certain degree of sparsity and all the algorithms end up with the same sparsity. I measured the time including initialization time and excluding initialization time, but since I ended up using the same initialization for each test group, the initialization did not affect the time rank. The rankings are shown on Table 3.5 through 3.8.

Graphs showing the rank difference as input size, rank and weight varies are in the Appendix A.

Algorithm	Error Rank	Time Rank	Sparsity Rank
Analytical	1.81	**0.27**	1.88
TNC	2.1	2.93	**2.13**
L-BFGS-B	1.87	1.97	1.24
M-SNMF	**0.22**	1.55	0

Table 3.5: Mean rank comparison between all algorithms solving SNMF

Algorithm	Error Rank	Time Rank
Analytical	1.55	**0.28**
TNC	1.71	2.56
L-BFGS-B	1.5	1.05
M-SNMF	**1.24**	2.09

Table 3.6: Mean rank comparison between all algorithms solving SNMF with 0.5 sparsity projection

Algorithm	Error Rank	Time Rank
Analytical	**1.12**	**0.35**
TNC	1.23	2.57
L-BFGS-B	**1.11**	1.03
M-SNMF	2.54	2.04

Table 3.7: Mean rank comparison between all algorithms solving SNMF with 0.9 sparsity projection

Algorithm	Error Rank	Time Rank	Sparsity Rank
Analytical	**0.49**	2.05	0.65
TNC	0.53	2.92	0.37
L-BFGS-B	2.51	1.03	1.99
M-WSNMF	2.47	**0.0**	**2.99**

Table 3.8: Mean rank comparison between all algorithms solving WSNMF

Algorithm	Error Rank	Time Rank
Analytical	**0.23**	1.47
TNC	0.87	2.85
L-BFGS-B	2.88	0.52
M-WSNMF	2.01	**0.39**

Table 3.9: Mean rank comparison between all algorithms solving WSNMF with 0.5 sparsity projection

Algorithm	**Error Rank**	**Time Rank**
Analytical	**0.45**	1.49
TNC	1.16	2.89
L-BFGS-B	2.40	**0.43**
M-WSNMF	1.99	1.19

Table 3.10: Mean rank comparison between all algorithms solving WSNMF with 0.9 sparsity projection

3.2.3 Friedman Test and Post-hoc Wilcoxon Test

The results were verified with the Friedman Test, a non-parametrical statistical test to detect differences in treatments across multiple test attempts. For each algorithm batch, I applied the Friedman test to determine whether there were any statistically relevant difference between all the algorithms tested. Since there were statistically significant differences detected in all batches, I then applied the post-hoc Wilcoxon test to each pair of algorithms to determine whether they produced statistically relevant differences from one another. The P-Values with values less than 0.005 on tables 3.11 through 3.16 show that the differences in rank obtained from the respective experiments are significant and not produced by chance.

Algorithms	Error P-Value	Time P-Value	Sparsity P-Value
L-BFGS-B & Analytical	0.9	0	0.34
L-BFGS-B & TNC	0.02	0	0.14
L-BFGS-B & M-SNMF	0	0	0
Analytical & TNC	0.002	0	0.009
Analytical & M-SNMF	0	0	0
TNC & M-SNMF	0	0	0

Table 3.11: P-Values of the Pairwise Wilcoxon Test for the algorithms solving SNMF without sparsity projection

Algorithms	Error P-Value	Time P-Value
L-BFGS-B & Analytical	0.326	0
L-BFGS-B & TNC	0.317	0
L-BFGS-B & M-SNMF	0.024	0
Analytical & TNC	0.724	0
Analytical & M-SNMF	0.024	0
TNC & M-SNMF	0.024	0.002

Table 3.12: P-Values of the Pairwise Wilcoxon Test for the algorithms solving SNMF with 0.5 sparsity projection

Algorithms	Error P-Value	Time P-Value
L-BFGS-B & Analytical	0.277	0
L-BFGS-B & TNC	0.895	0
L-BFGS-B & M-SNMF	0	0
Analytical & TNC	0.811	0
Analytical & M-SNMF	0	0
TNC & M-SNMF	0	0

Table 3.13: P-Values of the Pairwise Wilcoxon Test for the algorithms solving SNMF with 0.9 sparsity projection

Algorithms	Error P-Value	Time P-Value	Sparsity P-Value
M-WSNMF & L-BFGS-B	0.034	0	0
M-WSNMF & Analytical	0	0	0
M-WSNMF & TNC	0	0	0
L-BFGS-B & Analytical	0	0	0
L-BFGS-B & TNC	0	0	0
Analytical & TNC	0	0	0

Table 3.14: P-Values of the Pairwise Wilcoxon Test for the algorithms solving WSNMF

Algorithms	Error P-Value	Time P-Value
M-WSNMF & L-BFGS-B	0	0.001
M-WSNMF & Analytical	0	0
M-WSNMF & TNC	0	0
L-BFGS-B & Analytical	0	0
L-BFGS-B & TNC	0	0
Analytical & TNC	0	0

Table 3.15: P-Values of the Pairwise Wilcoxon Test for the algorithms solving WSNMF with 0.5 sparsity projection

Algorithms	Error P-Value	Time P-Value
M-WSNMF & L-BFGS-B	0.002	0
M-WSNMF & Analytical	0	0
M-WSNMF & TNC	0	0
L-BFGS-B & Analytical	0	0
L-BFGS-B & TNC	0	0
Analytical & TNC	0	0

Table 3.16: P-Values of the Pairwise Wilcoxon Test for the algorithms solving WSNMF with 0.9 sparsity projection

3.2.4 Discussion of the results

For the SNMF without sparsity projection batch, M-SNMF is shown to have the best error rank. The statistical analysis on Table 3.11 shows that the the difference in rank between M-SNMF and the other alternatives is significant, but there was not much difference detected between Analytical and L-BFGS-B. With respect to time, the analytical solution outperformed the others and the difference is clearly significant. The M-SNMF solution had the second best time performance. With respect to sparsity, M-SNMF clearly performed worse than the algorithms using Constrained Alternating Least Squares, which had similar sparsity mean scores. From that I conclude that Constrained Alternating Least Squares tends to produce sparser results than Multiplicative Updates for SNMF.

For the SNMF with 0.5 sparsity projection batch, the multiplicative update algorithm (M-SNMF) performed better than the others with respect to the Error Rank. With respect to time, the analytical solution had the best results. But with a greater sparsity projection being enforced, performance of M-SNMF degraded and the other algorithms performed better, but not very different from one another. It can be seen from the statistical tests that only the comparison between M-SNMF and the others were shown to be significant. It is reasonable to conclude that when a high degree of sparsity is enforced, Constrained Alternating Least Squares algorithm often converge to a common local optima. As for M-SNMF, the worse performance can be explained by the fact that the projection to a high degree of sparseness may force many coefficients of G to go to zero. Since it is a multiplicative update algorithm, elements reaching zero tend to remain zero, diminishing the available optimization options and thus being unable to reach the same local optima reached by the other algorithms. In the graph at page 57 of the Appendix A, it can be seen how the error performance for M-SNMF deteriorates as the input size increases, in the case of high sparseness projection.

For the WSNMF without projection batch, as in the non-weighted case, the Analytical also had the best error rank, though very close to TNC. The results mirror the non-weighted case (SNMF without sparsity projection). Since there were more test cases for the weighted case, except for the difference between the error rank in Analytical and TNC algorithms, all differences in rank are shown to be significant.

For the WSNMF with 0.5 and 0.9 sparsity projection, all the differences in rank are shown to be significant. The Analytical solution clearly outperforms the others in error rank. Table 3.9 (WSNMF with 0.5 sparsity projection) and Table 3.10 (WSNMF with 0.9 sparsity projection) are similar, except from the shift of position between L-BFGS-B and M-WSNMF in the Time Rank. With 0.5 sparsity projection, M-WSNMF is faster than L-BFGS-B and with 0.9 sparsity projection, L-BFGS-B is faster.

It is also important to notice that there is a time performance drop for the M-SNMF algorithm between without sparseness and with sparseness projection. This is because the multiplicative update algorithm takes much more iterations to converge (around 100 iterations) than Constrained Alternating Least Squares (2 to 4 iterations). Since projection is performed at the end of each iteration, the time cost for the projection is much higher for M-SNMF.

Examining the graphs on the Appendix A, one can see some interesting tendencies.

In the graph at page 51, measuring time versus input size for SNMF without sparseness projection, it can be seen that the performance of the analytical solution gets increasingly better than the others, showing that the analytical solution scales better for the SNMF problem. The same is not through for WSNMF. Adding weights to the algorithms makes Constrained Alternating Least Squares more complex than the multiplicative update algorithm equivalent. It can be seen from the graphs at pages 61, 65 and 70 that M-SNMF scales better as input size increases.

Chapter 4

Clustering Experiments

4.1 Experimental Methodology

For the clustering experiments, I created a synthetic data set from a mixture of gaussians, following the methodology in [17]. For each target number of clusters k, I constructed a data set of 1000 elements in 500 dimension as follows:

1. Construct mean vectors $m_1, m_2, ..., m_k \in \mathbf{R}^{500\times 1}$. For each row index $i = 1, 2, ..., 500$,
   - Randomly pick an index q from $\{1, 2, ..., k\}$ and d from $\{1, 2, 3\}$
   - set $m_q(i) = d$ and $m_j(i) = 0$ for all $j \neq q$ where $m_j(i)$ is i-th element of m_j
2. Then, for each $j = 1, 2, ..., k$, set the covariance matrix $Cov_j \in \mathbf{R}^{500\times 500}$ as $Cov_j(i, i) = 0.3$ if $m_j(i) \neq 0$ and $Cov_j(., .) = 0$ for all others.
3. Generate mixture of gaussians from $m_1, m_2, ..., m_k$ and $Cov_1, Cov_2, ..., Covk$ with balanced weights.

The first step in the algorithm generates k mean vectors where for each of the 500 dimensions, only one of the vectors will have a non-zero value assuming a magnitude of 1, 2 or 3 (chosen at random). Each mean vector represents the center of a multivariate gaussian. The second step generates a covariance matrix for each of the k mean vectors. This covariance matrix is a 500×500 diagonal matrix with zeroed elements along the diagonal whenever the corresponding dimension is zero in the mean vector and 0.3 otherwise. This means that each gaussian mixture generates elements only along the axis which are not zero in its corresponding mean vector, and since each non zero entry in the mean vector is guaranteed to be zero in the other mean vectors, we ensure that there is no intersection between the gaussians mixtures.

This approach creates a clearly separable high dimensional data set. Because the data set was created in a way that the clusters are clearly separated, the optimal clustering assignments can be considered as ground truth. Data sets were generated for $k = 2, 3, ..., 8$. For each k I did 100 trials and recorded how many times the algorithms achieved 100% accuracy. The algorithms were also tested with different percentages of missing values. I tested five different algorithm variations: WSNMF, M-WSNMF, SNMF and M-SNMF

with sparseness enforced and K-Means. In all the matrix factorization based methods, a sparseness degree of 0.5 was enforced. For each case, the algorithms that achieved the best minimization error from the previous tests were chosen.

For each k and each of the 100 trials, the following was done:

1. Generate artificial high-dimensional data set
2. Create random weight matrix with 0, 25%, 50% and 70% of zeroes and the corresponded amount of ones.
3. For each weight matrix, run the five algorithms. When they support weighting(M-WSNMF and WSNMF), use the weight matrix directly. When they do not, impute the missing data with the mean of the corresponding column.
4. Measure accuracy of each algorithm.

For WSNMF, I applied the analytical optimization on the left-hand side and L-BFGS-B on the right-hand-side, as in Algorithm 8. M-WSNMF was run as in Algorithm 9.

Algorithm 8 WSNMF with sparseness projection

1: Initialize F and G using K-Means.
2: Fix G and find $F_i = (G^T W_i G)^{-1} G^T W_i X_i$ for each row i.
3: Fix F and find G using L-BFGS-B gradient descent subject to non-negativity constraints.
4: Project G such that it has sparseness equal to 0.5.
5: Repeat steps 2 and 3, 4 times or until the variation of the error norm stabilizes.

Algorithm 9 Multiplicative Updates solving WSNMF (M-WSNMF)

1: Initialize F and G using K-Means.
2: $R = ((W \otimes X)G) \oslash ((W \otimes FG^T)G)$
3: Apply update rule for F: $F = F \otimes R$.
4: $R_1 = ((W^T \otimes X^T)F)^+ + ((W^T \otimes GF^T)F)^-$
5: $R_2 = ((W^T \otimes X^T)F)^- + ((W^T \otimes GF^T)F)^+$
6: Apply update rule for G: $G \otimes R_1 \oslash R_2$.
7: Project G such that it has sparseness equal to 0.5.
8: Repeat steps 2 through 6 100 times or until the variation of the error norm stabilizes.

The matrix $G_{n \times k}$ represents the cluster assignments. Since matrix factorization algorithms generate soft clustering assignments, the i^{th} feature is assigned to cluster j if $argmax(G_i) = j$, where G_i is the i^{th} row of matrix G.

4.2 Experiment Results and Discussion

The results of the clustering experiment are shown from Table 4.1 through 4.7. For each cluster $k = 2, 3...8$ and each missing rate (0, 0.25, 0.5 0.7) I measure the score (number of times the algorithms got 100% accuracy) and the mean accuracy from all the 100 runs.

Missing (%)	0		0.25		0.5		0.7	
Method	**Acc.**	**Score**	**Acc.**	**Score**	**Acc.**	**Score**	**Acc.**	**Score**
K-Means	1.00	100	1.00	100	1.00	100	1.00	100
M-WSNMF	1.00	100	1.00	100	1.00	100	1.00	100
M-SNMF	0.54	0	0.55	0	0.55	0	0.55	0
SNMF	1.00	100	1.00	100	1.00	100	1.00	100
WSNMF	1.00	100	1.00	100	1.00	100	1.00	100

Table 4.1: Number of clusters K=2

Missing (%)	0		0.25		0.5		0.7	
Method	**Acc.**	**Score**	**Acc.**	**Score**	**Acc.**	**Score**	**Acc.**	**Score**
K-Means	0.83	61	0.85	65	0.88	71	0.90	77
M-WSNMF	0.81	57	0.89	74	0.88	71	0.88	72
M-SNMF	0.70	24	0.75	26	0.72	28	0.76	27
SNMF	0.86	58	0.88	65	0.86	59	0.89	66
WSNMF	0.87	65	0.89	68	0.90	70	0.92	77

Table 4.2: Number of clusters K=3

Missing (%)	0		0.25		0.5		0.7	
Method	**Acc.**	**Score**	**Acc.**	**Score**	**Acc.**	**Score**	**Acc.**	**Score**
K-Means	0.76	32	0.83	50	0.82	46	0.80	43
M-WSNMF	0.74	33	0.80	49	0.82	51	0.77	41
M-SNMF	0.85	40	0.88	48	0.87	35	0.88	52
SNMF	0.84	38	0.84	40	0.83	35	0.83	37
WSNMF	0.78	33	0.85	49	0.86	49	0.86	48

Table 4.3: Number of clusters K=4

Missing (%)	0		0.25		0.5		0.7	
Method	**Acc.**	**Score**	**Acc.**	**Score**	**Acc.**	**Score**	**Acc.**	**Score**
K-Means	0.72	19	0.76	28	0.76	25	0.74	25
M-WSNMF	0.68	16	0.71	20	0.73	20	0.74	26
M-SNMF	0.84	23	0.86	29	0.84	17	0.85	23
SNMF	0.83	26	0.81	21	0.77	9	0.80	15
WSNMF	0.74	19	0.81	26	0.83	32	0.83	27

Table 4.4: Number of clusters K=5

Missing (%)	0		0.25		0.5		0.7	
Method	**Acc.**	**Score**	**Acc.**	**Score**	**Acc.**	**Score**	**Acc.**	**Score**
K-Means	0.69	13	0.69	11	0.68	8	0.73	17
M-WSNMF	0.64	7	0.70	12	0.70	14	0.69	13
M-SNMF	0.86	14	0.84	9	0.83	13	0.86	15
SNMF	0.81	10	0.80	10	0.81	14	0.80	8
WSNMF	0.74	10	0.78	19	0.76	13	0.80	16

Table 4.5: Number of clusters K=6

Missing (%)	0		0.25		0.5		0.7	
Method	**Acc.**	**Score**	**Acc.**	**Score**	**Acc.**	**Score**	**Acc.**	**Score**
K-Means	0.65	5	0.69	6	0.68	10	0.68	5
M-WSNMF	0.63	3	0.65	2	0.68	7	0.67	6
M-SNMF	0.81	7	0.83	8	0.82	3	0.83	10
SNMF	0.80	3	0.81	12	0.79	5	0.80	6
WSNMF	0.71	3	0.77	6	0.77	7	0.75	7

Table 4.6: Number of clusters K=7

Missing (%)	0		0.25		0.5		0.7	
Method	**Acc.**	**Score**	**Acc.**	**Score**	**Acc.**	**Score**	**Acc.**	**Score**
K-Means	0.65	0	0.66	4	0.65	3	0.67	2
M-WSNMF	0.62	2	0.64	4	0.65	3	0.65	1
M-SNMF	0.81	5	0.81	6	0.82	7	0.81	1
SNMF	0.79	1	0.78	7	0.79	5	0.78	3
WSNMF	0.71	1	0.73	0	0.74	4	0.75	3

Table 4.7: Number of clusters K=8

The most notable aspect of the results is that M-SNMF was clearly superior than the others for $k > 3$(Table 4.3-4.7) in terms of mean accuracy. For $k = 2$(Table 4.1), however, M-SNMF was the only algorithm that didn't get 100% accuracy on all test runs. In terms of score, however, the result was mixed. K-Means scored in most cases worse than the best performing matrix factorization algorithm.

Between the two algorithms using weights, the multiplicative update version (M-WSNMF) and the constrained alternate least squares version (WSNMF), WSNMF showed superior performance on the vast majority of the tests. For the versions that do not use weights, however, the multiplicative update algorithm (M-SNMF) was better than SNMF.

One curious phenomenon that was noticed in this experiment was that in many cases, when the percentage of missing data increased, performance of the algorithms in terms of mean accuracy and score also increased. One hint towards explaining this phenomenon

might be the peculiarity of the artificial dataset generated, which guaranteed clearly separable clusters in a high-dimensional space. On the other hand, generating artificial datasets with ambiguous clusters also has its problems, as there is no ground truth for determining accuracy and it would be necessary to resort to other metrics for measuring cluster quality. As noted in Section 1.2.1, there is no unique way of measuring cluster quality and the best measure is ultimately the suitability to a given problem.

4.2.1 Statistical Test

As with the experiment in Chapter 3, I applied the Friedman test for the clustering experiment, to check whether the experimental data was sufficient to determine a statistical significance between the different methods. The results of the Friedman test in table 4.8 show the resulting P-value for each set of number of clusters and missing rate. Unfortunately, all P-values were above 0.05, indicating that there might not be sufficient data to be certain of a conclusion.

K	Missing Rate	Accuracy P-Value
2	0	0.48
2	0.25	0.48
2	0.5	0.48
2	0.7	0.48
3	0	0.94
3	0.25	0.64
3	0.5	0.86
3	0.7	0.11
4	0	0.82
4	0.25	0.87
4	0.5	0.42
4	0.7	0.45
5	0	0.14
5	0.25	0.48
5	0.5	0.92
5	0.7	0.23
6	0	0.12
6	0.25	0.43
6	0.5	0.09
6	0.7	0.86
7	0	0.31
7	0.25	0.74
7	0.5	0.26
7	0.7	0.67
8	0	0.34
8	0.25	0.32
8	0.5	0.39
8	0.7	0.52

Table 4.8: Results of Friedman test for the clustering experiment

4.2.2 Comparison with other works

I have found no other works that perform quite the same algorithm comparison. However, the method for generating the artificial data set was taken from the work "Sparse Nonnegative Matrix Factorization for Clustering"[17], in which the authors compare K-Means with Sparse NMF and Sparse SNMF algorithms implemented with Alternating Non-negative Least Squares. However, they did not test for different missing rates. In this work, the enforcing of sparseness is done by adding regularization parameters in the objective function, as in Eq. 1.3 from the introduction.

The work tested K-Means, Sparse NMF and Sparse SNMF for a number of clusters varying from 3 to 30 and measured only the number of times each algorithm achieved

perfect accuracy, but not the mean accuracy. The scores obtained for number of clusters varying from 3 to 8 are on table 4.9. Similar to my results, performance of K-Means dropped rapidly as the number of clusters increased. However, the results for Sparse NMF and Sparse SNMF had much better scores than the ones I got for similar algorithms. The results may differ because the algorithms are in fact different and regularization is a better approach to ensure sparseness than sparseness projection in this case. This hypothesis needs to be properly tested, though. Both works indicate, however, that matrix factorization algorithms with sparseness constraints are more fitted for high dimensional data clustering than K-Means.

K	3	4	5	6	7	8
K-Means	53	37	13	3	4	1
NMF	69	62	66	65	72	76
SNMF	100	100	100	100	100	100

Table 4.9: Score results of the related work experiment for cluster number 3 to 8

Chapter 5

Conclusions

5.1 Minimization Error, Time and Sparseness Experiment

Several conclusions can be drawn from the results of the first experiment. Concerning SNMF, I found that the multiplicative update wins over the multiplicative update alternative when no sparseness projection takes place and also when a sparseness projection of 0.5 is enforced, but the multiplicative update algorithm performance deteriorates as the desired sparseness gets higher.

On the WSNMF cases, the Analytical solution consistently performed better that the alternatives in terms of the minimization of the error. Restricting sparseness, in fact, widened the difference between the other algorithms. The Multiplicative Update algorithm, however, had a much better time performance. Also, I found that K-means seems to be a good choice of initialization for WSNMF.

This empirical study suggests that the choice between *Multiplicative Updates* and Constrained *Alternating Least Squares* for solving NMF problems should be carefully considered. The results suggest that for SNMF, when low to medium sparseness is required by the application, it is preferable to choose M-SNMF over Constrained Alternating Least Squares algorithm. For high degree of sparseness, the Analytical version of Constrained Alternating Least Squares is preferred. When time is critical, the analytical version should be preferred. For WSNMF applications in which time is not essential, constrained *Alternating Least Squares* methods should be more appropriate to be used over *Multiplicative Update* algorithms. Also, using an analytical solution when available is shown to produce smaller error results for both SNMF and WSNMF.

5.2 Clustering Experiment

Regarding the clustering experiment, it can be concluded, conforming to the literature on the subject, that matrix factorization algorithms with sparseness constraints outperform K-Means clustering, while also remaining much more versatile.

The M-SNMF algorithm turned out to be very robust with respect to the number of clusters k. This agrees with the previous experiment that shows that M-SNMF also

produced the smallest error ranking when sparseness was enforced. Having lower error in the objective function does not necessarily mean that the algorithm will perform well for clustering, since objective function alone is not a good indicative of clustering performance (Section 1.2.1). But as seen in Section 2.2, even though sparseness constraints (Section 2.4) do not contribute towards error minimization of the objective function, they help in alleviating the uniqueness problem, reaching a factorization that is more sparse, and better structured for a clustering solution.

The algorithms using weights overall did not perform better than their weightless counterparts when missing values were present. Imputation worked out better for this experiment than the addition of the weights. This may be due to the peculiarities of the artificial data set chosen. One should note, however, that the addition of weights is not useful only for modelling missing values, but also for providing different confidence values for the data or modelling error in the data measurement. This could hardly be achieved by using imputation methods.

5.3 Wrapping up

From both experiments it can be seen that multiplicative update algorithm for SNMF (M-SNMF) performed better in comparison with the other SNMF algorithms and also achieved better performance in the clustering experiment. It can be seen from tables 3.7 that the analytical SNMF solution had the smallest error ranking (tied with L-BFGS-B) when sparseness of 0.9 was enforced and had best time performance overall, while M-SNMF had the best error ranking for low and medium sparseness projection ((Tables 3.5 and 3.6) and average time performance.

It can be seen that time performance of M-SNMF (and M-WSNMF) worsen when sparseness projection is applied. This happens because multiplicative updates algorithm needs much more iterations to converge and sparseness projection is applied at the end of each iteration. It can be conjectured that taking smaller steps toward minimizing the objective function and then projecting at the end of each step leads to better local optima solutions. However, more tests are necessary to reach this conclusion.

Regarding the comparison between the algorithms solving WSNMF, the constrained alternating least squares analytical implementation performs better than the multiplicative update on both experiments. From the experiments alone, it is hard to infer why this happens. One possible explanation could be the fact that multiplicative update algorithm have weak convergence properties, while constrained alternating least squares converge really fast.

The literature often states that constrained alternating least squares is more time efficient and has better convergence properties, while multiplicative updates are simple to implement. This might be an oversimplification. Being a difficult problem (NP-Hard) and having many different possible variations, parameters and heuristics, it is not always the case that one method is better than the other, as can be seen from the experiments that were made.

5.4 Future Work

As shown in the bibliographic research and related works(Chapter 2), matrix factorization methods are used in many different applications from distinct fields. In particular, the addition of weights to model uncertainty, missing data or confidence levels in the objective function of a factorization has shown to be useful in real applications. Also, the possibility to deal seamlessly with negative input data, as opposed to regular NMF, has also shown to be very valuable.

Therefore, I believe the WSNMF formulation could be used successfully to model different kinds of problems, but this remains to be tested. A possible direction of future work would be to improve the overall performance of the WSNMF algorithms and test it in applications that are very sensitive to weights and have negative values as input, such as product recommendation or gene expression analysis.

Bibliography

[1] Russell Albright, James Cox, David Duling, Amy N Langville, and C Meyer. Algorithms, initializations, and convergence for the nonnegative matrix factorization. Technical report, Tech. rep. 919. NCSU Technical Report Math 81706. http://meyer. math. ncsu. edu/Meyer/Abstracts/Publications. html, 2006. url: http://citeseerx. ist. psu. edu/viewdoc/download, 2006.

[2] C. Boutsidis and E. Gallopoulos. SVD based initialization: A head start for nonnegative matrix factorization. *Pattern Recognition*, 41(4):1350–1362, April 2008.

[3] J.P. Brunet, P. Tamayo, T.R. Golub, and J.P. Mesirov. Metagenes and molecular pattern discovery using matrix factorization. *Proceedings of the National Academy of Sciences*, 101(12):4164–4169, 2004.

[4] Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A Limited Memory Algorithm for Bound Constrained Optimization, 1995.

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

[6] M. Cooper and J. Foote. Summarizing video using non-negative similarity matrix factorization. In *Multimedia Signal Processing, 2002 IEEE Workshop on*, pages 25–28. IEEE, 2002.

[7] C. Ding, X. He, and H.D. Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *Proc. SIAM Data Mining Conf*, pages 606–610, 2005.

[8] Chris Ding. Nonnegative matrix factorizations for clustering: A survey. *Data Clustering: Algorithms and Applications*, page 148, 2013.

[9] Chris H. Q. Ding, Tao Li, and Michael I. Jordan. Convex and semi-nonnegative matrix factorizations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(1):45–55, 2010.

[10] Konstantinos Drakakis, Scott Rickard, Ruairí De Fréin, and Andrzej Cichocki. Analysis of financial data using non-negative matrix factorization. In *International Mathematical Forum*, volume 3, pages 1853–1870. Journals of Hikari Ltd, 2008.

[11] Nicolas Gillis and François Glineur. Low-Rank Matrix Approximation with Weights or Missing Data Is NP-Hard. *SIAM Journal on Matrix Analysis and Applications*, 32(4):1149–1165, October 2011.

[12] Nicolas Gillis and Abhishek Kumar. Exact and Heuristic Algorithms for Semi-Nonnegative Matrix Factorization. page 18, October 2014.

[13] Liyun Gong and Asoke K. Nandi. An enhanced initialization method for non-negative matrix factorization. In *2013 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, September 2013.

[14] ND Ho, P Van Dooren, and V Blondel. Weighted nonnegative matrix factorization and face feature extraction. *submitted to Image and Vision Computing*, 2007.

[15] P.O. Hoyer. Non-negative matrix factorization with sparseness constraints. *The Journal of Machine Learning Research*, 5:1457–1469, 2004.

[16] Andreas Janecek and Ying Tan. *Using Population Based Algorithms for Initializing Nonnegative Matrix Factorization*, volume 6729 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[17] Jingu Kim and Haesun Park. Sparse nonnegative matrix factorization for clustering. Technical report, 2008.

[18] Yong-Deok Kim and Seungjin Choi. Weighted nonnegative matrix factorization. *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 1541–1544, April 2009.

[19] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37, August 2009.

[20] Amy N Langville, Carl D Meyer, Russell Albright, James Cox, and David Duling. Initializations for the nonnegative matrix factorization. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 23–26. Citeseer, 2006.

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

[22] Stan Z Li, XinWen Hou, HongJiang Zhang, and QianSheng Cheng. Learning spatially localized, parts-based representation. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–207. IEEE, 2001.

[23] C.J. Lin. Projected gradient methods for nonnegative matrix factorization. *Neural computation*, 19(10):2756–2779, 2007.

[24] W.-S. Lu, S.-C. Pei, and P.-H. Wang. Weighted low-rank approximation of general complex matrices and its application in the design of 2-D digital filters. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 44(7):650–655, July 1997.

[25] Ivan Markovsky and Mahesan Niranjan. Approximate low-rank factorization with structured factors. *Computational Statistics & Data Analysis*, 54(12):3411–3420, December 2010.

[26] Quanyi Mo and Bruce A. Draper. Semi-nonnegative matrix factorization for motion segmentation with missing data. In Andrew W. Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *ECCV (7)*, volume 7578 of *Lecture Notes in Computer Science*, pages 402–415. Springer, 2012.

[27] P. Paatero and U. Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 2006.

[28] Sun Park, Ju-Hong Lee, Deok-Hwan Kim, and Chan-Min Ahn. Multi-document summarization based on cluster using non-negative matrix factorization. In *SOFSEM 2007: Theory and Practice of Computer Science*, pages 761–770. Springer, 2007.

[29] Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD cup and workshop*, volume 2007, pages 5–8, 2007.

[30] V Paul Pauca, Farial Shahnaz, Michael W Berry, and Robert J Plemmons. Text mining using non-negative matrix factorizations. In *SDM*, volume 4, pages 452–456, 2004.

[31] Paul Sajda, Shuyan Du, Truman R Brown, Radka Stoyanova, Dikoma C Shungu, Xiangling Mao, and Lucas C Parra. Nonnegative matrix factorization for rapid recovery of constituent spectra in magnetic resonance chemical shift imaging of the brain. *Medical Imaging, IEEE Transactions on*, 23(12):1453–1465, 2004.

[32] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl. Application of dimensionality reduction in recommender system – a case study. In *IN ACM WEBKDD WORKSHOP*, 2000.

[33] D. Seung and L. Lee. Algorithms for non-negative matrix factorization. *Advances in neural information processing systems*, 13:556–562, 2001.

[34] N. Srebro and T. Jaakkola. Weighted low rank approximation, 2003.

[35] Nathan Srebro and Tommi Jaakkola. Weighted low-rank approximations. In Tom Fawcett and Nina Mishra, editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 720–727. AAAI Press, 2003.

[36] John A. Tropp. *Topics in Sparse Approximation.* PhD thesis, University of Texas, 2004.

[37] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman. Missing value estimation methods for DNA microarrays. *Bioinformatics*, 17(6):520–525, June 2001.

[38] Stephen A. Vavasis. On the Complexity of Nonnegative Matrix Factorization. *SIAM Journal on Optimization*, 20(3):1364–1377, January 2010.

[39] Fei Wang, Tao Li, Xin Wang, Shenghuo Zhu, and Chris Ding. Community discovery using nonnegative matrix factorization. *Data Mining and Knowledge Discovery*, 22(3):493–521, 2011.

[40] Jim Jing-Yan Wang, Xiaolei Wang, and Xin Gao. Non-negative matrix factorization by maximizing correntropy for cancer clustering. *BMC bioinformatics*, 14(1):107, 2013.

[41] Yu-Xiong Wang and Yu-Jin Zhang. Nonnegative Matrix Factorization: A Comprehensive Review. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1336–1353, June 2013.

[42] Stefan Wild, James Curry, and Anne Dougherty. Improving non-negative matrix factorizations through structured initialization. *Pattern Recognition*, 37(11):2217–2232, November 2004.

[43] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 267–273. ACM, 2003.

[44] Yun Xue, Chong Sze Tong, Ying Chen, and Wen-Sheng Chen. Clustering-based initialization for non-negative matrix factorization. *Applied Mathematics and Computation*, 205(2):525–536, November 2008.

[45] Sheng Zhang, Weihong Wang, James Ford, and Fillia Makedon. Learning from incomplete ratings using non-negative matrix factorization. In *SDM*, volume 6, pages 548–552. SIAM, 2006.

[46] Chun-Hou Zheng, De-Shuang Huang, D Zhang, and Xiang-Zhen Kong. Tumor clustering using nonnegative matrix factorization with gene selection. *Information Technology in Biomedicine, IEEE Transactions on*, 13(4):599–607, 2009.

[47] Zhonglong Zheng, Jie Yang, and Yitan Zhu. Initialization enhancer for non-negative matrix factorization. *Engineering Applications of Artificial Intelligence*, 20(1):101–110, February 2007.

Appendix A

Graphs

A.1 SNMF without sparsity

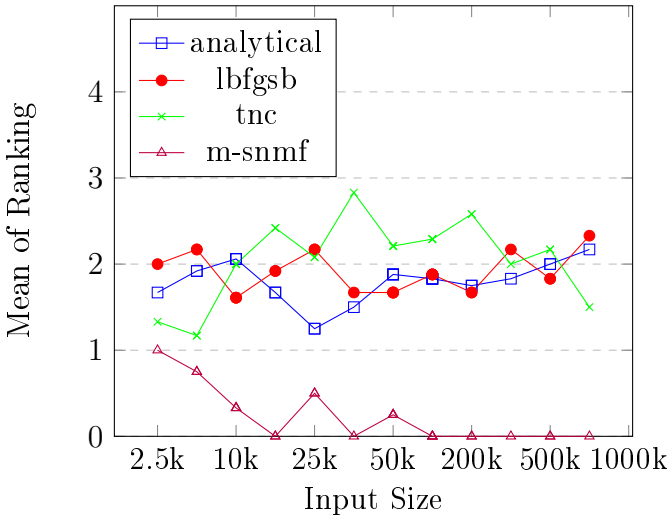

Figure A.1: Mean of error rankings separated by Input Size for SNMF without sparseness constraints

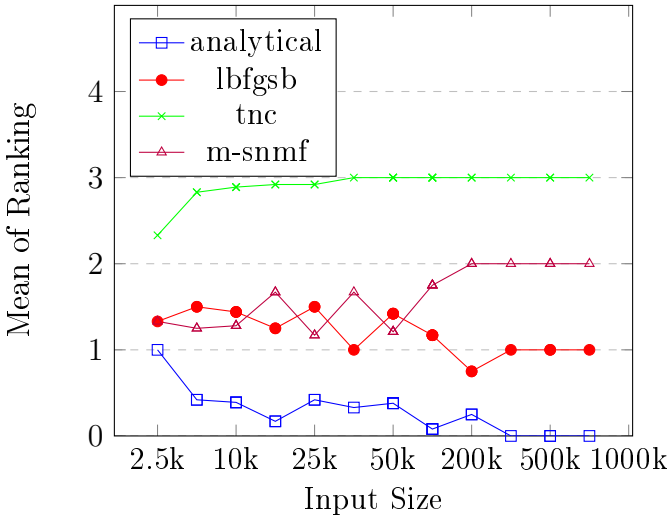

Figure A.2: Mean of time rankings separated by Input Size for SNMF without sparseness constraints

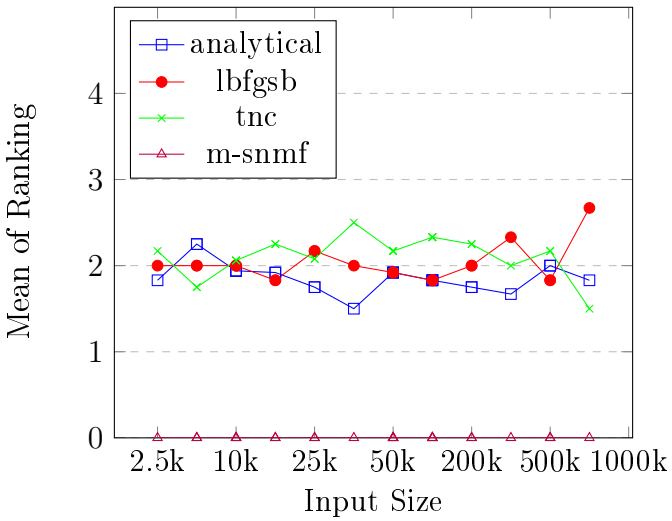

Figure A.3: Mean of sparseness rankings separated by Input Size for SNMF without sparseness constraints

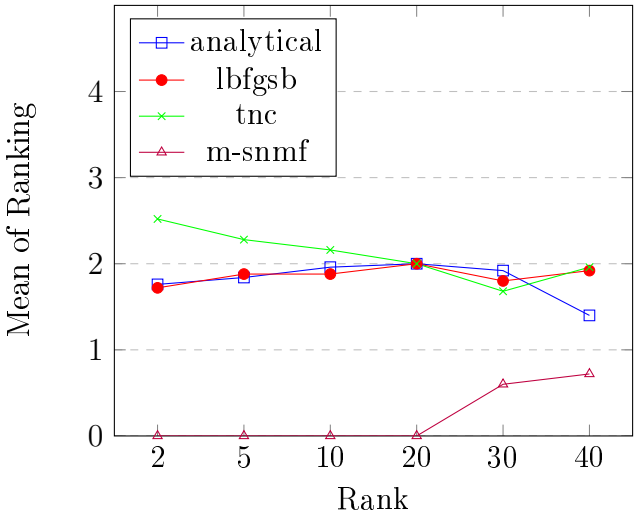

Figure A.4: Mean of error rankings separated by Rank for SNMF without sparseness constraints

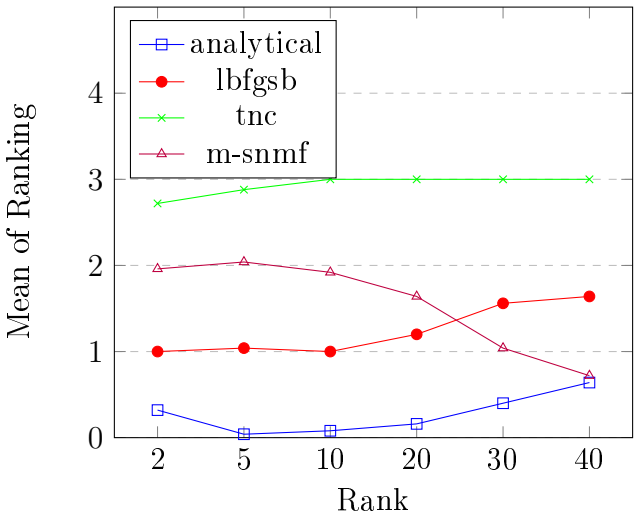

Figure A.5: Mean of time rankings separated by Rank for SNMF without sparseness constraints

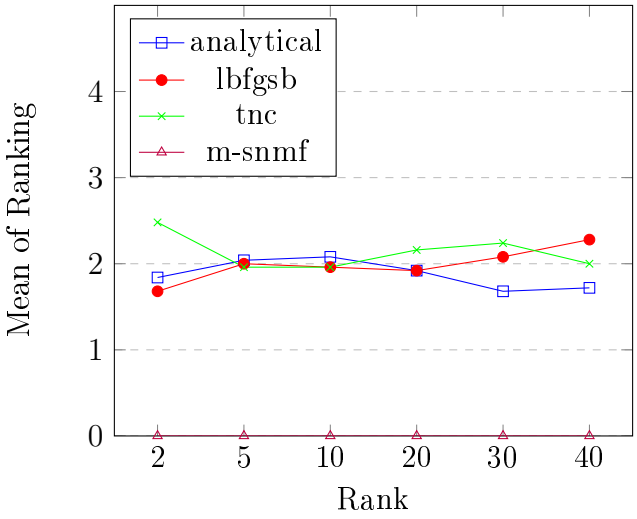

Figure A.6: Mean of sparseness rankings separated by Rank for SNMF without sparseness constraints

A.2 SNMF with medium sparsity

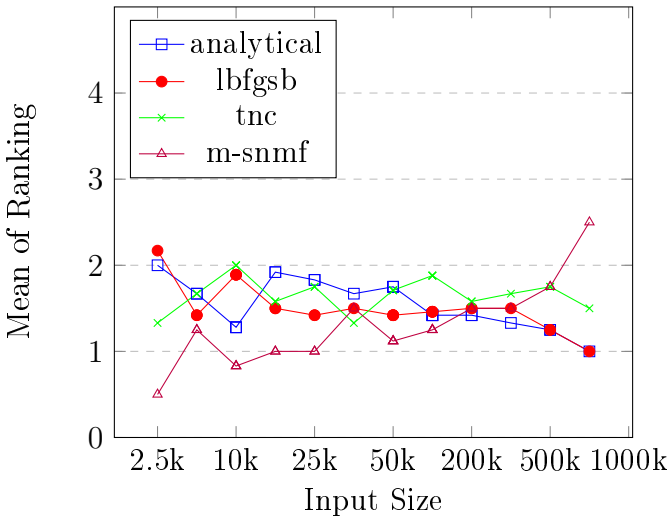

Figure A.7: Mean of error rankings separated by Input Size for SNMF with 0.5 sparseness projection

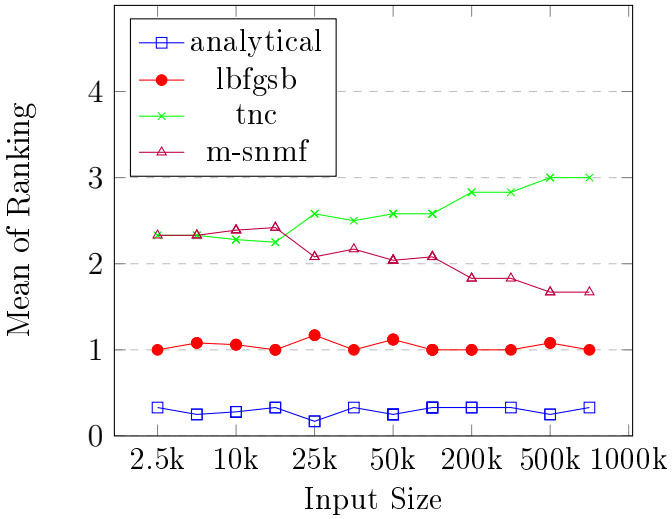

Figure A.8: Mean of time rankings separated by Input Size for SNMF with 0.5 sparseness projection

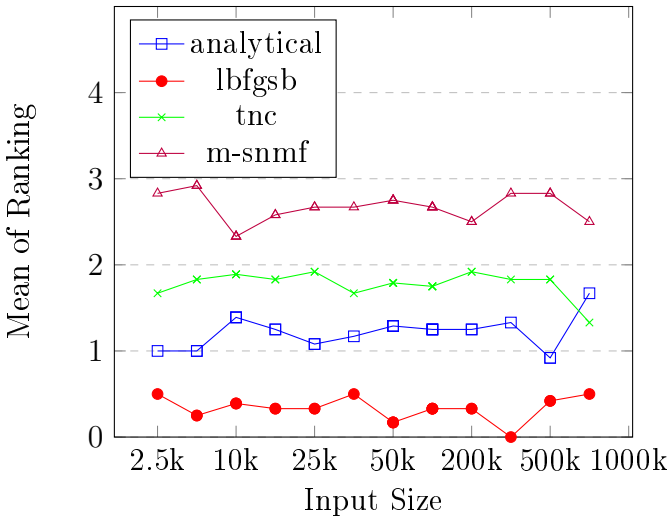

Figure A.9: Mean of sparseness rankings separated by Input Size for SNMF with 0.5 sparseness projection

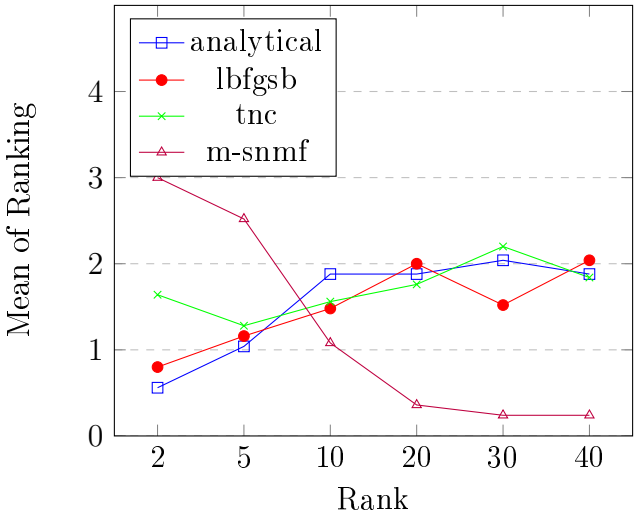

Figure A.10: Mean of error rankings separated by Rank for SNMF with 0.5 sparseness projection

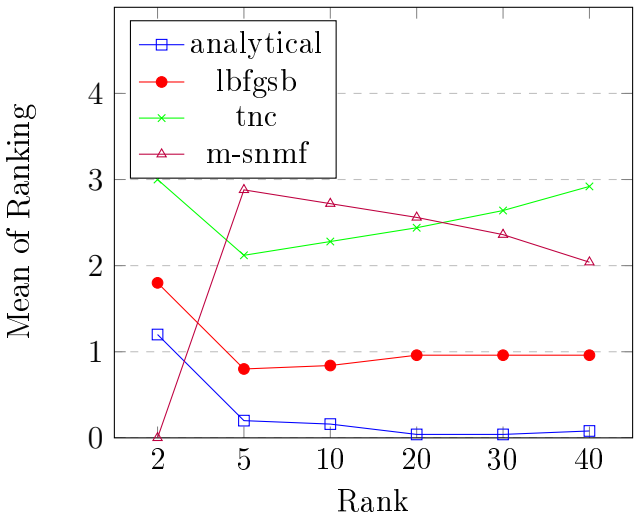

Figure A.11: Mean of time rankings separated by Rank for SNMF with 0.5 sparseness projection

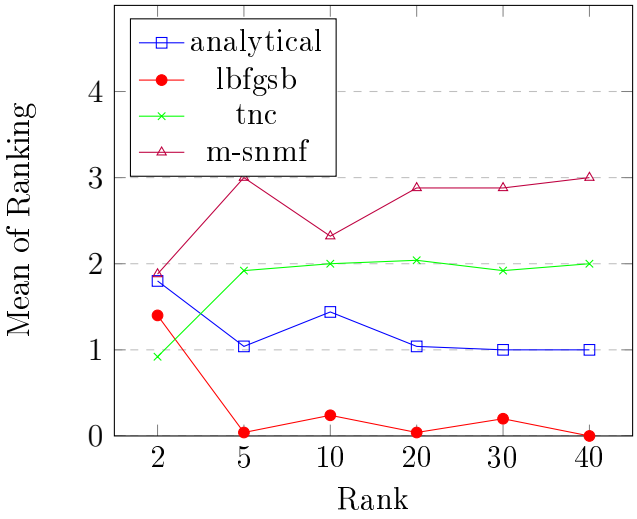

Figure A.12: Mean of sparseness rankings separated by Rank for SNMF with 0.5 sparseness projection

A.3 SNMF with high sparsity

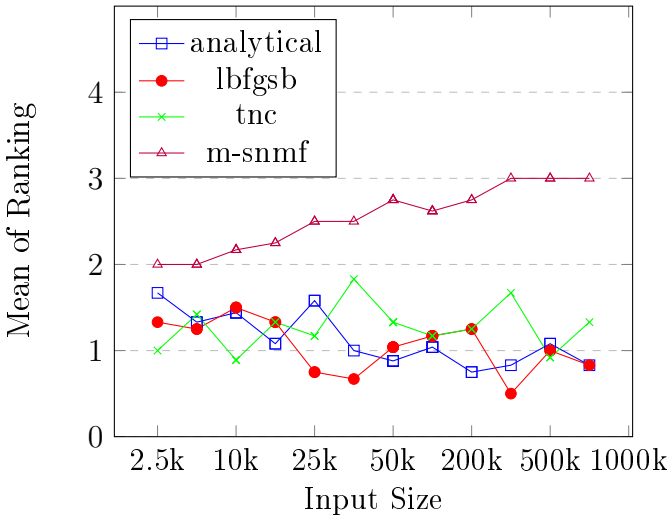

Figure A.13: Mean of error rankings separated by Input Size for SNMF with 0.9 sparseness projection

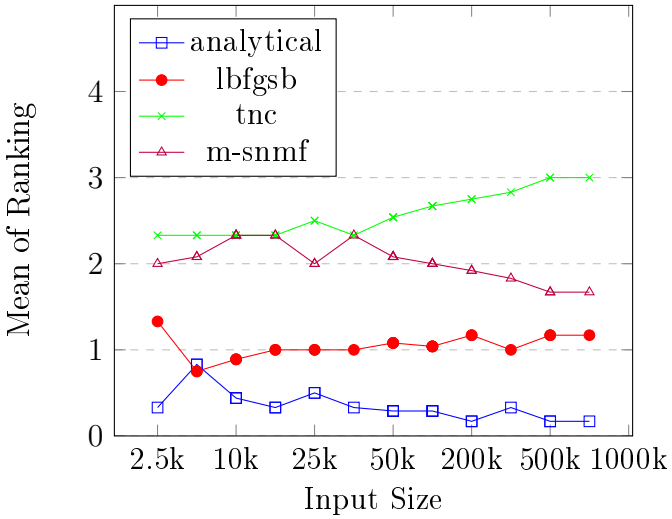

Figure A.14: Mean of time rankings separated by Input Size for SNMF with 0.9 sparseness projection

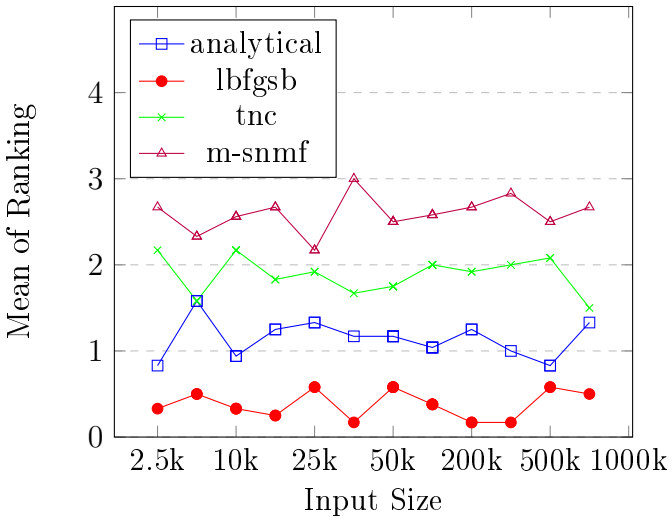

Figure A.15: Mean of sparseness rankings separated by Input Size for SNMF with 0.9 sparseness projection

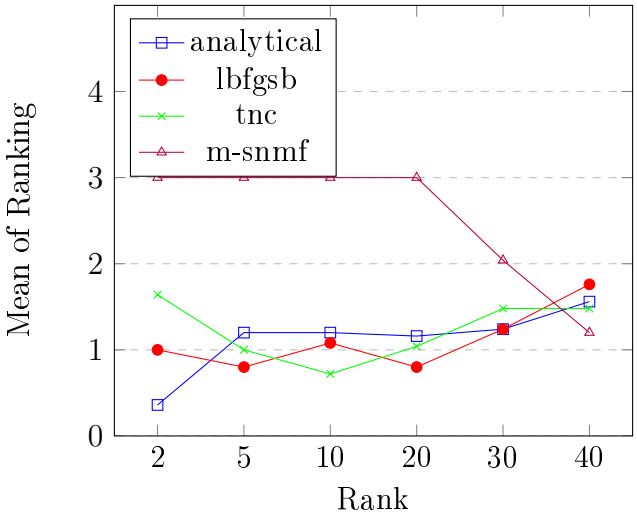

Figure A.16: Mean of error rankings separated by Rank for SNMF with 0.9 sparseness projection

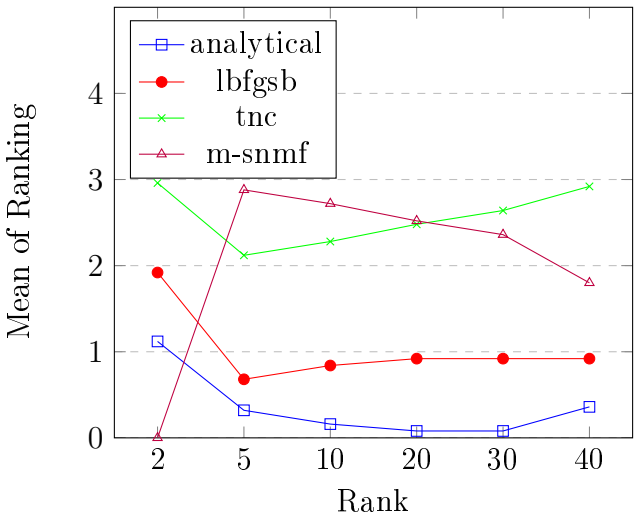

Figure A.17: Mean of time rankings separated by Rank for SNMF with 0.9 sparseness projection

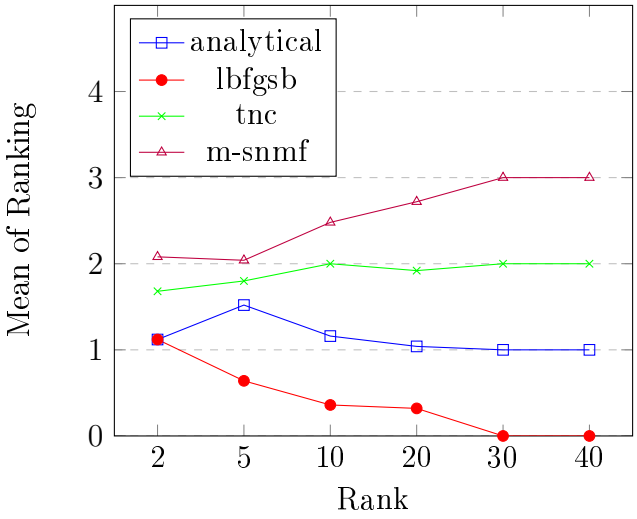

Figure A.18: Mean of sparseness rankings separated by Rank for SNMF with 0.9 sparseness projection

A.4 WSNMF with no sparsity

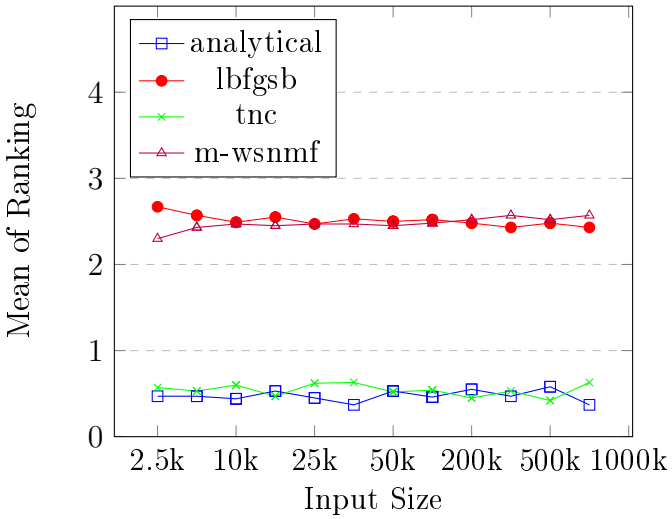

Figure A.19: Mean of error rankings separated by Input Size For WSNMF without sparseness projection

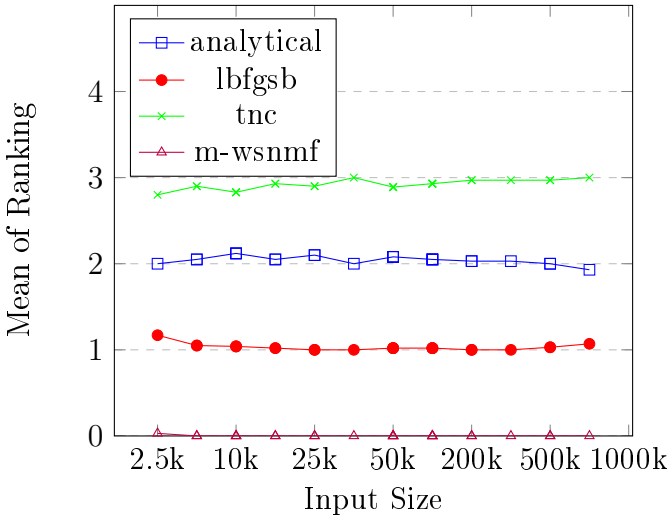

Figure A.20: Mean of time rankings separated by Input Size For WSNMF without sparseness projection

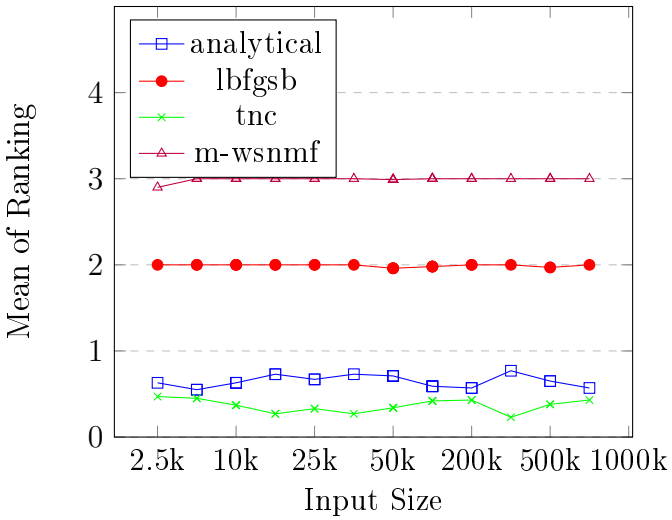

Figure A.21: Mean of sparseness rankings separated by Input Size For WSNMF without sparseness projection

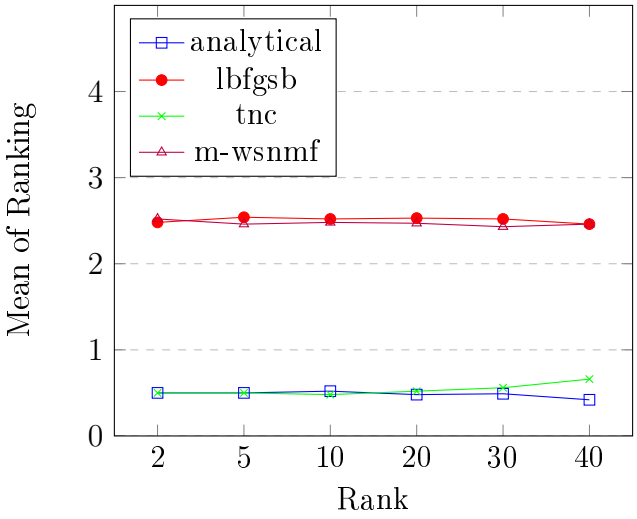

Figure A.22: Mean of error rankings separated by Rank For WSNMF without sparseness projection

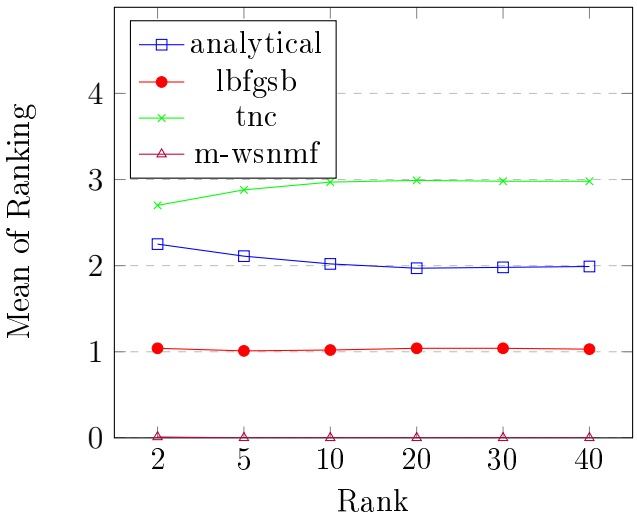

Figure A.23: Mean of time rankings separated by Rank For WSNMF without sparseness projection

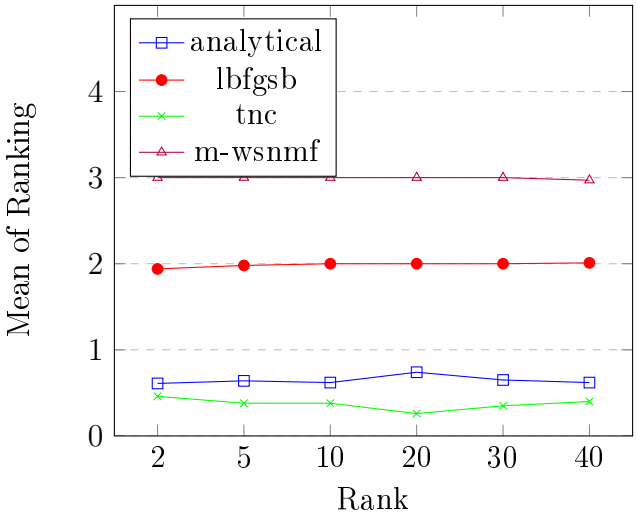

Figure A.24: Mean of sparseness rankings separated by Rank For WSNMF without sparseness projection

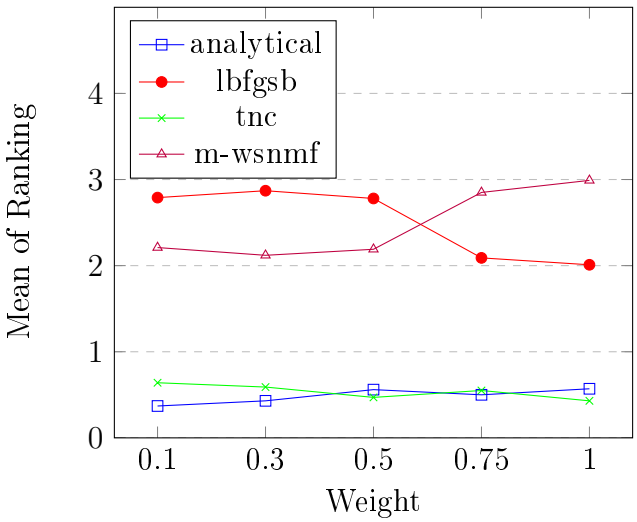

Figure A.25: Mean of error rankings separated by Weight For WSNMF without sparseness projection

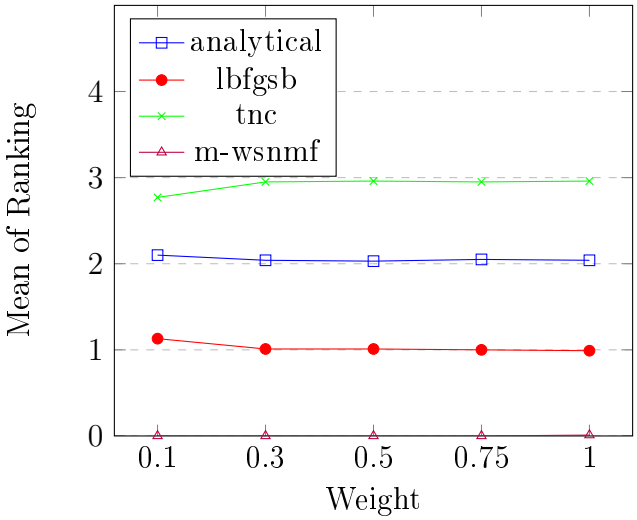

Figure A.26: Mean of time rankings separated by Weight For WSNMF without sparseness projection

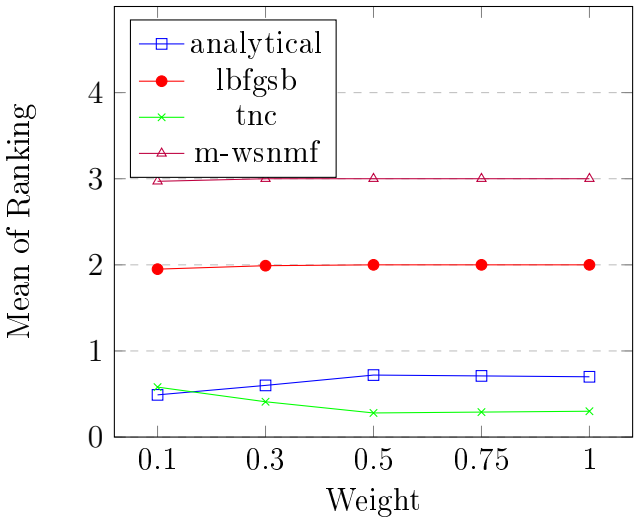

Figure A.27: Mean of sparseness rankings separated by Weight For WSNMF without sparseness projection

A.5 WSNMF with medium sparsity

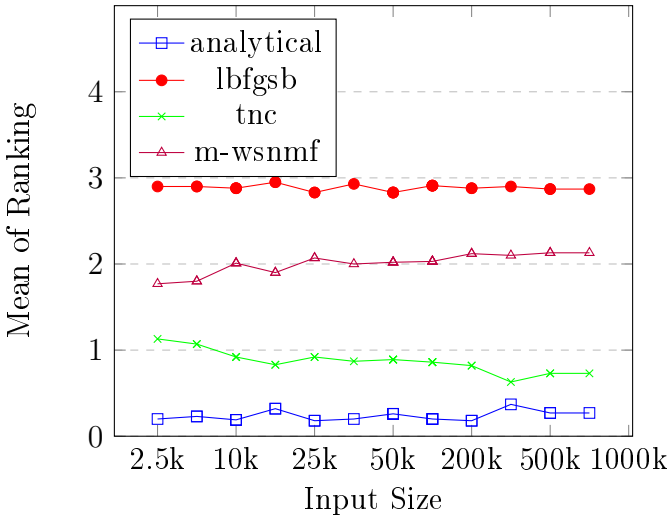

Figure A.28: Mean of error rankings separated by Input Size for WSNMF with 0.5 sparseness projection

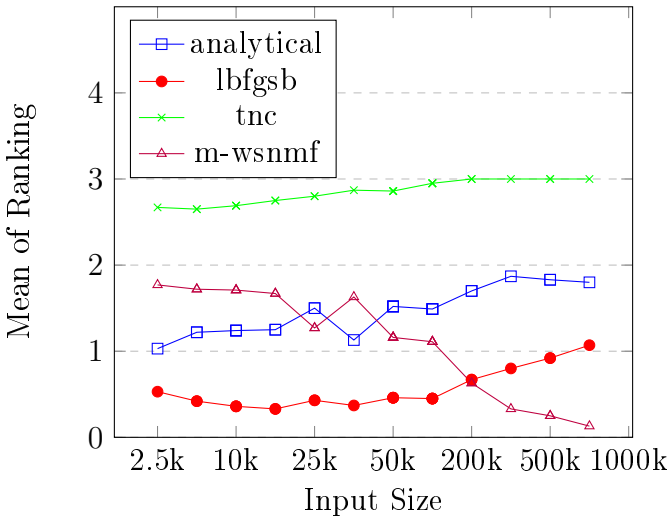

Figure A.29: Mean of time rankings separated by Input Size for WSNMF with 0.5 sparseness projection

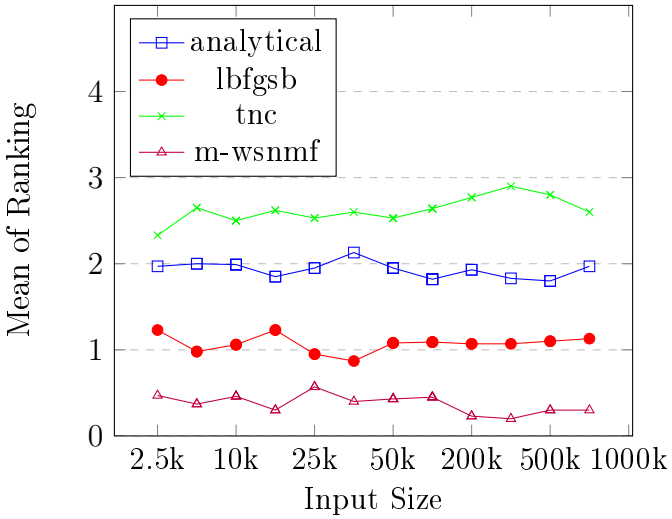

Figure A.30: Mean of sparseness rankings separated by Input Size for WSNMF with 0.5 sparseness projection

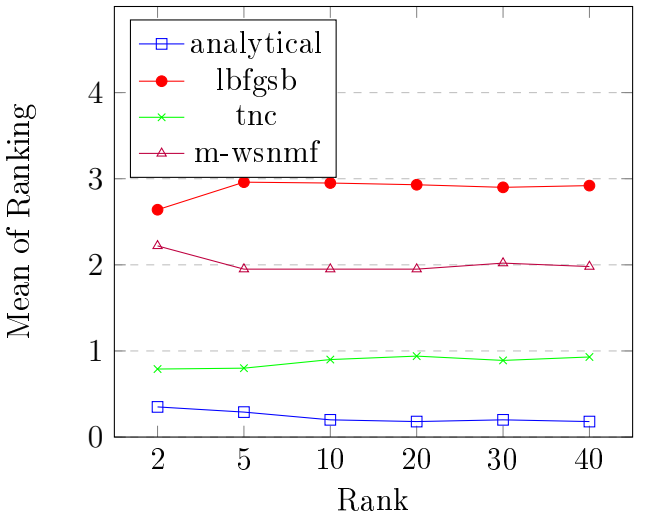

Figure A.31: Mean of error rankings separated by Rank for WSNMF with 0.5 sparseness projection

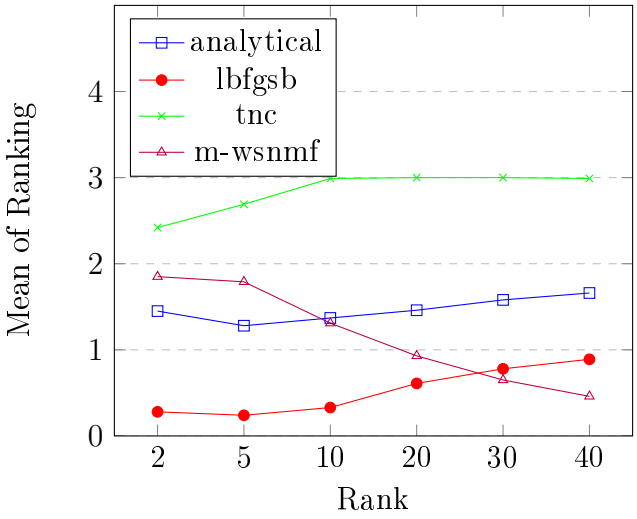

Figure A.32: Mean of time rankings separated by Rank for WSNMF with 0.5 sparseness projection

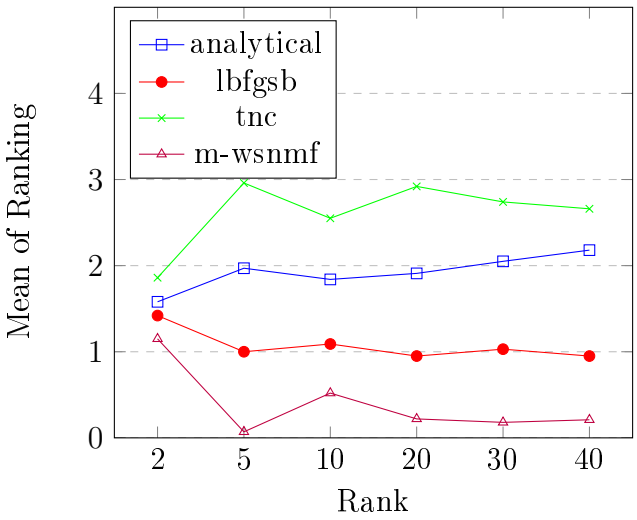

Figure A.33: Mean of sparseness rankings separated by Rank for WSNMF with 0.5 sparseness projection

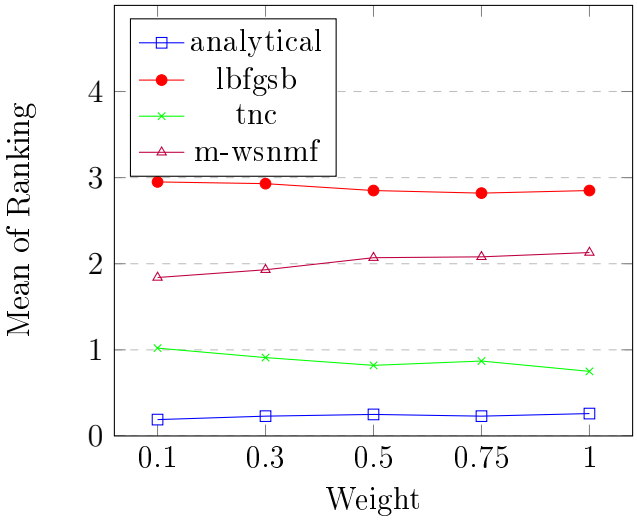

Figure A.34: Mean of error rankings separated by Weight for WSNMF with 0.5 sparseness projection

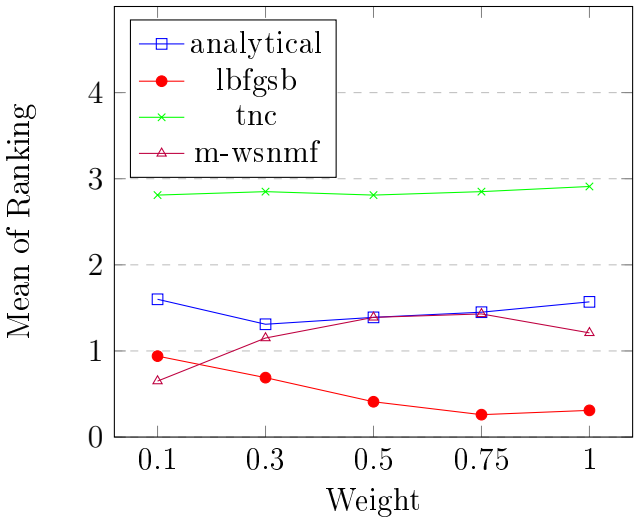

Figure A.35: Mean of time rankings separated by Weight for WSNMF with 0.5 sparseness projection

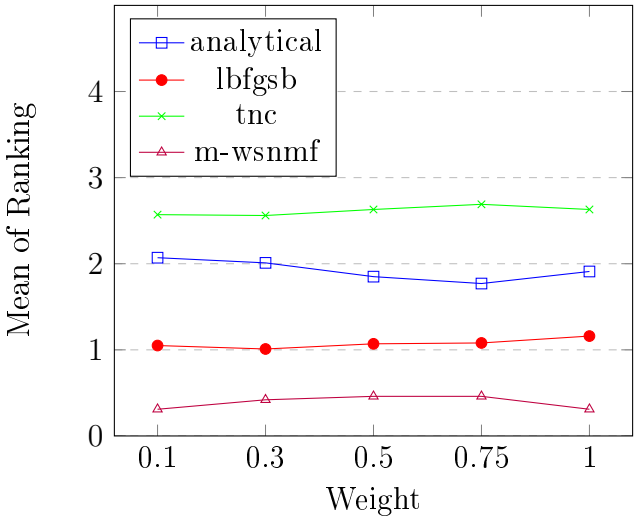

Figure A.36: Mean of sparseness rankings separated by Weight for WSNMF with 0.5 sparseness projection

A.6 WSNMF with High Sparsity

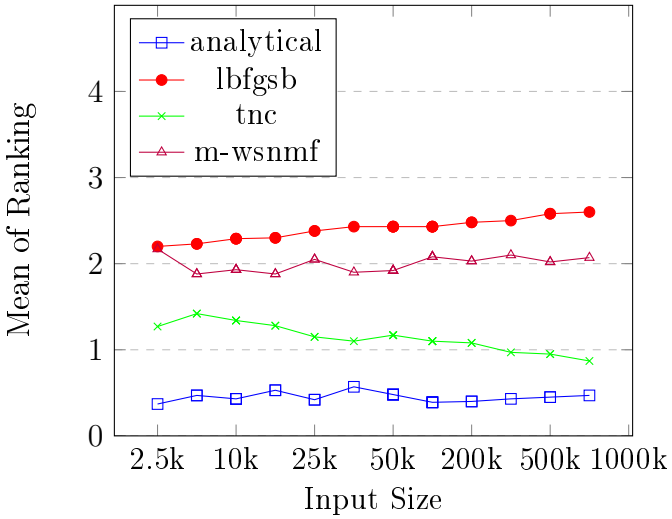

Figure A.37: Mean of error rankings separated by Input Size for WSNMF with 0.9 sparseness projection

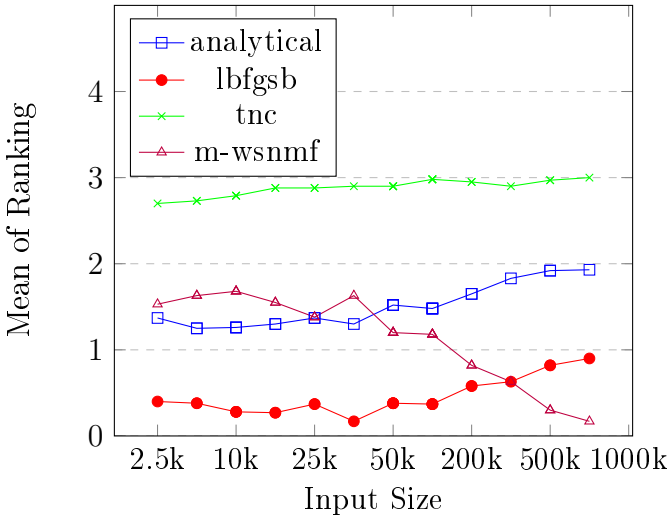

Figure A.38: Mean of time rankings separated by Input Size for WSNMF with 0.9 sparseness projection

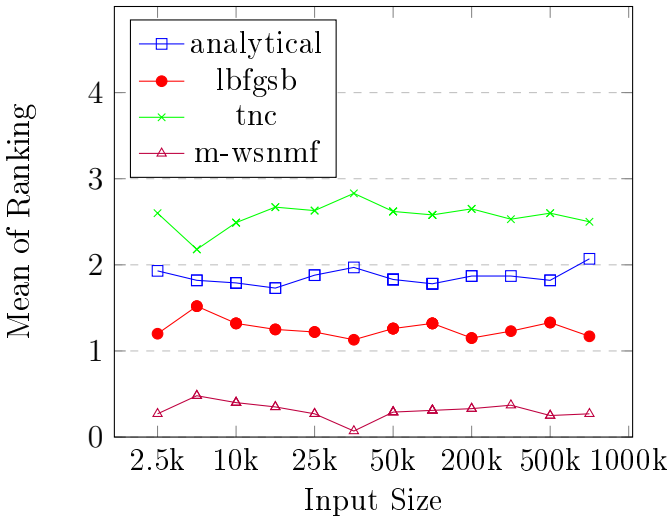

Figure A.39: Mean of sparseness rankings separated by Input Size for WSNMF with 0.9 sparseness projection

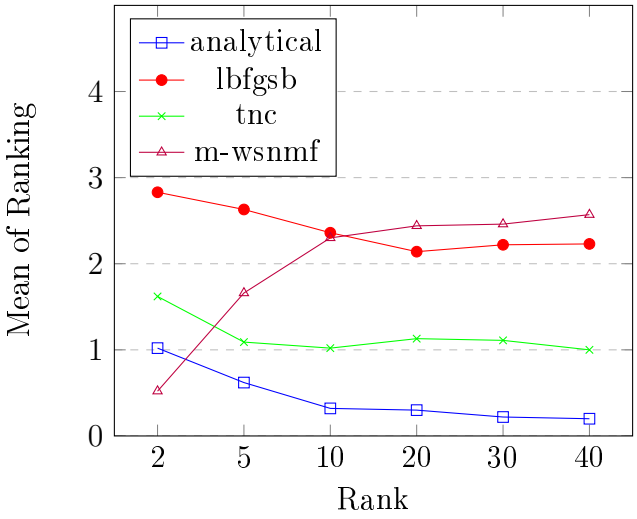

Figure A.40: Mean of error rankings separated by Rank for WSNMF with 0.9 sparseness projection

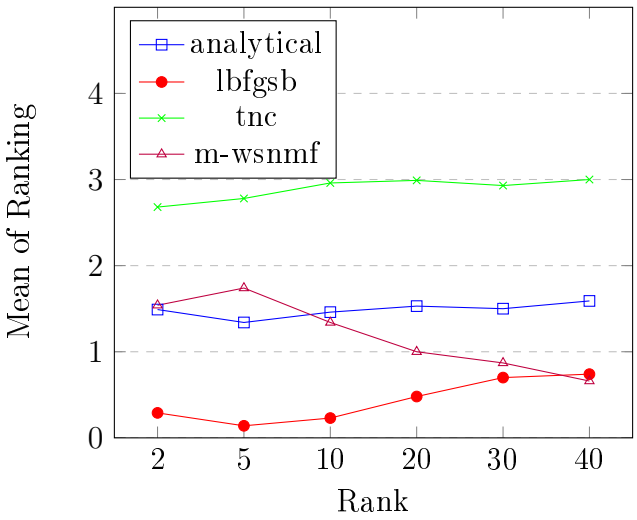

Figure A.41: Mean of time rankings separated by Rank for WSNMF with 0.9 sparseness projection

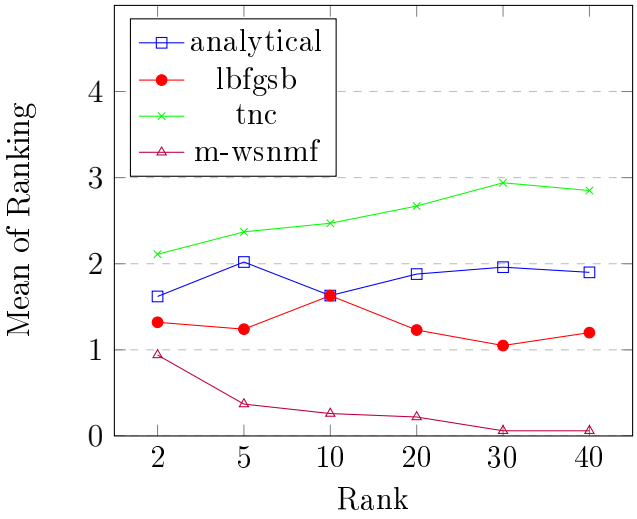

Figure A.42: Mean of sparseness rankings separated by Rank for WSNMF with 0.9 sparseness projection

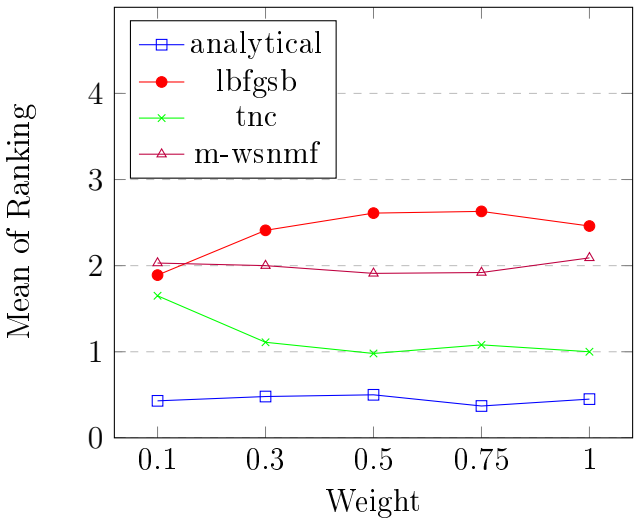

Figure A.43: Mean of error rankings separated by Weight for WSNMF with 0.9 sparseness projection

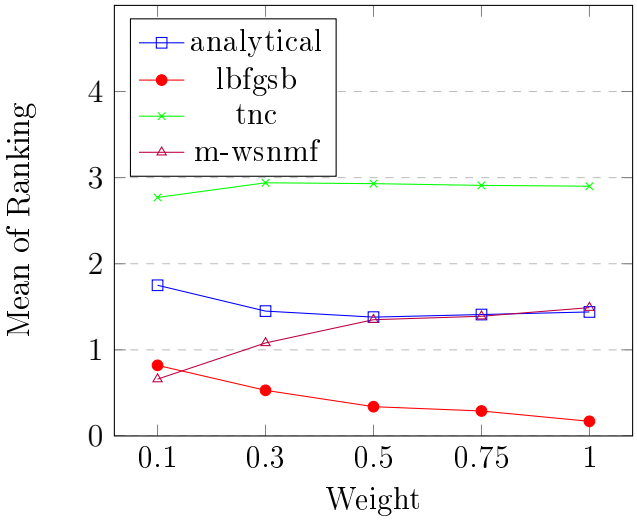

Figure A.44: Mean of time rankings separated by Weight for WSNMF with 0.9 sparseness projection

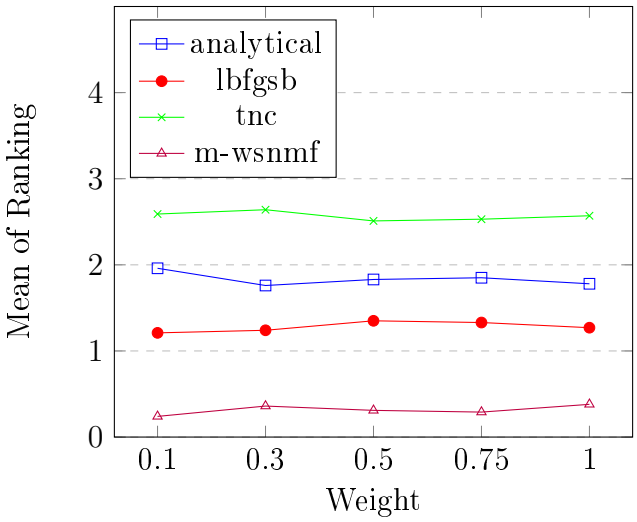

Figure A.45: Mean of sparseness rankings separated by Weight for WSNMF with 0.9 sparseness projection