Swarm Intelligence for Dimensionality Reduction: How to Improve the Non-negative Matrix Factorization with Nature-inspired Optimization Methods

Andreas Janecek [1]* and Ying Tan [2]

[1] *University of Vienna, Research Group Theory and Applications of Algorithms, 1090 Vienna, Austria*

[2] *Key Laboratory of Machine Perception (MOE), and Department of Machine Intelligence; School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, China*

Email: andreas.janecek@univie.ac.at; ytan@pku.edu.cn * Corresponding author

ABSTRACT:

Low-rank approximations allow for compact representations of data with reduced storage and runtime requirements *and* reduced redundancy and noise. The **Non-negative Matrix Factorization (NMF)** is a special low-rank approximation which allows for additive parts-based, interpretable representation of the data. Various properties of NMF are similar to Swarm Intelligence (SI) methods: indeed, most NMF objective functions and most SI fitness functions are non-convex, discontinuous, and may possess many local minima. This chapter summarizes our efforts on improving convergence, approximation quality, and classification accuracy of NMF using five different meta-heuristics based on SI and evolutionary computation. We present (*i*) new initialization strategies for NMF, and (*ii*) an iterative update strategy for NMF.

The applicability of our approach is illustrated on data sets coming from the areas of spam filtering and email classification. Experimental results show that both optimization strategies are able to improve NMF in terms of faster convergence, lower approximation error, and/or better classification accuracy.

Keywords: *Nonnegative Matrix Factorization, NMF Initialization, Differential Evolution, Particle Swarm Optimization, Genetic Algorithms, Fireworks Algorithm, Fish School Search, Email Classification, Spam Filtering, Multiplicative Update*

INTRODUCTION

Low-rank approximations are utilized in several content based retrieval and data mining applications, such as text and multimedia mining, web search, etc. and achieve a more compact representation of the data with only limited loss in information. They reduce storage and runtime requirements, and also reduce redundancy and noise in the data representation while capturing the essential associations. The Non-negative Matrix Factorization (NMF, (Lee and Seung 1999)) leads to a low-rank approximation which satisfies non-negativity constraints. NMF approximates a data matrix A by $A \approx WH$, where W and H are the *NMF factors*. NMF requires *all entries* in A, W and H to be zero or positive. Contrary to other low-rank approximations such as the Singular Value Decomposition (SVD), these constraints force NMF to produce so-called "additive parts-based" representations. This is an impressive benefit of NMF, since it makes the interpretation of the NMF factors much easier than for factors containing positive and negative entries (Berry, Browne et al. 2007) (Janecek and Gansterer 2010) (Lee and Seung 1999).

The NMF is usually not unique if different initializations of the factors W and H are used. Moreover, there are several different NMF algorithms which all follow different strategies (e.g. mean squared error, least squares, gradient descent, etc.) and produce different results. Mathematically, the goal of NMF is to find a "good" (ideally the best) solution of an optimization problem with bound constraints in the form $\min_{x \in \Omega} f(x)$, where $f : \mathbb{R}^N \to \mathbb{R}$ is the nonlinear objective function of NMF, and Ω is the feasible region (for NMF, Ω is restricted to non-negative values). f is usually not convex, discontinuous and may possess many local minima (Stadlthanner, Lutter et al. 2007). Since meta-heuristic optimization algorithms are known to be able to deal well with such difficulties they seem to be a promising choice for improving the quality of NMF. Over the last decades nature-inspired meta-heuristics, including those based on swarm intelligence, have gained much popularity due to their applicability for various optimization problems. They benefit from the fact that they are able to find acceptable results within a reasonable amount of time for many complex, large and dynamic problems (Blackwell 2007). Although they lack the ability to guarantee the optimal solution for a given problem (comparably to NMF), it has been shown that they are able to tackle various kinds of real-world optimization problems (Chiong 2009). Meta-heuristics as well as the principles of NMF are in accordance with the *law of sufficiency* (Eberhart, Shi et al. 2001): If a solution to a problem is good enough, fast enough and cheap enough, then it is sufficient.

In this chapter we present two different strategies for improving the NMF using five optimization algorithms based on swarm intelligence and evolutionary computing: Particle Swarm Optimization

(PSO), Genetic Algorithms (GA), Fish School Search (FSS), Differential Evolution (DE), and Fireworks Algorithm (FWA). All algorithms are population based and can be categorized into the fields of *swarm intelligence* (PSO, FSS, FWA), *evolutionary algorithms* (GA), and a combination thereof (DE). The goal is to find a solution with smaller overall error at convergence, and/or to speed up convergence of NMF (*i.e.* smaller approximation error for a given number of NMF iterations) compared to identical NMF algorithms without applied optimization strategy. Another goal is to increase the classification accuracy in cases where NMF is used as dimensionality reduction method for machine learning applications. The concepts of the two optimization strategies are the following: In the first strategy, meta-heuristics are used to initialize the factors W and H in order to minimize the NMF objective function *prior* to the factorization. The second strategy aims at iteratively improving the approximation quality of NMF during the first iterations.

The proposed optimization strategies can be considered successful if they are able to improve the NMF in terms of either (*i*) faster convergence (*i.e.* better accuracy per runtime) (*ii*) lower final approximation error, (*iii*) or better classification accuracy. The optimization of different rows of W and different columns of H can be split up into several partly independent sub-tasks and can thus be executed concurrently. Since this allows for a parallel and/or distributed computation of both update strategies, we also discuss parallel implementations of the proposed optimization strategies. Experimental results show that both strategies, the initialization of NMF factors as well as an iterative update during the first iterations, are able to improve the NMF in terms of faster convergence, lower approximation error, and/or better classification accuracy.

Related Work

The work by Lee and Seung (Lee and Seung 1999) is known as a standard reference for NMF. The original *Multiplicative Update* (MU) algorithm introduced in this article provides a good baselines against which other algorithms (e.g. the *Alternating Least Squares* algorithm (Paatero and Tapper 1994), the *Gradient Descent* algorithm (Lin 2007), ALSPGRAD (Lin 2007), quasi Newton-type NMF (Kim and Park 2008), fastNMF and bayesNMF (Schmidt and Laurberg 2008), etc.) have to be judged. While the MU algorithm is still the fastest NMF algorithm per iteration and a good choice if a very fast and rough approximation is needed, ALSPGRAD, fastNMF and bayesNMF have shown to achieve a better approximation at convergence compared to many other NMF algorithms (Janecek, Schulze-Grotthoff et al. 2011).

NMF initialization. Only few algorithms for non-random NMF initialization have been published. (Wild, Curry et al. 2004) used spherical k -means clustering to group column vectors of A as input for W. A similar technique was used in (Xue, Tong et al. 2008). Another clustering-based method of structured initialization designed to find spatially localized basis images can be found in (Kim and Park 2008). (Boutsidis and Gallopoulos 2008) used an initialization technique based on two SVD processes called

nonnegative double singular value decomposition (NNDSVD). Experiments indicate that this method has advantages over the centroid initialization in (Wild, Curry et al. 2004) in terms of faster convergence.

NMF and meta-heuristics. So far, only few studies can be found that aim at combining NMF and meta-heuristics, most of them are based on Genetic Algorithms (GAs). In (Stadlthanner, Lutter et al. 2007), the authors have investigated the application of GAs on sparse NMF for microarray analysis, while (Snásel, Platos et al. 2008) have applied GAs for boolean matrix factorization, a variant of NMF for binary data based on Boolean algebra. However, the methods presented in these studies are barely connected to the techniques presented in this chapter. In two preceding studies (Janecek and Tan 2011), (Janecek and Tan 2011) we have introduced the basic concepts of the proposed update strategies. Very recently, the study in (LI, YANG et al. 2013) proposed to replace the two NMF algorithm presented in (Lee and Seung 1999) with the quantum-behaved PSO (QPSO) for speech signal processing. QPSO is used to extract non-negative components with low cross-talk error and high SNR. Although the authors mention that QPSO is able to achieve better results than the two classic NMF algorithms, unfortunately no runtime comparisons are performed to check whether approximation per runtime can be improved. We performed similar experiments and found that when the complete NMF is replaced with SI methods the runtime increases significantly. Thus, we expect a significant growth of computation time also in the case of QPSO. Another very recent study (DAI 2013) uses an approach very similar to the NMF initialization presented in this chapter. We note that our first publication on this topic (Janecek and Tan 2011) appeared two years earlier. In their abstract, the authors mention that their approach uses the output of nonsmooth NMF as initial values for PSO to avoid blind search. However, at the moment this article is only available in Chinese characters. Similarly to the study (DAI 2013), no runtime measurements are available.

In this chapter we extend our preliminary work in several ways by the following new contributions. At first, we evaluate our methods on synthetic data as well as on data sets coming from the areas of spam filtering/email classification. This allows us to evaluate the proposed methods in the application context of the applied data sets. In other words, we are now able to investigate the quality of the NMF not only in terms of approximation accuracy but also in terms of *classification* accuracy achieved with the approximated data sets as well as with the basic vectors of the NMF factor W. Within this evaluation process we consider two different classification settings, a static setting where NMF is computed on the complete data set (training and test data), and a dynamic setting where NMF can be applied dynamically to new data. Moreover, we present a detailed evaluation of the runtime performance of the proposed update strategies, and, finally, we are able to compare the performance of our strategies with each other using the same parameter settings, data sets, and hardware set-up.

Notation

A matrix is represented by an uppercase italic letter (A, B, Σ, …), a vector by a lowercase bold letter ($\mathbf{u}$, $\mathbf{x}$, , $\mathbf{q}_1$, …), and a scalar by a lowercase Greek letter (λ, μ, …). The i^{th} row vector of a matrix D is

represented as $\mathbf{d}_i^r$, and the j^{th} column vector of D as $\mathbf{d}_j^c$. Matrix-matrix multiplications are denoted by "*", element-wise multiplications by "$\cdot$", and element-wise divisions by "./".

Synopsis

In the following section we briefly review low-rank approximations and NMF algorithms. Then, we summarize the swarm intelligence algorithms used in this chapter, and present the proposed optimization strategies for NMF based on them. Moreover, we discuss different classification methods based on NMF. Finally, we evaluate our methods, discuss the achieved results, and conclude our work and summarize ongoing and future research activities in this area.

LOW RANK APPROXIMATIONS

Given a data matrix $A \in \mathbb{R}^{m \times n}$ whose n columns represent instances and whose m rows contain the values of a certain feature for the instances, most low-rank approximations reduce the dimensionality by representing the original data as accurately as possible with linear combinations of the original instances and/or features. Mathematically, A is replaced with another matrix A_k with usually much smaller rank. In general, a closer approximation means a better factorization. However, it is highly likely that in some applications specific factorizations might be more desirable compared to other solutions.

The most important low-rank approximation techniques are the Singular Value Decomposition (SVD, (Berry 1992)) and the closely related Principal Component Analysis (PCA, (Jolliffe 2002)). Traditionally, the PCA uses the eigenvalue decomposition to find eigenvalues and eigenvectors of the covariance matrix Cov(A) of A. Then the original data matrix A can be approximated by $A_k := AQ_k$, with $Q_k = [\mathbf{q}_1, ..., \mathbf{q}_k]$, where $\mathbf{q}_1, ..., \mathbf{q}_k$ are the first k eigenvectors of Cov(A). The SVD decomposes A into a product of three matrices such that $A = U\Sigma V^\top$, where Σ contains the singular values along the diagonal, and U and V are the singular vectors. The reduced rank SVD to A can be found by setting all but the first k largest singular values equal to zero and using only the first k columns of U and V, such that $A_k := U_k \Sigma_k V_k^\top$. Other well-known low-rank approximation techniques comprise Factor Analysis, Independent Components Analysis, Multidimensional Scaling such as Fastmap or ISOMAP, or Locally Linear Embedding (LLE), which are all summarized in (Tan, Steinbach et al. 2005).

Amongst all possible rank k approximations, the approximation A_k calculated by SVD and PCA is the best approximation in the sense that $|| A - A_k ||_F$ is as small as possible (cf. (Berry, Drmac et al. 1999)). In other words, SVD and PCA give the closest rank k approximation of a matrix, such that $|| A - A_k ||_F \leq || A - B_k ||_F$, where B_k is *any* matrix of rank k, and $|| . ||_F$ is the Frobenius norm, which is

defined as $(\sum |a_{ij}|^2)^{1/2} = \| A \|_F$. However, the main drawback of PCA and SVD refers to the interpretability of the transformed features. The resulting orthogonal matrix factors generated by the approximation usually do not allow for direct interpretations in terms of the original features because they contain positive ***and*** negative coefficients (Zhang, Berry et al.). In many application domains, a negative quantification of features is meaningless and the information about how much an original feature contributes in a low-rank approximation is lost. The presence of negative, meaningless components or factors may influence the entire result. This is especially important for applications where the original data matrix contains only positive entries, e.g. in text-mining applications, image classification, etc. If the factor matrices of the low-rank approximation were constrained to contain only positive or zero values, the original meaning of the data could be preserved better.

Non-negative Matrix Factorization (NMF)

The NMF leads to special low-rank approximations which satisfy these non-negativity constraints. NMF requires that all entries in A, W and H are zero or positive. This makes the interpretation of the NMF factors much easier and enables NMF a non-subtractive combination of parts to form a whole (Lee and Seung 1999). The NMF consists of reduced rank *nonnegative* factors $W \in \mathbb{R}^{m \times k}$ and $H \in \mathbb{R}^{k \times n}$ with $k \ll min\{m, n\}$ that approximate a matrix $A \in \mathbb{R}^{m \times n}$ by $A \approx WH$, where the approximation WH has rank at most k . The nonlinear optimization problem underlying NMF can generally be stated as

$$\min_{W,H} f(W,H) = \min_{W,H} \frac{1}{2} \| A - WH \|_F^2 . \qquad (1.1)$$

The Frobenius norm $\| . \|_F$ is commonly used to measure the error between the original data A and the approximation WH, but other measures such as the Kullback-Leibler divergence are also possible (Lee and Seung 2001)). The error between A and WH is usually stored in a distance matrix $D = A - WH$ (cf. *Figure 1*). Unlike the SVD, the NMF is not unique, and convergence is not guaranteed for all NMF algorithms. If they converge, then usually to local minima only (potentially different ones for different algorithms). Nevertheless, the data compression achieved with only local minima has been shown to be of desirable quality for many data mining applications (Langville, Meyer et al. 2006). Moreover, for some specific problem settings a smaller residual $D = A - WH$ (a smaller error) may not necessarily improve of the solution of the actual application (e.g. classification task) compared to a rather coarse approximation. However, as analyzed in (Janecek and Gansterer 2010) a closer NMF approximation leads to qualitatively better classification results and turns out to achieve significantly more stable results.

Figure 1 *- Scheme of very coarse NMF approximation with very low rank k. Although k is significantly smaller than m and n, the typical structure of the original data matrix can be retained (note the three different groups of data objects in the left, middle, and right part of A).*

NMF Initialization. Algorithms for computing NMF are iterative and require initialization of the factors W and H . NMF unavoidably converges to local minima, probably different ones for different initialization (cf. (Boutsidis and Gallopoulos 2008)). Hence, random initialization makes the experiments unrepeatable since the solution to Equ.1.1 is not unique in this case. A proper non-random initialization can lead to faster error reduction and better overall error at convergence. Moreover, it makes the experiments repeatable. Although the benefits of good NMF initialization techniques are well known in the literature, most studies use random initialization (cf. (Boutsidis and Gallopoulos 2008). Since some initialization procedures can be rather costly in terms of runtime the trade-off between computational cost in the initialization step and the computational cost of the actual NMF algorithm need to be balanced carefully. In some situations, an expensive preprocessing step may overwhelm the cost savings in the subsequent NMF update steps.

General structure of NMF. In the basic form of NMF (see *Algorithm 1*), W and H are initialized randomly and the whole algorithm is repeated several times (*maxrepetition*). In each repetition, NMF update steps are processed until a maximum number of iterations is reached (*maxiter*). These update steps are algorithm specific and differ from one NMF variant to the other. Termination criteria: If the approximation error drops below a pre-defined threshold, or if the shift between two iterations is very small, the algorithm might stop before all iterations are processed.

Algorithm 1 *– General structure of NMF algorithms*

Multiplicative update (MU) algorithm. To give an example of the update steps for a specific NMF algorithm we provide the update steps for the MU algorithm in *Algorithm 2*. MU is one of the two original NMF algorithms presented in (Lee and Seung 1999) and still one of the fastest NMF algorithms per iteration. The update steps are based on the mean squared error objective function and consist of multiplying the current factors by a measure of the quality of the current approximation. The divisions in *Algorithm 2* are to be performed *element-wise*. ε is used to avoid division by zero ($\varepsilon \approx 10^{-9}$).

Algorithm 2 *– Update steps of the multiplicative update algorithm.*

SWARM INTELLIGENCE OPTIMIZATION

Optimization techniques inspired by swarm intelligence (SI) have become increasingly popular and benefit from their robustness and flexibility (Chiong 2009). Swarm intelligence is characterized by a decentralized design paradigm that mimics the behavior of swarms of social insects, flocks of birds, or schools of fish. Optimization techniques inspired by swarm intelligence have shown to be able to successfully deal with increasingly complex problems (Blackwell 2007). In this chapter we use five different optimization algorithms. Particle Swarm Optimization (PSO, (Kennedy and Eberhart 1995)) is a classical swarm intelligence algorithm, while Fish School Search (FSS, (Bastos Filho, Lima Neto et al. 2009)) and Fireworks Algorithm (FWA, (Tan and Zhu 2010)) are two recently developed swarm intelligence methods. These three algorithms are compared to a Genetic Algorithm (GA, (Haupt and Haupt 2005)), a classical evolutionary algorithm, and Differential Evolution (DE, (Price, Storn et al. 2005)), which shares some features with swarm intelligence but can also be considered as an evolutionary algorithm. Since PSO, GA and DE are well known optimization techniques we will not summarize them here; instead the interested reader is referred to the references given above.

Fish School Search is a recently developed swarm intelligence algorithm that mimics the movements of schools of fish. The main operators are *feeding* (fish can gain/lose weight, depending on the region they swim in) and *swimming* (there are three different swimming movements).

Algorithm 3 *– Pseudo code of the Fish School Search algorithm*

The ***Fireworks Algorithm*** is a novel swarm intelligence algorithm that is inspired by observing fireworks explosion. Two different types of explosion (search) processes are used in order to ensure diversity of resulting sparks, which are similar to particles in PSO or fish in FSS.

Algorithm 4 *– Pseudo code of the Fireworks Algorithm*

IMPROVING NMF WITH SWARM INTELLIGENCE OPTIMIZATION

Before describing our two optimization strategies for NMF based on swarm intelligence, we discuss some properties of the Frobenius norm (cf. (Berry, Drmac et al. 1999)). We use the Frobenius norm (1.1) as NMF objective function (*i.e.* to measure the error between A and WH) because it offers some properties that are beneficial for combining NMF and optimization algorithms. The following statements about the Frobenius norm are valid for any real matrix. However, in the following we assume that D refers to a distance matrix storing the distance (error of the approximation) between the original data and the approximation, $D = A - WH$. The Frobenius norm of a matrix $D \in \mathbb{R}^{m \times n}$ is defined as

$$\| D \|_F = \left(\sum_{i=1}^{min(m,n)} \sigma_i \right)^{1/2} = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} | \mathbf{d}_{ij} |^2 \right)^{1/2}, \qquad (1.2)$$

where σ_i are the singular values of D, and $\mathbf{d}_{ij}$ is the element in the i^{th} row and j^{th} column of D. The Frobenius norm can also be computed row wise or column wise. The *row wise* calculation is

$$\| D \|_F^{RW} = \left(\sum_{i=1}^{m} | \mathbf{d}_i^r |^2 \right)^{1/2}, \tag{1.3}$$

where $| \mathbf{d}_i^r |$ is the norm of the i^{th} row vector of D, *i.e.* $| \mathbf{d}_i^r | = (\sum_{j=1}^{n} | r_j^i |^2)^{1/2}$, and r_j^i is the j^{th} element in row i. The *column wise* calculation is

$$\| D \|_F^{CW} = \left(\sum_{j=1}^{n} | \mathbf{d}_j^c |^2 \right)^{1/2}, \tag{1.4}$$

with $| \mathbf{d}_j^c |$ being the norm of the j^{th} column vector of D, *i.e.* $| \mathbf{d}_j^c | = (\sum_{i=1}^{m} | c_i^j |^2)^{1/2}$, and c_i^j being the i^{th} element in column j. Obviously, a reduction of the Frobenius norm of any row or any column of D leads to a reduction of the total Frobenius norm $\| D \|_F$.

Figure 2 – *Illustration of the optimization process for row l of the NMF factor W. The l^{th}row of A ($\mathbf{a}_l^r$) and all columns of H0 are the **input** for the optimization algorithms. The **output** is a row-vector $\mathbf{w}_l^r$ (the l^{th}row of W) which minimizes the norm of $\mathbf{d}_l^r$, the l^{th}row of the distance matrix D. The norm of $\mathbf{d}_l^r$ is the fitness function for the optimization algorithms (minimization problem).*

In the following we exploit these properties of the Frobenius norm for the proposed NMF optimization strategies. While strategy 1 aims at finding heuristically optimal starting points for the NMF factors, strategy 2 aims at iteratively improving the quality of NMF during the first iterations. All meta-heuristics mentioned in Section 0 can be used within both strategies. Before discussing the optimization strategies we illustrate the basic optimization procedure for a specific row (row l) of W in Figure 2. This procedure is similar for both optimization strategies.

Parameters: Global parameters used for all optimization algorithms are upper/lower bound of the search space and the initialization, the number of particles (chromosomes, fish, ...), and maximum number of fitness evaluations. Parameter settings are discussed in Sections 0. For all meta-heuristics, the problem dimension is equal to the rank k of the NMF. *I.e.* if, for example, k = 10, a row/column vector with 10 continuous entries is returned by the optimization algorithms.

Optimization Strategy 1 – Initialization

Algorithm 5 *– Pseudo code for the initialization procedure for NMF factors W and H. The two for-loops in lines 4 and 10 can be executed concurrently. SIO = Swarm Intelligence Optimization*

The goal of this optimization strategy is to find heuristically optimal starting points for the rows of W and the columns of H, respectively, *i.e. prior* to the factorization process. *Algorithm 5* shows the pseudo code for the initialization procedure. In the beginning, $H0$ needs to be initialized randomly using a non-negative lower bound (preferably 0) for the initialization. In the first loop, W is initialized row wise, *i.e.* row $\mathbf{w}_i^r$ is optimized in order to minimize the Frobenius norm of the i^{th} row $\mathbf{d}_i^r$ of D, which is defined as $\mathbf{d}_i^r = \mathbf{a}_i^r - \mathbf{w}_i^r H0$. Since the optimization of any row of W is independent to the optimization of any other row of W, all $\mathbf{w}_i^r$ can be optimized concurrently. In the second loop, the columns of H are initialized using on the previously computed and already optimized rows of W, which need to be gathered beforehand (in line 7 of the algorithm). H is initialized column wise, *i.e.* column $\mathbf{h}_j^c$ is optimized in order to minimize the Frobenius norm of the j^{th} column $\mathbf{d}_j^c$ of D, which is defined as $\mathbf{d}_j^c = \mathbf{a}_j^c - W\mathbf{h}_j^c$. The optimization of the columns of H can be performed concurrently as well.

Optimization Strategy 2 – Iterative Optimization

Algorithm 6 *- Pseudo code for the iterative optimization for the Multiplicative Update algorithm. SIO = Swarm Intelligence Optimization. The methods used in this algorithm are explained below.*

The second optimization strategy aims at iteratively optimizing the NMF factors W and H *during the first iterations* of the NMF. Compared to the first strategy not all rows of W and all columns of H are optimized – instead the optimization is only performed on selected rows/columns. In order to improve the approximation as fast as possible we identify rows of D with highest norm (the approximation of this row is worse than for other rows of D) and optimize the corresponding rows of W. The same procedure is used to identify the columns of H that should be optimized. Our experiments showed that not all NMF algorithms are suited for this iterative optimization procedure. For many NMF algorithms there was no improvement with respect to the convergence or a reduction of the overall error after a fixed number of iterations. However, for the multiplicative update (MU) algorithm – which is one of the most widely used NMF algorithms – this strategy is able to improve the quality of the factorization. Hence, *Algorithm 6* shows the pseudo code for the iterative optimization of the NMF factors during the first iterations using the update steps of the MU algorithm described in Section 0. As shown in Section 0, this update strategy is able to significantly reduce the approximation error per iteration for the MU algorithm. Due to the relatively high computational cost of the meta-heuristics the optimization procedure is only applied in the first *m* iterations and only on *c* selected rows/columns of the NMF factors. Similar to strategy one the

optimization of all rows of W are independent from each other (identical for columns of H), which allows for a parallel implementation of the proposed method. In the following we describe the variables and functions (for updating rows of W) of *Algorithm 6*. Updating columns of H is similar to updating the rows of W .

- m: the number of iterations in which the optimization using meta-heuristics is applied
- c : the number of rows and/or columns that are optimized in the current iteration.
- Δc : the value of c is decreased by Δc in each iteration. $\Delta c = round(c_{initial} / m)$
- $[Val, IX\_W] = sort(norm(\mathbf{d}_i^r), 'descend')$: returns the values Val and the corresponding indices (IX_W) of the norm of all row vectors $\mathbf{d}_i^r$ of D in descending order.
- $IX\_W = IX\_W(1:c)$: returns only the first c elements of the vector IX_W .
- $\text{minimize} \,||\, \mathbf{a}_i^r - \mathbf{w}_i^r H \,||_F$: see Figure 2 and optimization strategy 1

Using NMF for Classification Problems

As already mentioned before, we also investigate the performance of NMF when applied for classification tasks. In this article, we use two different classification methods for evaluating the classification accuracy of NMF based on the optimization strategies discussed in Sections 0 and 0. Both classification methods have shown to work well for different application areas (Janecek 2010).

Static classification: In the first approach we analyze the classification accuracy achieved with the basis vectors (*i.e.* features in W). In this setting the NMF needs to be computed on the complete dataset (training and test data) which makes this technique only applicable on test data that is already available before the approximation/classification. However, the advantage of this approach is that any freely chosen classification method can be applied on the basis features.

If the original data matrix $A \in \mathbb{R}^{m \times n}$ is an instance × feature matrix, then the NMF factor W is a $m \times k$ matrix, where every instance is described by k basis *features, i.e.* every column of W corresponds to a basis feature. Note that this setup is different to the one discussed at the beginning of Section 0! By applying a classification algorithm on the rows of W instead on the rows of A we can significantly reduce the dimension of the classification problem and thus decrease the computational cost for both, building the classification model and testing new data.

Dynamic classification: The second approach can be applied dynamically to new data. Here the factorization of the data (NMF) and the classification process are separated from each other (*i.e.* the NMF is performed on labeled training data – the unlabeled test data does not have to be available at the time of performing the NMF). This approach is called *NMF-LSI* and is based on an adaptation of latent semantic indexing which is a variant of the well-known vector space model.

A *vector space model* (VSM, (Raghavan and Wong 1999)) is a widely used algebraic model for representing objects as vectors in a potentially very high dimensional metric vector space. The distance of a query vector **q** to all objects in a given *feature* × *instance* matrix A are usually measured in terms of the cosines of the angles between **q** and the columns of A such that $cos\varphi_i = \frac{e_i^\top A^\top q}{\| Ae_i \|_2 \| q \|_2}$.

Latent semantic indexing (LSI, (Berry, Drmac et al. 1999)) is a variant of the basic VSM that replaces the original matrix A with a low-rank approximation A_k of A. In the standard version of LSI the SVD (see Section 0) is used to construct A_k, and $cos\varphi_i$ can be approximated as $cos\varphi_i \approx \frac{e_i^\top V_k \Sigma_k U_k^\top q}{\| U_k \Sigma_k V_k^\top e_i \|_2 \| q \|_2}$. LSI has computational advantages resulting in lower storage and computational cost, and often gives a cleaner and more efficient representation of the (latent) relationship between data elements.

NMF-LSI: The approximation within LSI can be replaced with other approximations. Instead of using the truncated SVD ($A_k := U_k \Sigma_k V_k^\top$), we approximate A with $A_k := W_k H_k$ (the NMF). When using NMF, the value of *k* must be fixed prior to the approximation. The cosine of the angle between **q** and the i^{th} column of A can then be approximated as $cos\varphi_i \approx \frac{e_i^\top H_k^\top W_k^\top q}{\| W_k H_k e_i \|_2 \| q \|_2}$. In order to save computational cost, the left term in the numerator ($e_i^\top H_k^\top$) and the left part of the denominator ($\| W_k H_k e_i \|_2$) can be computed a priori. In all three methods (VSM and both LSI variants) a query instance **q** is assigned to the same class as the majority of its *k*-closest (in terms of cosine similarity) instances in A.

SETUP

Software: All software is written in Matlab. We used only publicly available NMF implementations: Multiplicative Update (MU, Matlab's Statistics Toolbox since v6.2, *nnmf()*). ALS using Projected Gradient (ALSPG, (Lin 2007)), BayesNMF and FastNMF (both (Schmidt and Laurberg 2008)). Matlab code for NNDSVD (Section 0) is also publicly available (cf. (Boutsidis and Gallopoulos 2008)). Codes for PSO and DE were adapted from (Pedersen 2010), and code for GA from the appendix of (Haupt and Haupt 2005). For FWA we used the same implementation as in the introductory paper (Tan and Zhu 2010), and FSS was self-implemented following the algorithm provided in (Bastos Filho, Lima Neto et al. 2009).

Hardware: All experiments were performed on a SUN FIRE X4600 M2 with eight AMD Opteron quad-core processors (32 cores overall) with 3.2 GHz, 2MB L3 cache, and 32GB of main memory (DDR-II 666).

Parallel implementation: We implemented parallel variants of the optimization algorithms exploiting Matlab's parallel computing potential. Matlab's Distributed Computing Server (which requires a separate license) allows for parallelizing the optimization process over a large number (currently up to 64) of workers (threads). These workers can be nodes in multi-core computers, GPUs, or a node in a cluster of simple desktop PCs. Matlab's Parallel Computing Toolbox (which is included in the basic version of Matlab) allows to run up to eight workers concurrently, but is limited to local workers, *i.e.* nodes on a multi-core machine or local GPUs, but no cluster support.

Parameter setup: The dimension of the optimization problem is always identical to the rank k of the NMF (cf. Section 0). The upper/lower bound of the search space was set to the interval $[0,(4*max(A))]$ and upper/lower bound of the initialization to $[0,max(A)]$. In order to achieve fair results which are not biased due to excessive parameter tuning we used the same parameter settings for all data sets. These parameter settings were found by running a self-written benchmark program that tested several parameter combinations on randomly generated data. For some optimization strategies (PSO, FSS and FWA) the recommended parameter settings from the literature worked fine. However, for GA and DE the parameter settings that were used in most studies in the literature did not perform very well. For GA we found that a very aggressive (high) mutation rate highly improved the results. For DE we observed a similar behavior and found that the maximum crossover probability (1) achieved the best results. For all experiments in this paper, the following parameter settings were used:

- GA: mutation rate of 0.5; selection rate of 0.65
- PSO: (G_{best} topology) following (Bratton and Kennedy 2007) ω = 0.8, and $c_1 = c_2 = 2.05$
- DE: crossover probability (pc) set to upper limit 1
- FSS: $step_{ind\_initial} = 1$, $step_{ind\_final} = 0.001$, $W_{scale} = 10$
- FWA: number of sons (*sonnum*) set to 10

Data sets: We used three different data sets to evaluate our methods. *DS-RAND* is a randomly created, fully dense 100×100 matrix which is used in order to provide unbiased results. To evaluate the proposed methods in a classification context we further used two data sets from the area of email classification (spam/phishing detection). Data set *DS-SPAM1* consists of 3000 e-mail messages described by 133 features, divided into three groups: spam, phishing and legitimate email. An exact description of this data set can be found in (Janecek and Gansterer 2010). Data set *DS-SPAM2* is the *spambase* data set taken from (Kjellerstrand 2011) which consists of 1813 spam and 2788 non-spam messages. DS-SPAM1 represents a ternary classification problem; DS-SPAM2 represents a typical binary classification problem.

EXPERIMENTAL EVALUATION

The evaluation is split up into two parts. First we evaluate the two optimization strategies proposed in Section 0 and Section 0, then we evaluate the quality of NMF in a classification context.

Evaluation of Optimization Strategy 1

Initialization: Before evaluating the improvement of the NMF approximation quality as such, we first measure the initial error after initializing W and H (*before* running the NMF algorithm). **Figure 3** and Figure 4 show the average approximation error (*i.e.* Frobenius norm / fitness) per row (left) and per column (right) for data set DS-RAND.

Figure 3 – *Left hand-side: average approximation error per row (after initializing rows of W). Right hand-side: average approximation error per column (after initializing of H). NMF rank k = 5. Legends are ordered according to approximation error (top = worst, bottom = best).*

Figure 4 – ***Similar information as for Figure 3, but for NMF rank k = 30.***

The figures on the left side show the *average* (mean) approximation error per *row* after initializing the rows of W (first loop in *Algorithm 5*). The figures on the right side show the *average* (mean) approximation error per *column* after initializing the columns of H (second loop in *Algorithm 5*). The legends are ordered according to the average approximation error achieved after the maximum number of function evaluations for each figure (top = worst, bottom = best). When the NMF rank k is small (see **Figure 3**, k=5) all optimization algorithms except FWA achieve similar results. Except FWA, all optimization algorithms quickly converge to a good result. With increasing complexity (*i.e.* increasing rank k) FWA clearly improves its results, as shown in Figure 4. The gap between the optimization algorithms is much bigger for larger rank k. Note that GA needs more than 2000 evaluations to achieve a low approximation error for initializing the rows of W. When initializing the columns of H, PSO and GA suffer from their high approximation error during the first iterations, which is caused by the relatively sparse factor matrix W for PSO and GA. Although PSO is able to reduce the approximation error significantly during the first 500 iterations, FSS and GA achieve slightly better final results. Generally, FSS achieves the best approximation accuracy after the initialization procedure for large k. However, as shown later the initial approximation error is *not necessarily an indicator* for the approximation quality of NMF or the resulting classification accuracy.

Runtime performance: When parallelizing a sequential algorithm over p processors the speed-up indicates how much the parallel algorithm can perform specific tasks faster than the sequential algorithm. Speed-up is defined as $S_p = ET_{sequential} / ET_{parallel}$, where ET is the execution time. A linear speed-up is achieved when S_p is equal to p. Efficiency is another metric that estimates how well-utilized the processors are in solving the problem, compared to the cost of communication and synchronization.

Efficiency is defined as $E_p = S_p / p$. For algorithms with linear speed-up the efficiency is 1, for algorithms with lower speed-up ratio it is between 0 and 1.

Figure 5 – *Runtime and speed-up measurement/estimation for DS-RAND using 1500 function evaluations per row/column for k= 5. As a reference, NNDSVD needs about 0.16 seconds for k=5. This indicates that if the number of workers is larger than 12, the proposed optimization strategy is faster than NNDSVD.*

Figure 5 shows the runtime behavior for optimization strategy 1 with increasing number of Matlab workers. Runtimes are shown for the FSS optimization algorithm – however, all optimization algorithms have rather similar runtimes. Due to license limitations we only had Matlab's Parallel Computing Toolbox available which is limited to 8 workers (cf. Section 0). We measured runtimes and speed-up for up to 8 workers (average efficiency of about 0.95) and estimated the behavior of speed-up and runtime for a larger number of workers (based on this efficiency). Upgrading to Matlab's Distributed Computing Server is possible without any code-changes and thus only a license issue. When using eight workers, the NNDSVD initialization (the best NMF initialization strategy from the literature, Section 0) is a bit faster, but estimation shows that the proposed initialization strategy is faster when 12 or more workers are used. NNDSVD is already optimized and cannot be parallelized further in its current implementation.

Figure 6 – *Approximation error archived by different NMF algorithms using different initialization variants (k=30, after 1500 fitness evaluations)*

Approximation quality: For evaluating the approximation results achieved by NMF using the factors W and H initialized by the optimization algorithms, we compare our results to random initialization as well as to NNDSVD. Figure 6 shows the approximation error on the *y*-axis (log scale) after a given number of NMF iterations for four NMF algorithms using different initialization methods (for DS-RAND). The initialization methods in the legend are ordered (top = worst, bottom = best). Since the MU algorithm (A) has low cost per iteration but converges slowly, the first 100 iterations are shown (for all other algorithms the first 25 iterations are shown). For MU, all initialization variants achieve a smaller approximation error than random initialization. NNDSVD shows slightly better results than PSO and FWA, but GA, DE and especially FSS are able to achieve a smaller error per iteration than NNDSVD. For ALSPG (B), the new initialization strategy achieves better results than random initialization and also achieves a better approximation error than NNDSVD. This improvement is independent of the actual optimization algorithm. The same behavior can be seen for FastNMF (C) and BayesNMF (D). It has to be mentioned that FastNMF and BayesNMF were developed after the NNDSVD initialization. Surprisingly, when using FastNMF, NNDSVD achieves a lower approximation than random initialization. When comparing the different meta-heuristics, FSS achieves the best results amongst all optimization algorithms and achieves the closest approximation after 100 (MU) and 25 (ALSPG, FastNMF, BayesNMF) iterations,

respectively. DE and GA follow with a small gap since they are not as stable as FSS (i.e. they achieve good results for some, but not for all NMF algorithms.

Evaluation of Optimization Strategy 2

Figure 7 shows the convergence curves for the NMF approximation using optimization strategy 2 for different values of rank *k* (data set DS-RAND). Due to the relatively high computational cost of the meta-heuristics we applied our optimization procedure here only on the rows of W, while the columns in H remained unchanged. Experiments showed that with this setting the loss in accuracy compared to optimizing both, W *and* H, is relatively small while the runtime can be increased significantly. m was set to 2 which indicates that the optimization is only applied in the first two iterations, and c was set to 20. As can be seen, the approximation error per iteration can be reduced when using optimization strategy 2. For small rank *k* (left side of Figure 7) the improvement is significant but decreases with increasing values of *k* (see right side of Figure 7). For larger *k* (larger than 10) the improvement over the basic MU is only marginal.

Figure 7 – *Accuracy per Iteration when updating only the row of W, m=2, c=20. Left: k=2, right: k=5*

Runtime performance: Figure 8 shows the reduction in runtime for different rank *k* when the same accuracy as for basic MU should be achieved. Runtimes are shown for a parallel implementation using 32 Matlab workers. Basic MU sets the baseline (1 = 100%), the runtimes of the optimization strategy 2 (using different optimization algorithms) are given as $t_{opt-XX} / t_{Basic\ MU}$. For example, for small rank *k* the runtime can often be reduced by more than 50%. With increasing rank *k* the runtime savings get smaller and are only marginal for *k*=10. For rank *k* larger than 12 the basic MU algorithm is faster than optimization strategy 2.

Figure 8 – *Proportional runtimes for achieving the same accuracy as basic MU after 30 iterations for different values of k when updating only the rows of W. (m=2, c=20)*

Evaluation of the Classification Accuracy

Since optimization strategy 1 (initialization, Sections 0 and 0) achieves a faster, closer, and more stable approximation as optimization strategy 2 (iterative update, Sections 0 and 0) we evaluate the classification accuracy for this strategy. In the following, we measure the quality of optimization strategy 1 as pre-processing step for the two classification approaches mentioned in Section 0. Within the *static classification approach* any machine learning algorithm can be used for classification, but the approximation used for reducing the dimensionality of the data set (SVD, PCA, NMF) needs to be applied on the complete data set. Contrary, the *dynamic classification approach* can be applied on the training data, the test data does not need to be available at the time of computing the approximation. However, this approach cannot be applied to all classification methods.

		J4.8			*k*NN(1)			SVM (SMO)		
		all features: 0,973			all features: 0,977			all features: 0,976		
NMF Alg	*Init*	*k = 30*	*k = 15*	*k = 5*	*k = 30*	*k = 15*	*k = 5*	*k = 30*	*k = 15*	*k = 5*
ALSPG	DE	0,968	0,972	0,965	0,974	0,972	0,968	**0,973**	0,956	0,940
ALSPG	FSS	0,961	0,972	0,967	0,971	0,972	**0,969**	**0,973**	0,954	0,939
ALSPG	FWA	**0,973**	0,969	**0,970**	0,972	**0,973**	0,968	0,964	0,954	0,938
ALSPG	GA	0,970	0,968	0,969	0,973	0,970	0,968	**0,973**	**0,957**	**0,947**
ALSPG	PSO	0,971	0,972	0,969	**0,977**	0,971	0,968	0,972	0,954	0,937
ALSPG	NNDSVD	0,963	**0,976**	0,964	0,969	0,972	0,968	0,966	0,952	0,938
ALSPG	RAND	*0,943*	*0,938*	*0,935*	*0,952*	*0,940*	*0,938*	*0,948*	*0,942*	*0,913*
BAYES	DE	**0,971**	0,970	0,970	0,974	0,973	0,968	0,971	0,954	0,946
BAYES	FSS	0,966	**0,973**	**0,971**	**0,976**	0,971	0,969	**0,975**	0,953	**0,947**
BAYES	FWA	0,970	0,970	0,968	0,972	**0,974**	0,968	0,957	0,954	0,941
BAYES	GA	0,966	0,971	0,968	0,974	0,973	0,969	0,972	0,955	**0,947**
BAYES	PSO	0,968	0,967	0,969	0,970	0,971	**0,970**	0,966	**0,957**	0,937
BAYES	NNDSVD	0,968	0,972	0,968	0,970	0,973	0,969	0,966	0,952	0,947
BAYES	RAND	*0,952*	*0,941*	*0,953*	*0,961*	*0,951*	*0,947*	*0,958*	*0,937*	*0,926*
FAST	DE	0,966	0,969	0,969	**0,977**	0,973	0,968	0,970	0,955	0,946
FAST	FSS	0,967	0,971	**0,970**	0,976	0,971	0,969	**0,975**	0,953	**0,947**
FAST	FWA	**0,968**	0,970	0,969	0,971	**0,974**	0,968	0,957	0,954	0,941
FAST	GA	0,966	0,965	0,968	0,973	0,971	0,969	0,973	0,955	**0,947**
FAST	PSO	**0,968**	0,970	**0,970**	0,974	0,971	**0,970**	0,973	**0,956**	0,937
FAST	NNDSVD	0,966	**0,973**	**0,970**	0,970	0,973	0,968	0,966	0,952	0,939
FAST	RAND	*0,954*	*0,949*	*0,937*	*0,958*	*0,951*	*0,941*	*0,957*	*0,935*	*0,917*
MU	DE	0,955	0,952	0,965	0,966	0,959	0,968	0,962	0,953	0,940
MU	FSS	**0,965**	0,960	0,967	0,967	0,964	**0,969**	**0,966**	0,952	0,939
MU	FWA	0,949	0,956	**0,970**	0,964	0,966	0,968	0,959	**0,955**	0,938
MU	GA	0,954	0,961	0,969	0,966	0,966	0,968	0,961	0,944	**0,947**
MU	PSO	0,958	0,939	0,969	0,949	0,946	0,968	0,953	0,940	0,937
MU	NNDSVD	0,964	**0,967**	0,964	**0,972**	**0,973**	0,968	0,963	0,954	0,938
MU	RAND	*0,941*	*0,937*	*0,947*	*0,948*	*0,941*	*0,951*	*0,951*	*0,930*	*0,927*

Table 1 – *Classification results (static classification) for DS-SPAM1*

Static classification: We used three classification algorithms from the freely available WEKA toolkit (Witten and Frank 2005) to compare the classification accuracies achieved with the NMF factor W based on different NMF initializations: A support vector machine (SVM) based on the sequential minimal optimization (SOM) algorithm using a polynomial kernel with an exponent of 1; a k-nearest neighbor (*k*NN) classifier; and a J4.8 decision tree based on the C4.5 decision tree algorithm. Results were achieved using a 10-fold cross-validation, *i.e.* by randomly partitioning the data sets into 10 subsamples and then iteratively using one 9 subsamples as training data and 1 for testing.

Table 1 shows the overall classification results achieved with data set DS-SPAM1 using three different values of rank *k* and the three different classification methods mentioned above. The overall classification accuracy is computed as the number of correct classified email messages divided by the total number of messages. The most-left column indicates the NMF algorithm and the second column the initialization strategy used for computing the NMF (RAND = random initialization). Note that the number of features is reduced to 30, 15 and 5, respectively, compared to 133. This reduction in the number of features significantly speeds up both, the process of building the classification model and the classification process itself. The best result for each NMF algorithm and each rank *k* is highlighted in bold letters. The proposed initialization strategies achieve better classification results as the state-of-the-art initialization method NNDSVD and significantly better results as random NMF initialization. Among the applied optimization algorithms there is not much difference, though FSS achieves a larger number of best results then the other algorithms. Results for J4.8 and *k*NN are very stable even for *k*=5 and are almost identical to the classification result achieved with all features. For SVM, the classification result tends to decrease with decreasing rank *k*. This behavior has been observed in another study (Janecek, Gansterer et al. 2008) where SVM has been applied on data sets from other dimensionality reduction methods (PCA). However, compared to NNDSVD and random initialization the proposed initialization methods achieve better results for all ranks of *k*. Comparing the different NMF algorithms it can be seen the MU achieves lower classification accuracy compared to ALSPG, FastNMF and BayesNMF.

		J4.8			*k*NN(1)			SVM (SMO)		
		all features: 0,921			all features: 0,907			all features: 0,904		
NMF Alg	*Init*	*k = 30*	*k = 15*	*k = 5*	*k = 30*	*k = 15*	*k = 5*	*k = 30*	*k = 15*	*k = 5*
FAST	DE	0,918	0,893	0,863	**0,902**	0,880	0,821	**0,905**	0,865	0,798
FAST	FSS	0,920	**0,920**	0,773	0,895	0,889	0,826	0,894	0,880	0,773
FAST	FWA	0,916	0,916	0,864	0,887	**0,898**	0,797	0,893	0,885	0,757
FAST	GA	0,918	0,914	0,865	0,889	0,896	0,827	0,896	**0,891**	0,778
FAST	PSO	**0,921**	0,911	**0,878**	0,895	0,892	**0,850**	0,896	0,881	**0,827**
FAST	NNDSVD	0,919	0,911	0,811	0,895	0,894	0,816	0,894	0,882	0,766
FAST	RAND	*0,907*	*0,908*	*0,813*	*0,885*	*0,886*	*0,803*	*0,887*	*0,864*	*0,752*

Table 2 – *Classification results (static classification) for DS-SPAM2 (FastNMF)*

Table 2 shows the static classification results achieved with data set DS-SPAM2. Results are shown for the FastNMF, which achieved the most stable results of all NMF algorithms for this data set. Again, the proposed initialization strategy again achieves better results as NNDSVD and random initialization. Compared to DS-SPAM1, the results for this data set tend to decrease with decreasing rank *k*. This indicates that it is important to find a good trade-off between classification accuracy and computational cost.

Dynamic classification: Table 3 shows the classification results achieved with the dynamic classification approach described in Section 0 for DS-SPAM1. In general, the classification accuracies achieved for

data set DS-SPAM2 using the dynamic classification approach are rather similar to the results for DS-SPAM1 shown in Table 3. The baseline to which the NMF-LSI variants are compared are given by a standard LSI classification using SVD as approximation algorithm (see Section 0). A basic vector space model achieves a classification accuracy of 0.911, while LSI achieves 0.911, 0.914 and 0.887, respectively, for rank *k* set to 30, 15 and 5. Similar to Table 2 (DS-SPAM2) the results are sensible with respect to the value of rank *k*. For very small values of k (5) the classification results generally tend to decrease. Overall, the initialization strategy based on meta-heuristics achieve much better classification accuracy as NNDSVD and random initialization, and also outperform basic LSI in many cases. The best results are again highlighted in bold letters. Especially GA and FWA achieve good classification results.

Baseline	*LSI*	*0,911*	*0,914*	*0,887*		*LSI*	*0,911*	*0,914*	*0,887*
NMF Alg	*Init*	*k = 30*	*k = 15*	*k = 05*	*NMF Alg*	*Init*	*k = 30*	*k = 15*	*k = 05*
ALSPG	*DE*	*0,911*	*0,898*	*0,889*	*FAST*	*DE*	*0,912*	*0,895*	*0,888*
ALSPG	*FSS*	***0,943***	*0,899*	*0,877*	*FAST*	*FSS*	*0,926*	*0,897*	*0,879*
ALSPG	*FWA*	*0,930*	***0,914***	*0,883*	*FAST*	*FWA*	*0,913*	*0,912*	***0,891***
ALSPG	*GA*	*0,927*	*0,901*	***0,896***	*FAST*	*GA*	***0,927***	***0,914***	*0,875*
ALSPG	*PSO*	*0,918*	*0,889*	*0,885*	*FAST*	*PSO*	*0,923*	***0,914***	*0,847*
ALSPG	*NNDSVD*	*0,914*	*0,911*	*0,840*	*FAST*	*NNDSVD*	*0,911*	*0,913*	*0,846*
ALSPG	*RAND*	*0,901*	*0,886*	*0,874*	*FAST*	*RAND*	*0,898*	*0,899*	*0,838*
BAYES	*DE*	*0,911*	*0,906*	*0,888*	*MU*	*DE*	*0,893*	*0,897*	*0,834*
BAYES	*FSS*	*0,926*	*0,897*	*0,879*	*MU*	*FSS*	*0,892*	*0,882*	*0,807*
BAYES	*FWA*	*0,914*	*0,911*	***0,891***	*MU*	*FWA*	***0,913***	*0,882*	***0,843***
BAYES	*GA*	***0,930***	***0,916***	*0,875*	*MU*	*GA*	*0,899*	*0,899*	*0,795*
BAYES	*PSO*	*0,922*	*0,915*	*0,848*	*MU*	*PSO*	*0,922*	*0,900*	*0,812*
BAYES	*NNDSVD*	*0,904*	*0,913*	*0,846*	*MU*	*NNDSVD*	*0,906*	***0,908***	*0,795*
BAYES	*RAND*	*0,898*	*0,896*	*0,854*	*MU*	*RAND*	*0,876*	*0,889*	*0,817*

Table 3 – *Dynamic Classification using DS-SPAM1. Basic Vector Space Model (all features): 0,911*

CONCLUSION

In this chapter we presented two new optimization strategies for improving the NMF using optimization algorithms based on swarm intelligence. While strategy one uses swarm intelligence algorithms to initialize the factors W and H *prior* to the factorization process of NMF, the second strategy aims at iteratively improving the approximation quality of NMF *during* the first iterations of the factorization. Overall, five different optimization algorithms were used for improving NMF: Particle Swarm Optimization (PSO), Genetic Algorithms (GA), Fish School Search (FSS), Differential Evolution (DE), and Fireworks Algorithm (FWA).

Both optimization strategies allow for efficiently computing the optimization of single rows of W and/or single columns of H in parallel. The achieved results are evaluated in terms of accuracy per runtime and per iteration, final accuracy after a given number of NMF iterations, and in terms of the classification

accuracy achieved with the reduced NMF factors when being applied for machine learning applications. Especially the initialization strategy (optimization strategy 1) is able to significantly improve the approximation results of NMF compared to random initialization and state-of-the-art methods. Among the different optimization algorithms, the recently developed fish school search algorithm achieves slightly better results than the other heuristics. The iterative strategy (optimization strategy 2) can improve one of the basic NMF algorithms (the multiplicative update strategy) for very small rank k and can thus be used if a rough and very fast approximation method is needed. Moreover, the NMF subsets achieved with optimization strategy 1 have shown to clearly improve the classification accuracy of NMF compared to state-of-the-art initialization strategies, and also achieve better results as feature subsets computed with other low-approximation techniques.

Future work: Our investigations provide several important and interesting directions for future work. First of all, we will set the focus on developing optimization strategies that update the factor matrices W and H concurrently instead of applying an alternating update fashion where one factor is fixed and the other one is optimized. Moreover, we will apply the optimization strategies on NMF problems were sparseness constraints are enforced, i.e. the optimization strategies are enforced to compute solutions with a certain percentage of zero values. We also plan to use different NMF optimization functions (not based on the Frobenius norm) for our optimization methods and several recently developed NMF algorithms (HALS, multilayer NMF, etc.).

Acknowledgments: This work was supported by National Natural Science Foundation of China (NSFC), Grant No. 61375119, 61170057 and 60875080. Andreas wants to thank the *Erasmus Mundus External Coop. Window*, Lot 14 (2009-1650/001-001-ECW).

REFERENCES

Bastos Filho, C. J. A., F. B. Lima Neto, A. J. C. Lins, A. I. S. Nascimento and M. P. Lima (2009). Fish School Search. In R. Chiong (Ed.), *Nature-Inspired Algorithms for Optimisation* (pp. 261-277), Springer, .

Bastos Filho, C. J. A., F. B. Lima Neto, M. F. C. Sousa, M. R. Pontes and S. S. Madeiro (2009). On the influence of the swimming operators in the Fish School Search algorithm. *In Proceedings of Systems, Man and Cybernetics* (pp. 5012-5017), San Antonio, TX: IEEE.

Berry, M. W. (1992). Large Scale Singular Value Computations. *International Journal of Supercomputer Applications*, 6(1), 13-49.

Berry, M. W., M. Browne, A. N. Langville, P. V. Pauca and R. J. Plemmons (2007). Algorithms and Applications for Approximate Nonnegative Matrix Factorization. *Computational Statistics & Data Analysis*, 52(1), 155-173.

Berry, M. W., Z. Drmac and E. R. Jessup (1999). Matrices, Vector Spaces, and Information Retrieval. *SIAM Review*, 41(2), 335-362.

Blackwell, T. (2007). Particle Swarm Optimization in Dynamic Environments. In Yang et al. (Eds.) *Evolutionary Computation in Dynamic and Uncertain Environments* (pp. 29-49). Berlin, Heidelberg: Springer

Boutsidis, C. and E. Gallopoulos (2008). SVD based Initialization: A Head Start for Nonnegative Matrix Factorization. *Pattern Recogn,* 41(4), 1350-1362.

Bratton, D. and J. Kennedy (2007). Defining a Standard for Particle Swarm Optimization. *In Proceedings of Swarm Intelligence Symposium*, (pp. 120-127). Honolulu, HI:IEEE

Chiong, R. (2009). *Nature-Inspired Algorithms for Optimisation*, Berlin, Heidelberg: Springer.

Dai H, and X. Wang and H. HU and Wang Y. (2013). Nonsmooth Nonnegative Matrix Factorization Algorithm Based on Particle Swarm Optimization. *Computer Engineering*, 39(1), 204-207.

Eberhart, R. C., Y. Shi and J. Kennedy (2001). *Swarm Intelligence*, San Francisco, CA:Morgan Kaufmann.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Boston, MA: Addison-Wesley Longman.

Haupt, R. L. and S. E. Haupt (2005). *Practical Genetic Algorithms (2nd ed.)*, Hoboken, NJ: John Wiley & Sons, Inc.,

Janecek, A. (2010). Efficient Feature Reduction and Classification Methods: Applications in Drug Discovery and Email Categorization, PhD Thesis, University of Vienna.

Janecek, A. and W. N. Gansterer (2010). Utilizing Nonnegative Matrix Factorization for E-mail Classification Problems. In M. W. Berry (Ed.) *Survey of Text Mining III: Application and Theory,*(pp. 57-80), Hoboken, NJ: John Wiley & Sons, Inc

Janecek, A., W. N. Gansterer, M. Demel and G. F. Ecker (2008). On the Relationship Between Feature Selection and Classification Accuracy. *JMLR: Workshop and Conference Proceedings,* 4(1), 90–105.

Janecek, A., S. Schulze-Grotthoff and W. N. Gansterer (2011). libNMF - A library for nonnegative matrix factorizatrion. *Computing and Informatics*, 30(2), 205-224.

Janecek, A. and Y. Tan (2011). Iterative Improvement of the Multiplicative Update NMF Algorithm using Nature-inspired Optimization. *In Proceedings of the 7th International Conference on Natural Computation* , (pp. 1668-1672), Shanghai, China: IEEE.

Janecek, A. and Y. Tan (2011). Using Population Based Algorithms for Initializing Nonnegative Matrix Factorization. In Proceedings of the 2nd International Conference on Swarm Intelligence, Part II, LNCS 67239, (pp. 307-316), Chongqing, China: Springer.

Jolliffe, I. T. (2002). *Principal Component Analysis*, New York: Springer.

Kennedy, J. and R. C. Eberhart (1995). Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks*, (pp. 1942-1948), Perth, WA:IEEE.

Kim, H. and H. Park (2008). Nonnegative Matrix Factorization Based on Alternating Nonnegativity Constrained Least Squares and Active Set Method. *SIAM J. Matrix Anal. Appl.* 30(1), 713-730.

Kjellerstrand, H. (2011). "hakanks hemsida.". Retrieved from http://hakank.org/weka

Langville, A. N., C. D. Meyer and R. Albright (2006). Initializations for the Nonnegative Matrix Factorization. *In Proceedings of the 12th ACM Int. Conf. on Knowledge Discovery and Data Mining,* (pp. 1-18), Philadelphia, PE: ACM

Lee, D. D. and H. S. Seung (1999). Learning Parts of Objects by Non-negative Matrix Factorization. *Nature*, 401(6755), 788-791.

Lee, D. D. and H. S. Seung (2001). Algorithms for Non-negative Matrix Factorization. *Advances in Neural Information Processing Systems,* 13(1), 556-562.

Li, H., J. Yang and C. Hao (2013). Non-negative Matrix Factorization of Mixed Speech Signals based on Quantum-behaved Particle Swarm Optimization. *Journal of Computational Information Systems,* 9(2), 667- 673.

Lin, C.-J. (2007). Projected Gradient Methods for Nonnegative Matrix Factorization. *Neural Comput.* 19(10), 2756-2779.

Paatero, P. and U. Tapper (1994). Positive Matrix Factorization: A Non-negative Factor Model With Optimal Utilization of Error Estimates of Data Values. *Environmetrics,* 5(2), 111-126.

Pedersen, M. E. H. (2010). "SwarmOps - Numeric & Heuristic Optimization Source-Code Library, Retrieved from: http://hvass-labs.org/projects/swarmops/cs/files/SwarmOpsCS1_0.pdf

Price, K. V., R. M. Storn and J. A. Lampinen (2005). *Differential Evolution: A Practical Approach to Global Optimization*, Secaucus, NJ: Springer.

Raghavan, V. V. and S. K. M. Wong (1999). A Critical Analysis of Vector Space Model for Information Retrieval. *Journal of the American Society for Information Science*, 37(5), 279-287.

Schmidt, M. N. and H. Laurberg (2008). Non-negative matrix factorization with Gaussian process priors. *Comp. Intelligence and Neuroscience*, 2008(1), 1-10.

Snásel, V., J. Platos and P. Krömer (2008). Developing Genetic Algorithms for Boolean Matrix Factorization. In Proceedings of the Dateso 2008 Annual International Workshop on DAtabases, TExts, Specifications and Objects, (pp. 1-10), Desna, Czech Republic: CEUR-WS.org

Stadlthanner, K., D. Lutter, F. Theis and et al. (2007). Sparse Nonnegative Matrix Factorization with Genetic Algorithms for Microarray Analysis. *In Proceedings of the International Joint Conference on Neural Networks*, (pp. 294-299), Orlando, FL: IEEE.

Storn, R. and K. Price (1997). Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimizatio,n* 11(4), 341-359.

Tan, P.-N., M. Steinbach and V. Kumar (2005). *Introduction to Data Mining*, Boston, MA: Addison Wesley.

Tan, Y. and Y. Zhu (2010). Fireworks Algorithm for Optimization. In Tan et al. (Eds), *Advances in Swarm Intelligence* (pp.355-364), Beijing, China: Springer.

Wild, S. M., J. H. Curry and A. Dougherty (2004). Improving non-negative matrix factorizations through structured initialization. *Patt. Recog,.* 37(11), 2217-2232.

Witten, I. H. and E. Frank (2005). *Data Mining: Practical Machine Learning Tools and Techniques*, San Francisco, CA: Morgan Kaufmann.

Xue, Y., C. S. Tong, Y. Chen and W. Chen (2008). Clustering-based initialization for non-negative matrix factorization. *Appl. Math. & Comput.*, 205(2), 525-536.

Zhang, Q., M. W. Berry, B. T. Lamb and T. Samuel (2009). A Parallel Nonnegative Tensor Factorization Algorithm for Mining Global Climate Data. *In Proceedings of the 9th International Conference on Computational Science*, (pp. 405-415), Berlin Heidelberg: Springer

DEFINITION OF KEYTERMS

Non-negative Matrix Factorization: The Non-negative Matrix Factorization (NMF, (Lee and Seung 1999)) leads to a low-rank approximation which satisfies non-negativity constraints by approximating a data matrix A by $A \approx WH$, where W and H are the *NMF factors*. NMF requires *all entries* in A, W and H to be zero or positive.

NMF Initialization: Algorithms for computing NMF are iterative and require initialization of the factors W and H since NMF unavoidably converges to local minima, probably different ones for different initialization. Contrary to random initialization, a proper non-random initialization can lead to faster error reduction and better overall error at convergence.

Differential Evolution (DE): In DE, (Storn and Price 1997, Price, Storn et al. 2005) a particle is moved around in the search-space using simple mathematical formulation, if the new position is an improvement the particles' position is updated, otherwise the new position is discarded.

Particle Swarm Optimization (PSO): In PSO, (Kennedy and Eberhart 1995) each particle in the swarm adjusts its position in the search space based on the best position it has found so far as well as the position of the known best fit particle of the entire swarm.

Genetic Algorithms (GA): GAs, (Goldberg 1989) are global search heuristics that operate on a population of solutions using techniques encouraged from evolutionary processes such as mutation, crossover, and selection.

Fireworks Algorithm (FWA): FWA, (Tan and Zhu 2010) is a recently developed swarm intelligence algorithm that simulates the explosion process of fireworks. Two types of sparks are generated, based on uniform and Gaussian distribution, respectively.

Fish School Search (FSS): FSS, (Bastos Filho, Lima Neto et al. 2009, Bastos Filho, Lima Neto et al. 2009)is based on the behavior of fish schools. The main operators are *feeding* (fish can gain/lose weight, depending on the region they swim in) and *swimming* (which mimics the collective movement of all fish).

Multiplicative Update (MU) NMF algorithm. MU-NMF is one of the two original NMF algorithms presented in (Lee and Seung 1999) and still one of the fastest NMF algorithms per iteration. The update steps are based on the mean squared error objective function and consist of multiplying the current factors by a measure of the quality of the current approximation.

INDEX

Fireworks Algorithm, FWA

Fish School Search, FSS