Speeding up tensor based recommenders with clustered tag space and improving quality of recommendations with Non-negative tensor factorization

Martin Leginus and Valdas Žemaitis

Department of Computer Science, Aalborg University,
Selma Lagerlofs Vej 300,
9220 Aalborg East, Denmark
{mlegin09, vzemai09}@student.aau.dk

Resume

Social tagging facilitates users to annotate and categorize items (Web links, pictures, movies, etc.). Assigned tags express the user preferences. Social tagging systems (STS) have become popular hence various tag based recommenders have been developed. The state-of-the-art STS model three types of entities (i.e. tag-user-item) and relationships between them are encoded into a 3-order tensor. Latent relationships and patterns can be discovered by applying tensor factorization techniques like Higher Order Singular Value Decomposition (HOSVD), Canonical Decomposition (CD), etc. STS accumulate large amount of data that significantly slows down the process of a tensor factorization.

Firstly, we propose to reduce tag space by exploiting clustering techniques so that execution time is improved and memory requirements are decreased while preserving the quality of the recommendations. The clustering is motivated by the fact that many tags in a tag space are semantically similar thus the tags can be grouped. Two approaches of computing tags similarities are investigated. The former one utilizes tag pair cooccurrence similarity measures. The latter expresses tags as feature vectors and uses standard distance measures. The best prediction quality is achieved with the Spectral K-means clustering that employs cooccurrence tag pair similarity. The best trade-of between prediction accuracy and execution time is when the number of clusters equals to 50% of the original tag space size.

Secondly, we propose to incorporate the personal prior knowledge to increase the precision of tensor based recommenders. In addition, we take an advantage of Non-negative Tensor Factorization (NTF) to get rid of negative values from the factorized tensor that are difficult to interpret. Possibility to run NTF in parallel is explored and the minor improvement of the execution time is achieved.

Finally, we combine all the approaches to improve the quality and time of computations and present the promising experimental results.

Speeding up tensor based recommenders with clustered tag space and improving quality of recommendations with Non-negative tensor factorization

Martin Leginus* Valdas Žemaitis*

Abstract

Social tagging facilitates users to annotate and categorize items (Web links, pictures, movies, etc.). Assigned tags express the user preferences. Social tagging systems (STS) have become popular hence various tag based recommenders have been developed. The state-of-the-art STS model three types of entities (i.e. tag-user-item) and relationships between them are encoded into a 3-order tensor. Latent relationships and patterns can be discovered by applying tensor factorization techniques like Higher Order Singular Value Decomposition (HOSVD), Canonical Decomposition (CD), etc. STS accumulate large amount of data that significantly slows down the process of a tensor factorization. Firstly, we propose to reduce tag space by exploiting clustering techniques so that execution time is improved and memory requirements are decreased while preserving the quality of the recommendations. The clustering is motivated by the fact that many tags in a tag space are semantically similar thus the tags can be grouped. Secondly, we propose to incorporate the personal prior knowledge to increase the precision of tensor based recommenders. In addition, we take an advantage of Non-negative Tensor Factorization (NTF) to get rid of negative values from the factorized tensor that are difficult to interpret. Finally, we combine all the approaches to improve the quality and time of computations and present the promising experimental results.

Keywords: personal tensor factorization, tags, higher order singular value decomposition, non-negative tensor factorization, clustering, prior knowledge, parallel

1 Introduction

Collaborative tagging has become an important and popular way of categorizing and retrieving various content within different Web 2.0 sites. Due to the simplicity and massive users engagement, tagging [41] has been incorporated into the leading social web systems such as Facebook, Flickr, Delicious. A user has possibility to classify and characterize a particular content item by annotating it with an arbitrary term – tag. This allows a convenient retrieval and searching for the content according to a defined tag.

A tagging activity expresses additional knowledge about user interests about the item [32]. These implicit ratings (or voting) can be used to extend the capabilities of the state-of-the-art recommendation systems based on explicit ratings [15]. A collaborative tagging enables a construction of tag-based users profiles which can be utilized for personalised content recommendations. There are different families of systems that can compute the recommendation purely based on tag profiles of users. The promising tag-based recommenders are based on tensor factorization [36, 44] which are a concern of our study.

Tensor based recommenders build 3-dimensional matrix (tensor) by reflecting relationships between all users, items and tags from STS. Afterwards, a factorization technique is performed on the constructed tensor. The tensor approximation usually reveals latent relations between the involved objects. The main advantages of these recommenders are as follows:

- Outperforms other tag-aware state-of-the-art recommendation algorithms like: Fusion [46], Item-based collaborative filtering [51] and for tag recommendations FolkRank [23] as was shown in [44] and [36].
- Generates recommendations of items, tags or users from the same approximated tensor [44] – it is enough to perform a factorization only once for item, tag or user recommendations. Most often items are recommended to a user. Tag recommendations can be utilized to suggest possible tags when tagging is performed.
- Takes into account all three dimensions i.e., items, users and tags altogether, in comparison with algorithms [46] that split 3-dimensional data into three 2-dimensional relations {user, item}, {user, tag},

*Department of Computer Science, Aalborg University, Selma Lagerlofs Vej 300, 9220 Aalborg East, Denmark. Email: {mlegin09, vzemai09}@student.aau.dk

{tag, item} – such transformation causes a partial loss of interactions between objects that were captured by the 3-dimensional datastructure e.g., a particular tag has different meanings for different users, users have different preferences for items and items have different facets.

However, there are many practical difficulties that restrict usage of tensor based recommenders in real world application. Below is a list the most significant problems that are addressed in this paper:

- A factorization is computationally demanding process and most of the tensor based recommenders [36, 44] calculate tensor approximation in the offline mode. When new users, items or taggings are inserted into a system there is a need to recompute tensor approximation so the appropriate recommendations are generated. However, there are proposed extensions like *folding-in* [39] and *incremental Singular Value Decomposition* (SVD) [7] which enable less frequent recomputations of a tensor factorization. The former one is suitable only when the size of update is small enough otherwise the space will not be orthogonal which negatively affects recommendation results. The latter one is computationally faster than standard batch SVD. The main limitations of these extensions are that they can be applied only to HOSVD. Other factorization techniques like Non-negative Tensor Factorization (NTF), Ranking with Tensor Factorization (RTF) [36] and Canonical decomposition need to fully recompute tensor approximation when new objects enter the STS.

- Excessive memory demands when large datasets are used.

- Final factorized tensor may contain negative values that are confusing and hardly interpretable when generating recommendations. This decreases the accuracy of predictions.

1.1 Contributions

Above mentioned problems are addressed in this paper with exploitation of the non-negative tensor factorization and clustering techniques that reduce tag space and improve time performance of the approximation process. The main novel aspects of proposed techniques are described below.

1.1.1 Non-negative tensor factorization

NTF algorithm [16], [9] is adjusted and utilized in this work. The main benefits of this approach are:

- NTF generates a factorized tensor with positive values only – an interpretation of such values is clear and straightforward. The majority of other factorization methods produce an approximated tensor that can contain negative values. The negative values are omitted because they signal the recommendation is not relevant for the user. But it is not always true, there exist cases when negative values make sense and should be used while constructing the list of the recommendations [30, 48]. Unfortunately, there is no rule how to handle negative values. The NTF eliminates this problem and therefore it is simple to interpret the approximated tensor.

- A prior knowledge about the user interests and preferences can be incorporated to achieve a higher prediction accuracy. The factor matrices can be constructed using random numbers or using a specific algorithm to encode the prior knowledge about the user. This helps to exploit the fact that users future interests should remain related to the prior interests.

- We explore whether the time needed to compute the approximated tensor can be reduced by implementing a simple approach – to calculate the factor matrices in parallel. This helps to improve the overall time performance of factorization process and the precision stays the same. The current trend in computers hardware implies that the future computers will not have more powerful processor (CPU) but will have a higher number of CPUs installed on a single machine. The NTF algorithm with the possibility to be executed in parallel helps to utilize all computational power of the modern computer hardware.

1.1.2 Reducing tag space with clustering

The factorization is computationally demanding and when new users, items or tags enter the system, there is a need to recompute tensor approximation such that new objects will be incorporated into the tensor. This procedure is time demanding and it prevents often recomputations of approximated tensor and as a consequence not relevant recommendations are generated. Our approach addresses this issue by clustering a tag space. Majority of the STS contain a lot of semantically related tags (rarely used tags are commonly just extensions of more frequent ones) and they can be grouped. Hence, this reduces the size of the tag space. Clustering of a tag space results into the following improvements:

- The size of the tensor is smaller so that time performance of a tensor approximation is better.

- Memory requirements are significantly decreased.

- The similar accuracy of recommendations is preserved.
- The factorization can be recomputed more often and recommendations will embrace the new entered objects. Moreover, the approach can be also combined with the *folding-in* and *incremental SVD* methods for HOSVD factorization.

To the best of our knowledge, we are first that introduce clustering of tag space to reduce a dimension of the tensor to improve execution time of the factorization.

In this paper, we consider only recommending items to a particular user according to tags that he has in common with other similar users. The proposed NTF factorization can generate either tag or user recommendations from the same approximated tensor. Clustering a tag space does not restrict user or item recommendations. However, a certain post-filtering method has to be applied for tags prediction as there is a need to appropriately extract tags from clusters.

1.2 Outline

The structure of the paper is organized as follows. In Section 2 are introduced different tensor factorization and clustering techniques. The principles of HOSVD and NTF factorization are described in the section 3. In Section 4 is explained the extended NTF factorization that incorporates a prior knowledge and can be executed in parallel. Also, the approach of reducing tag space by clustering to improve time performance of tensor factorization. In Section 5 are presented results of experiments and in the last section is summarized the whole work.

2 Related work

2.1 Recommendation systems based on tensor factorization

Symenonidis et al. [43] introduce recommendation algorithm based on Higher Order Singular Value Decomposition (HOSVD) where the relationships between users, items and tags are represented in three dimensional matrix called tensor. Each tagging activity for a given item from a particular user is represented by value 1 in the initial tensor, all other cases are represented with 0. The core idea is the factorization of an initial tensor utilizing the HOSVD to obtain an approximated tensor which reveals the latent relationships and patterns of the users. The tensor is split into three mode matrices by applying different perspectives to the initial tensor. The Singular Value Decomposition (SVD) [29] is the factorization method used for all the three mode matrices. The approximated tensor is computed by multiplying the results from SVD of the mode matrices. The recommendations are obtained from the approximated tensor by inspecting the entries that belong to a given user, item and tag. We utilize this recommendation system and use it as the baseline for the comparison of precision and time performance with our proposed techniques.

[27] proposes another tensor based system – called multiverse recommendation, which employs HOSVD technique and supports relations between users, item and more contextual objects. Context can be expressed as a number of variables (like time, location, tag) hence the method is also called as N-dimensional tensor factorization. For the simplicity authors present only a model with a single contextual variable – it is only 3-dimensional tensor factorization. Authors define objective function which consists of loss function and regularization part, the final tensor is approximated by the optimization of the objective function. The factorized tensor can contain negative values that can be hardly interpretable. As authors propose to incorporate more contextual variables – it also enlarges tensor dimensions therefore, we consider our clustering solution suitable to group space of a particular textual context variable. Rendel et al. [36] present tag recommender – Ranking with Tensor Factorization (RTF). The system exploits a custom post-based ranking interpretation scheme and utilizes a similar factorization technique as HOSVD. The main differences between our two considered factorization techniques (the baseline HOSVD based algorithm and proposed NTF factorization) and the RTF are:

- The interpretation scheme (function which is used when the initial tensor is created) is different. Authors point out the three possible situations – a positive case (a user tagged a particular item); negative case (a user has explored a particular item but has not tagged it with a given tag); not observed case (a user has not explored a given item yet). This interpretation is semantically more accurate and provides better recommendation results. However, the interpretation constraints must be trained.
- As the interpretation constraints have to be trained, the outcome from this learning process are the model parameters – matrices that are used for the computation of the approximated tensor.

In our work, we use a naive interpretation scheme proposed by [43] instead of incorporating the post based ranking. We avoid the latter one approach because it generates only tag recommendations. Integrating

clustering with the RTF would require additional adjustments of the post-based ranking interpretation scheme.

Canonical Decomposition (CD) also known as Parallel Factor Analysis (PARAFAC) is widely used in the different research areas – chemometrics, psychometrics and signal processing [10]. The main advantage in comparison to the Tucker Decomposition [47] is linear computation time for a construction of factorized tensor as the core tensor is diagonal. A factorized tensor is approximated according to the three component matrices also called as loading factors. These matrices have to be trained, the most common algorithm for the learning is Alternating Least Squares (ALS) [45]. The components are estimated iteratively until the solution converges. However, training can be computationally demanding.
The NTF algorithm, we implemented, is based on PARAFAC, therefore the construction time of the tensor is linear and it can be controlled by adjusting the size of factors. However, the usage of too small factors can result in lower precision. The impact of this issue can be minimized by incorporating the prior knowledge instead of randomly generating factor matrices – therefore, the size of the factor matrices can be reduced with the improved time performance and with preserved quality of the recommendations.

The problem of the mentioned methods is negative values in the final approximated tensor. In the following paragraph, we describe NTF based techniques that produce only positive weights in the factorized tensor and approximate the initial tensor within the acceptable error rate [10].

The approach presented by [16] is based on PARAFAC decomposition using a diagonal core tensor. The factor matrices are initialized using random numbers r where $r \in (0, 1)$. The NTF algorithm [16] uses Bound-Constrained Linear least-Squares solver (BCLS) [6] that is an extension of ALS. The main purpose of this solver is to restrict the range of values (which approximate initial tensor) in factor matrix and to minimize the rest of the values (which are not significant for the approximation) – in such way the factor matrices are trained (the important values are purified while the rest of them are minimized). Also, it is known [16], [52] that each column of the factor matrices can be trained in parallel.

FacetCube [9] is the recommendation system based on NTF and it extends NTF with incorporating prior knowledge of the users. The domain ontology is constructed for a particular user to reduce the search space of the recommendations. Such approach offers 3 distinct modes of the tensor approximation: with unrestricted search space (no prior knowledge is incorporated); from a sub-space which is determined by the prior knowledge of the users and a fixed space that is provided by the users. Our approach does not restrict the search space but promotes the facts of a user behaviour from the past. Therefore the obtained recommendations capture not only the preferences of a user but also the predictions based on the revealed latent relations.

Authors of [31] incorporate a lexical prior knowledge into a non-negative matrix tri-factorization to perform sentiment analysis – automatic identification whether a given web item is considered as positive or negative one. The technique uses a context-independent sentiment lexicon (positive or negative textual expressions i.e., bad, great, etc.) to construct the prior knowledge about a certain users query (e.g. when searching for some product, movie etc.). The lexical prior knowledge is incorporated into two matrices – matrix of prior knowledge for words and for documents. The approach can be utilized for various sentiment analysis tasks. Our technique is different as we analyze the three-dimensional data structures and use personal prior knowledge of a user based on his past behaviour.

The research study [52] proposes to use the parallelism while adjusting the values of each column for factor matrices. It is possible due to the ALS algorithm – the original NTF problem is divided into the three sub-problems. The goal of each sub-problem is to compute the factor matrix. At the given time only one factor matrix is being computed while the other two factor matrices stay unchanged (in other words – only one sub-problem is being solved at the given time). We take an inspiration from this proposal and provide the solution to compute each of the factors in parallel (as the result, we start 3 processes in parallel for each NTF iteration). The main difference is that authors of [52] adjust each single column of factor matrix in parallel. However, the factor matrices are still computed in a linear mode.

We choose to implement the algorithm proposed by [16], to incorporate the prior knowledge and provide the possibility to run the algorithm in parallel.

2.2 Clustering techniques

Cluster analysis groups data items into certain amount of clusters, such that items within the same group are more similar than items in different groups. Clustering is commonly used for data dimensional reductions

hence processing such clustered data spaces is computationally less expensive as not clustered data [1]. Clustering techniques are commonly utilized within recommendation and personalization algorithms. [2], [38] propose to group customers with the similar preferences, once clusters are generated predictions for a particular user can be made by averaging the opinions of other customers in that cluster. Techniques based on clustering usually produce less-personal recommendations but performance is significantly better. Similarly, our approach is based on pre-clustering that provides reasonable trade-off between accuracy and time performance. However, instead of aggregating users or content items into groups, we concentrate on clustering tags. Many papers explore possibilities to cluster tags for different purposes. [42] proposes to cluster tags and consequently the technique builds ontologies from generated tag clusters. The ontologies can enhance various tasks in social tagging systems – query extension when tag search is performed, better visualization of related tags to the searched tag and also improved tag suggestion when tagging is performed. Before clustering, tags are preprocessed such that infrequent, isolated and unusual tags are filtered out as the motivation is to cluster only tags with a more general applicability as they will be part of generated ontologies. Such pre-processing is infeasible in our approach as we mainly want to cluster infrequent and unusual tags. Their approach is similarly based on the computing co-occurrences between tag pairs. The another difference is that tag pairs within a cluster are evaluated firstly by querying a semantic search engine, i.e, Swoogle whether both tags belong to the same ontology. In case the answer to the query is not resolved, Wikipedia and Google services are invoked to verify whether a given tag was not misspelled. The approach provides better and semantically more related clusters, however it is computationally more expensive and mainly it filters out rare and unusual tags. Therefore, it is not suitable for our application. Begelman et al. [5] group tags into clusters to improve search, exploration and subscription of content. They argue that users tagging behaviour is divergent thus tagspace contains many semantically similar tags which should be clustered to provide better retrieval services. The claim also promotes our approach – appropriate clustering of tags can be useful and does not have to decrease the quality of recommendations significantly. The technique builds similarity matrix where an affinity between two tags is based on their co-occurence (amount of items that were annotated with given tag pair). Afterwards, the spectral clustering is performed. Our method similarly computes affinities between tag pairs, however we exploit different similarity measures (Dice coefficient, Jaccard and Cosine similarities) that incorporate co-occurence of tag pairs. Wu et al. [50] express a tag, user or item in d-dimensional vector space where each dimension represents some knowledge and describes connection between a given object and particular knowledge or category. It is probabilistic generative model and probabilities that a particular object is conditioned by a given category are estimated according to the expectation-maximization algorithm (EM). A user, item or tag can be expressed as d-dimensional vector, for each category then can be retrieved top-k mostly semantically related tags, items or users. The method is infeasible due to time complexity of EM algorithm and even when vector representation of tags is obtained – another cluster analysis would have to be performed. In the work [18] tags are expressed as feature vectors and in a such way a tag space is clustered. Each tag is encoded as a vector over the set of items and a tag frequency for the particular item is used as a weight on the corresponding position in the vector. Clustering techniques Mean Shift and K-means also utilize the same approach of expressing each tag as a vector over set of resources.

3 Preliminaries

In this section, we describe the fundamental principles of the HOSVD and NTF as our proposed techniques are based on these two factorization methods.

3.1 HOSVD

A higher-order singular value decomposition (HOSVD) is an extended version of the SVD applied to the multidimensional matrices. SVD [13] computes matrix approximation for any matrix $F_{D_1 \times D_2}$ in the following way:

$$F_{D_1 \times D_2} = U_{D_1 \times D_1} . S_{D_1 \times D_2} . V^T_{D_2 \times D_2} \quad (1)$$

where $U_{D_1 \times D_1}$ contains left singular vectors (eigenvectors of FF^T), $V_{D_2 \times D_2}$ contains right singular vectors (eigenvectors of $F^T F$) and $S_{D_1 \times D_2}$ is diagonal matrix with singular values – square roots of the non-zero eigenvalues of FF^T sorted in descended order. SVD is used in different statistical and analysis tasks. In the area of information retrieval a well known technique called Latent semantic indexing (LSI) also utilizes SVD – it reveals latent relations between words and documents from a corpus. LSI addresses synonymy and polysemy of words - which is also crucial for the HOSVD. It is common to process only first c top singular values and corresponding singular vectors ($c \leq min(D_1, D_2)$) to achieve better approximation of the matrix F – such approach removes noise and preserves only the most important semantic information from the original matrix.

Tensor of n-th order – is multidimensional array with N indices, denoted as $\mathcal{A} \in R^{I_1 \times I_2 \times \ldots, \times I_N}$. In this

work we consider only 3-rd order tensors.
Tensor fiber – one dimensional fragment of a tensor (column vector), such that all indices are fixed except for one. For matrix (2nd tensor) a matrix column is a mode-1 fiber and matrix row is mode-2 fiber. For 3rd order tensor there are column, row and tube fibers [10].

A tensor can be converted into so called mode matrices by arranging particular fibers of a tensor as columns of mode matrices. Tensor of 3rd order can be unfolded into three different mode matrices with the following dimensions:

$A_1 \in R^{I_1 \times I_2 I_3}$ – column fibers of $\mathcal{A}$ as columns of A_1
$A_2 \in R^{I_2 \times I_1 I_3}$ – row fibers of $\mathcal{A}$ as columns of A_2
$A_3 \in R^{I_3 \times I_1 I_2}$ – tube fibers of $\mathcal{A}$ as columns of A_3

Mode-n multiplication of tensor by matrix – a mode-n multiplication $\mathcal{Y} = \mathcal{A} \times_n F$ of a tensor $\mathcal{A} \in R^{I_1 \times I_2 \times \ldots \times I_N}$ by a matrix $F \in R^{D_n \times I_n}$ is a tensor $\mathcal{Y} \in R^{I_1 \times I_2 \times \ldots \times I_{n-1} \times D_n \times I_{n+1} \times \ldots \times I_N}$ with elements:

$$y_{i_1,i_2,\ldots,i_{n-1},d_n,i_{n+1},\ldots,i_N} = \sum_{d_n=1}^{D_n} = a_{i_1,i_2,\ldots,i_N} f_{i_n,d_n} \quad (2)$$

HOSVD of 3rd order tensor is defined as:

$$\mathcal{A}' = \mathcal{S} \times_1 U^1_{c1} \times_2 U^2_{c2} \times_3 U^3_{c3} \quad (3)$$

where U^1_{c1}, U^2_{c2} and U^3_{c3} are matrices with the top c_i left singular vectors from the SVD of $1, 2, 3$ mode matrices respectively. Core tensor S is obtained according to:

$$\mathcal{S} = \mathcal{A} \times_1 (U^{(1)}_{c1})^T \times_2 (U^{(2)}_{c2})^T \times_3 (U^{(3)}_{c3})^T \quad (4)$$

The factorized tensor $\mathcal{A}'$ is the approximation of the initial tensor $\mathcal{A}$.

3.1.1 HOSVD in Social Tagging Systems

Symenonidis et al. (2008) [43] utilize HOSVD to analyze and detect relationships and patterns between tags, users and information items. HOSVD based approach explores the complex 3-dimensional relations and detects latent associations which provide better quality of recommendations. The usage data of a recommendation system are represented by 3rd order tensor – $\mathcal{A}$ where for a particular user with a selected information item and an assigned tag is stated a weight 1 and for all other cases where is not created relation a weight is 0 :

$$a_{u,i,t} \in \mathcal{A}, a_{u,i,t} = \begin{cases} 1, \text{ exists an association for } (u,i,t) \\ 0, \text{ no association between } (u,i,t) \end{cases} \quad (5)$$

$$\begin{pmatrix} -0.53 & 0.00 & -0.85 \\ -0.85 & 0.00 & 0.53 \\ 0.00 & 1.00 & 0.00 \end{pmatrix}$$

Figure 2: Application of the SVD to the 1st mode matrix - U^1 matrix

$$\begin{pmatrix} 1.62 & 0.00 & 0.00 \\ 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.62 \end{pmatrix}$$

Figure 3: Application of the SVD to the 1st mode matrix - S^1 matrix

We are including an illustrative example to better describe the HOSVD factorization. Let us assume a social tagging system with 3 different users, 3 different information items – articles and 3 tags. The associations between these objects are showed in the Table 1. The initial tensor A is constructed according to the

Users	Information items	Tags	Weights
U_1	I_1	T_1	1
U_2	I_1	T_1	1
U_2	I_2	T_2	1
U_3	I_3	T_3	1

Table 1: The associations between the objects

usage data (Table 1) and it is depicted in the Figure 1. The tensor is unfolded into the three mode matrices, denoted as the 1-mode, 2-mode and 3-mode respectively. The unfolded mode matrices from the initial tensor $\mathcal{A}$ are subject of the SVD. It results into creation of U^n, S^n, V^n matrices (see Figures 2 and 3) with the U^1 and S^1 matrices respectively, $V^{1\,T}$ is not depicted due to the huge size and is not required in the further computations), the most important are U^1, U^2, U^3 as they contain the left singular vectors of the 1-mode, 2-mode and 3-mode matrices.

The algorithm stores top c_i singular values of $i-th$ mode matrices with corresponding left singular vectors in order to construct the core tensor and the approximated tensor $\mathcal{A}'$ – depicted in the following Figure 4.

From the new tensor $\mathcal{A}'$, the recommendation system is able to suggest tags or information items with the highest weights to a given user. Readers can observe that a new association was discovered between User 1 and Tag 2 and Information Item 2.

3.2 NTF

NTF is another technique that extracts characteristics jointly from the different data dimensions [9]. It means that the latent relations can be revealed from the given data set knowing some facts from the past.

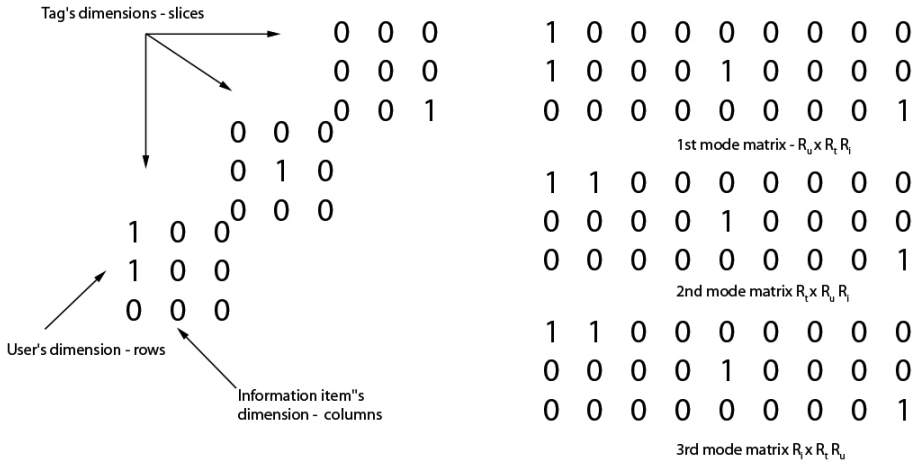

Figure 1: Tensor construction according to the users preferences table and i-th mode matrices

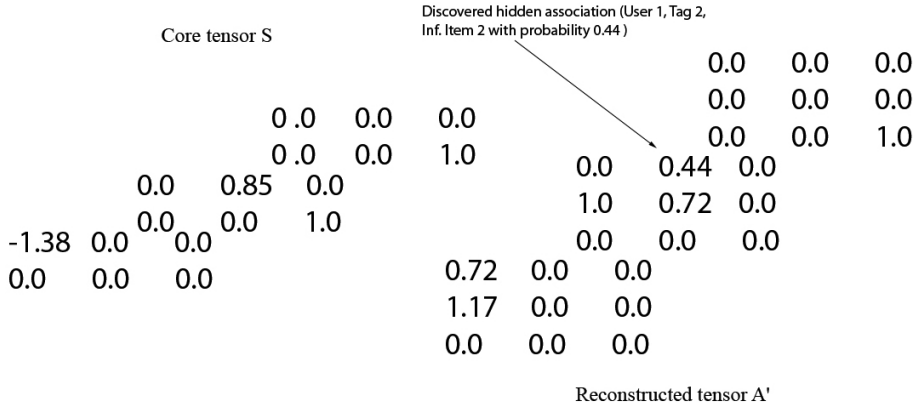

Figure 4: The core tensor S and the reconstructed tensor A'

To apply tensor factorization we need to construct the initial data tensor (n-dimensional matrix, where $n = 3$). There can be used any kind of triplets to construct the initial tensor (for example: {*user, item, tag*}). The third-order initial tensor $\mathcal{A} \in \mathcal{R}_{+}^{I \times J \times K}$ represents the data where I, J and K are the dimensions of users, items and tags, respectively.

The goal of NTF is to approximate the initial tensor such that it reflects patterns of the original data. The quality of the approximation can be measured by the error rate between the initial and factorized tensor. The most common error measure is Kullback-Leibler divergence (KL) [12], [40]. KL is a measure how close a probability distribution $p = \{p_i\}$ is to a model (or candidate) $q = \{q_i\}$:

$$D_{KL}(p\|q) = \sum_i p_i \log_2 \frac{p_i}{q_i} \tag{6}$$

In this study NTF is considered as the following problem [16]:

$$\text{NTF: } \underset{\mathcal{C},X,Y,Z}{minimize} \frac{1}{2}\|\mathcal{C} \times_1 X \times_2 Y \times_3 Z - \mathcal{A}\|_F^2 \quad \text{subject to } \mathcal{C}, X, Y, Z \geq 0 \tag{7}$$

where $\mathcal{C}$ is a diagonal core tensor, $\mathcal{A}$ is the initial tensor, X, Y and Z are the factor matrices, initialized with positive random values (each factor has the same amount of rows as corresponding dimension of the initial tensor).

The factor matrices are trained using BCLS (which is an extension of ALS (Formula 8)) algorithm.

$$\underset{x}{minimize}\ f(x) \text{ subject to } x \in X \tag{8}$$

BCLS algorithm (Formula 9) ensures that the values of factor matrices stays within the lower and upper bound-

aries.

$$\underset{x\in\mathcal{R}^n}{minimize}\ \frac{1}{2}\|Mx-b\|_2^2+c^Tx+\frac{1}{2}\gamma^2\|x\|_2^2 \quad \text{subject to } l\le x\le u \tag{9}$$

where M is $m \times n$ factor matrix, b, c are m- and n-vectors, respectively. The n-vectors l and u are lower and upper bounds on variables x, respectively. γ is a non-negative regularization parameter that can be used to control the optimality of the final solution. As the matrices are being trained, the error rate is decreasing. The training is executed until the error rate converges. The convergence condition is computed using Frobenius norm. If the condition is not met after the number of iterations k, then the factorized tensor will not approximate the initial tensor within the acceptable error rate. The factorized tensor $\mathcal{T}$ is computed using mode-n multiplication of core tensor $\mathcal{C}$ by factor matrices:

$$\mathcal{T} = \mathcal{C} \times_1 X \times_2 Y \times_3 Z \tag{10}$$

The outcomes of NTF technique are as follows [16]:

- the first outcome is three *factor* matrices that represents the most significant data characteristics for a corresponding tensor dimension, i.e. matrices X, Y and Z represent the users, items and tags, respectively. The number of the factor matrices is the same as the order of a tensor. Each row of each factor matrix represents exactly one user, item or tag of a corresponding dimension of the initial tensor. For example, the factor matrix for tagged items by users are: $X \in \mathcal{R}_+^{I\times L}$, $Y \in \mathcal{R}_+^{J\times M}$ and $Z \in \mathcal{R}_+^{K\times N}$ where each row of X, Y and Z represents the most important (active) users, mostly tagged items by the most popular tags.

- the second outcome is a core *tensor* $\mathcal{C}$ which is in the same order as the initial tensor, just usually much smaller when measuring the sizes. This core tensor represents all the correlations among the factors in all data dimensions: $\mathcal{C} \in \mathcal{R}_+^{L\times M\times N}$ where L, M and N are the number of factor size for users, items and tags, respectively.

- the main outcome is a factorized tensor $\mathcal{T}$ – the product of the core tensor $\mathcal{C}$ and the trained factor matrices X, Y, Z (Formula 10).

4 Proposals

We propose the improvements for the existing tensor based recommendation systems to increase the quality of the recommendations and speed up the time performance of recommendations. We focus on non-negative tensor factorization to get rid of negative values and incorporate prior knowledge to improve the quality of predictions. Clustering of tag space is another extension which improves the time performance of the factorization process and reduces the memory demands.

4.1 Proposals to extend standard NTF algorithm

4.1.1 Motivation

The popular techniques of data analysis like Principal component analysis (PCA) and SVD produce approximations that contain negative values. Authors of [48] and [30] argue that for the data relations with positive values only (e.g. pixel encoding) it is not possible to interpret the negative approximated values. During our experimental work we observed that the factorized tensor is sparse as it contains only $\sim 1\%$ entries that are not equal to 0. From these non-zero entries almost $\sim 50\%$ are negative values. In the recommendations context the negative values can imply that a recommendation is not relevant for the user at all. However, there are cases when negative values are relevant. So, negative values have the undesirable impact on the quality of recommendations. Also, these values can be called ballast – it gives no advantage for the factorization process. Such phenomena (when the positive values only are used as input for the tensor, but as result the negative values are produced) can not be resolved unless non-negative tensor factorization is employed.
The studies [9], [31] imply that prior knowledge can be successfully exploited during the factorization process to improve the quality of the recommendations. We focus on incorporating the *personal* prior knowledge of a user. It is a very common situation when a user does not have a rich training dataset and as the result the quality of the recommendations drops significantly. There can be used pre-filtering technique (to filter out not related objects for the user [34]) but it would restrict the search space for the recommendations. Incorporating prior knowledge into the NTF does not constrain the search space while the benefits of tensor based recommenders are still preserved.
A tensor constructed using a current interpretation scheme from the STS data is usually sparse ($\sim 99\%$ of tensor entries are zeros) [36]. Tensor factorization techniques based on HOSVD and ALS approximate dense data accurately as they minimize the element-wise loss between the entries of the initial and factorized tensor. Therefore, the techniques do not perform well on sparse STS data. The incorporation of a prior knowledge pre-processes the factor matrices hence it improves the approximation of a sparse tensor. The final tensor is approximated faster with a lower error rate.

We propose the extensions for the NTF algorithm as follows:

- incorporate the prior knowledge in factor matrices;
- possibility to run the algorithm in parallel.

4.1.2 Prior knowledge

We take the inspiration from FacetCube framework [9] for the prior knowledge incorporation. The main difference between this approach and FacetCube is that the important aspects of the initial tensor are *promoted* instead of reducing the search space for the recommendations. The model of prior knowledge is universal as it reflects the personal facts of each user behaviour in the past. Such approach enables to benefit from a prior knowledge and at the same time not to restrict the search space. The proposed approach does not suppress the discovery of latent relations that is the main advantage of tensor based recommenders. It is possible to incorporate the personal prior knowledge of a concrete user only. The generalized or averaged prior knowledge of all users would produce noisy approximated tensor.

Prior knowledge model The prior knowledge is a set of tagging activities, i.e., tags assigned by a user to items in the past. The prior knowledge is promoted and incorporated into the factor matrices however, the initial data tensor is not modified. We are interested in two factor matrices that reflect items and tags. Each row of a factor matrix is expressing the importance of a concrete item or tag. The significance of the object can be controlled by adjusting the values of exact row. The higher sum of a row reflects the more distinctive promotion of prior knowledge. This enables to differentiate the prior knowledge – it is natural that some parts of the knowledge are more important than the others i.e., items are preferred over tags because we consider only items recommendations.
The procedure to incorporate the prior knowledge:

1. promote the items the user has tagged in the past using a parameter r_k where k is the index of user item;
2. promote the tags user has used in the past using a parameter t_i where i is the index of a tag from the set of all distinct tags used by the user;
3. promote the items that were tagged by the other users using the same tags as our user. This type of promotion is *half* personal because we promote the items user has not tagged, but the promotion is based on the tags the user and other users have used. The promotion is expressed using the parameter $otherItems$.

Aforementioned parameters must be adjusted according to the concrete dataset to reflect its structure. Generally it should hold: $r_k > t_i > 1 > otherItems$ (because our recommender is item-based and usually each item is associated with multiple tags) where r_k for active users is smaller than r_k for not active users, i.e., $r_k^{\text{Active users}} < r_k^{\text{Not active users}}$.
For the experimental studies the parameter $r_k = \frac{|\text{all items}|}{|\text{user items}|}$ is used. The parameter t_i is expressed as the term frequency-inverse document frequency and it is defined as follows:

$$t_i = \frac{freq_{user}(\text{tag}_i)}{|\text{all tags}_{user}|} \times \log\left(\frac{|\text{all tags}_{corpus}|}{freq_{corpus}(\text{tag}_i)}\right) \quad (11)$$

where $freq_{user}(\text{tag}_i)$ represents how frequently the user utilized a tag_i and $freq_{corpus}(\text{tag}_i)$ is the frequency of a tag_i in the system. This weight promotes more popular tags of a user that are not frequently used by the other users in the system. This type of promotion better expresses user interests.
It is very important to choose the correct parameters to not over-promote prior knowledge as it can suppress the latent relations. In case of over-promotion there is no need to use a recommender based on a tensor factorization because we could simply generate the recommendations based on the tags usage of the users. To verify the optimality of the values for chosen parameters we investigate whether not promoted items (the revealed latent relations) appeared in the top-N list of recommendations. Our naive approach for setting values for the parameters should be further more investigated in the future work whether the latent relations are not suppressed.

Example of incorporating prior knowledge The Figure 5 represents a single slice of the initial tensor.

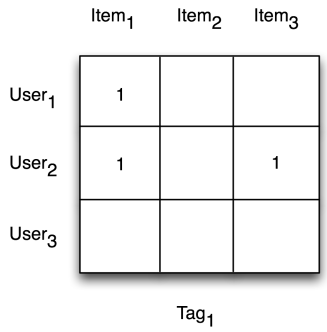

Figure 5: The slice nr. 1 of the initial tensor

The slice itself represents a distinct tag. Empty relations are omitted by convenience. $User_1$ has tagged the $Item_1$ and $User_2$ has tagged the items 1 and 3 using

the same Tag_1.
The rows of the items factor (Figure 6) represent the set of items encoded in the initial tensor.

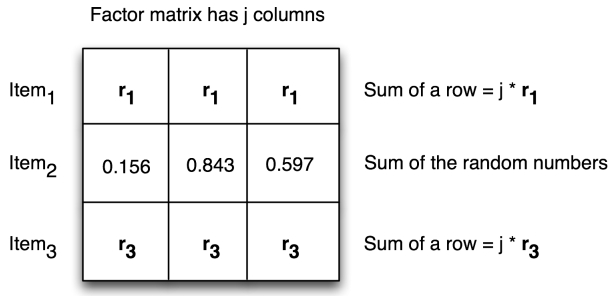

Figure 6: The items factor and the prior knowledge

The personal prior knowledge of $User_2$ is promoted by changing the values of the exact factors rows – we promote this personal prior knowledge by assigning the values r_1 and r_3 for each of the cell of the corresponding factor rows (in this case – rows 1 and 3). It is possible to compute the distinct values for r_k to differentiate the prior knowledge. However, we use the same value for all r_k as the quality of recommendations is acceptable.
In the same way we express the prior knowledge for the Tag_1 using a variable t_1 (computed according to the Formula 11) in the tags factor matrix for the $User_2$ (Figure 7).

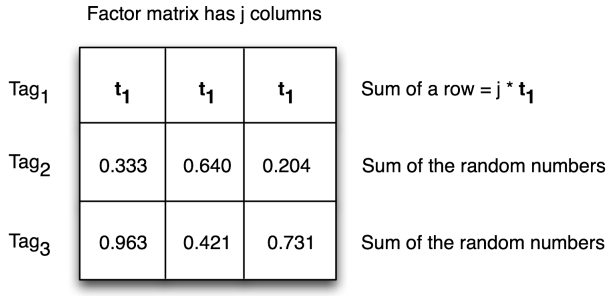

Figure 7: The tags factor and the prior knowledge

It is also known $User_1$ tagged the $Item_1$ with the same tag as $User_2$, therefore the value of r_1 is increased: $r_1 \leftarrow r_1 + otherItems * freq$ where $freq$ is a number of the common tag usage by the other users for the same item as originally tagged by our user.

4.1.3 Algorithm of NTF

In the following paragraph, we describe the utilized NTF algorithm (Algorithm 1). The input for the algorithm is initial tensor $\mathcal{A}$ and the option whether the factor matrices should be trained in parallel. The output is a factorized tensor $\mathcal{T}$ which is computed according to the Formula 10.

Algorithm 1 The algorithm for NTF with possibility to run in parallel

Data: $\mathcal{A} \in \mathbb{R}^{I \times J \times K}$, $useParallelism \in \{false, true\}$
Result: $\mathcal{T} \in \mathbb{R}^{I \times J \times K}$

Initialize factors with prior knowledge:
$X \in \mathbb{R}^{I \times N}$, $Y \in \mathbb{R}^{J \times N}$, $Z \in \mathbb{R}^{K \times N} \geq 0$
Initialize core tensor: $\mathcal{C} \in \mathbb{R}^{K \times K \times K} = \mathcal{I}$
Reset iterations counter: $k \leftarrow 0$ **repeat**
  if *useParallelism* **then**
    Start parallel processes $p_i, i = 1, 2, 3$
    $X \leftarrow p_1(solve(NTF_X))$
    $Y \leftarrow p_2(solve(NTF_Y))$
    $Z \leftarrow p_3(solve(NTF_Z))$
  end
  else
    $X \leftarrow solve(NTF_X)$
    $Y \leftarrow solve(NTF_Y)$
    $Z \leftarrow solve(NTF_Z)$
  end
  [compute column scales]
  D^1_{k+1} for X
  D^2_{k+1} for Y
  D^3_{k+1} for Z
  [compute scaled factor matrices]
  $X_{k+1} \leftarrow X D^1_{k+1}$
  $Y_{k+1} \leftarrow Y D^2_{k+1}$
  $Z_{k+1} \leftarrow Z D^3_{k+1}$
  for $n = 1, 2, 3$ **do**
    [update diagonal core tensor $\mathcal{C}$]
    $\mathcal{C}_{k+1} \leftarrow C_k \times_n (D^n_{k+1})^{-1}$
  end
  $k \leftarrow k + 1$
until *converged*;
$\mathcal{T} \leftarrow \mathcal{C}_k \times_1 X_k \times_2 Y_k \times_3 Z_k$

Firstly, the factor matrices and a core tensor are initialized, the prior knowledge is encoded into the tags (X) and items (Z) factor matrices. The factorization process is being repeated until the optimal solution is found or the number of iterations is exceeded. To train each of the factor matrix the BCLS algorithm is executed. After the training phase the values of factor matrices are scaled – small and insignificant values tend to move to lower bound and the important values are preserved. The core tensor is updated using mode-n multiplication and the trained factor matrices. Finally, if the optimality condition is satisfied (or the number of iterations is exceeded), the approximated tensor is computed.
The linear execution of NTF algorithm (Algorithm 1)

takes time T^{NTF}_{linear} (to execute code in lines $5-7$ takes $\sim 95\%$ of all execution time) (Formula 12):

$$T^{NTF}_{linear} = (t_1 + t_2 + t_3) * iter_{linear} \quad (12)$$

where t_1, t_2, t_3 is time needed to compute factor matrices 1, 2, 3, respectively and $iter_{linear}$ is a number of iterations needed to achieve the optimal solution when NTF is executed linearly.
We expect the NTF algorithm to terminate in time $T^{NTF}_{parallel}$ (Formula 13) when the parallel approach is applied.

$$T^{NTF}_{parallel} = (\frac{t_1 + t_2 + t_3}{3} + os_{time}) * iter_{parallel} \quad (13)$$

where os_{time} is a time needed for the operating system to handle the parallel processes, manage the memory and $iter_{parallel}$ is number of iterations needed to achieve the optimal solution when NTF is executed in parallel. It is very likely there will be needed more iterations for parallel execution than linear execution ($iter_{parallel} \geq iter_{linear}$) to approximate a tensor because the training process of the factor matrices is slightly slower.
We implement a provided NTF algorithm (that originally developed in Matlab and C by the authors of [16]) in Java with the possibility to run it in parallel.

4.1.4 Discussion

The main advantage of NTF algorithm is that the approximated tensor contains only the positive values. Hence the interpretation of the results is clear. Our attempt to incorporate the personal prior knowledge by promoting the facts from the past of the exact user helps to produce the recommendations of better quality. This model overcomes the problems that arise when pre-filtering is used: the search space for the recommendations is not reduced. The parallel execution of the algorithm improves the time performance as it utilizes the modern computer hardware more efficiently.
The major drawback of this model is a need to compute NTF factorization for each user because the factor matrices are built using the personal prior knowledge of the concrete user in a system. Every time a new object (e.g. item, tag or user) is introduced into the system a factorization has to be recomputed for each user to provide up to date recommendations. In comparison with the HOSVD approximation, there are no techniques for updating factorized tensor like *folding-in* or *incremental SVD*. These limitations constrain the usage of the NTF with prior knowledge in the real world applications as the demands for the resources, i.e., memory and CPU power are high.

The usage of the personal prior knowledge can be restricted due to the absence of the user behaviour in the past. Also, it is a challenge to choose the promotion variables for the aspects of the personal prior knowledge to not suppress the latent relations. Furthermore, the execution time of NTF can drop significantly when more iterations are executed or bigger factor matrices are used.
The prior knowledge model (step 3) can be extended by computing the similarities (based on syntax and semantic as presented by [14]) between the tags instead of using just the same tags. Also, the clustering techniques can be applied to group the similar tags in order to enrich the prior knowledge.

4.2 Reducing tag space with clustering

We propose to utilize a cluster analysis on the tag space to group similar tags into clusters. Such reduced tag space causes smaller initial tensor and in consequence the better time performance is achieved while the quality of recommendations is preserved. Before describing technical details of our method, we provide motivation for clustering and describe our approach with the illustrative example.

4.2.1 Motivation

The majority of tags used within the social tagging systems are assigned and used rarely. These infrequently used tags cause unnecessary high memory demands e.g. 28145 tags from the BibSonomy dataset were assigned just once, and an initial tensor will contain 28145 slices (each slice of the tensor corresponds to a particular tag) of the size $|U| \times |I|$ however each such slice will contain only one value. We explore the MovieLens and BibSonomy datasets 5.2 and evaluate the amount of such rarely used tags in the Table 2. Our findings are supported also by the [28] where authors have shown that around 30% of all distinct tags were used only once within the Delicious system. Their analysis also shows that synonyms, acronyms and spelling variations occur frequently among tags. Many tags differ only in the spelling variations – upper or lower case initial letters of tags, singulars or plurals, spelling mistakes. Therefore, majority of the rare tags can be grouped with the

Dataset	Number of tags at each frequency	Number of uses of tag
MovieLens	9248	1
MovieLens	2515	2
MovieLens	1134	3
BibSonomy	28145	1
BibSonomy	8806	2

Table 2: Amount of rarely used tags in the BibSonomy and MovieLens datasets.

more frequent because of the mentioned reasons. For instance, infrequent tags from the BibSonomy dataset like: *123flickr*, *flickr+avivamagnolia*, *flickr.com*, *flickr+tamevolcano*, *flickrInspector*, *flickrbits*, *flickr-feeds*, *flickrites*, *toflickr* could be grouped into one cluster with more frequent tag *flickr*. On the other hand, there are rarely used tags with the unique and specific meaning that cannot be clustered. It is the task of the clustering to appropriately distinguish which tags can (not) be clustered.

Let us present the following motivational example with the 3 users (*u1, u2, u3*), 3 web items (*flickr.com, www.wallpapers.org, cnn.com*) and 6 distinct tags (*flickr.com, toflickr, flickr, wallpapers, pictures, news*). The tagging posts of the users assigned to the web items are depicted in the Table 3: Obviously,

User	Web item	Tag	Weight
u1	flickr.com	flickr.com	1
u1	flickr.com	toflickr	1
u1	flickr.com	flickr	1
u1	wallpapers.com	wallpapers	1
u1	flickr.com	pictures	1
u2	flickr.com	flickr	1
u3	cnn.com	news	1

Table 3: Tagging posts of the example.

the tags *flickr.com, flickr, toflickr* are semantically related and our approach groups them into one cluster *flickr cluster* = {*flickr.com, flickr, toflickr* }.

Grouping the similar tags into the cluster provides new relations as follows:

User	Web item	Tag	Weight
u1	flickr.com	flickr cluster	3
u1	wallpapers.com	wallpapers	1
u1	wallpapers.com	pictures	1
u2	flickr.com	flickr cluster	1
u3	cnn.com	news	1

Table 4: Tagging relations when tags about flickr are grouped into one cluster.

After the HOSVD factorization is computed, the following scores are obtained for the given triplets. HOSVD factorization reveals a latent relation between the user *u2* and the web item *wallpapers.com*. Therefore, the recommendation system recommends item *wallpapers.com* to the user *u2*. The similar result would be obtained if the tags *flickr.com, flickr, toflickr* would not be merged however our approach removes 2 slices of the tensor and in consequence it improves time performance of the factorization and decreases memory requirements.

User	Web item	Tag	Weight
u1	flickr.com	flickr cluster	3.0435
u1	wallpapers.com	wallpapers	0.9287
u2	wallpapers.com	wallpapers	0.2572
u1	wallpapers.com	pictures	0.9287
u2	wallpapers.com	pictures	0.2572
u2	flickr.com	flickr cluster	0.8429
u3	cnn.com	news	1

Table 5: Results of the factorization with the revealed relation between user *u2* and web item *wallpapers.com*.

4.2.2 Cluster analysis of tags

We exploit and evaluate 4 different clustering techniques that are adjusted to group similar tags into a cluster. The general proposed approach consists of the following steps:

1. Perform cluster analysis of a tag space with the selected clustering method from the 4 proposed techniques.

2. Build an initial tensor where a tag dimension has the same size as the amount of obtained clusters. A tagging performed by a user u to a item i with a tag t is a triplet *(u, i, t)*. All such triplets are encoded to corresponding positions in the initial tensor with the initial weight 1 and a tag t is mapped to the matching cluster. When two or more triplets share the same item and user – differ only in tags and these tags belong to the same one cluster a final weight in the tensor is the amount of such triplets, e.g. given two triplets: *(u, i, t_1), (u, i, t_2)* and tags t_1, t_2 belong to the same tag cluster, then an initial tensor will contain weight 2 at the position (a row for a user u, a column for a item i and slice corresponding to tag cluster with tags t_1, t_2).

3. Compute tensor factorization for the constructed initial tensor. Finally, items recommendations are generated according to the sorted weights from the factorized tensor for the given user and all not observed items.

In the following sections, we describe 4 different clustering techniques – first two proposed approaches cluster tags according to their co-occurrence based similarities, K-means and Mean shift algorithms consider each tag from a tag space as feature vector.

4.2.3 Correlated Feature Hashing

We propose to reduce a tag space with hashing function that is similar to the proposed technique in [4] where authors successfully reduced dictionary size by utilizing

hashing. The idea is to share and group tags with the similar meaning. We sort the tags used within the system according to the frequency of usage such that t_1 is the most frequent tag and t_T is the least frequent. For each tag $t_i \in 1,\ldots,T$ is calculated *DICE* coefficient with respect to each tag $t_j \in 1,\ldots,K$ among the top K most frequent tags. The *DICE* coefficient is defined as:

$$\mathrm{DICE}(t_i,t_j) = \frac{2.\mathrm{cocr}(t_i,t_j)}{\mathrm{ocr}(t_i)+\mathrm{ocr}(t_j)} \tag{14}$$

where $\mathrm{cocr}(t_i,t_j)$ denotes the number of co-occurrences for tags t_i and t_j, $\mathrm{ocr}(t_i)$ and $\mathrm{ocr}(t_j)$ is the total number of tag t_i, t_j assignments respectively. For each tag t_i, we sort the K scores in descending order such that $S_p(t_i) \in 1,\ldots,K$ represents the tag of the p-th largest DICE score $\mathrm{DICE}(t_i,S_p(t_i))$. We can then use hash kernel approximation defined as:

$$\bar{\Phi}_{t_j}(x) = \sum_{t_i \in T: h(t_i)=t_j} \Phi_{t_i}(x) \tag{15}$$

and given by hash function

$$h(t_i) = S_1(t_i) \tag{16}$$

The described approach is replacing each tag t_i with the tag $S_1(t_i)$. Obviously, we have reduced tag space from all T tags to the K most frequent tags.

4.2.4 Spectral K-means clustering

We utilize Spectral K-means clustering technique. Firstly, we encode tag relations into the affinity matrix W, such that $w_{i,j}$ entry represents affinity between tag t_i and tag t_j. The similarity matrix can be also interpreted as undirected weighted graph G where tags represent nodes and weights are expressed as similarities between given tags. We exploit Jaccard (17) and Cosine(18) similarity measures denoted as $\mathrm{JAC}(t_i,t_j)$ and $\mathrm{COS}(t_i,t_j)$.

$$\begin{aligned}\mathrm{JAC}(t_i,t_j) &= \frac{|t_i| \cap |t_j|}{|t_i| \cup |t_j|} \\ &= \frac{\mathrm{cocr}(t_i,t_j)}{\mathrm{ocr}(t_i)+\mathrm{ocr}(t_j)-\mathrm{cocr}(t_i,t_j)}\end{aligned} \tag{17}$$

$$\mathrm{COS}(t_i,t_j) = \frac{|t_i| \cap |t_j|}{\sqrt{|(t_i| \times |t_j|}} = \frac{\mathrm{cocr}(t_i,t_j)}{\sqrt{\mathrm{ocr}(t_i) \times \mathrm{ocr}(t_j))}} \tag{18}$$

where $\mathrm{cocr}(t_i,t_j)$ is the sum of all co-occurrences for tags t_i and t_j, $\mathrm{ocr}(t_i)$ and $\mathrm{ocr}(t_j)$ is the total number of tag t_i, t_j occurrences respectively. The introduced *DICE* 14 similarity measure is also considered. Once the similarity matrix W is created, we then proceed to find (sub) clusters of tags that address the same topic.

To obtain clusters, we rely on a spectral clustering algorithm which input is the undirected weighted graph G. The spectral clustering algorithm partitions the graph G based on its spectral decomposition into subgraphs. In order to run the spectral clustering, we perform the following steps:

1. We build the Laplacian matrix $L = D^{-1/2}WD^{-1/2}$ derived from the affinity matrix W. The D is $n \times n$ diagonal matrix whose $(i,i)-th$ element is the sum of W's $i-th$ row, in other words it is the degree of a given node i - sum of all weights corresponding to the edges that are connected to a given node i. The Laplacian matrix L is symmetric and has identical size as affinity matrix W.
2. We compute the k largest eigenvectors of L, these obtained top k eigenvectors are used as columns to create a new matrix $U \in \mathcal{R}^{n \times k}$. We consider each row of U as a point in $\mathcal{R}^k$, hence we can apply standard K-means algorithm to cluster these points into k clusters.
3. Finally, we map original node i to the cluster j if and only if row i from matrix U belongs to the same cluster j. We obtained disjoint groups of similar and related tags and we are able to build initial tensor with the reduced tag dimension and then proceed with the factorization.

We have tried to integrate and explore another spectral non-parametric clustering algorithm Eigencuts [8]. Similarly to the spectral K-means algorithm it computes eigendecomposition of the Laplacian matrix that is derived from the affinity matrix. The main intuition of this algorithm is to identify bottleneck edges within the graph structure by performing random walk and observe how a probability flow changes when a given weight for the edge is replaced. However, we have struggled with infeasible time and memory demands therefore it can be concern of our future work.

4.2.5 K-means

The following two clustering techniques differ from the previous in a such way that each tag is expressed in n-dimensional vector space where i-th dimension corresponds to the i-th item res_i (in a similar way as in [24, 35]). We denote $T = \{t_1, t_2 \ldots, t_{|T|}\}$ as the set of all distinct tags that are clustered and $R = \{res_1, res_2 \ldots res_n\}$ the set of all items that are tagged with tags from T. Let $f(t,res_i)$ be equal to a frequency of a tag t assignements to item res_i otherwise it is equal to 0. Then, the vector representation of tag t is:

$$t = (f(t,res_1), f(t,res_2), \ldots, f(t,res_n)) \tag{19}$$

Once, tags from T are expressed as n-dimensional vectors, we proceed with the cluster analysis.

The K-means is a simple well known clustering technique that groups objects from a given set into k clusters (given a priori). The algorithm starts with generating k random centroids. Then for each object from a dataset, i.e., a tag from T the nearest centroid is found thus a given tag is associated with a particular centroid. When all tags are processed, centroids have to be recomputed such that new centroid is the mean value of the vectors for the given cluster. Again for all objects the nearest centroids are identified and objects are clustered with them. This process repeats until locations of centroids do not change.

Clustering of a tag space with the K-means algorithm is computed as follows:

1. Each tag from a tag space T is expressed as n-dimensional vector. According to the size of the tag space and user requirements an amount of clusters is set to k.

2. It randomly places k centroids such that a distance from each other is maximized.

3. Each tag from the tag space is binded to the nearest centroid.

4. New centroids are computed as the mean value of tags vectors grouped with a given centroid. It continues with the step 3, until new centroids are identical with the centroids from the previous iteration.

We obtained k disjoint clusters of tags so we can proceed with the tensor factorization. The results of K-means algorithm depend on used distance measure - we exploit Cosine, Manhattan, Euclidean and Jaccard distances.

4.2.6 Mean shift clustering

The Mean shift [11] is a nonparametric clustering as it does not require a prior definition of the number of clusters. The feature vector space is considered as probability density function. Input objects for clustering are considered as they would be sampled from this probability density function. The density function is estimated according to the Kernel density estimation method (also known as Parzen Window technique) which is defined as:

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K(\frac{x - x_i}{h}) \tag{20}$$

where n is an amount of tags, d is dimension of the vector space and kernel function $K(x)$ is defined as:

$$K(x) = c_{k,d} k(||x||^2)$$

where $c_{k,d}$ is constant that assures integration of $K(x)$ to 1 and $k(x)$ is specific kernel profile. Mean shift vector $m_h(x)$ is computed for each object and it points toward the direction of the maximum density increase. It results into the replacement of the window to the new position given as: $x = x + m_h(x)$. These two steps repeat for each object until stationary point ($\nabla f(x) = 0$) is not reached. The final clusters (dense regions) correspond to these local maximums of the given density function. Each final cluster contain only objects that are associated to the same stationary point.

In the context of tags clustering, for each tag that is expressed as d-dimensional vector a window is placed around a given tag such that a window size is conditioned with parameter h and a given point is centroid of that window. Mean shift computes mean for all tags within the given window (mean shift vector points to the computed mean). Centroid of the window is moved to the obtained mean and this repeats until window position stabilize.

The main advantage of the technique is to automatically determine the number of clusters, however this depends on the window size which is given by bandwidth parameter h and it has to be tuned. The smaller parameter is set more final clusters are obtained. The drawbacks are computational expensiveness and when a feature space with large amount of dimensions is considered it is demanding to identify points that belong within the particular window.

4.2.7 Theoretical complexity analysis

We compare complexity analysis of the basic HOSVD factorization with the proposed approach where the tag dimension is reduced. When the HOSVD is performed, always 3 different Singular Value Decomposition (for each mode matrix) must be computed. According to [7], [20, p. 254] a computation complexity of full SVD for a matrix $M_{p\times q}$ is $O(pq^2 + p^2q + q^3)$ To obtain a core tensor and then a final approximated tensor, a tucker product (n-mode multiplication) must be calculated. The complexity of this operation is $O(C.|U|.|I|.|T|)$ where $C = \frac{1}{3}(|U| + |I| + |T|)$ [3]. A complexity of these sub operations of the HOSVD is showed in the Table 6.

For better readability, we substituted a term $[U^2R + R^2U + RU + CUR]$ with $\mathcal{A}$, a term $[UR^2 + RU^2 + RU]$ with $\mathcal{B}$, $[R^3 + U^3]$ with $\mathcal{C}$ and R^3U^3 with $\mathcal{D}$. Obviously, computational complexity is decreased when a tag space is clustered. Less clusters are generated thus the execution time of the HOSVD factorization is improved. Let us assume that $|T_c| = k.|T|$ where $k \leq 1$ (percentage how many clusters should be created from the original tag space $|T|$), in such case with our proposed approach we improve computational complexity

Operation	Full HOSVD	Clustered HOSVD
SVD of $U^1_{\|U\|\times\|\text{tags}\|\|I\|}$	$\|U\|(\|T\|\|I\|)^2 + \|U\|^2(\|T\|\|I\|) + \|T\|\|I\|^3)$	$\|U\|(\|T_c\|\|I\|)^2 + \|U\|^2(\|T_c\|\|I\|) + \|T_c\|\|I\|^3)$
SVD of $U^2_{\|I\|\times\|\text{tags}\|\|U\|}$	$\|I\|(\|T\|\|U\|)^2 + \|I\|^2(\|T\|\|U\|) + \|T\|\|U\|^3)$	$\|I\|(\|T_c\|\|U\|)^2 + \|I\|^2(\|T_c\|\|U\|) + \|T_c\|\|U\|^3)$
SVD of $U^3_{\|\text{tags}\|\times\|I\|\|U\|}$	$\|T\|(\|I\|\|U\|)^2 + \|T\|^2(\|I\|\|U\|) + \|I\|\|U\|^3)$	$\|T_c\|(\|I\|\|U\|)^2 + \|T_c\|^2(\|I\|\|U\|) + \|I\|\|U\|^3)$
Tucker product	$C.\|U\|.\|I\|.\|T\|$	$C.\|U\|.\|I\|.\|T_c\|$
Total	$T\mathcal{A} + T^2\mathcal{B} + T^3\mathcal{C} + \mathcal{D}$	$T_c\mathcal{A} + T_c^2\mathcal{B} + T_c^3\mathcal{C} + \mathcal{D}$

Table 6: Complexity analysis of HOSVD and clustered HOSVD

with

$$(T - T_c)\mathcal{A} + (T - T_c)^2\mathcal{B} + (T - T_c)^3\mathcal{C} + \mathcal{D}$$

Our approach also decreases memory requirements, when the size of the initial and factorized tensor is notably smaller. Let us assume the initial size of the used distinct tags $|T|$ amount of users $|U|$, items $|I|$ and $|K|$ is the amount of clusters. Then clustering saves:

$$|T - T_c| \times (|U|.|I|) \times 8\text{bytes}$$

in comparison to the initial tensor where a tag space is not clustered.

4.3 Discussion

One could argue that applying clustering to reduce tag dimension before computing the tensor factorization is unnecessary as HOSVD is also dimensional reduction technique. However, the low rank approximation [22] is not reducing tag dimension in the same sense as clustering. The low rank SVD is only removing noisy data, so that approximated tensor better reflects the patterns in the data. The importance of clustering is to shrink tag space in such way that an initial tensor contains less slices and the factorization is faster. Once the tag space is clustered each mode matrix is smaller. In case only the low rank SVD is applied user and item mode matrices reflect the original tag space and only the tag mode matrix is reduced.

5 Experiments

We investigate the prediction quality and time performance of the proposed techniques and compare it with the baseline results of the HOSVD recommender [44].

5.1 Experimental environment and implementation

All the experiments are conducted on Windows 7 64-bit operating system running on Intel Core 2 Quad CPU @ 2.66GHz with 8,00 GB (7,87 GB usable) RAM.
The HOSVD, NTF and clustering techniques are implemented in Java 6 and source code is available on the website[1]. The main parts of Spectral K-means, Mean shift and K-means are provided by Apache Mahout[2].

5.2 Datasets

The proposed techniques are evaluated on the BibSonomy [25] and MovieLens [37] datasets. Similarly, as in [44] the datasets are preprocessed with p-core filtering so that data are more dense. P-core constraints data such that each user, item and tag has to appear at least p times in the dataset [17]. The following p-core values are applied:

- 5 for the BibSonomy dataset (B)
- 59 for the MovieLens dataset (M)

Due to the limited memory resources on the test computer it is not possible to run NTF with lower p-core value than 59 for MovieLens dataset.
The BibSonomy contains 116 distinct users, 361 items and 412 tags. The total number of tagging posts is 10148.
The MovieLens dataset contains 288 unique items tagged by 221 different users with 265 unique tags. In total, there are tagged 8412 items using at least one or more tags by the users.

5.3 Classes of users

Users are sorted according to their tagging activity and 3 classes of users are created: *Active users* – top 10 most active users, *Average users* – 10 average active users and *Not active users* – 10 least active users. The classes are used to investigate the quality of the proposed techniques more precisely.

5.4 Experimental protocol

The same methodology is followed as in [44] in order to compare the enhancements of our approaches. For a test user we split his tagging posts into training and evaluation set in the ratio 50%:50%. The task of the recommender is to predict the items that corresponds to the tagging posts in the evaluation set. The models are

[1] `http://www.cs.aau.dk/~mleginus/thesis`
[2] `http://mahout.apache.org`

trained on the training set and the recommendations quality is measured on the test set. The prediction quality is evaluated according to the following common evaluation metrics.

- Precision is the ratio of the number of recommended relevant items (we consider only items that correspond to the tagging posts in the evaluation part of the data set) from the top-N list of entries to N.

$$\mathrm{P}_N = \frac{|\{\text{relevant items}\} \cap \{\text{top-N items}\}|}{|\{\text{top-N items}\}|} \quad (21)$$

- Recall is the ratio of the number of recommended items from the top-N list of entries to the total number of items for a given user from the evaluation part of data set.

$$\mathrm{R}_N = \frac{|\{\text{relevant items}\} \cap \{\text{top-N items}\}|}{|\{\text{relevant items}\}|} \quad (22)$$

- F-measure is a metric which involves the previously described recall and precision indicators. There is computed an average of both metrics. The best results of the recommendations are achieved when F-measure reaches one.

$$\text{F-measure} = 2 \,.\, \frac{\mathrm{P}_N.\mathrm{R}_N}{\mathrm{P}_N + \mathrm{R}_N} \quad (23)$$

5.5 Results for the HOSVD

We conduct experiments using the HOSVD recommender and both datasets. The results are used as the baseline for the comparison with our proposed techniques. We measure the quality of the items recommendations, also the execution time of the factorization.

For the BibSonomy dataset the average precision for all 116 users is 0.3373 and 0.2075 for 221 users from the MovieLens dataset (Table 7). The best prediction quality is achieved when top $(60, 105, 225)$ singular values and corresponding left singular vectors are preserved for the 1st, 2nd and 3rd mode matrices respectively for the BibSonomy dataset. For the MovieLens dataset the following values are used: $(44, 103, 69)$. We achieve the similar precision for the BibSonomy dataset as is reported in [44], however the achieved recall is worse, because the BibSonomy dataset contains significantly more items (+115) and the amount of users is similar. The precision for MovieLens dataset is worse, compared with the results for the BibSonomy, even using more dense data. For the least active users, the precision is 0 and it is not acceptable. The reason of such accuracy is that these users have posted a few tags to the items constrained by p-core.

The splitting into training and evaluation set is performed with ratio 50% : 50%, the evaluation set contains many items and in consequence the recall is lower. The average execution time of HOSVD factorization for BibSonomy is 110 s and 102 s for MovieLens.

Class of users	Precision	Recall	F-measure
All users^B	0.3373	0.1266	0.1637
Active users^B	0.7	0.052	0.0966
Average users^B	0.3000	0.138	0.182
Not active users^B	0.155	0.155	0.155
All users^M	0.2075	0.0738	0.0983
Active users^M	0.42	0.0435	0.0788
Average users^M	0.1628	0.1261	0.1421
Not active users^M	0	0	0

Table 7: Baseline results for the HOSVD

We incorporate the prior knowledge into the mode matrices of HOSVD to verify whether the proposed model is universal. The preliminary experimental study shows that the quality of precision is increased by $\sim 3\%$ for the BibSonomy dataset. However, this approach requires a more detailed analysis.

5.6 Clustered tag space

In this section, we evaluate quality of recommendations when a tag space is clustered and the sets of users and items have the original size. HOSVD is applied on such compressed tensor where tags dimension is reduced from the original size to the certain number of clusters. The goal is to find the best trade-off between the accuracy of recommendations and time performance.

5.6.1 Distance and similarity measures

Before performing clustering, a tag space has to be preprocessed so that tags are expressed either as feature vectors (K-means, Mean Shift) or in the affinity matrix where each entry represents similarity between two tags (Spectral Clustering, Correlated Feature Hashing).

When a tag space is clustered with K-means algorithm and Mean Shift, we explore and compare different similarity measures. The experiments are conducted with Cosine, Euclidean, Manhattan and Jaccard distance measures. We observe how a given distance measure influences the quality of recommendations. The results are compared in the following table:

Distance measure	Precision	Recall	F-measure
Cosine	0.1806	0.0607	0.0908
Euclidean	0.1293	0.0454	0.0672
Manhattan	0.1732	0.0495	0.0769
Jaccard	0.1474	0.0481	0.0806

Table 8: Comparison of the distance measures and their impact on the recommendation quality for K-means amount of clusters set to (40%) of the tag space

The cosine distance produces the best recommendations results. Many tags are placed into one large cluster when Euclidean and Manhattan distances are used thus the quality is significantly decreased. The observation is similar to the [42] as authors identify Cosine similarity as the most suitable. It is sensitive to the small variations in more elements. Euclidean and Manhattan distances are more sensitive to significant changes in a few elements of feature vectors. Based on this results, we use the cosine similarity for all K-means and Mean shift clustering computations.

For the spectral K-means clustering and feature hashing, a coocurrence similarity has to be computed for each tag pair in the original tag space. We evaluate several similarity measures: Dice 14, Jaccard 17 and Cosine 18. The impact of selected measures on the prediction quality is shown in the Table 9.

Similarity measure	Precision	Recall	F-measure
Cosine	0.2722	0.0926	0.1381
Dice	0.285	0.104	0.1523
Jaccard	0.283	0.1007	0.1484

Table 9: Comparison of the similarity measures and their impact on the recommendation quality for spectral K-means clustering amount of clusters set to (50%) of the tag space

The evaluated measures do not affect significantly the quality of recommendations therefore, in this work the Dice similarity is used.

5.6.2 Clustering techniques

We describe corresponding settings and parameters for each clustering technique that are used to run them.

Correlated feature hashing: A tag space is split into *top k most frequent tags set* (MFTS) and the rest of a tag space – *not frequent tags set* (NFTS). For each tag from the MFTS is computed similarity with all tags in the NFTS. Afterwards, each tag from the NFTS is mapped to a tag with the highest similarity score from the MFTS. Some tags from the MFTS do not have mapping with infrequent tags such tags are mapped to themselves so they generate single-tag clusters. The dice coefficient is used for computations of tag pairs similarities.

Spectral K-means clustering: For each tag in a tag space a co-occurrence is computed with all other tags. The Dice distance measure is used and tag pairs similarities are encoded into the affinity matrix. The amount of clusters is given by input parameter k.

K-means clustering: Each tag is expressed as feature vector, where a weight on a particular dimension expresses relation between the tag and a particular item. The number of clusters k is set in advance.

Mean shift clustering: The algorithm does not require setting the number of clusters a priori, but a window size has to be specified in advance. The window size is the main parameter that affects the amount of clusters. The second parameter determines whether two clusters are close enough, if so they are merged into one bigger cluster.

5.6.3 Results

For each technique, the number of clusters is iteratively changed: starting from the 40% of the original tag space size, after each iteration amount of clusters is increased by 10% untill 80%. We plot the precision value versus amount of clusters in Figure 8. We also measure improvement of time performance. The execution time of the factorization is significantly decreased as is showed in the following Figure 9.

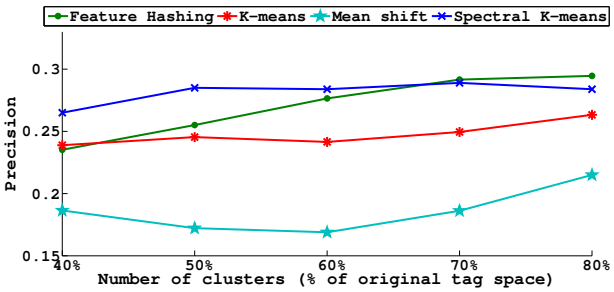

Figure 8: Precision comparison of clustering techniques

The spectral K-means outperforms correlated feature hashing, K-means and also Mean Shift almost in all cases. The Mean Shift attains the worst precision for all number of clusters as the distribution of tags in clusters is not uniform. The K-means and spectral K-means methods get higher precision as the number of clusters is increased but the precision of recommendations does not grow so significantly as could be expected.

The best trade-off between accuracy and execution time is reached when the number of clusters is 50% of

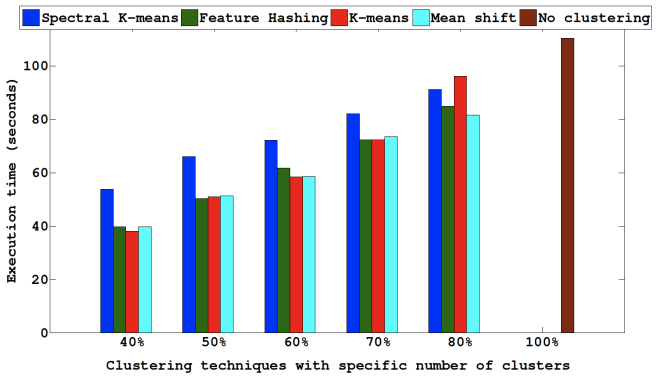

Figure 9: Comparison of time performance for clustering techniques

the original tag space. In such case the average precision for all users is decreased with only $\sim$ **0.05** but the execution time of the factorization is decreased about **40.2**%. The results of clustering techniques when number of clusters is set to 50% are in the Tables 10, 11.

Clustering method	Precision	Recall	F-measure	Execution time
HOSVD baseline	0.3373	0.1266	0.1841	110.326 s
Feature hashing	0.2551	0.0873	0.1301	50.301 s
Spectral K-means	0.2850	0.1040	0.1523	65.956 s
K-means	0.2544	0.1045	0.1465	48.813 s
Mean Shift	0.1722	0.0697	0.0992	51.312 s

Table 10: Accuracy and execution time results for clustering techniques, number of clusters is set to 50% of the tag space.

The best prediction quality is achieved with the spectral K-means clustering. The accuracy for active class of users is increased but for average and not active classes of users accuracy drops by 0.02 and 0.032 respectively.

Class of users	Precision	Recall	F-measure
All users	0.2850	0.1040	0.1523
Active users	0.74	0.0544	0.1013
Average users	0.28	0.1212	0.1692
Not active users	0.123	0.123	0.123

Table 11: Accuracy results of spectral K-means clustering (number of clusters is set to 50% of the tag space) for 3 classes of users: *Active users*, *Average users*, *Not active users*.

The examples of different tag clusters for the Bibsonomy dataset are shown in the Table 12. The feature hashing creates clusters that always contain at least one frequent tag. Such approach is not always suitable, e.g., tags *swikig, eswc*2006 are mapped with the tag *iccexample* that describes an example of color profile. The Mean shift generates not uniformly distributed clusters, e.g., a cluster about social bookmarking contains tags about web2.0. The spectral K-means and K-means methods produce reasonable clustering results. However, the former one slightly outperforms the latter method and that also affects the prediction accuracy.

The proposed approach significantly decreases execution time of the factorization as is shown by the theoretical complexity analysis and also proved by conducted experiments. We observe that 50% is the optimal number of clusters such that the execution time is improved and the quality of the recommendations is acceptable. The methods based on a tag pair co-occurrence similarity outperform the techniques that express tags as feature vectors. The Mean shift is not suitable for tags clustering because of the complicated tuning of window size parameter. The best results are achieved with the spectral K-means clustering.

5.7 Results for the NTF

We conduct the following experiments using the NTF algorithm, both datasets and all users:

- the basic NTF – NTF_b;
- NTF with prior knowledge – NTF_k;
- NTF with prior knowledge and with parallelism – NTF_{kp};
- NTF with prior knowledge and clustering – NTF_{kc}.

All the results are presented in Table 13.

Case	Precision	Recall	F-measure	Execution time
NTF_b^B	0.0465	0.0138	0.0212	34.485 s
NTF_b^M	0.0524	0.0140	0.0220	70.584 s
NTF_k^B	0.8508	0.4440	0.5834	84.576 s
NTF_k^M	0.9687	0.5572	0.7074	144.600 s
NTF_{kp}^B	0.8594	0.4494	0.5901	82.494 s
NTF_{kp}^M	0.9776	0.5591	0.7113	108.292 s
NTF_{kc}^B	0.7686	0.3968	0.5233	32.787 s
NTF_{kc}^M	0.8596	0.5094	0.6397	56.656 s

Table 13: The experimental results for the NTF

Correlated feature hashing	Spectral K-means	K-means	Mean Shift
informationretrieval, lecture, retrieval	retrieval, lecture, evaluation, informationretrieval	ranking, folkrank, information, retrieval, ir, informationretrieval	retrieval, information, ir
iccsexample, swikig, eswc2006	semwiki2006, swikig, eswc2006	wiki, swikig, semwiki2006, eswc2006	swikig, semwiki2006, eswc2006
bookmarks2.0, bookmark, system, socialnetworking	bookmark, system, bookmarks2.0, bookmarklet	bookmarks2.0, bookmarks, socialbookmarking, bookmarklet	bookmarks, software, ajax, socialbookmarks, web2.0, bookmarking, delicious, socialbookmarking, blog, online, tagging, folksonomy
eclipse, rails, code, develop	develop, informatik, rails, code, python, eclipse	code, svn, source, subversion	code, svn, source, subversion

Table 12: Different cluster examples when amount of clusters is set to 50% of the original tag space

5.7.1 NTF_b vs. HOSVD

The basic NTF algorithm produces the recommendations with unacceptable precision (0.0465 for the BibSonomy and 0.0524 for MovieLens) compared to the HOSVD baseline results (Section 5.5). The time performance, compared to the basic HOSVD results are improved $\sim$ 3.2 times for the BibSonomy and $\sim$ 1.5 for MovieLens. The average precision for all the users is poor because the factor matrices are initialized with random numbers only. The results imply that random values can be used for the active users only (the classes of users are defined in a Section 5.3). The active users have rich enough training sets to minimize the negative impact of random values and the precision rates from ~ 0.5 to ~ 1. However, for the average users and not active users (that is usually a majority and a common case in the real world applications) have too small training sets for the basic NTF to produce the recommendations within the acceptable precision, recall and f-measure metrics.

5.7.2 NTF_k vs. NTF_b

According to the results, our proposal to incorporate the personal prior knowledge and to promote the important aspects of the triplets significantly improves the average precision (0.8508 for the BibSonomy vs. 0.0465 and 0.9687 for MovieLens vs. 0.0524) for all the users. The precision and other measures are improved for the both datasets therefore we claim that the prior knowledge model is universal. The prior knowledge technique has to be adjusted for a particular dataset, e.g. the items in MovieLens dataset are tagged with more tags in comparison to the BibSonomy thus the variables for expressing the prior knowledge are adapted. NTF_k attains higher precision on MovieLens dataset than the BibSonomy. That is caused by more dense MovieLens dataset because the higher p-core value is used.
The execution time is improved almost twice compared to the basic NTF algorithm. It is caused by the extra time needed for incorporating the prior knowledge. The additional time is consumed by querying the database to fetch the prior knowledge for each user. We strongly believe the workload to resolve the prior knowledge could be optimized to achieve the similar time performance as for the basic NTF.

5.7.3 NTF_{kp} vs. NTF_k

The results of NTF_{kp} suggest that we achieve a slight improvement in time performance of NTF using our parallel approach. Compared to NTF_k, the execution time is improved by ~ 1.025 for the BibSonomy dataset and ~ 1.3 times for the MovieLens dataset.
The precision (also the recall and f-measure) for both datasets are almost the same, compared to the results of NTF_k. The slight differences can be argued by the fact that NTF algorithm does not guarantee the unique solution using the same initial tensor as the factor matrices are initialized with random values.
The minor improvement of time performance could be argued by the set of issues:

- To execute NTF_{kp} the Parallel Java Library [26] is used. Our implementation is not optimal because a large amount of memory (RAM) is required ($\sim$ 2 GB per process) for the data structures. Frequent memory allocations and deallocations are causing the time delay.
- Too many processes are started that are striving for the limited memory resources therefore, the execution time is not optimal. Parallel Colt framework [49] is using threads to execute the computations. The framework is heavily utilized for the

operations with matrices and tensors therefore, the additional processes are introduced.

- The standard Java Virtual Machine is not the optimal technology [21] to exploit the parallel processors. Java, compared to C/C++, can not control kernel threads to fully utilize the hardware.

5.7.4 NTF_{kc} vs. NTF_{kp}

We combine NTF prior knowledge approach with Spectral K-means clustering technique (the number of clusters is 50% of the original tag space) using Dice similarity measure. The previous results show that HOSVD and clustering technique with these settings produce the most accurate recommendations (Table 10). The results of NTF_{kc} infer that the execution time of tensor factorization can be significantly improved because of the reduced tag space. In comparison with the results of NTF_{kp}, the speed-up of the factorization is ~ 2.5 times for the BibSonomy dataset and ~ 1.9 times for the MovieLens dataset. The quality of recommendations is slightly decreased: $\sim 9\%$ for the BibSonomy and $\sim 12\%$ for the MovieLens data. The trade-off between the precision and time performance can be adjusted by changing the number of clusters.

6 Conclusion and future work

In this work, we propose to utilize clustering techniques for reducing tag space that significantly improves the execution time of tensor factorization and decreases the memory demands preserving the acceptable quality of the recommendations.
Two approaches of computing tags similarities are investigated. The former one utilizes tag pair co-occurrence similarity measures. The latter expresses tags as feature vectors and uses standard distance measures. The best prediction quality is achieved with the Spectral K-means clustering that employs co-occurrence tag pair similarity. The best trade-off between prediction accuracy and execution time is when the number of clusters equals to 50% of the original tag space size. Such reduced tag space improves time performance of a factorization and the prediction quality insignificantly decreases.
The model of the personal prior knowledge is presented. The facts about the users preferences are incorporated into the factor matrices of NTF to improve the quality of the recommendations. The approach is adjustable for a concrete dataset – the importance of a prior knowledge aspects can be differentiated.
Possibility to run NTF in parallel is explored and the minor improvement of the execution time is achieved.
The experimental results of a combined solution (NTF with a prior knowledge and clustering) strongly imply the time performance of a factorization is improved, the quality of the recommendations is acceptable.

As a future work, we intend to extend the clustering techniques with ability to detect tags with multiple meanings (polysemy), e.g., tag *java* can be considered as an island or a programming language. The extension would allow a particular factorization technique to correctly address polysemy of tags and as consequence the quality of recommendations would be improved. The second aim of a future work is to ensure whether the tags from the same cluster are semantically similar utilizing different semantic sources, i.e., WordNet dictionary and ontologies from open linked data on the Web [14]. The semantically enriched clusters should improve accuracy of the recommendations.
The process of incorporating a personal prior knowledge into a tensor should promote also the semantically similar tags. The proposal could exploit the proposed clustering techniques. Also, the more complex study for incorporating prior knowledge into the n-mode matrices of HOSVD should be investigated.
It is worthy to explore how to use a prior knowledge of all users in the initial tensor. Such approach would not require to recompute NTF algorithm for each user. The promotion variables should be computed according to the activity of all the users in the past.
The last goal of a future work is to improve our parallel implementation by training each of a column of factor matrices in parallel.

7 Acknowledgments

We are thankful to our supervisor Peter Dolog for the ideas and comments. We also would like to thank Daniel Sira, Frederico Durão and Darius Šidlauskas from the Aalborg University for the comments.

References

[1] Xavier Amatriain, Alejandro Jaimes*, Nuria Oliver, and Josep M. Pujol. Data mining methods for recommender systems. In Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor, editors, *Recommender Systems Handbook*, pages 39–71. Springer US, 2011. $10.1007/978 - 0 - 387 - 85820 - 3_2$.

[2] J.A.K. andJohn Riedl. Collaborative Filtering: Supporting Social Navigation in Large, Crowded Infospaces. *Designing Information Spaces: The Social Navigation Approach*, page 43, 2003.

[3] R. Badeau and R. Boyer. Fast multilinear singular value decomposition for structured tensors.

SIAM Journal on Matrix Analysis and Applications, 30:1008, 2008.

[4] B. Bai, J. Weston, D. Grangier, R. Collobert, K. Sadamasa, Y. Qi, O. Chapelle, and K. Weinberger. Learning to rank with (a lot of) word features. *Information retrieval*, 13(3):291–314, 2010.

[5] G. Begelman, P. Keller, and F. Smadja. Automated tag clustering: Improving search and exploration in the tag space. In *Collaborative Web Tagging Workshop at WWW2006, Edinburgh, Scotland*, pages 15–33. Citeseer, 2006.

[6] M. Bierlaire, Ph. L. Toint, and D. Tuyttens. On iterative algorithms for linear least squares problems with bound constraints. *Linear Algebra and its Applications*, 143:111 – 143, 1991.

[7] M. Brand. Incremental singular value decomposition of uncertain data with missing values. *Computer VisionECCV 2002*, pages 707–720, 2002.

[8] C. Chennubhotla and A.D. Jepson. Half-lives of eigenflows for spectral clustering. *Advances in Neural Information Processing Systems*, pages 705–712, 2003.

[9] Yun Chi and Shenghuo Zhu. Facetcube: a framework of incorporating prior knowledge into nonnegative tensor factorization. In *CIKM*, pages 569–578, 2010.

[10] A. Cichocki, R. Zdunek, A.H. Phan, and S. Amari. *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation*. Wiley, 2009.

[11] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on pattern analysis and machine intelligence*, pages 603–619, 2002.

[12] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley series in telecommunications. Wiley, New York, NY [u.a.], 1991.

[13] Datta, B. *Numerical Linear Algebra and Application*. Brooks/Cole Publishing Company, 1995.

[14] Frederico Durao and Peter Dolog. Extending a hybrid tag-based recommender system with personalization. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1723–1727, New York, NY, USA, 2010. ACM.

[15] C.S. Firan, W. Nejdl, and R. Paiu. The benefit of using tag-based profiles. 2007.

[16] Michael P. Friedlander and Kathrin Hatz. Computing non-negative tensor factorizations. *Optimization Methods Software*, 23:631–647, August 2008.

[17] J. Gemmell, T. Schimoler, M. Ramezani, and B. Mobasher. Adapting k-nearest neighbor for tag recommendation in folksonomies. In *Proc. 7th Workshop on Intelligent Techniques for Web Personalization and Recommender Systems*. Citeseer, 2009.

[18] J. Gemmell, A. Shepitsen, B. Mobasher, and R. Burke. Personalization in folksonomies based on tag clustering. *Intelligent techniques for web personalization & recommender systems*, 12, 2008.

[19] D.E. Goldberg. Genetic algorithms in search, optimization, and machine learning. 1989.

[20] G.H. Golub and C.F. Van Loan. *Matrix computations*. Johns Hopkins Univ Pr, 1996.

[21] Jordi Guitart, Xavier Martorell, Jordi Torres, and Eduard Ayguad. Efficient execution of parallel java applications, 2001.

[22] P.C. Hansen. The truncatedsvd as a method for regularization. *BIT Numerical Mathematics*, 27(4):534–553, 1987.

[23] A. Hotho, R. J
"aschke, C. Schmitz, and G. Stumme. Information retrieval in folksonomies: Search and ranking. *The Semantic Web: Research and Applications*, pages 411–426, 2006.

[24] A. Huang. Similarity measures for text document clustering. In *Proceedings of the Sixth New Zealand Computer Science Research Student Conference (NZCSRSC2008), Christchurch, New Zealand*, pages 49–56, 2008.

[25] Robert Jschke, Leandro Marinho, Andreas Hotho, Lars Schmidt-Thieme, and Gerd Stumme. Tag recommendations in social bookmarking systems. *AI Communications*, 21(4):231–247, December 2008.

[26] Alan Kaminsky. Parallel programming in java: pre-conference workshop. *J. Comput. Small Coll.*, 22:5–6, June 2007.

[27] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *RecSys '10: Proceedings of the fourth ACM conference on Recommender systems*, pages 79–86, New York, NY, USA, 2010. ACM.

[28] M.E.I. Kipp and D.G. Campbell. Patterns and inconsistencies in collaborative tagging systems: An examination of tagging practices. *Proceedings of the American Society for Information Science and Technology*, 43(1):1–18, 2006.

[29] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM J. Matrix Anal. Appl*, 21:1253–1278, 2000.

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

[31] Tao Li, Yi Zhang, and Vikas Sindhwani. A non-negative matrix tri-factorization approach to sentiment classification with lexical prior knowledge. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, pages 244–252, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.

[32] A.K. Milicevic, A. Nanopoulos, and M. Ivanovic. Social tagging in recommender systems: a survey of the state-of-the-art and possible extensions. *Artificial Intelligence Review*, 33(3):187–209, 2010.

[33] M. Mitchell. *An introduction to genetic algorithms*. The MIT press, 1998.

[34] U. Panniello, A. Tuzhilin, M. Gorgoglione, C. Palmisano, and A. Pedone. Experimental comparison of pre-vs. post-filtering approaches in context-aware recommender systems. In *Proceedings of the third ACM conference on Recommender systems*, pages 265–268. ACM, 2009.

[35] D. Ramage, P. Heymann, C.D. Manning, and H. Garcia-Molina. Clustering the tagged web. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 54–63. ACM, 2009.

[36] S. Rendle, L. Balby Marinho, A. Nanopoulos, and L. Schmidt-Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 727–736. ACM, 2009.

[37] GroupLens Research. Movielens data sets — grouplens research, 2010. [Online; accessed 01-December-2010].

[38] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[39] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Proceedings of the 5th International Conference in Computers and Information Technology*. Citeseer, 2002.

[40] Jonathon Shlens. Notes on kullback-leibler divergence and likelihood theory, 2007.

[41] G. Smith. *Tagging: people-powered metadata for the social web*. New Rider Pr, 2008.

[42] L. Specia and E. Motta. Integrating folksonomies with the semantic web. *The semantic web: research and applications*, pages 624–639, 2007.

[43] P. Symeonidis, A. Nanopoulos, and Y. Manolopoulos. Tag recommendations based on tensor dimensionality reduction. In *Proceedings of the 2008 ACM conference on Recommender systems*, pages 43–50. ACM, 2008.

[44] P. Symeonidis, A. Nanopoulos, and Y. Manolopoulos. A unified framework for providing recommendations in social tagging systems based on ternary semantic analysis. *IEEE Transactions on Knowledge and Data Engineering*, pages 179–192, 2009.

[45] Yoshio Takane, Forrest Young, and Jan de Leeuw. Nonmetric individual differences multidimensional scaling: An alternating least squares method with optimal scaling features. *Psychometrika*, 42:7–67, 1977. 10.1007/BF02293745.

[46] K.H.L. Tso-Sutter, L.B. Marinho, and L. Schmidt-Thieme. Tag-aware recommender systems by fusion of collaborative filtering algorithms. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1995–1999. ACM, 2008.

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

[48] Max Welling and Markus Weber. Positive tensor factorization, 2001.

[49] Piotr Wendykier and James G. Nagy. Parallel colt: A high-performance java library for scientific computing and image processing. *ACM Trans. Math. Softw.*, 37:31:1–31:22, September 2010.

[50] X. Wu, L. Zhang, and Y. Yu. Exploring social annotations for the semantic web. In *Proceedings of the 15th international conference on World Wide Web*, pages 417–426. ACM, 2006.

[51] Z. Xu, Y. Fu, J. Mao, and D. Su. Towards the semantic web: Collaborative tag suggestions. In *Collaborative web tagging workshop at WWW2006, Edinburgh, Scotland*. Citeseer, 2006.

[52] Qiang Zhang, Michael W. Berry, Brian T. Lamb, and Tabitha Samuel. A parallel nonnegative tensor factorization algorithm for mining global climate data. In *Proceedings of the 9th International Conference on Computational Science*, ICCS 2009, pages 405–415, Berlin, Heidelberg, 2009. Springer-Verlag.

A Appendix: Genetic algorithms - a way to tune tensor based recommenders

In this section, we propose a diagnostic tool for estimating optimal parameters for tensor based recommenders. The approach is based on Genetic Algorithm (GA) [19, 33] and it identifies the optimal parameters for HOSVD based recommender so that the best possible accuracy is attained. Below is described the problem of tuning the parameters, also introduced short related work and finally our solution is described.

A.1 Motivation

Tensor based recommenders are complex algorithms that require detailed tuning and adjustments of particular parameters or settings to provide the most accurate results. For HOSVD based recommenders [44], there must be provided a number of preserved top singular values for each mode matrix. These 3 parameters c_1, c_2, c_3 determine dimensions of the core tensor as only c_1, c_2, c_3 top left singular vectors from the SVD of $1, 2, 3$ mode matrices respectively are used for construction of the core tensor. The motivation to filter out small singular values and corresponding left singular vectors is to achieve better approximation of mode matrices. This removes noise and preserves only important semantic information. The common approach to determine the parameters is to empirically estimate a percentage of original diagonal of matrix with singular values, e.g., let us assume a diagonal of the original singular values matrix contains the following sorted singular values $1.2, 1.15, 0.95$ that sum to $(1.2 + 1.15 + 0.95) = 3.3$, we want to preserve 70 percent of the original diagonal therefore the parameter c_1 is set to 2 as $(1, 2 + 1.15)/3.3 \sim 70\%$.

Symeonidis et. al [44] provide an analysis about the parameters and their significant impact on the precision of the recommendations. According to their empirical results, a 70% of original diagonal matrices provides the best quality of recommendations. However, the given setting does not hold in our work as for the BibSonomy dataset the best results are achieved with the parameters set to ($c_1 \sim 74\%, c_2 \sim 47\%$ and $c_3 \sim 78\%$). Rendle et. al [36] also describe that recommenders based on HOSVD are sensitive to small changes of the 3 parameters. In both works, the parameters are tuned manually without usage of any diagnostic methods. Such approach is computationally expensive when large dataset is considered. This naive approach search through a defined parameters space, that requires to execute 3 inner loops as there is no correlation between the parameters. Such finding of the optimal parameters is slow and not efficient.

We utilize GA to speed up the search of the optimal parameters as it can faster explore the parameters search space.

A.2 Our approach

The GA is search heuristic that we use to find the optimal parameters. The GA are adapted to our search problem in the following way:

- Each parameter represents a gene.
- The parameters c_1, c_2, c_3 are genes that are together encoded in a chromosome.
- A population of possible solutions consists of k different chromosomes.
- Fitness function is the average precision for considered users and for the provided chromosome (parameters c_1, c_2, c_3).

Below are described steps of the process of searching for the optimal parameters with the GA.

1. A population is randomly generated in a such way that each chromosome consists of 3 genes. Each gene represents a particular parameter and it is randomly initialized (parameter c_i belongs to the defined interval $(0.4, 0.8)$.
2. Calculate the fitness function of each chromosome from the population. The result of the fitness function is average precision for the given parameters.
3. New population is created such that chromosomes that produce best average precision are reproduced. New chromosomes are created with mutation and cross-over operations.
4. Go to step 2, repeat until fixed number of iterations is reached.

Once the searching process is finished, GA returns a chromosome with the best average precision (result of the fitness function). The given chromosome contains the optimal parameters c_1, c_2, c_3.

A.3 Preliminary results

We provide a short study of the technique and compare it with the naive approach for searching the optimal parameters. The experiments are conducted on MovieLens dataset. Values of the parameters range from 0.4 to 0.8 (preserve from 40% to 80% of diagonal of original mode matrix).

A.3.1 Naive approach

The search space for the parameters is represented as set of the possible values that range from 0.4 to 0.8, i.e.,

$$\text{SearchSpaceSet} = \{0.35, 0.4, 0.45, 0.5, \ldots, 0.75, 0.8\}$$

The goal is to find 3 different parameters that belong to *SearchSpaceSet* hence there must be executed 3 inner loops so that all possible combinations of c_1, c_2 and c_3 are explored. Therefore, execution time of the search for parameters is defined as

$$|\text{SearchSpaceSet}|^3 \times t_{\text{fitness function}}$$

According to aforementioned formula, the search for optimal parameters of HOSVD recommender on Movielens dataset takes $10^3 \times t_{\text{fitness function}}$ as the size of the $|\text{SearchSpaceSet}| = 10$ and $t_{\text{fitness function}}$ is computation time for selected users for a particular combination of the parameters.

A.3.2 Our approach

The approach based on GA of searching for optimal parameters takes the following execution time:

$$\text{iterations} \times \text{PopulationSize} \times t_{\text{fitness function}}$$

where the number of iterations is denoted as iterations and size of the population is given by PopulationSize. The number of iterations is set to 20 and size of the population to 5 different triples of parameters. Therefore, the execution time of search for the parameters with GA on MovieLens dataset takes $20 \times 5 \times t_{\text{fitness function}}$

Obviously, the approach based on GA outperforms the naive one. It improved almost 10 times the execution time of search for the optimal parameters. Generally, is expected that term iterations $\times$ PopulationSize $\ll |\text{SearchSpaceSet}|^3$ and therefore in most cases it significantly speed up search process for the optimal parameters.
The found parameters c_1, c_2 and c_3 for the MovieLens dataset are $0.431, 0.553, 0.44$ respectively.

A.4 Conclusion

We propose a novel approach of tuning tensor based recommenders. The solution is based on GA, that more optimally explore search space of the parameters for HOSVD based recommenders. Aforementioned preliminary results prove the contribution of the approach. The given technique could be easily adopted for other tensor based recommendation systems, e.g., NTF recommender requires to specify one dimension of factor matrices a priori. The approach would be adjusted in such way that sizes of factor matrices would be encoded into chromosomes and the fitness function would compute average precision for given users and provided chromosome that contains factor matrices sizes.

B Appendix: Design and implementation

In this part, the design of the implemented recommendation system based on the HOSVD and NTF tensor factorization techniques are presented. Firstly, the general structure of the system is introduced. Secondly, we present each part of the system more precisely. Moreover, the main data structures and algorithms are defined. Finally, we state which open source frameworks are used in our system.

B.1 Architecture

The architecture of our system is composed of several layers. The general structure is shown in the Figure 10.

The system is designed to be easily extended with the multiple recommendation techniques while using the same database, data structure, utilities, statistics and the presentation layers. We designed and implemented the p-core and tag-space clustering extensions that are available for HOSVD based and NTF based recommenders. Also, prior knowledge extension is available for NTF.

B.2 Detailed overview

B.2.1 Database layer

We use the samples of the real world data sets (as described in Section 5.2) to generate recommendations. The data is stored in the databases. We choose to use Hibernate framework[3] for the object/relational persistence and query service. We are able to manipulate with the data as the objects. Also, we are able to bind a number of the databases to the system – each of the data set is stored in the different databases. Moreover, the databases can be different, e.g., MySQL, PostgreSQL, Oracle, etc. The structure of a database layer is presented in the Figure 11.
We designed an importer of the data sets in case data set is not available in a dump file format. The importer can be extended to implement the database scheme of a concrete data set. The concrete importer is invoked to import data when a dataset is chosen by the user using Graphical User Interface (GUI) of the system.
We designed a simple but efficient cache technique to minimize the creation time of the initial tensor. We store the results of each different query for each

[3] `http://www.hibernate.org`

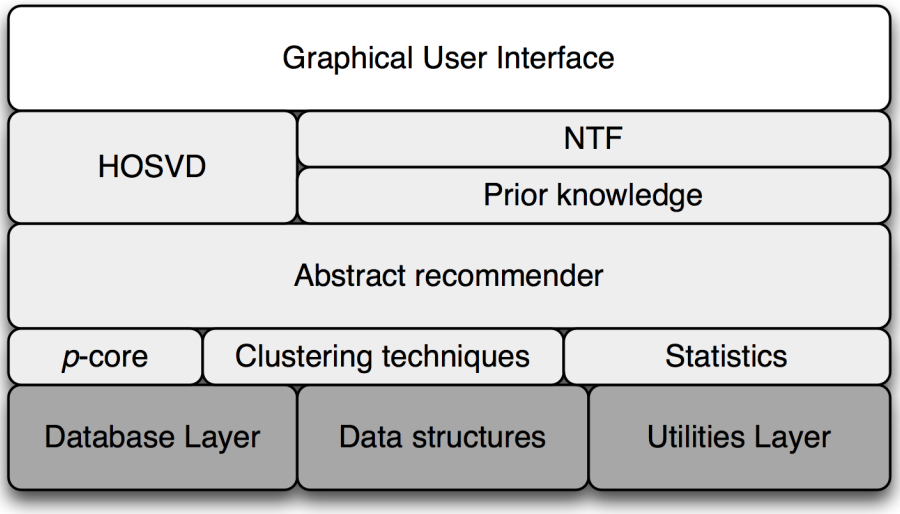

Figure 10: General overview of the system

different persistence manager (i.e. for each different database) in a memory to improve the performance. It is possible because we use `read` only operations to construct the initial tensor, provide recommendations and the other actions. The data is imported before using the Queries API and this is executed only once when data is not imported. Caching technique could be easily extended in case we would need to use `write/update` operations for the other actions than data import.

B.2.2 Data structures

We divide the data structures that are required for our system, into the two parts: the structures needed for the database layer and the others (mostly used in the statistics layer).

The data structures of the database layer capture the relations of triplets *user-tag-item* (Figure 12). There is an interface `DefaultEntity` that identifies an object of a database. The abstract classes `DefaultUser`, `DefaultTag` and `DefaultResource` implement `DefaultEntity` and represent user, tag and item respectively. There are unique implementations for each of the abstract class per each data set.

The other data structures as `Recommendation`, `Statistics`, `AverageStatistics` are used to represent the generated recommendation for the user and to provide information about the statistics of the recommendation(s).

B.2.3 Utilities layer

There is a number of utilities (helpers) classes used:

- `Settings` – used for generating the recommendations: identifies the currently selected data set.
- `RandomUtil` – used for generating the recommendations: helps to split the data set of a user into the training and the evaluation sets (as described in Section 5.2).
- `IOUtil` – used for managing `Input/Output` operations. Also, writes the average statistics to the file.
- `CollectionsUtil` – used for manipulating the objects of a type `Collection`.

B.2.4 Abstract recommender

The recommender layer can be split into the several parts:

- Generating initial tensor
- Computing tensor factorization
- Generating a list of recommended items

Generating initial tensor To generate the initial tensor, a user must be provided (for who the recommendations will be computed) and a number of users to be used. We provide the algorithm (see Algorithm 2) of the initial tensor creation.
We describe the main ideas of the algorithm:

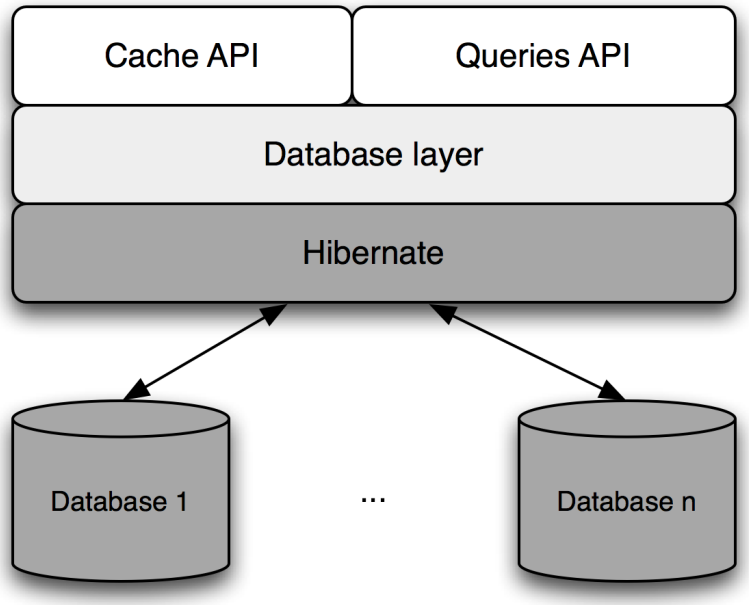

Figure 11: Database layer with caching service

- Users set *users* is generated.
- Tags set *tags* is resolved. These are the tags used by the *users*.
- Items set *items* is resolved. These are the items tagged with the *tags* by the *users*.
- Iterate through all the *users*, *tags* and *items*. Resolve if a current *item* is tagged by the current *tag* and *user*. If so, the existing relationship is marked in the tensor.
- Store the empty relations (if a user does not tag a item with a tag) of a current user. These relations will be used to resolve the recommendations.

Algorithm 2 The algorithm for creating the initial tensor

Data: User *user*, number of users *users*
Result: Initial tensor *tensor*

```
15 Generate a set of users
   for i ← 0 to users do
16 |  get user u, add it to the users set users
17 end
18 Get a set of tags used by the users
   Get a set of items tagged with the tags by the users

19 tensor = new double[tags][users][items]

20 int uIndex = 0;
   foreach user u ∈ users and uIndex < users do
21 |  for tIndex ← 0 to tags do
22 |  |  for rIndex ← 0 to items do
23 |  |  |  if a current user tagged a current item with
   |  |  |  a current tag then
24 |  |  |  |  tensor[tIndex][uIndex][rIndex]++
25 |  |  |  end
26 |  |  end
27 |  end
28 |  if current user == u then
29 |  |  StoreEmptyRelations (tensor)
30 |  end
31 |  uIndex++;
   |
32 end
```

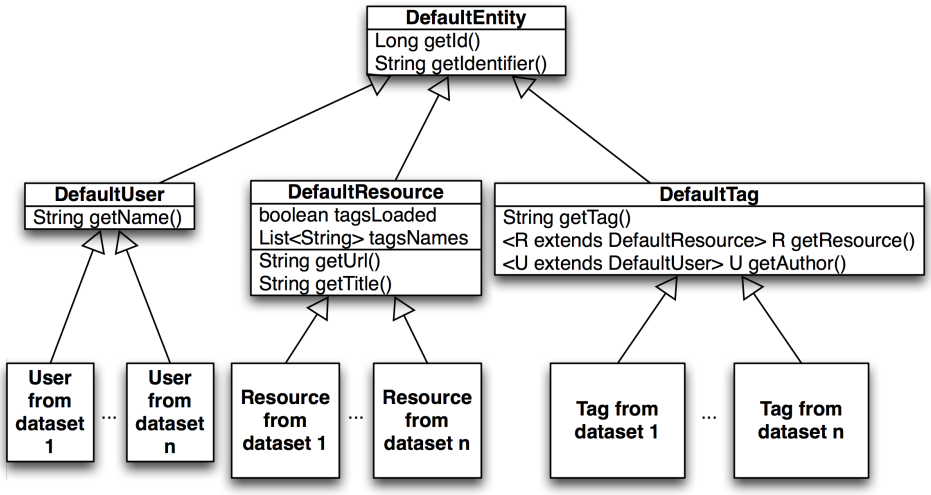

Figure 12: Data structures of the database layer

Computing tensor factorization To compute a tensor factorization, the initial tensor has to be defined. Currently, the recommendations can be computed using HOSVD (Algorithm 3) or NTF algorithms (as presented in Section 4.1.3).
We describe the key points of HOSVD algorithm:

- Firstly, the initial tensor is split into the three mode matrices and the Singular Value Decomposition is computed for each mode matrix.
- Secondly, the dimensions are reduced for the matrices that are results from the Singular Value Decomposition for each mode matrix. These reduced matrices are multiplied to compute a core tensor.
- Finally, the reduced matrices are transformed, and with n-mode multiplication the factorized tensor is computed.

Algorithm 3 The algorithm for tensor factorization

Data: Initial `tensor`
Result: Factorized `tensor`

33 Splits initial tensor into the first, second and third mode matrices and computes SVD for each mode matrix
`splitIntoModeMatrices` (tensor)

34 Computes a core tensor: applies dimensional reduction, executes matrices multiplication
`computeCoreTensor` (){
`matrixDimensionalReduction`
`multiplyDifferentMatrices`
}

35 Computes factorized tensor: transforms the three reduced matrices, applies matrices multiplication
`computeFinalTensor` (){
`transformMatrix` $(redMat1, fMatDim)$
`transformMatrix` $(redMat2, sMatDim)$
`transformMatrix` $(redMat3, tMatDim)$
`multiplyDifferentMatrices`
}

Generating a list of recommended items When the factorized tensor is computed, the recommendations can be resolved. It is possible to do this because the empty relations are known. The value of an empty relation is the coordinate in an approximated tensor. It is iterated through the list of the empty relations and checked if a score in a concrete coordinate of the factorized tensor is positive. If so – the item is resolved and an object of `Recommendation` is created and added to the list of recommendations. Finally, the list of the

recommendations is sorted by the scores.

B.2.5 p-core

The main goal of this extension to make a data set more dense. The members of triplets (*users*, *tags* and *items*) are generated as it is described in Section 5.2. More dense data is forwarded to the recommender layer to create initial tensor or the clustering techniques.

B.2.6 Clustering techniques

Clustering techniques reduces a tag space as described in Section 4.2. The extension can be combined with any recommender available (currently HOSVD and NTF) in the system.

B.2.7 Prior knowledge

This extension is designed for NTF (as explained in Section 4.1.2) and can be utilized by HOSVD recommender. It may be possible to re-use this extension by the recommenders based on various factorization techniques.

B.3 Statistics layer

This is a collection of the simple helper classes that computes metrics and provides the additional information about the recommendations. The metrics are: precision, recall and f-measure (as described in Section 5.4). While analyzing the additional information it is possible to know which tags were used by the user and which tags were used to tag a item. Also, the training, evaluation sets and a top-N list can be resolved.

B.4 Graphical User Interface

Using the GUI a user is able to select the data set, a particular user for who the recommendations will be generated, the factorization method and whether the clustering technique should be used. In case basic method is selected, the user has to provide a number of random users to be used to construct the initial tensor. Otherwise, a user has to provide the value for the variable p for p-core based factorization method.
The screenshots of the application are provided in the Figures 13, 14 and 15.

B.5 The Open source frameworks

We use the following frameworks for our recommender system:

- **Prallel Colt**[4] [49] – to execute various mathematical operations on tensors and matrices. It has a rich API and performs well. It is developed by the scientists of CERN[5] and it is used for the CERN projects.
- **Parallel Java Library**[6] [26] – to utilize the parallelism with Java. The rich API provides a variety of options to execute code in parallel.
- **Apache Mahout**[7] – to exploit the clustering techniques like k-Mean, Spectral k-Mean, Mean shift. During our work, we attempted to integrate Eigencut clustering technique. However, we faced some problems with the implementation of the method. We found and reported one error in the code to the developers of the Apache Mahout framework.
- **Hibernate** – to manipulate data as the objects of Java. We describe our choice of Hibernate in the section B.2.1.

B.6 Conclusion

We present the design of Recommender application. HOSVD and NTF factorization techniques are implemented. Also, the extensions of p-core, clustering techniques and prior knowledge are implemented. The main advantages of this system are as follows:

- Database layer enables to manipulate data items as objects. A number of the different databases can be bound to the system.
- Flexible Abstract recommender layer can be easily extended or the new factorization techniques can be implemented.
- Statistics layer can represent the statistics for any factorization technique. New metrics can be easily added.
- Graphical User Interface is simple to use.
- We use the powerful open source frameworks to minimize the efforts and time when manipulating with data and tensors (also matrices).

[4] http://sites.google.com/site/piotrwendykier/software/parallelcolt

[5] http://www.cern.ch

[6] http://www.cs.rit.edu/~ark/pj.shtml

[7] http://mahout.apache.org/

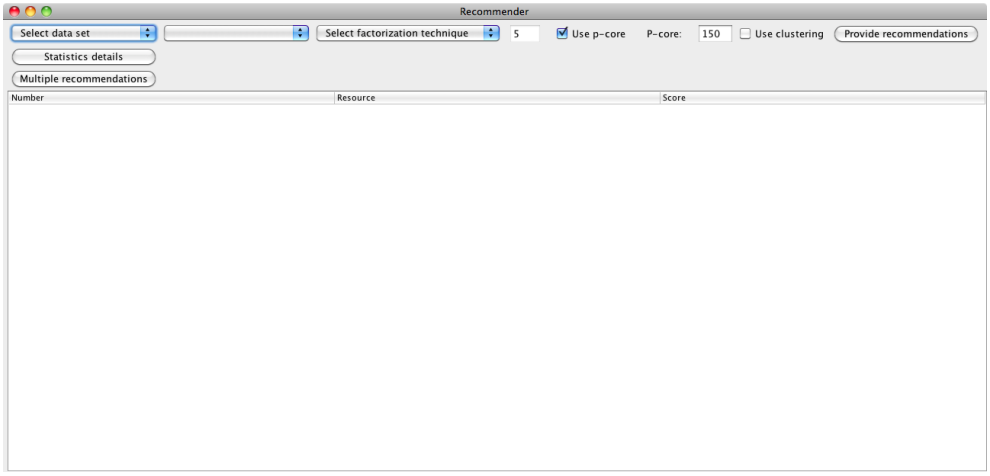

Figure 13: The main window of a recommender system

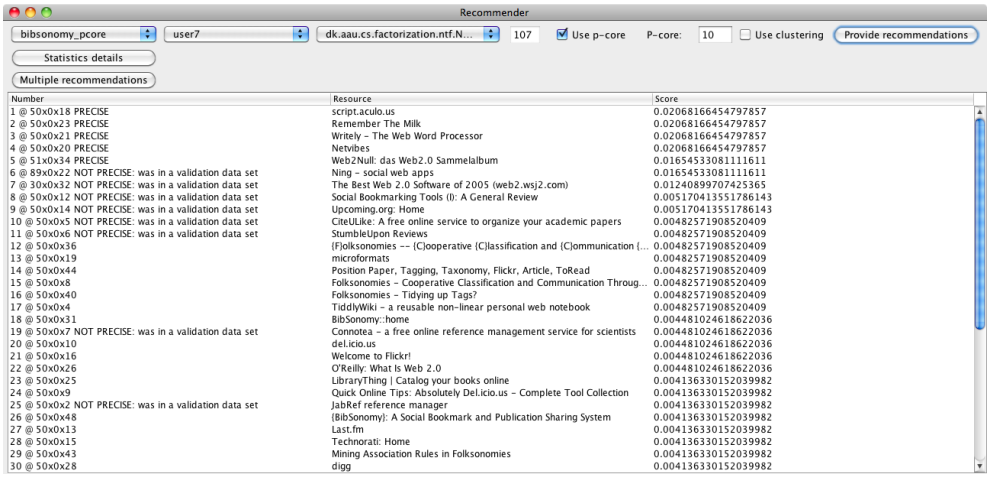

Figure 14: The main window with the generated recommendations

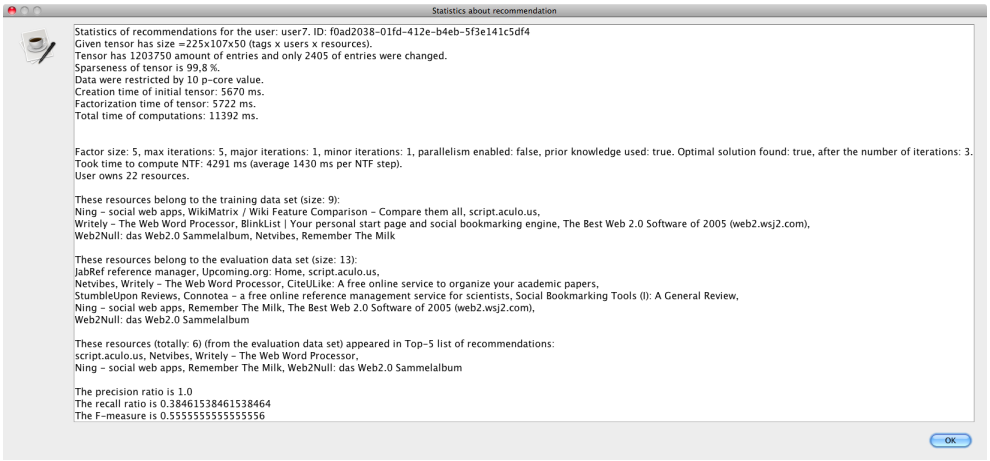

Figure 15: The statistics window for the generated recommendations