AALBORG UNIVERSITY

PROJECT REPORT

Analysis of the recommendation systems based on the tensor factorization techniques, experiments and the proposals

Group:
d519a

Authors:
Martin Leginus Valdas Žemaitis

Department of Computer Science
January 4, 2011

AALBORG UNIVERSITET

Department of Computer Science

Title:
Analysis of the recommendation systems based on the tensor factorization techniques, experiments and the proposals

Project Period:
1st of September to
11th of January, 2011

Project Group:
d519a

Project Members:
Martin Leginus
Valdas Žemaitis

Supervisor:
Peter Dolog

Number of copies printed: 4

Pages in report: 67

Pages in total: 70

Abstract:

We analyse the state-of-the-art recommendation systems based on the tensor factorization techniques. The comparison of these systems is provided, the possible problems and the drawbacks are identified. We developed the HOSVD recommendation system. Using the implemented recommender system the experiments are conducted and the results are reported. The majority of the identified problems are verified during the experiments. Moreover, we propose a number of the possible solutions and improvements to the problems. Finally, we describe the main goals of our work during the next semester.

Preface

A PDF version of this report, the source code of our implementation and compiled application are available on this website:

`http://cs.aau.dk/~mleginus/recommender`

We would like to thank our supervisor Peter Dolog and Frederico Durao from the Department of Computer Science at Aalborg University.

CHAPTER
1

Introduction

A common user of the internet has to struggle through a constantly increasing amount of the irrelevant information. Nowadays, recommendation systems try to attack this problem providing the personalised recommendations to the user. These applications try to guide the user with the personalised and adjusted information based on her interests.

Recommendation systems analyse users preferences and habits for the better estimations of the personal recommendations. The more informative users interests are, the more accurate information can be recommended. The state-of-the-art recommendation systems are able to reveal the recommendations that have no or little textual similarities to the users preferences. This is possible because of the semantic similarities between them.

To simplify a management (sorting, categorizing and also recommending) of the information and the content, a new innovative technique, called tagging, has emerged. This practice allows to utilize a collaborative and social power of the users, where each user can freely assign a tag to the content item (e.g. web link, photo, video, article).

Tensor factorization is a widely spread technique that has been applied in the various domains and recently has become popular in the area of the recommendation systems. The triple relations of user-tag-item can be captured by the tensors. During the factorization process there can be revealed the latent relations between the involved objects.

The goals of our project are to analyse the recommendation systems based on tags and the tensor factorization techniques, identify the shortcomings of them and propose the possible solutions to overcome them. We present the analysis of a number of the techniques in this area. We design and implement the recommendation system based on the some ideas presented in analysis. The drawbacks of the analysed techniques are identified during the experiments. Based on the gained knowledge and the conducted experiments we will propose the extensions to these systems. Finally, we will identify the most promising directions for the further improvements.

CHAPTER
2

Analysis

In this chapter an analysis for our project will be made. Firstly, we will introduce the main types of the recommendation systems, where we try to point out the main advantages and disadvantages of a particular system. In the second part, we will present the state-of-the-art recommenders – the majority of them are based on the tensor factorization techniques. However we also cover a single hybrid tag-based recommender system extended with a semantic factor. In the end of this chapter, the recommenders will be compared and evaluated from the different aspects. Finally, we will provide the statements for the chosen direction of our project.

2.1 Overview of the recommendation systems

The main types of the recommendation systems will be analysed in the following section. We will try to include the most important advantages and disadvantages of each introduced type.

2.1.1 Content-based

Content based (CB) [16] recommenders analyse titles of items and profiles of users. The profile of a user shows his/her preferences. The recommendations are generated by computing the similarities between the user profile and the items known by a system. Such recommenders can be used for various purposes, e.g. to recommend items to buy, articles to read, historical objects to visit.
It is easy to implement such systems and they provide the recommendations within the acceptable time frame.
However, the quality of the recommendations depends on the textual information – it can be affected negatively because of the differences (of the textual representation) between the items and the profiles. The textual representations can be dissimilar at all, but the meanings can be semantically similar. Therefore, extensions should be introduced to improve the quality of the recommendations. Furthermore, CB recommenders tend to perform poorly when the profiles of the users are less informative – systems must take care of so called cold-start ([10]) problem when little or almost nothing is known about the preferences of the user.

Case-based

Case-based recommenders [22] are special type of CB systems. These systems use some concrete criteria (cases) (that are common for a group of objects) to decide about the similarities between items and users profile. For example, a user can specify which cases (features) should be satisfied for an item (e.g. size, weight, color etc.), not caring about titles of the items. Systems can easily identify the items that satisfy the cases.

2.1.2 Collaborative Filtering

Collaborative Filtering (CF) [21] [14] is a process of filtering for particular information or patterns from a data set that is provided by the collaborative work of users (e.g. virtual agents, human beings). The recommendation systems use collected data by the collaborative users to provide the suggestions what a particular user may be interested in by knowing the preferences (e.g. likes or dislikes) of the user. In other words, the recommenders try to find the behaviour patterns for a particular user as similar as possible to the behaviour of the other users in a collaboratively combined data set based on the similar preferences between the users. The assumption is that "those who agreed in the past tend to agree again in the future". The example of a recommendation system based on CF could be a system that proposes to read a book for a user knowing what types of the books this particular user likes or dislikes and analysing all the likes and dislikes of the users known by the system.
It is one of the most popular method to provide the recommendations. The biggest advantage of such systems is the usage of the users opinions that are considered as important and relevant for the others. It is also easy to implement such systems and they perform well using dense data set ([23]). The main disadvantages occur when data is sparse – most often it affects the performance negatively. Also, the quality of the recommendations depends on the opinions of the users, i.e. if there are only the few common interests it is possible that the system will recommend the items without common interests.

2.1.3 Other systems

The systems as demographic, utility-based and knowledge-based are not so popular these days. However, we will briefly describe each of them:

- **Demographic** system categorizes users based on the personal attributes (preferences). It is able to provide the recommendations for a demographic group knowing the preferences of the group and the features of the items.

- **Utility-based** system provides the recommendations based on the utility. The utility is computed for each item and a particular user. The most problematic and challenging step is to compute and estimate the utility function.
- **Knowledge-based** system uses a function that can compute how a specific item meets the preferences for a given user. Recommendations are provided based on the output of the functions.

2.1.4 Hybrid-based

Hybrid-based recommendation systems [3] combine the advantages of two or more recommendation systems to minimize the disadvantages of each of them and to benefit for the quality and performance of the recommendations. Most often CB, CF systems are combined to achieve the better recommendations.

2.1.5 Tag-based

Tagging is an activity when a user associates an item with the arbitrary word(s), called tag(s). The tags describe a content of the item and interests of the user. Tagging process can be modelled as:

$$Tagging : (U, T, I) \tag{2.1}$$

where U is a user that tags an item I with a tag T.
This activity simplifies a process of categorizing and retrieving content and takes advantage of the collaborative tagging – where more users are involved into the tagging of the same content. Tag-based recommenders [15] analyse tags, discover preferences of a given user and provides suggestions for the user which items could be interesting. The main advantage of the tag-based recommenders is that user preferences and interests are expressed by used tags of the given person. Therefore, these recommenders provide more accurate and personalized recommendations.
On the other hand, majority of the tag-based recommenders consider only textual (syntactical) similarities among tags. It causes problems when there are tag synonyms and according to the syntactical similarity these relations will not be revealed. The similar problem can occur when a given tag has more different meanings – so called polysemy. These issues are handled by various techniques which extend standard tag-based recommenders and provide semantically more accurate recommendations (as analysed in the sections 2.2, 2.3.1).

Folksonomy

It is common that tagging systems enable social connectivity (e.g. Delicious, Flickr) via common interests which are revealed using the same tags. A term *folksonomy* ([15]) means a "user-generated and distributed classification system" that evolves when a group of the users "collectively tag resources". The main features of the folksonomies are ability to adapt (change) rapidly, flexibility, "free-for-all collaborative customisation and their serendipity", information classification. The benefits of the folksonomies are as follows:

- able to index and cluster the information;
- able to identify the communities;
- allow searching and browsing of the information;
- are utilised as data sources for the collaborative recommender systems.

2.2 Hybrid tag-based recommender system with personalization

The authors of an article [8] introduce a hybrid extension for a tag-based recommender system. The extension is based on the semantic factor (as explained in a section 2.2.1): the recommender system is able to provide the recommendations based on the semantic meanings of tags, not only the syntax similarities between the tags. A number of the recommendation systems rely on the syntactical similarities between the tags. Such systems may not be able to discover the semantic differences between the tags like *Java* (may have a meaning of a programming language or an island), *apple* (a fruit) and *Apple* (a company), *jaguar* (an animal) and *Jaguar* (a car).
A hybrid approach embraces both: a similarity calculus and a semantic factor. The similarity calculus is enriched with the factors as follows:

- **Tag Similarity (TS)** is a combination of cosine similarity ($CosSim$) for the text and a semantic similarity ($SemSim$) between the tags which is presented in a section 2.2.2. TS formula is:

$$TS_{(D_i;\, D_{ii})} = CosSim(T_{D_i}; T_{D_{ii}}) * SemSim(T_{D_i}; T_{D_{ii}}), \tag{2.2}$$

  where T is a set of the tags of a particular document, D_i and D_{ii} are the particular documents from a set of the documents D.

- **Tag Popularity (TP)** is a total number of tag appearances in the available resources. The more times a tag appears in the resources, the more popular it is.

- **Tag Representativeness (TR)** reflects how good a particular tag represents a document (the tag was assigned to a given document).

- **Affinity between user and tag (AF)** measures how often a tag is used by a user. The more often tag is used by a user, more precisely it reflects the interests of a user. This is an important factor for the personalization of the recommendation results. AF formula is:

$$\begin{aligned} AF_{(u,\,t)} = card\{r \in Documents \mid (u,t,r) \in R, R \subseteq U \times T \times D\} \\ /card\{t \in T \mid (t,u) \in R_u, R_u \subseteq U \times T\}, \end{aligned} \tag{2.3}$$

  where *card* (cardinality) is the size of a set, u is a particular user, t is a particular tag, r is a particular resource, R is a set of resources, U is a set of users, T is a set of tags and D is a set of documents.

The final hybrid similarity score (HS) is calculated as follows:

$$HS_{(D_i;\,D_{ii})} = [(Ds_{D_i} + Ds_{D_{ii}}) \times TS_{(D_i;\,D_{ii})}] \times AF_{(u,\,t)}, \tag{2.4}$$

where Ds is a document score, defined as follows:

$$Ds = \sum_{i=1}^{n} TP_i \times \sum_{i=1}^{n} TR_i, \tag{2.5}$$

where n is a total number of the known tags by a system, TP_i is a tag popularity of a particular tag and TR_i is a tag representativeness of a particular tag.

2.2.1 Semantic similarity factor

The semantic similarity factor reflects the relations between the syntactically different tags in the different tag sources. There is considered two types of the relations between the tags:

- **synonyms**, e.g. *house* and *apartment*;

- **equivalency** in a context, e.g. *java*, *programming.*

2.2.2 Semantic similarity

There are utilized the two data sources for the semantic relations: WordNet dictionary [28] and "ontologies from open linked data published on the Web" [2]. To compute a semantic similarity score, the tags must be found out in the semantic source and then the score (*SemSim*) can be computed by the following formula:

$$SemSim(s,\,t) = WNSim(s,\,t) \times OntoSim(s,\,t) \mid \{(s,\,t) \in T\}, \tag{2.6}$$

where s and t are the particular tags from the tags set T, $WNSim$ is a similarity score retrieved using WorldNet and *OntoSim* is a similarity score computed using the ontologies:

- $WNSim$ is issued when two tags are chosen for the comparison. This score is calculated using a formula proposed by Wu and Palmer [29]:

  $$WNSim(s, t) = 2 * depth(LCS)/\left[depth(s) + depth(t)\right], \tag{2.7}$$

  where s and t are the source and target words that are compared; "$depth(s)$ is the shortest distance from a root node to a node S on the taxonomy"; "LCS denotes the least common sub-submer of s and t".

- *OntoSim* considers the tags as the classes. This formula is calculating three types of the relations between the two classes (tags). The types are as follows:

  - *Hierarchical relation* (HR)
  - *Negative relation* (NR)
  - *Positive relation* (PR)

  The relations are retrieved using Sparql queries[1]. Finally, the ontological similarity is calculated as follows:

  $$OntoSim(s, t) = \sum_{i=1}^{n} HR(s, t) * \sum_{i=1}^{n} NR(s, t)/\sum_{i=1}^{n} PR(s, t), \tag{2.8}$$

  where s and t are the tags from the set of tags T.

2.2.3 Evaluations

The biggest advantage of the semantic extension for the recommender system using tags is ability to consider the semantic relations between the syntactically different tags. The results of the experiments show that in most of the cases, the semantic similarity score was higher than the syntax similarity score between the sets of the tags that shared at least one a syntactically equivalent tag. Furthermore, the formulas used by the recommender are not computationally expensive that should benefit to the overall performance of the recommender. The usage of WorldNet and linked data resources assure that rich and constantly being updated dictionaries are used to calculate the semantic similarity score. However, the results of the experiments show that 59% of the recommendations were accepted and 41% were rejected by the users. These may indicate that the recommender needs an improvement to provide the results of a higher acceptance level. Also, the recommender

[1]http://www.w3.org/TR/rdf-sparql-query

can not deal with the misspelled tags that affect negatively the quality of the recommendations. The usage of the WorldNet and linked data resources may cause the performance problems because queries must be executed via internet and the response time to the queries can affect negatively the overall performance of the recommender. Finally, the authors of [8] suggest to extend a semantic factor "by measuring the *tag pair co-occurrence*" that should help to provide the better recommendation results.

2.3 HOSVD based recommender systems

In this section, we will analyse the recommendation systems based on tensor factorization using Higher Order Singular Value Decomposition (HOSVD). Furthermore, the possible extensions for HOSVD systems will be introduced and analysed.

2.3.1 Recommendation systems based on tensors dimensionality reduction

The state-of-the-art recommendation systems exploit the provided data (users - u, information items - i, tags - t) only in 2-dimensional relations. These pairs: *(users, tags), (users, information items), (tags, information items)* are analysed by the different types of the algorithms [3] which try to discover the most relevant and suitable content – tags or information items for the users. However, the mentioned algorithms do not reflect 3-dimensional base of the provided data and therefore, they are not able to analyse all the relationships between the triplets of data.
Researcher Symenonidis et al. (2008) [24] realised that involving and exploring existent relationships between tags, users and information items can reveal more relevant outcomes. This approach is exploring the complex 3-dimensional relations and is able to detect the latent associations which provide the better recommendations. The technique is based on a Singular Value Decomposition (SVD) [7] which computes matrix approximation. Usage data of a recommendation system are represented by a 3-dimensional tensor – 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, see the following function:

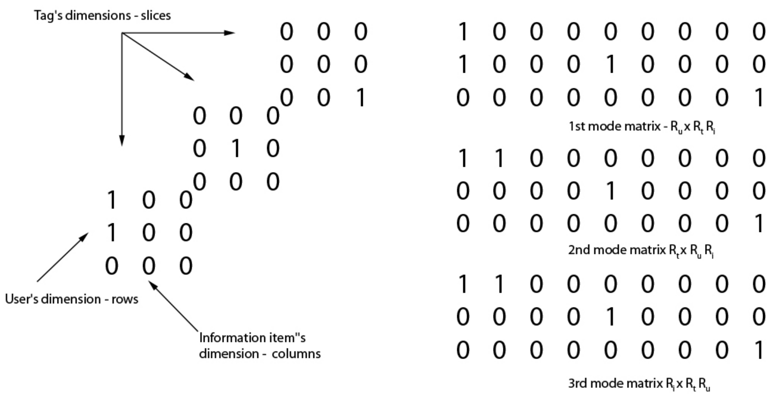
$$a_{u,i,t} \in A, a_{u,i,t} = \begin{cases} 1, & (u,i,t) \text{ is an existing relationship in a system} \\ 0, & \text{no association between } (u,i,t) \end{cases} \tag{2.9}$$

We are including an illustrative example to better describe the tensor reduction technique. We assume a small recommendation system – with 3 different users, 3 different information items – articles and 3 tags. The associations between these objects are showed in the Table 2.1.

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 2.1: The associations between the objects

According to the usage data, we are able to construct the basic tensor A which is depicted in the Figure 2.1. The constructed tensor is unfolded into the three new matrices (A_1, A_2, A_3) (each is put together from the basic tensor where the different perspectives are applied).
From the depicted tensor there were created the three new matrices which are denoted as the 1-mode, 2-mode and 3-mode matrices.


Tag's dimensions - slices
User's dimension - rows
Information item"s dimension - columns
1st mode matrix - R_u x R_t R_i
2nd mode matrix R_t x R_u R_i
3rd mode matrix R_i x R_t R_u

$$\begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}$$

$$\begin{matrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}$$

$$\begin{matrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{matrix}$$

Figure 2.1: Tensor construction according to the users preferences table and i-th mode matrices

A higher-order singular value decomposition is an extended version of the SVD applied to the multi-dimensional matrices. The basic SVD for a matrix $F_{D_1 \times D_2}$ is expressed by this formula:

$$F_{D_1 \times D_2} = U_{D_1 \times D_1}.S_{D_1 \times D_2}.V^T_{D_2 \times D_2} \tag{2.10}$$

The unfolded $i-th$ mode matrices from the constructed tensor A are subject of the SVD. It results into creation of U^n, S^n, V^n matrices (see Figures 2.2 and 2.3) 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 orthonormal vectors – singular vectors of 1-mode, 2-mode and 3-mode matrices.

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

Figure 2.2: 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 2.3: Application of the SVD to the 1st mode matrix - S^1 matrix

We are storing the $c - most$ singular values of $i - th$ mode matrices to be able to construct the core tensor S:

$$S = A \times_1 (U_{c1}^{(1)})^T \times_2 (U_{c2}^{(2)})^T \times_3 (U_{c3}^{(3)})^T \tag{2.11}$$

We are able to reconstruct A' which is approximation of the basic tensor A according to the formula:

$$A' = S \times_1 U_{c1}^1 \times_2 U_{c2}^2 \times_3 U_{c3}^3 \tag{2.12}$$

In the following Figure 2.4, there are showed a core tensor S and the reconstructed tensor A'.

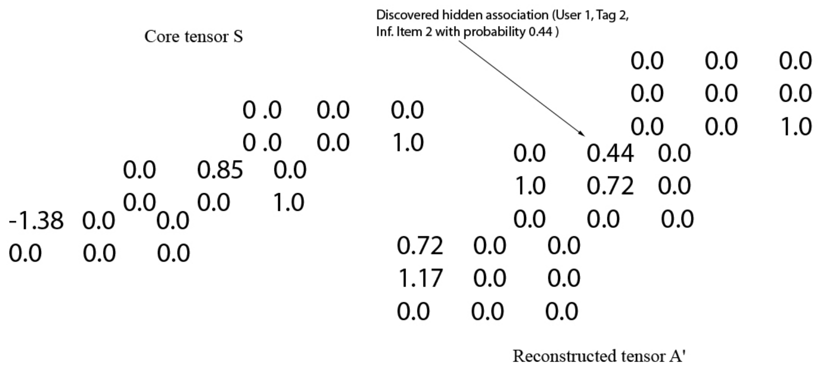


Figure 2.4: The core tensor S and the reconstructed tensor A'

From the new tensor A' we are able to recommend tags or information items with the highest weights to a given user. We can observe that a new association was discovered between User 1 and Tag 2 and Information Item 2.

Evaluations

The proposed method was compared with the two state-of-the-art recommendation algorithms: Folkrank [13] and Collaborative Tag Suggestions [30]. The latter one is denoted also as PR (Penalty Reward algorithm). Authors of the tensor reduction algorithm evaluated the given algorithms on two different data sets – Bibsonomy is a social bookmarking system for scientific publications and Last.fm is a music website with well developed collaborative tagging recommendation system. Each data set was divided into two different sets (a testing set – 25% of tags and a training set – 75% of all tags). According to the training set, algorithms suggested possible tags for a particular user and a given information item. The results showed that the tensor reduction clearly overcomes the other methods. The main advantage of the described algorithm is that it exploits the relationships between all the three different types of the objects (users, tags and information items) of a provided data set and also reveals the latent relationships between these objects.

2.3.2 Multiverse recommendation using N-dimensional tensors factorization

Authors of [14] present Collaborative Filtering (see section 2.1.2) method based on Tensor Factorization (TF). TF is based on a Matrix Factorization (MF) and it enables to model User-Item-Context N-dimensional tensor. This model is called *Multiverse Recommendation* (MR). It is possible to include N various aspects (variables) of a context, like geo-location, time, the gender of an author etc. As a result, N-dimensional tensor can be constructed and factorized. After the factorization of such tensors there can be provided context-aware recommendations.

Model

To be able to provide the better recommendations based on various settings it is crucial to extend "standard" approach of two-dimensional matrix of *user-item* relations. This framework proposes to extend a two dimensional matrix with the third dimension of *context* relation and this results into a tensor (see Figure 2.5). To be able to support N context variables, their own TF method is used. This model is called *Multiverse Recommendation* because of an ability to bind possibly N different contexts.

Matrix Factorization

It is assumed that ratings (likes or dislikes) of a user on items can be represented in a sparse matrix $\mathcal{Y}^{n \times m}$ where n is a number of users and m is a

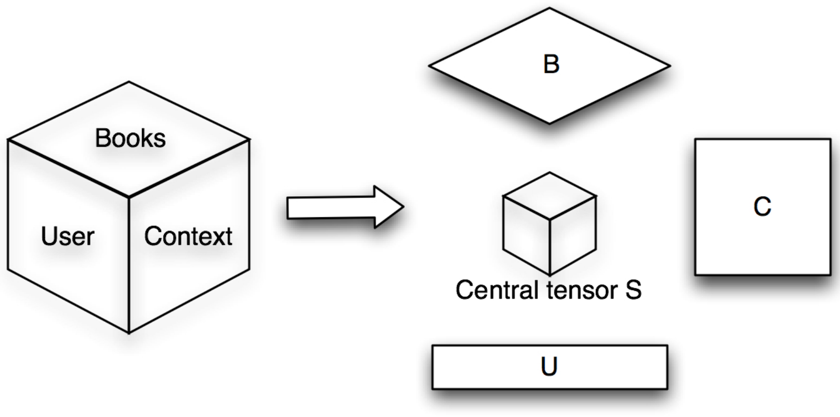


Figure 2.5: HOSVD tensor factorization model

number of items. A concrete value for a particular user on a particular item is denoted by $Y \in \mathcal{Y}^{n \times m}$. The MF process results into two matrices: $U \in \mathbb{R}^{n \times d}$ and $B \in \mathbb{R}^{m \times d}$ such that $F := UB^{\mathsf{T}}$ approximates $\mathcal{Y}$ where U is a concrete value for a user and B is a concrete value for an item (in this case – book).

Tensor Factorization

Tensor Factorization extends the MF in such way that the N-dimensional TF is able to handle N-dimensional data (N variables of a context).
For the simplicity the authors of [14] describe a tensor with a single variable C for the context. Therefore, the tensor $\mathcal{Y}$ will be three-dimensional and it is denoted as follows: $\mathcal{Y}^{n \times m \times c}$ where n is a number of users, m is a number of items and c is a number of contextual variables where $c_i \in \{1, ..., c\}$. The value of a rating $Y \in \mathcal{Y}^{n \times m \times c}$ can vary from 0 to 5. Then the rating $Y \in \{0, ..., 5\}^{n \times m \times c}$ where 0 means a rating for a particular item by a particular user is unknown.
There are used two tensor operations to compute a binary tensor D:

- *tensor-matrix multiplication* denoted by $\times_U$ where subscript indicates the direction of a multiplication.
- *tensor outer product* denoted by $\bigotimes$

So, the binary tensor is $D \in \{0; 1\}^{n \times m \times c}$ whenever entries $\mathcal{Y}_{ijk}$ are known where i is user, j is item and k is context.

HOSVD-decomposition

The illustration of High Order Singular Value decomposition (HOSVD) is shown in the Figure 2.5. Three-dimensional tensor is decomposed (during the process of factorization) into the three matrices: $U \in \mathbb{R}^{n \times d_U}$, $B \in \mathbb{R}^{m \times d_B}$, $C \in \mathbb{R}^{c \times d_C}$ and a central tensor $S \in \mathbb{R}^{d_U \times d_B \times d_C}$. Then a factorization function is:

$$F_{ijk} = S \times_U U_{i*} \times_B B_{j*} \times_C C_{k*} \tag{2.13}$$

This function enables to control dimensionality of the factors by adjusting parameters d_U, d_B, d_C.

Finalization

A number of operations must be executed to accomplish the algorithm of the tensor factorization:

- *Loss function* $L(F, \mathcal{Y})$ that penalizes the distance between the estimated tensor F and observed tensor $\mathcal{Y}$. Loss function takes into account just the observed values – the missing values are not included into the calculations.

- *Regularization* $\Omega[U, B, C]$ is needed to ensure that a loss function is not leading to overfitting and that a complexity of a model (composed of U, B, C and C) does not grow boundless.

- *Optimization* $R[U, B, C, S]$ is minimizing a regularized risk that is a combination of $L(F, Y)$ and $\Omega[U, B, C]$.

Algorithm

Algorithm is "easy to implement" and it can be executed in parallel. The complexity is $\mathcal{O}(K d_U d_B d_C)$ where K is a number of ratings and d_U, d_B, d_C are the dimensions of the factors.

Evaluations

Multiverse Recommendation framework enables to model N-dimensional tensor and therefore, it supports N contextual variables. Because of this feature, the recommendations can be made with considering various settings, like: geo-location, time etc. Besides of it the algorithm can be implemented in parallel that would improve time performance. However, as more contextual variables will be used, a performance will decrease. Furthermore, unlike most of the recommender systems, this framework can handle all the data set with all the involved contextual information: there is no need to make pre-filtering nor post-filtering for the data set. Because of this no data is lost and the better recommendations can be made. Also, it should be possible to tune up the algorithm to perform better and/or provide better results when applying it to the concrete data set and a type of recommendations. The experiments show that this framework is able to provide the more accurate recommendations because of the usage of the contextual variables. It also outperforms the non-contextual recommenders and even the "current state-of-the-art context-aware recommendation approaches".

2.3.3 Extensions for HOSVD based recommender systems

Authors of an article [25] further investigate tensor based recommendation systems and provide "a unified framework for providing recommendations in social tagging systems based on ternary semantic analysis". The most notable features that are introduced for the tensor based recommendation systems (as analysed in a section 2.3.1) are as follows:

- the usage of kernel-SVD smoothing technique to reduce a data sparsity;
- the reduction of dimensions for matrices;
- a handling of arrival of new users, tags and items.

Kernel-SVD smoothing technique

Sparsity for three dimensional data is a problem that can affect the results of an algorithm. To overcome this problem kernel-SVD can be applied instead of basic SVD (as analysed in a section 2.10) in three unfolded matrices (A_1, A_2, A_3). Kernel-SVD is an operation of SVD "in the Kernel-defined feature space".
The entries of each unfolding A_i ($1 \leq i \leq 3$) are mapped to a higher dimensional space using a function ϕ. Therefore, each matrix A_i can be encoded in a matrix F_i where each element a_{xy} of A_i can be mapped to the corresponding element f_{xy} of F_i:

$$f_{xy} = \phi(a_{xy}) \tag{2.14}$$

Now, SVD can be applied using F_i:

$$F_i = U^{(i)} S^{(i)} (V^i)^T \tag{2.15}$$

The resulting $U^{(i)}$ matrices can be used to construct a core tensor S (as analysed in a section 2.11).
The authors of [25] introduce one more optimization: to avoid expensive computation of F_i, "all computations must be done in the form of inner products". Because we are interested to compute only the matrices with the left singular vectors, we can define a matrix B_i:

$$B_i = F_i F_i^T \tag{2.16}$$

Because B_i is computed using the inner products of F_i, we can express B_i using the results of a kernel function. This technique is called "kernel trick" and enables to avoid the computations of an expensive F_i. We know each $U^{(i)}$, $V^{(i)}$ are orthogonal and $S^{(i)}$ is diagonal, it follows (from the Formulas 2.15 and 2.16):

$$B_i = (U^{(i)}S^{(i)}(V^i)^T)(U^{(i)}S^{(i)}(V^i)^T)^T = U^{(i)}(S^{(i)})^2(U^i)^T \tag{2.17}$$

Therefore, each $U^{(i)}$ can be computed diagonalising each matrix B_i "and taking its eigen-vectors".
Authors explicitly mark that for the experiments they used Gaussian kernel:

$$K(x,y) = e^{\frac{\|x-y\|^2}{c}} \tag{2.18}$$

that is a common function to compute kernel SVD for many applications. Parameter c is computed as a "standard deviation in each matrix unfolding".

Reduction of dimensions

To filter out noisy approximations, the dimensions of a matrix can be reduced. It will contribute for the performance and the quality of the recommendations. This can be done eliminating "the small singular values that introduce noise". So, resulted matrices from the SVD are reduced to the c higher singular values and the corresponding singular vectors. This reduction operation is called *thin*-SVD which is optimal in that way that it computes rank-c approximation with the minimum Frobenius norm. The reduced matrix is denoted as rank-c approximation.
The c_i-top singular values and the corresponding left singular vectors from $U^{(i)}$ are taken from computed SVD "on the unfolded matrix A_i of a i-mode". The selected sizes of c_1, c_2 and c_3 determine the size of a core tensor S.
However, to select sizes for c_1, c_2 and c_3 is a difficult task. A practical approach would be to choose sizes for c_1, c_2 and c_3 by preserving some amount (expressed by percentages) of information of original $S^{(1)}$, $S^{(2)}$ and $S^{(3)}$. Authors give a hint that at least 70% of original data should be retained in $S^{(1)}$, $S^{(2)}$ and $S^{(3)}$.

Arrival of new Users, Tags and Items

Normally, as the new users, tags or items are inserted into the system, the expensive computations (as shown in the Formulas 2.11 – 2.12) should be re-executed and a tensor A' (that gives the recommendations) should be re-created. There is a possibility to avoid expensive re-computations by the following solutions which depends on the size of an update (how many new users, tags and items are inserted):

- for the small amount of new data, $folding - in$ technique is recommended;
- for the larger updates, Incremental SVD technique is recommended.

Folding-in technique When adding a new user, the 1-mode matrix unfolding A_1 must be re-computed. As a result, a new row (u) will be appended to the matrix A_1. Because of the changes to A_1, its SVD has to be computed. To avoid the expensive computations, we can re-use the "existing basis $U_{c1}^{(1)}$ of left singular vectors, to project the u row onto the reduced $c1$-dimensional space of users in the A_1 matrix". This operation is called $folding - in$:

$$u_{new} = u \cdot V_{c1}^{(1)} \cdot (S_{c1}^{(1)})^{-1} \tag{2.19}$$

where u_{new} is a new mapped row that will be appended to the end of $U_{c1}^{(1)}$. $V_{c1}^{(1)}$ and $(S_{c1}^{(1)})^{-1}$ are the dimensionally reduced matrices computed when SVD was applied on a original A_1 (before inserting a new user).
To update tensor A' we have to execute the operations identified by the Formula 2.16. We must note that only $U^{(1)_{c1}}$ has been changed. Because of it, A' update can be done:

$$[S \times_2 U_{c2}^{(2)} \times_3 U_{c3}^{(3)}] \times_1 U_{c1}^{(1)} \tag{2.20}$$

where the left factor (in the brackets) is unchanged and could be preserved in memory.
For inserting new items and tags the folding-in (Formula 2.19) must be applied for 2-mode unfolding (A_2) and 3-mode unfolding A_3 of a tensor A, respectively.

Incremental SVD technique When applying folding-in technique to update SVD, the space becomes not orthogonal and it can cause incorrect recommendations. When the size of update is not big, this problem may not occur.
The incremental SVD technique is proposed to handle large updates, but due to the complexity of this method, the details of it can be found in the article [25] and section 4.4.2.

2.4 Recommender systems based on other factorization techniques

In this section we will analyse the recommendation systems based on tensor factorization using Ranking with Tensor Factorization (similar approach to the HOSVD), Canonical Decomposition and Non-Negative Tensor Factorization techniques.

2.4.1 Tensor factorization with post-based ranking interpretation scheme

Rendel et al. [17] proposed a different approach for creating the initial tensor which expresses user - item - tag associations. Instead of using the 0/1 interpretation scheme (as introduced in the formula 2.9), they use so called post based ranking interpretation. They observed and noticed the following drawbacks of 0/1 scheme:

- The semantics are incorrect because the case when a user has observed a given item but not tagged with a particular tag t and the situation when the user has never explored a particular item are the same. In both situations the initial tensor is filled with 0.
- Approach of fitting values 1 and 0 into the tensor is an unnecessary constraint. The important is only to distinguish between positive and negative cases (the positive relation should be represented with a larger value than the negative one).
- In the most of data sets the amount of 0 reaches about 99% of all entries in the tensor and it is causing a sparsity problem.

Post-based Ranking Interpretation (PBRI) scheme distinguishes three different situations to achieve more accurate semantics. It reflects the following associations:

- The positive case – a user marked a given item with a particular tag. Formally defined as:

$$T^+_{u,i} := \{t|(u,i) \in P_s \wedge (u,i,t) \in S\}$$

- The negative case – a user has observed a given item but has not tagged it.

$$T^-_{u,i} := \{t|(u,i) \in P_s \wedge (u,i,t) \notin S\}$$

- The not observed case – a user has not visited a given item.

where the tagging information from the past is represented by $S \subseteq U \times I \times T$. The observed items by users are denoted as: $P_s := \{(u,i)|\exists t \in T : (u,i,t) \in S\}$ The ranking constraint is defined in this way:

$$y^p_{u,i,t_1} > y^p_{u,i,t_2} \leftrightarrow (u,i,t_1) \in T^+_{u,i} \wedge (u,i,t_2) \in T^-_{u,i}$$

The introduced interpretation solves the mentioned drawbacks of the 0/1 scheme. From the semantic point of view, it distinguishes three different cases and makes difference between not observed relationships and negative associations. The constraint requires only the smaller values for the negative

cases and it does not have to be set to 0 weight. The not observed values are not considered during the training. However, this approach has to be optimized to satisfy as many rankings as possible. Authors proposed to use AUC – area under ROC-curve (receiver operating characteristic) to find optimal rankings constraints [26].
A tensor factorization model called Ranking with Tensor Factorization (RTF) is similar to the HOSVD. An estimated tensor $\hat{Y}$ is constructed by multiplying three different features matrices $\hat{U}, \hat{I}, \hat{T}$ with a core tensor $\hat{C}$.

$$\hat{Y} = \hat{C} \times_u \hat{U} \times_i \hat{I} \times_t \hat{T}$$

These matrices and the core tensor are model parameters with the following sizes:

$$\begin{array}{ll} \hat{C} \in \mathcal{R}^{k_U \times k_I \times k_T}, & \hat{U} \in \mathcal{R}^{|U| \times k_U} \\ \hat{I} \in \mathcal{R}^{|I| \times k_I} & \hat{T} \in \mathcal{R}^{|T| \times k_T} \end{array}$$

The low rank matrices $\hat{U}, \hat{I}, \hat{T}$ are conditioned by these k_U, k_I, k_T dimensions respectively. The model parameters are denoted as 4-tuple: $\hat{\theta} := (\hat{C}, \hat{U}, \hat{T}, \hat{I})$. They have to be learned according to the optimization criterion that uses the PBRI and maximizes the ranking by AUC for a given tagging of an user u for an item i as:

$$AUC(\hat{\theta}, u, i) = \frac{1}{|T^+_{u,i}||T^-_{u,i}|} \sum_{t^+ \in T^+_{u,i}} \sum_{t^- \in T^-_{u,i}} s(\hat{y}_{u,i,t_+} - \hat{y}_{u,i,t_-})$$

Where the s is s-shaped logistic function:

$$s(x) = \frac{1}{1 - e^{-x}}$$

Then, the optimization task is defined as:

$$argmax_{\hat{\theta}} \sum_{(u,i) \in P_s} AUC(\hat{\theta}, u, i)$$

The optimization is done only with observed data (P_s). The intention is to minimize the training error for models which can lead to overfitting. Therefore, authors proposed to use regularization where an objective function is minimized by gradient descent. The details of the regularization can be found in the paper [17].

Comparison with HOSVD

This factorization technique – solves the following important issues of the HOSVD:

- HOSVD cannot deal with missing values, they are filled with 0.
- To prevent over-fitting, HOSVD should use regularization.

Both models compute tensor factorization in off-line mode, and then recommending is based on the estimated tensor in on-line mode. Therefore, it is able to produce relatively good results in short time.

Evaluation

Authors conducted experiments on the two different data sets – BibSonomy and Last.fm. The algorithm was compared with HOSVD, FolkRank and PageRank. Almost in all the experiments RFT overcomes other competitive techniques. Surprisingly, even when the dimensions of low rank matrices in RTF are smaller than dimensions of model matrices in HOSVD – RTF outperforms HOSVD.

2.4.2 Tensor factorization based on canonical decomposition

Recommendation systems that utilize tensor factorization techniques have been shown to provide high quality recommendations and quite often overcome the other state-of-the-art approaches like FolkRank, PageRank and Collaborative Filtering. The most common are the factorization models based on a Tucker Decomposition [27]. The factorization algorithms described in this analysis part: HOSVD 2.3.1, Ranking Tensor Factorization 2.4.1 are special extensions of the Tucker Decomposition (TD). The main disadvantage of TD is cubic computational time which is infeasible for large data sets. Therefore, we will try to inspect and describe another tensor factorization approach – Canonical Decomposition (CD) also known as Parallel Factor Analysis (PARAFAC). This decomposition technique is not so popular as HOSVD and to the best of our knowledge we know only one tag-based recommendation system (the main principles will be described in the end of this section) which utilizes an extension of CD [18].
CD is widely used in the different research areas – chemometrics, psychometrics and signal processing [6]. The main advantage in comparison to TD is linear computation time which depends on the dimension parameter (the differences between PARAFAC/CD and HOSVD techniques are more precisely described in the section 5.2.6).
The idea is for a given tensor $Y \in \mathcal{R}^{T \times U \times R}$ to find three component matrices (also called as the loading factors), such that:

- $T = [t_1, t_2, \ldots, t_{k_t}] \in \mathcal{R}^{T \times k_t}$
- $U = [u_1, u_2, \ldots, u_{k_u}] \in \mathcal{R}^{U \times k_u}$
- $R = [r_1, r_2, \ldots, r_{k_r}] \in \mathcal{R}^{R \times k_r}$

which participate in the following factorization:

$$\hat{Y} = \sum_{f=1}^{k} t_f \circ u_f \circ r_f + E \tag{2.21}$$

where $k = min(k_t, k_u, k_r)$ is the dimension parameter and $E \in \mathcal{R}^{T \times U \times R}$ is error tensor. Graphical representation is depicted in the Figure 2.6. The most popular method to find solution for the given model is to use alternating least squares (ALS) algorithm. It estimates loading factors T, U, R during more iterations until there is only little change in factor matrices. Each component is estimated by least squares regression according to other two components and approximated tensor (i.e. component matrix T is estimated by U, R and $\hat{Y}$). There exist different versions of ALS algorithms which are suitable for various situations. To find appropriate dimension parameter k, there must be considered a trade-off between computational accuracy and performance. The computational complexity is linear $\mathcal{O}(k)$ as the basic model contains only one sum over the $|k|$ entries. Obviously, PARAFAC/CD decomposition is faster than Tucker Decompositions (where three nested sums must be executed - $\mathcal{O}(k^3)$).

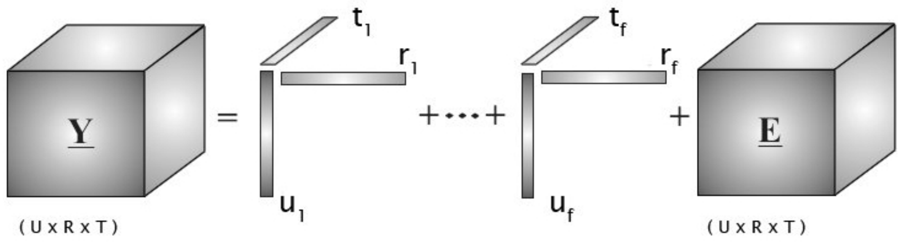


Figure 2.6: Graphical representation of the canonical decomposition.

Rendle et al. [18] proposed tag based recommendation system which exploits an extension of the Canonical Decomposition. The tensor is approximated by the following mathematical model called PairWise Interaction Tensor Factorization (PITF):

$$\hat{Y} = \sum_{f=1}^{k} \hat{u}_{u,f}.\hat{t}^{U}_{t,f} + \sum_{f=1}^{k} \hat{i}_{i,f}.\hat{t}^{I}_{t,f}$$

PITF is the special case of CD, where the dimension parameter is $2.k$ and the following holds:

$$\hat{u}^{CD}_{u,f} = \begin{cases} \hat{u}_{u,f}, & \text{if} f \leq k \\ 1, & \text{else} \end{cases}$$

$$\hat{i}^{CD}_{i,f} = \begin{cases} 1, & \text{if} f \leq k \\ \hat{i}_{i,f-k}, & \text{else} \end{cases}$$

$$\hat{t}^{CD}_{t,f} = \begin{cases} \hat{t}^{U}_{t,f}, & \text{if} f \leq k \\ \hat{t}^{I}_{t,f-k}, & \text{else} \end{cases}$$

The matrices $\hat{u}^{CD}, \hat{i}^{CD}$ and $\hat{t}^{CD}$ could be used in the basic factorization formula 2.21. We described the main principles of their factorization method, a reader that is interested in the learning algorithm for the model parameters $\hat{U}, \hat{I}, \hat{T}^{U}, \hat{T}^{I}$ and other details can find out more in a given paper [18].

Evaluations

The main contribution is significantly improved time performance compared to the other systems based on tensor factorization. As time complexity was improved some could expect decreased quality of recommendations, however in almost all cases this method outperformed the other state-of-the-art systems like HOSVD, RTF-TD, PageRank and FolkRank. The Canonical Decomposition/PARAFAC methods represent faster solution for the tensor factorization in comparison to the more popular HOSVD technique. On the other hand training a mathematical model for CD can be computationally demanding and it requires to find appropriate learning function which will be computationally not expensive and precise enough.

2.4.3 Recommendations based on non-negative tensor factorization

Authors of an article [4] introduce a framework for generating recommendations incorporating prior knowledge into the non-negative tensor factorization (NTF). The NTF is another decomposition technique for extracting relations from the multidimensional data. It means that the latent semantics can be revealed from the given data set knowing some facts from the past. Authors provide an example of 3-dimensional data relations: blog entries where *author*, *blog title* and a *timestamp* are the dimensions. The NTF is universal for extracting the relations from the multidimensional data and can be easily used for our domain where we analyse the relations of *author*, *resource* and *tag*.
NTF is a promising technique because it "extracts characteristics jointly from the different data dimensions". It is an advantage because each of the data dimension affects each other in a "joint way". The main advantage of the NTF is that approximated tensor will contain only positive values therefore, there will not be any problems with the interpretation of the results. The basic NTF does not use prior knowledge for extracting the relations. The authors propose an extension to NTF where the prior knowledge is

incorporated in such way that the extracted relations reflect (are compatible with) the prior knowledge. The extension is called *FacetCube*. For our model, the prior knowledge can be the known facts of a previous activity of a user, e.g. which resources were tagged and which tags were used for that. Knowing this, FacetCube framework should be able to reveal new relations that can be presented as the recommendations for the user, e.g. which resources should be interested for a user.

Standard NTF

To apply tensor factorization we need to construct a data tensor. We will use 3-dimensional data of users, resources and tags. The third-order tensor $\mathcal{A} \in \mathcal{R}_+^{I \times J \times K}$ will represent this data where I, J and K are the dimensions of the data of users, resources and tags, respectively. A value $(A)_{ijk} = a_{ijk}$ can represent, for example, how many times user i tagged a resource k with a tag j.
When a standard NTF [6] is applied and the outcomes are as follows:

- the first outcome is *facet matrices* that represents the most significant data characteristics for a single dimension. The number of the facet matrices is the same as the order of a tensor. Each column of each facet matrix represents exactly one facet of a corresponding dimension of data. For example, the facet matrix for tagged resources 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 column of X, Y and Z represents most important (active) users, mostly tagged resources by the most popular tags.

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

The goal of the NTF is to compute the core tensor $\mathcal{C}$, the facet matrices X, Y and Z in a such way that they all together $[\mathcal{C}, X, Y, Z]$ "approximate $\mathcal{A}$ in an optimal way". The quality of $\mathcal{A}$ can be measured by the error rate. The most common error measure is KL-divergence. So, the goal of NTF is to find $\mathcal{C}$, X, Y and Z with a minimal error rate:

$$error_{KL} = KL(\mathcal{A} \| [\mathcal{C}, X, Y, Z]). \tag{2.22}$$

FacetCube framework

FacetCube framework extends standard NTF by incorporating the prior knowledge to the process of a decomposition. There are presented two variants of a framework:

- Facets are restricted to be in a sub-space defined by the user.
- Facets are fixed by the user.

Also, FacetCube can be unrestricted (no prior knowledge is used) and in that case it would use standard NTF.

Restricted facets The assumption is that there is provided a sub-space from which the facets are derived. We can re-write the 2.22 equation:

$$error_{KL} = KL(\mathcal{A} \| [\mathcal{C}, I_I X, I_J Y, I_K Z]), \tag{2.23}$$

where I_I, I_J and I_K are the identity matrices of dimensions I, J and K, respectively. If we would consider every facet in Y as a point in a structure formed by $\{\vec{e_1}, ..., \vec{e_J}\}$ where $\vec{e_j}$ is a vector with dimension J (for example the j-th element being 1 and the others – zeros), the number of resources can be huge (space J) that makes difficult to find reliable facets. In this case, users prior knowledge can play significant role by reducing J space to a concrete sub-space. If we assume a given prior knowledge for resources as a set of basis $Y_B = \{\vec{b_1}, ..., \vec{b}_{M'}\}$ then we can assume that the search space is formed by Y_B. According to our example, if the prior knowledge of favourite (imported) resources is incorporated, the search space for the resources to recommend would be smaller. The Figure 2.7(b) illustrates it. We can extend the 2.23 equation:

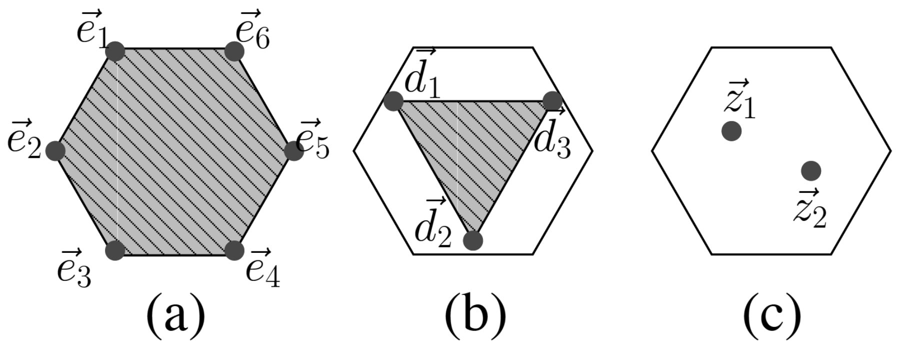


Figure 2.7: Variants of a FacetCube: (a) unrestricted search space, restricted search space (b) and fixed (c) data dimensions. Figure is taken from [4]

$$error_{KL} = KL(\mathcal{A} \| [\mathcal{C}, X, Y_B Y, Z]), \tag{2.24}$$

where Y is basis-restricted dimension with Y_B and all columns of Y_B sums to 1.

Fixed facets These may be fixed for the exact data dimensions. The example of a such fixation may be the requirement to generate recommendations from top-level resources only. The other example could be to generate recommendations only using a concrete set of tags. Therefore, we can extend standard NTF with the following equation:

$$error_{KL} = KL(\mathcal{A} \| [\mathcal{C}, X, Y, Z_B]), \tag{2.25}$$

where Z_B represents the fixed dimensions and each column of Z_B sums to 1.

Overall picture To summarize all three variants as mentioned above and shown in Figure 2.7, we can re-write standard NTF function:

$$error_{KL} = KL(\mathcal{A} \| [\mathcal{C}, X_B X, Y_B Y, Z_B Z]), \tag{2.26}$$

where X_B, Y_B and Z_B are the given prior knowledge; X, Y, Z and $\mathcal{C}$ need to be computed. We can observe that:

- when X_B, Y_B and Z_B are provided as the identity matrices we have standard NTF;
- when any of the X_B, Y_B or Z_B are provided and X, Y and Z need to be computed, we have basis-restricted search space;
- when any of the X_B, Y_B or Z_B are provided and any of X, Y or Z is fixed to be identity matrix, we have fixed search space.

An iterative algorithm Authors of a [4] propose an algorithm (Section 3.4) for "computing optimal solutions" using FacetCube framework. They present the algorithm in a general way – basis-constrained form. They also show it can be re-used in a case of the other two models (unconstrained and fixed).

Evaluations

The proposed "novel" extension of NTF, called FacetCube, "naturally" embraces 3-dimensional data relations. One of the major benefit of this framework is that it allows to incorporate a prior knowledge of the users. Because of it, the outcomes of the recommendations can be controlled (can be personal) and the search space are decreased and it also affects performance positively. Furthermore, authors claim that an optimal solution can be found using the algorithm they proposed. Even more, authors presented a pseudo code for the algorithm. They also conducted experiments on a large, real world data set (using proposed algorithm) and the results are promising. Finally, the approximated tensor contains only the positive values so there are no problems with the interpretation of the results.

2.5 Project domain

Recently tensor based recommendation systems have emerged [24], [14]. These systems are able to reveal the semantic relations between users, items and tags because they use three-dimensional relations of $\{user, tag, resource\}$. Existing algorithms analyse two-dimensional relations $\{user, tag\}$, $\{user, resource\}$, $\{resource, tag\}$ and are not able to provide the same quality of recommendations as tensor based ones [25]. Such systems can be used to generate various combinations of the recommendations [15]. There can be recommended:

- N tags with the highest score for a given user u and resource r;
- N resources with the highest score for a particular user u and a particular tag t;
- other users (that used a particular tag t) for a particular user u, that used the same tag t;

using tensor based recommender.
We choose to investigate tensor based recommendation systems, because:

- promising results are shown when comparing tensor based recommender with the state-of-the-art recommendation algorithms;
- most of the best performing models in the Neflix Challenge[2] competition were based on the matrix and tensor factorization [18].
- various combinations of the recommendations can be provided;
- there may be performance issues (as analysed in 5.1) when applying such systems for a real world applications. Our main goal is to investigate if and how the performance of an algorithm can be improved without loosing a quality of the recommendations or having a minimal impact on a quality.

2.6 Conclusion

2.6.1 Types of the recommendation systems

During the analysis we presented different types of the recommendation systems. These are as follows:

- Content based recommendation systems analyse the titles or descriptions of the resources and the profiles of the users to make the recommendations. Users profiles should be informative and reflect the

[2]http://www.netflixprize.com

preferences of the users. Based on this, the recommendations can be made. Such systems are quite popular due to the simplicity.

- Collaborative filtering systems capture the power of a collaborative actions. Such systems can recommend the items to the other users when knowing the preferences for the items of a current user. The main idea is that "those who agreed in the past tend to agree again in the future".
- Hybrid based systems are the combinations of two or more other recommendation systems. The combination is made using the benefits of each system and trying to solve the weak parts of them.
- Tagging is an action when user u tags an item i with a tag t. Tag is an arbitrary word describing the tagged item.
  Tag based recommenders are very flexible – they allow to make semantic recommendations. Also, the social factor is revealed – it is possible to find users with the common interests. Groups of users generate the classification systems for the various items and is called folksonomy.
  We described a hybrid tag-based recommendation system, which combines syntactical and semantic similarities among the tags to provide more accurate results. The semantic factor reveals relations among synonyms and also among the tags from the same domain (i.e. Java, programming).
- Other systems like (demographic, utility-based, knowledge-based) are not so popular these days.

2.6.2 Tensor based recommendation systems

We can divide tensor based recommendations into the systems that employ HOSVD technique for tensor factorization and the systems that use the other techniques.

HOSVD based recommender systems

Symenonidis et al. (2008) [24] introduced recommendation algorithm where the relationships between users, resources and tags are represented in three dimensional matrix called tensor. Each tagging activity for a given resource from a particular user is represented by value 1 in the initial tensor, all other cases are represented with 0. The core principle is with the usage of the decomposition technique HOSVD obtain an approximated tensor which should reveal the latent relationships and patterns of the users. The tensor is split into three so called mode matrices by applying different perspectives to the initial tensor. The Singular Value Decomposition 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 belonged to a given user and resource or tag.
Another novel system ([14]) was analysed – it supports relations between users, item and context. Context can be expressed as a number of variables (like time, location, tag). Therefore, it is called N-dimensional tensor factorization. For the simplicity authors presented a model just with a single contextual variable and therefore the factorization technique is quite similar to 3-dimensional tensor factorization. However, there are proposed to use various optimization functions like loss function, regularization and optimization to achieve the better performance and quality.

Extensions for HOSVD based recommender systems are as follows:

- **Handling sparsity**. Sparsity is a severe problem affecting performance and quality negatively. One of the possible solution is to use kernel-SVD smoothing – the contents of the matrices are mapped (encoded) to the new matrices. The new matrices are smaller and less sparse. SVD is applied on the new matrices.

- **Reducing the dimensions of matrices**. The low and similar values in a core tensor may introduce noise and affect performance negatively. To deal with it the analysed idea proposes to reduce the original size of data keeping 70% of the original data.

- **Handling the changes of data**. Due to the complexity of such systems it is required a lot of computational power to build the approximation tensor that is used to provide the recommendations. Furthermore, as the data changes (new users, resources and/or tags are introduced), the naive solution would be to re-compute the approximation tensor. However, we analysed the proposed extensions that can help to handle the changes of data – folding-in (for small changes) and incremental SVD (for big changes) techniques.

Recommendation systems based on the other factorization techniques

Introduced recommendation system by Rendel et al. [17] with post-based ranking interpretation scheme is using a similar factorization technique as HOSVD. The main differences 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 resource); negative case (a user has explored a particular resource but has not tagged it with a given tag); not observed case (a user has not explored a given resource). 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.

The recommendations are obtained in the same way as it is described in the section 2.3.1.

Recommendation systems based on a canonical decomposition The tensor is approximated according to the three component matrices also called as loading factors. These factors have to be trained, the most common algorithm for the learning is Alternating Least Squares. The components are estimated iteratively until there is only small change. The main advantage is linear computation time in comparison to the HOSVD, however training can be computationally demanding. This factorization technique is not so popular in the area of the recommendation systems. We analysed system with PairWise Interaction Tensor Factorization, that is an extension of the CD. The factorization model is composed from the two sums of the four different loading factors. This model clearly outperformed the other factorization techniques like HOSVD or RTF in time performance and almost in all cases provided the best accuracy of recommendations.

Recommendation systems based on a non-negative tensor factorization A novel framework, called FacetCube is based on non-negative tensor factorization proposed in the article ([4]). This framework is general enough to extract data characteristics for various purposes, including recommendation. NTF naturally fits for the multidimensional data because it is able to produce a good approximation model. FacetCube extends NTF incorporating prior knowledge of the users. It enables to reduce the search space of the data relations and benefits for the performance. There can be three modes of the recommendation generation: with unrestricted search space (no prior knowledge is used); from a sub-space which defined by the prior knowledge of the users and a fixed space that is provided by the users.

2.6.3 Comparison of tensor based techniques

We will compare the common features of all the analysed techniques based on tensor factorization. The shared aspects of the given models are presented[3] in the Table 2.2.

[3]The worst evaluation is one star ($\star$) and the best is five stars.

Technique	Time performance	Quality of the recommendations	Advantages	Disadvantages
HOSVD (2.3.1)	⋆	⋆⋆⋆	Reveals latent semantics	Runs slowly for real world data set. Big sparsity
Multiverse (2.3.2)	⋆⋆	⋆⋆⋆⋆	N different contextual variables are supported	Performance is affected negatively as the size of variables grow
RTF (2.4.1)	⋆⋆⋆	⋆⋆⋆⋆	More accurate interpretation scheme for data relations. Model parameters are trained	Unacceptable performance
PITF (2.4.2)	⋆⋆⋆⋆⋆	⋆⋆⋆⋆	Running time is linear	–
FacetCube (2.4.3)	⋆⋆	⋆⋆⋆⋆	Running time is linear. Incorporates prior knowledge	Unacceptable performance (if not based on CD)

Table 2.2: Comparison of tensor based techniques

The considered systems were precisely described in the previous sections, now we will elaborate on the features as time performance and quality of the recommendations:

- The **HOSVD** was the first technique utilizing the tensor factorization for the tag-based recommendations. Time performance of the proposed model is not optimized, because authors were assuming that factorized tensor will be computed in the offline mode. The algorithm outperforms the state-of-the-art recommenders. However, there were introduced other techniques that tend to perform better concerning time and quality.

- **Multiverse** performs better than HOSVD, but in case when more than 3 dimensions are considered – it can be practically impossible to apply this method to the real world application. The accuracy of the recommendations should be improved by the incorporating more contextual variables into the model.

- **RTF** performs significantly faster than HOSVD and also provide more precise recommendations. This is affected by the novel interpretation scheme. The performance is still not good enough for tensor factorization in online mode, also the interpretation scheme must be trained.

- **PITF** takes an advantage of the other type of tensor factorization technique (CD) which runs in a linear time. The quality of the recommendations is not significantly improved and is comparable to the RTF.

- **FacetCube** is able to incorporate the prior knowledge which improves the quality of the recommendations. It is important that approximated tensor contains only the positive values and the interpretation of the results is not complicated. The performance is not acceptable if the NTF is not based on the CD technique.

CHAPTER
3

Design

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

3.1 Architecture

We have chosen to implement a recommendation system based on the tensor factorization technique as analysed in the section 2.3.1.
The architecture of our system is composed of several layers. The general structure is shown in the Figure 3.1.

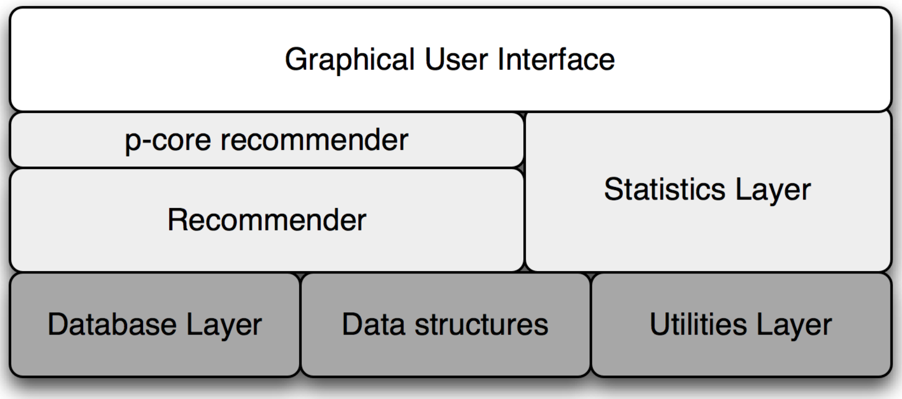


Figure 3.1: General overview of the system

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 extension for the basic HOSVD based recommender.

3.2 Detailed overview

3.2.1 Database layer

We use the samples of the real world data sets (as described in the section 4.1) to generate recommendations. The data is stored in the databases. We choose to use Hibernate framework[1] 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 are stored in the different databases. Moreover, the databases can be different – e.g. MySQL, PostgreSQL, Oracle. The structure of a database layer is presented in the Figure 3.2.
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 will be invoked to import data when a data set will be 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 different persistence manager (i.e. for each different database) in a memory to boost 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.

3.2.2 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 trinary relations of *user-tag-resource* (Figure 3.3). 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 resource respectively. There are the 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).

[1]http://www.hibernate.org

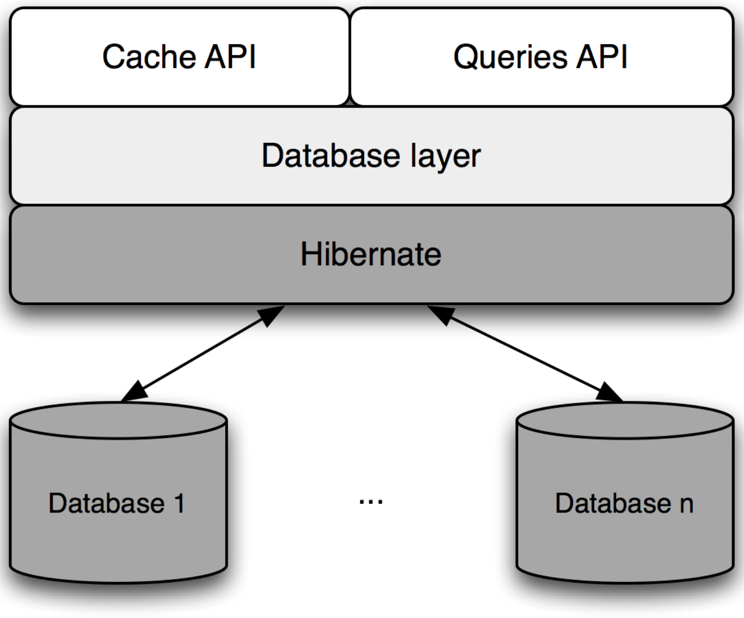


Figure 3.2: Database layer with caching service

3.2.3 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 the section 4.3.2).
- `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`.

3.2.4 Recommender

The recommender layer can be split into the several parts:

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

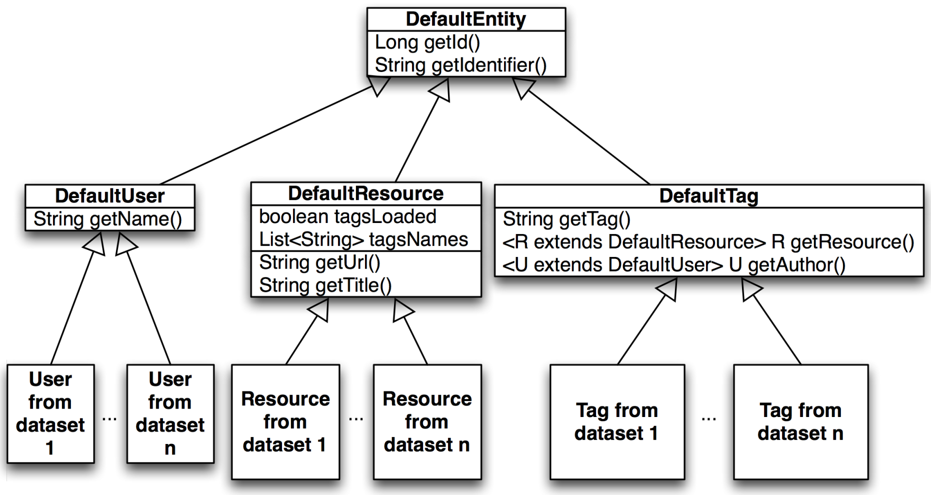


Figure 3.3: Data structures of the database layer

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 1) of the initial tensor creation.
We describe the main ideas of the algorithm:

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

Computing tensor factorization

To compute a tensor factorization, the initial tensor has to be defined. We provide the algorithm of the tensor factorization (see Algorithm 2).
We describe the key points of the algorithm:

```
Data: User user, number of users size
Result: Initial tensor initialTensor
1  Generate a set of users;
2  for i ← 0 to size do
3      get user u, add it to the users set users
4  end
5  Get a set of tags used by the users;
6  Get a set of resources tagged by the tags and by the users;
7  initialTensor = new
   double[sizeOfTags][size][sizeOfResources];
8  int userIndex = 0;
9  foreach user u ∈ users and userIndex < size do
10     for tagIndex ← 0 to sizeOfTags do
11         for resourceIndex ← 0 to sizeOfResources do
12             if a current user tagged a current resource with a current
               tag then
13                 initialTensor[tagIndex][userIndex][resourceIndex]++;
14             end
15         end
16     end
17     if current user == u then
18         StoreEmptyRelations (initialTensor);
19     end
20     userIndex++;
21 end
```

Algorithm 1: The algorithm for creating the initial tensor

- 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, multiplications are applied. The factorized tensor is computed.

Generating a list of recommended resources

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

```
Data: Initial tensor
Result: Factorized tensor
Splits initial tensor into the first, second and third mode matrices and
computes SVD for each mode matrix;
splitIntoModeMatrices (tensor);
Computes a core tensor: applies dimensional reduction, executes
matrices multiplication;
computeCoreTensor (){
 matrixDimensionalReduction ();
 multiplyDifferentMatrices ();
};
Computes factorized tensor: transforms the three reduced matrices,
applies matrices multiplication;
computeFinalTensor (){
 transformMatrix (reducedMatrix1,firstMatrixDimensions);
 transformMatrix (reducedMatrix2,secondMatrixDimensions);
 transformMatrix (reducedMatrix3,thirdMatrixDimensions);
 multiplyDifferentMatrices ();
};
```

Algorithm 2: The algorithm for tensor factorization

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 resource 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.

p-core recommender

This recommender uses the same algorithms as a basic recommender. The only difference is how the sets of *users*, *tags* and *resources* are generated – it is done as described in a section 4.3.3. As the sets are resolved, the algorithms of a basic recommender are used to provide the list of the recommendations.

3.2.5 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 4.2.1). 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 resource. Also, the training, evaluation sets and a top-N list (as described in 4.3.6) can be resolved.

3.2.6 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. 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 3.4, 3.5 and 3.6.

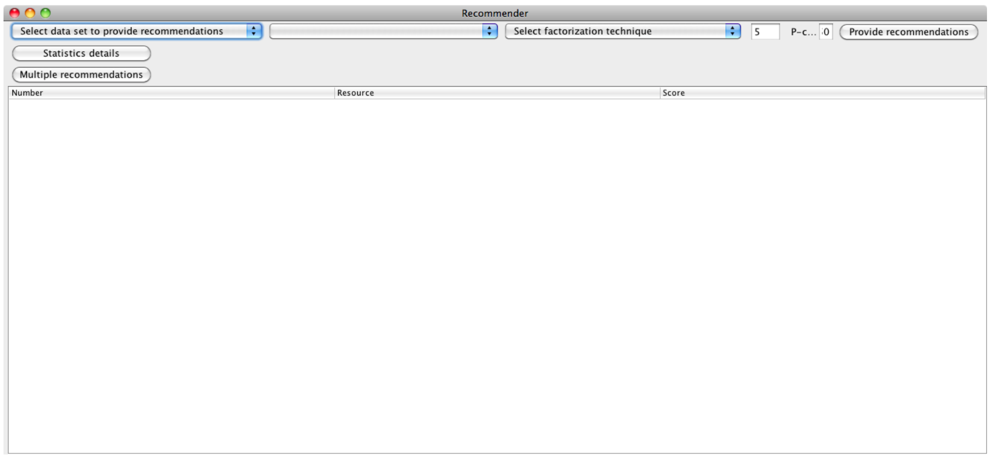
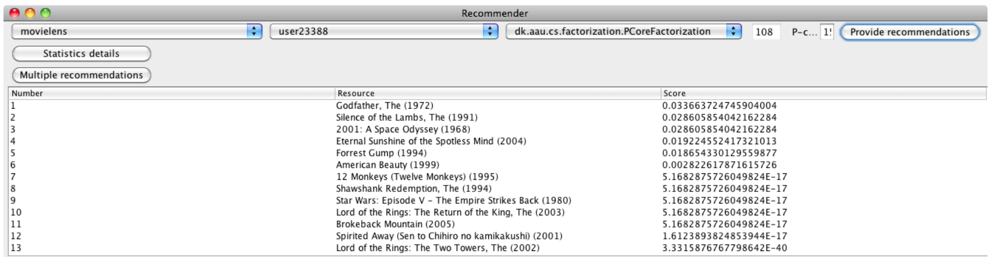


Figure 3.4: The main window of a recommender system



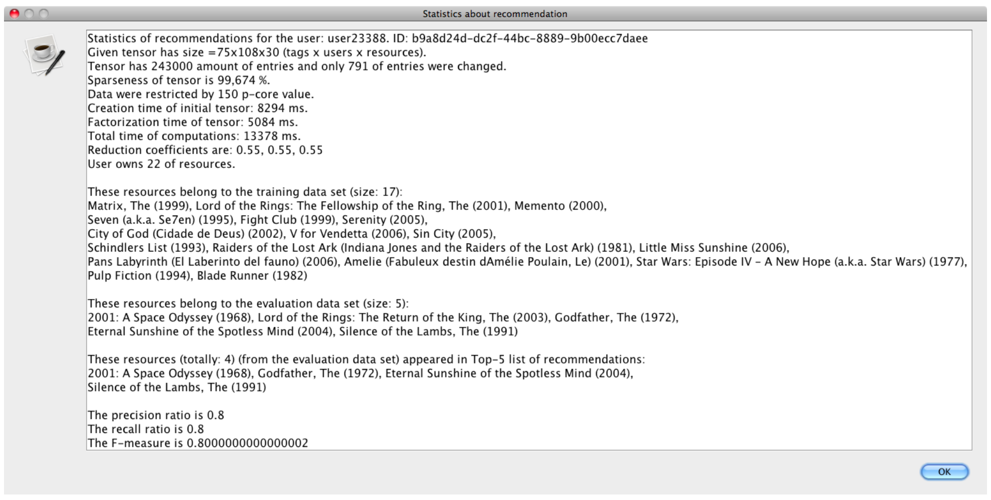
movielens | user23388 | dk.aau.cs.factorization.PCoreFactorization | 108 | P-c... 1!

Number	Resource	Score
1	Godfather, The (1972)	0.033663724745904004
2	Silence of the Lambs, The (1991)	0.028605854042162284
3	2001: A Space Odyssey (1968)	0.028605854042162284
4	Eternal Sunshine of the Spotless Mind (2004)	0.019224552417321013
5	Forrest Gump (1994)	0.018654330129559877
6	American Beauty (1999)	0.002822617871615726
7	12 Monkeys (Twelve Monkeys) (1995)	5.1682875726049824E-17
8	Shawshank Redemption, The (1994)	5.1682875726049824E-17
9	Star Wars: Episode V - The Empire Strikes Back (1980)	5.1682875726049824E-17
10	Lord of the Rings: The Return of the King, The (2003)	5.1682875726049824E-17
11	Brokeback Mountain (2005)	5.1682875726049824E-17
12	Spirited Away (Sen to Chihiro no kamikakushi) (2001)	1.6123893824853944E-17
13	Lord of the Rings: The Two Towers, The (2002)	3.3315876767798642E-40

Figure 3.5: The main window with the generated recommendations

3.3 The Open source frameworks

We use the following frameworks for our recommender system:



```
Statistics of recommendations for the user: user23388. ID: b9a8d24d-dc2f-44bc-8889-9b00ecc7daee
Given tensor has size =75x108x30 (tags x users x resources).
Tensor has 243000 amount of entries and only 791 of entries were changed.
Sparseness of tensor is 99,674 %.
Data were restricted by 150 p-core value.
Creation time of initial tensor: 8294 ms.
Factorization time of tensor: 5084 ms.
Total time of computations: 13378 ms.
Reduction coefficients are: 0.55, 0.55, 0.55
User owns 22 of resources.

These resources belong to the training data set (size: 17):
Matrix, The (1999), Lord of the Rings: The Fellowship of the Ring, The (2001), Memento (2000),
Seven (a.k.a. Se7en) (1995), Fight Club (1999), Serenity (2005),
City of God (Cidade de Deus) (2002), V for Vendetta (2006), Sin City (2005),
Schindlers List (1993), Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981), Little Miss Sunshine (2006),
Pans Labyrinth (El Laberinto del fauno) (2006), Amelie (Fabuleux destin dAmélie Poulain, Le) (2001), Star Wars: Episode IV - A New Hope (a.k.a. Star Wars) (1977),
Pulp Fiction (1994), Blade Runner (1982)

These resources belong to the evaluation data set (size: 5):
2001: A Space Odyssey (1968), Lord of the Rings: The Return of the King, The (2003), Godfather, The (1972),
Eternal Sunshine of the Spotless Mind (2004), Silence of the Lambs, The (1991)

These resources (totally: 4) (from the evaluation data set) appeared in Top-5 list of recommendations:
2001: A Space Odyssey (1968), Godfather, The (1972), Eternal Sunshine of the Spotless Mind (2004),
Silence of the Lambs, The (1991)

The precision ratio is 0.8
The recall ratio is 0.8
The F-measure is 0.8000000000000002
```

Figure 3.6: The statistics window for the generated recommendations

- **Colt**[2] – 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[3] and it is used for the CERN projects.
- **Hibernate** – to manipulate data as the objects of Java. We described our choice of Hibernate in the section 3.2.1.

3.4 Conclusion

We presented the design of the recommendation system based on the HOSVD tensor factorization technique. 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 Recommendation 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).

[2]http://acs.lbl.gov/software/colt/
[3]http://www.cern.ch

CHAPTER
4
Experiments

In this chapter we will present the goals, ideas and the results of the experiments. We will introduce the data sets and describe the methods of our experiments. Furthermore, we will present the results and the evaluations of the experiments.

4.1 Data sets

4.1.1 MovieLens

We have integrated a MovieLens data set provided by a GroupLens Research [19] to use as a data source when providing the recommendations. We have chosen a data set that contains 10680 unique resources tagged by 71567 different users with 16518 unique tags. In total, there are tagged 71155 resources using at least one or more tags by the users.

4.1.2 Delicious.com

To be able to compare the differences of a performance, quality and the statistics between the provided recommendations using a large data set (introduced in a section 4.1.1) and a small one, we have incorporated to use another data set provided by Choudhury, M. D. ([5]).
This data set contains tagged links by the users of a social bookmarking website Delicious.com.
The data set contains only 67 distinct resources, 1292 distinct tags, 2388 unique users and in totally there are 8915 relations of *user-tag(s)-resource.*

4.2 Experimental evaluation

We have to define the evaluation techniques precisely so we will be able to conclude the performance, the quality and the precision of the recommendations. Also, we will try to identify which settings – reduction coefficients suit the best when providing the recommendations using the real world data sets (as introduced in a section 4.1).

4.2.1 Evaluation methods

We will use the evaluation methods which are commonly used and accepted in this research area. The methods are used to evaluate our conducted experiments from the different aspects. We focus on the following aspects and methods:

- Time performance of a given method or an algorithm – we will measure the duration of a recommendation process.
- We will measure the sparsity of the used data to provide the recommendations.
- The accuracy and quality of the recommendations (concrete metrics can be found in the following section) – the intention is to measure and observe how the precise and accurate are personalised recommended items for a given user with his own interests and preferences.

The precision of the recommendations

We use the following commonly used metrics:

- Precision is the ratio of the number of recommended relevant items or tags (we consider only items from the evaluation part of the data set) from the top-N list of entries to N.

  $$\text{precision}_N = \frac{|\{\text{relevant documents}\} \cap \{\text{returned top-N documents}\}|}{|\{\text{returned top-N documents}\}|}$$

- Recall is the ratio of the number of recommmended items or tags from the top-N list of entries to the total number of items or tags set by a given user from the evaluation part of data set.

  $$\text{recall}_N = \frac{|\{\text{relevant documents}\} \cap \{\text{returned top-N documents}\}|}{|\{\text{relevant documents}\}|}$$

  Documents in the formulas represent resources or tags that were recommended to a particular user.

- 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.

  $$\text{F-measure} = 2 \; . \; \frac{\text{precision}_N . \text{recall}_N}{\text{precision}_N + \text{recall}_N}$$

4.3 Experimental configuration

4.3.1 Environment and programming language

All the experiments will be conducted on Windows 7 32–bit operating system running on Intel Core 2 Duo @ 2.66GHz CPU with $4,00$ GB ($3,25$ GB usable) RAM. The recommendations will be generated using our own implementation (as designed in a chapter 3).
The recommender is implemented using Java 6 programming language.

4.3.2 Data sets

To be able to conduct the experiments, we have to prepare the data sets. Each data set will be divided into the two parts: the first part will represent a training data which will be used for computing the personalised recommendations for a user. The second part of a data set – evaluation part – will be used to verify the quality and the accuracy of the recommendations – to find out how many tags or resources from the evaluation part were recommended for a given user. We will divide the original data set into the training and the evaluation parts with a ratio 75%:25%, respectively.
The data sets are split according to the data of a user, for whom the recommendations will be provided.

4.3.3 *p*-core

To make the data more dense we will apply so called *p*-core filtering – which means each user, resource and a tag has to appear at least *p* times in the data set [9]. Our implemented system will generate a list of the recommendations of the items for a selected user with the particular tags. We will use the following *p*-core values:

- 150, 120 and 90 for the MovieLens data set;
- 15 and 12 for the Delicious.com data set.

4.3.4 Reduction coefficients

To find out how the reduction coefficients c_1, c_2 and c_3 influence the quality of the recommendations, we will generate the recommendations for the same six users using all the combinations of the coefficients from the following set: $\{0.35, 0.45, 0.55, 0.65, 0.75, 0.85, 0.95\}$.
The coefficients are reducing the sizes of the resulting matrices from the SVD for the i-th mode matrices. For more details, please see the formulas 2.11, 2.12.

4.3.5 Users

Using a given *p*-core value and a combination of the coefficients, we will compute recommendations for the six different users:

- two the most active users;
- two users with an average activity;
- two the least active users.

The average values of the recommendation results for these users will be reported.

4.3.6 Top-N list

To be able to count precision, recall metrics and f-measure, we will consider the first top-10 recommendations as a top-N list. The recommendations will be sorted according to the values provided by the factorized tensor.

4.4 Results

4.4.1 Results of the experiments using MovieLens data

We conducted a number of the experiments using MovieLens data set. The results are presented in the following table: 4.1.

p-core	Size (users × resources × tags)	Reduction coefficients	Sparsity	Average precision	Average recall	F-measure	Average factorization time
150	108 × 30 × 75	$c_1 = 49$ $c_2 = 14$ $c_3 = 64$	99.672%	67.4%	67.4%	0.674	1549 ms
120	133 × 67 × 99	$c_1 = 86$ $c_2 = 23$ $c_3 = 55$	99.802%	36.6%	32.5%	0.343	14835 ms
90	167 × 120 × 159	$c_1 = 75$ $c_2 = 102$ $c_3 = 119$	99.884%	31.6%	24.3%	0.268	187642 ms

Table 4.1: Results for the MovieLens data set

According to the presented results we can conclude the following findings and observations:

- **Time performance** – when the size of a tensor increases, the recommendation process takes more time. The main reason is the computationally expensive tensor factorization.

- **Reduction coefficients** – when they are optimized, the more accurate results are obtained. The coefficients have to be estimated each time the size of the input data is changed. Also, we discovered that when the coefficients are not optimal, the quality and the time of the recommendations are affected negatively.
- **Training set** – when the size of a training set is relatively small, the precision and recall metrics decrease. Therefore, the considered least active users negatively influenced the average metrics.
- **Sparsity** – when the value of a *p*-core is greater, the more relations between the users, tags and resources are involved in the tensor factorization. Therefore, a sparsity decreases and the more precise results are provided.
- **Empty relations** – during the experiments, we could observe the cases when a particular user did not interact with the set of resources and tags. This resulted into involving useless information to the initial tensor and to the following factorization process.
- **Memory requirements** – we experienced that the HOSVD factorization requires a lot of memory resources. Due to the exceeded memory limits, the recommendations were not computed when the size of the tensor was greater than 251 users, 338 tags and 382 resources (*p*-core = 50).

4.4.2 Results of the experiments using Delicious.com data

We conducted a number of the experiments using Delicious.com data set. The results are presented in the following table: 4.2.

p-core	Size (users × resources × tags)	Reduction coefficients	Sparsity	Average precision	Average recall	F-measure	Average factor-ization time
15	55 × 67 × 125	$c_1 = 19$ $c_2 = 24$ $c_3 = 94$	99.869%	25%	25%	0.25	5831 ms
12	97 × 67 × 187	$c_1 = 34$ $c_2 = 27$ $c_3 = 140$	99.90%	33.3%	33.3%	0.333	27466 ms

Table 4.2: Results for the Delicious.com data set

Based on the results of the experiments, we can conclude that the observations described in the previous section (4.4.1) are still preserved.
However, we discovered that a problem of a small **training set** was more severe:

- **Training and Evaluation set** – most often the recommendations were computed using a very small training set and even smaller evaluation set (we split the users data to the training and the evaluation sets with a ratio 75%:25%, respectively). A small size of the evaluation set influences the precision metric negatively. This is because we restrict top-N list to the size of the evaluation set if the size is less than a constant N, e.g. if the size of the evaluation set is 1, the resource from this set must appear in the top-1 position. Such cases make precision and recall metrics inappropriate and unsuitable. Therefore, the average precision, the recall and the F-measure results in the previous table have no information value about the quality of the recommendations. This experience will force us to replace the given Delicious.com dataset with the larger one so the evaluation set will be big enough. We will exchange this data set during the next semester.

CHAPTER
5

Problems and proposals

The possible extensions and improvements for the recommendation systems based on a tensor factorization technique will be introduced. We will state the problems of these systems identified during the analysis and the experiments. Finally, we will present the proposals that could eliminate or minimize the impact of the described problems.

5.1 Recognized problems for tensor based recommenders

The introduced recommendation algorithms based on the tensor factorization overcome most of the state-of-the-art techniques and provide the accurate and precise results.
However, we identify the problems and issues of such systems. They are based on the theoretical knowledge gained during the analysis and the practical experience from the conducted experiments. We will describe them in the following sections.

5.1.1 Time performance

The introduced techniques in the sections 2.3.1, 2.3.2, 2.4.1 must compute tensor factorization in the offline mode because of the long-lasting computations. We verified this during the experiments as explained in the paragraph of the section 4.4.1. This problem depends on the size of the data set – using the real world data, a process of the tensor factorization becomes computationally excessive and expensive. Furthermore, when the additional dimensions are introduced the factorization turns into the unfeasible process (i.e. we assume a tensor with 4 dimensions, where each dimension has 100 items. The resulted tensor contains 100 millions entries). Therefore, four and more dimensional tensors can be only a concern of the theoretical studies. During the HOSVD factorization three distinct operations of the Singular Value Decomposition must be applied to the i-th mode matrices. The results are multiplied among each other to obtain a core tensor. In addition the results are multiplied with the core tensor to obtain the final approximated tensor which will be used for the recommendations. The execution time of the mentioned operations depends on the sizes of the involved matrices and tensors.

5.1.2 Incomplete interpretation scheme

The basic HOSVD model (2.3.1) interprets the input data and builds the initial tensor in a semantically restricted way. It considers only the two distinct interpretation cases, the former is when a particular user has tagged a given resource with a tag (positive case) and the latter is for all other situations. The second interpretation case is represented with a 0 weight and occupies over 99% of all entries in the initial tensor (2.4.1). This causes a sparsity problem which is described later in this section. The described interpretation scheme does not reflect correctly the preferences of a user and the recommendation results are not so accurate. There is no distinction between so called the negative cases – when a user has observed a resource but has not assigned any tag to it and a resource was not observed by the same user.

5.1.3 Sparsity

Sparsity is a severe problem that affects negatively the time performance of a recommender and mainly the quality of the recommendations – it is clear from the experimental results presented in the sections 4.4.1 and 4.4.2.
We observed the initial tensors have mostly empty relations (when the values in these tensors are zeros). The empty relations do not benefit to the time performance nor quality. The experimental results show that the least sparsity was greater than 99% – therefore, we consider reducing a sparsity as a key aspect to achieve more accurate recommendations.

5.1.4 Memory

Analysing the results of the conducted experiments we made an observation in the paragraph of the section 4.4.1 that a tensor factorization process requires a significant amount of the memory resources. The need for the resources increases more dramatically when there is used a bigger data set. We must to admit that the experiments were conducted using the samples of the real world data sets and we faced the memory problems generating recommendations using these snapshots. To be exact – the memory was consumed by the tensors which are the 3 dimensional matrices and the other matrices that are created and used during the computations. Each tensor/matrix is filled with the values of a type `double`. Each entry of a tensor/matrix consumes 8 bytes of a memory. During the factorization process there are created the following data structures:

- initial tensor (using a huge amount of data);
- factorized tensor (the same size as the initial one);
- core tensor (usually smaller than the initial one);

- three mode matrices (where the amount of rows is the same as 1 of the 3 dimensions in the initial tensor and the amount of columns is the result of the multiplication between other 2 dimensions of the initial tensor) and other temporary matrices as the inner results of the computations.

It is clear that the demands for a memory are very high when using the real world data sets and it complicates the process of a tensor factorization.

5.1.5 Training set

Almost every learning technique in the area of machine intelligence performs better when there is provided a big enough training set. It is crucial to provide as rich as possible data set for the recommender because the precision metrics directly depend on a size of the training set (as observed in the paragraph of the section 4.4.1).
However, it is important to avoid overfitting of the recommender with a training set – the quality of the recommendations and the time performance can decrease just because of a too large initial tensor.

5.1.6 Reduction coefficients

During the HOSVD factorization, the outcome matrices from the SVD of the i-th mode matrices are reduced by the reduction coefficients c_1, c_2, c_3 and used to construct a core tensor S by the formula 2.11. The authors propose this reduction to remove a 'noisy' information and suggest to set these parameters to the 70% of the most singular values. During the experiments we found out that this is not always the most accurate solution and it varies for each data set and even these parameters are distinct for the each mode matrix involved in the factorization. These parameters have to be tuned either when a new data set is introduced or when a size of the existing one is significantly changed. It is a computationally expensive process, but it must be considered to achieve the most accurate recommendations.

5.2 Proposed improvements

We mentioned the various issues for the algorithms based on the tensor factorization technique in the previous section 5.1. We will try to introduce the different techniques which could decrease the impact of the existing problems or even solve them. Also, we will try to involve pre- and post-filtering techniques to address the problems.
We describe the main principles of each method, how they could improve the existing techniques and how they could be implemented.

5.2.1 Pre-filtering

Proposal

Contextual information is used to select only the appropriate users, tags or resources and filter out not related information. For the recommendation systems based on a tensor factorization, there must be built a multi-dimensional tensor. Our idea is to apply pre-filtering in a such way it would reduce and minimize one or more dimensions of the initial tensor. The resulted tensor will contain only the relevant entries and then the HOSVD would be applied. This approach could be used when the contextual recommendation is required. The advantage of pre-filtering is that the amount of users, resources and tags will be smaller, but more dense as only the similar or close tags, users and resources will be involved. On the other hand a tensor factorization would be computed each time the recommendations are required by the user.

Addressed problems

This proposal could decrease the impact of the following problems:

- sparsity;
- memory;
- time performance;
- training set;
- cold start problem;
- eliminates the problem of handling data updates.

During the experiments we observed that a factorized tensor can be computed in about 1.5 second and provide the most precise recommendations compared with the other results. Also, the initial tensor was the least compared to the others used. However, the sparsity still was unacceptably high. There will be no need to implement the complex solutions to handle arrival of new users, tags and resources (as analysed in a section 2.3.3) because the tensor factorization will be re-computed each time when the recommendations are requested.
The Figures 5.1 and 5.2 represent the idea – a huge initial tensor with sparse data could be reduced to the small *personal* tensor with dense data.

Idea of the implementation

These facts inspire a proposal that may minimize or eliminate the mentioned problems:

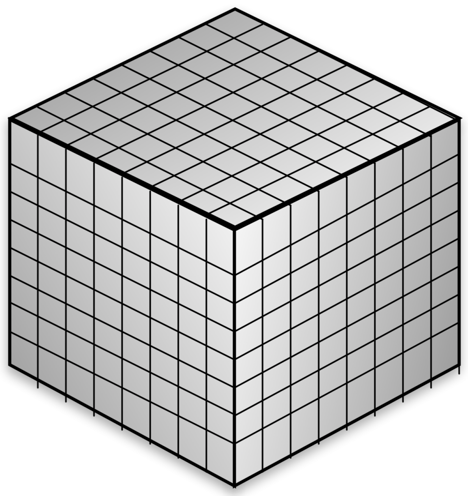


Figure 5.1: A huge initial tensor with sparse data

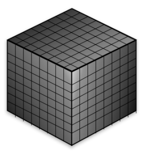


Figure 5.2: A small *personal* initial tensor with dense data

1. When generating the recommendations, build the *personal* initial tensor for the user for which the recommendations will be provided. Such tensor will be tuned to provide the recommendations for a single user only.

2. Personalized tensor must be small enough (having a similar number of the relations as the case of the experiments – $108 \times 30 \times 75$ (tags $\times$ users $\times$ resources)) – it will not consume a lot of memory, the computations will be executed within the acceptable time frame.

3. The training set must be enriched with the tags that have the highest similarity scores with the preferences of a current user (Tag Similarity, as analysed in the section 2.2, formula: 2.2). The resources, tagged by these tags, can be easily resolved and incorporated into the initial tensor. As a result – a rich training set will be used. It would also decrease a cold start problem.

4. More dense data should decrease the sparsity.

5. There will be no need to handle new data – the recommendations would be generated in 1.5 second or even faster – it means there is no need to store a computed factorized tensor in a memory nor manage it.

Because of a rich training set it would be possible to recommend up to $N - 1$ resources to the user (where N is a number of the distinct resources in the initial tensor and at least one resource should be tagged by the user to compute Tag Similarities). Such number of the recommendations should be large enough to satisfy the needs of the users.
Tag similarities can be computed in the off-line mode and stored for the re-use – there is no need to re-compute the similarities as the textual representation and the semantic meaning stay the same once computed.
We assume the proposed extensions for building personalized tensor will not be computationally expensive. Even more, they can be implemented in our current system (as designed in the section 3).

5.2.2 Post-filtering

Proposal

Similar proposal to the pre-filtering, the only difference is that filtering according to a given contextual information is performed after the tensor factorization is computed. This method alone would not improve time performance nor storage demands but could offer contextual based recommendations.
We propose to compute the recommendations for the p most active users using their data to create the initial tensor and apply post-filtering for the set of the recommended resources.

Addressed problems

We assume incorporating partly p-core filtering (when only the p most active users and their data is used) with post-filtering could help to improve the results and decrease the impact of these problems:

- memory;
- time performance;
- training set;
- eliminates the problem of handling data updates.

The experimental results showed that using the data of the most active users, the most popular tags and resources the time performance and quality of the recommender is promising. However, in most of the cases the user, for which the recommendations need to be computed, will not be among the most active users and the factorized tensor will not reveal the recommendations for a user. This problem may be solved finding the N most relevant recommendations for a current user between the top recommendations for the p most active users. The idea of a proposal is shown in the Figure 5.3

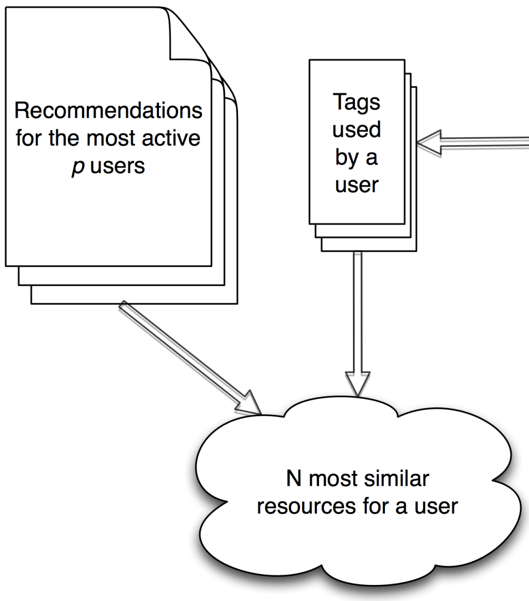

Recommendations for the most active *p* users
Tags used by a user
Resources used by a user
N most similar resources for a user

Figure 5.3: Post-filtering applied to the top recommendations for the p most active users and the tags used by the user

Idea of the implementation

These are the key points to realize the proposal:

1. Resolve all the tags used by the user.
2. Make the initial tensor using the data of the p most active users.
3. Compute the recommendations for these active users using their training sets.
4. Compute the similarities between the tags of a user and the tags of recommended items as analysed in the section 2.2.
5. Pick the N most similar tags and resolve the resources were tagged by them. This will be the list of the recommendations for a user.

This proposal does not address so many problems as a pre-filtering, but we think it is worth investigating it. Nevertheless, the memory and time performance problem will be minimized because of the partly applied p-core method. The most active users will have rich training sets. The sparsity should not be so high as using the random data. The recommendations can be computed from the scratch, because of the acceptable time performance – there is no need to handle the new data. However, it may be difficult to choose an optimal value for p.

5.2.3 Correlated feature hashing

Proposal

Another proposal is to reduce the size of the initial tensor using a correlated feature hashing function. The idea is to share and group tags with the similar meaning. To group similar tags we would use DICE coefficient:

$$DICE(i,j) = \frac{2.\text{cooccur}(i,j)}{\text{occur}(i) + \text{occur}(j)}$$

where each word i is from all tags and j belongs to the top F most frequent tags. We took inspiration from [1] where authors used this hashing function in information retrieval algorithm and reduced dictionary size from $2,5$ million words to 30000 most frequent words with relatively good results. Our idea is to reduce a number of all tags to the F most frequent and then apply tensor factorization.

Addressed problems

We believe hashing function would help to achieve an improvement because of these minimized problems:

- sparsity;
- memory;
- time performance.

It is clear that with a help of a hashing function the size of an initial tensor will be much smaller, so it should reduce sparsity and the needs for the memory. The recommendations should be also computed faster. The tensor factorization must be re-computed only when new tags are introduced or when tags frequency will be changed.
One of the possible drawbacks of this proposal may be the need of frequent clustering of the tags and the algorithm to identify the resources using hashed tags (when new tags will be introduced). We will expand on these issues in the following section.

Idea of the implementation

We present the key points of our proposal:

1. Identify the most frequently used tags.
2. Apply hash function and cluster the tags into the groups.
3. Build the initial tensor using only the most frequently used tags.
4. Execute tensor factorization.
5. Compose a list of recommendations.

The main challenge may be to identify a real recommended resource knowing a recommended hashed tag. It would be possible to do this if a unique identifier would be assigned to a tag. Also, the binary relations of the unique identifier and a value of a hash function on the real tag should be preserved. Furthermore, there is needed an algorithm to check if a current resource is tagged with one of the most frequently used tag. If not – if it belongs to the group of the tags identified by the current frequent tag.
Also, it is needed to handle the new tags – the hash function should be re-applied when a new tag is introduced or the frequencies of a tag usage had change.
Moreover, it is possible that this method could reduce the training set dramatically and affect negatively the quality of the recommendations.

5.2.4 Sparse matrices

Proposal

We designed and implemented the recommender based on a tensor factorization technique not caring about the density of the data. As it turned out, the real world data is sparse and it affects negatively the outcomes of a recommender. The idea is to take an advantage of the existing sparse data structures to present the initial tensor and all the other tensors and matrices.

Addressed problems

It is possible that using the adapted data structures for the sparse data we could reduce the following problems:

- sparsity;
- memory;
- time performance.

In stead of trying to solve the sparsity trying to use pre-filtering (as proposed in the section 5.2.1) or the hashing (as proposed in the section 5.2.3), this is a proposal to use the special data structures and algorithms that are specially designed to perform well using the sparse data.

Idea of the implementation

The open source framework Colt (as introduced in the section 3.3) offers the API for the sparse tensors and matrices. It is claimed it is very efficient handling data – the zero values do not require any memory. Moreover, the values that become zeros are ready to release the memory. The memory is reclaimed automatically by Colt, but it is possible to execute this operation manually. Furthermore, the time complexity issuing the basic operations on the matrices/tensors built using sparse data structures, is $\mathcal{O}(1)$.
There is a number of the recently proposed ideas how to improve the time performance when computing a factorization using the sparse data: [11], [12], [20]. There is suggested to take an advantage of a modern hardware of the computers – to employ the parallel computers or the processors of the multiprocessor workstations. It means the factorization algorithm must be highly scalable.

5.2.5 Different interpretation schemes

Proposal

The idea is to create the initial tensor with the different interpretation schemes. The 0/1 scheme given by the formula 2.9 in the section 2.3.1 is semantically restricted. During the experiments, where the basic 0/1 scheme was used – we could observe that the quality of the recommendations was not good enough. Therefore, we will consider the possible extensions of the 0/1 scheme and mainly we will try to integrate Post-Based Ranking Interpretation scheme (PBRI). PBRI interprets the input data in a different way and it solves the semantic drawbacks of the 0/1 scheme. The comparison of both interpretation methods is analysed deeply in the section 2.4.1. We believe that usage of this scheme could significantly improve the quality of the recommendations because of these arguments:

- Authors of the paper [17], clearly showed the contribution of the PBRI in the experiments part (this method outperformed HOSVD factorization with the 0/1 scheme and the other state-of-the-art recommendation systems).
- We could verify during the experiments that constructing the initial tensor should be applied with more than just two semantically distinct cases as the 0/1 scheme does.

We will try to modify and combine PBRI with the different factorization techniques that were introduced in the analysis part. According to the conducted experiments we will propose the most appropriate interpretation function.

Addressed problems

This proposal could partially solve the following problems:

- sparsity;
- quality of the recommendations.

Idea of the implementation

The idea is to replace the current 0/1 function with the proposed schemes and in combination with the different factorization techniques try to observe the quality of the recommendations. The described PBRI consists of the ranking constraints and the learning phase – therefore, we will have to adjust our architecture to allow learning process before a factorization will be applied.

5.2.6 Various factorization techniques

Proposal

We will inspect and explore the possible tensor decomposition techniques. Most of the current approaches are based on Higher Order Singular Value Decomposition (HOSVD). This technique decomposes the initial tensor into the core tensor and the factor matrices for each dimension (users, tags and resources). The approximated tensor is the result of a multiplication between the core tensor and the factor matrices (see the formula 2.12 in the section 2.3.1). This multiplication is implemented as a nested sum of degree 3 and we can conclude that time complexity for predicting one relation between user, tag and resource is $\mathcal{O}(k^3)$, where $k = \min(c_1, c_2, c_3)$. However, there exist the other factorization methods like: Non-Negative Tensor Factorization (NTF) and Canonical Decomposition (CD) also called Parallel Factor Analysis (PARAFAC). The main advantage of CD is a better time complexity as was showed in the section 2.4.2 – the proposed recommender has a linear time complexity and provides at least as good results as the other systems based on HOSVD factorization. The difference between the HOSVD factorization and CD technique is that a core tensor is a diagonal tensor, therefore the following holds for the entries of the tensor:

$$\hat{s}_{u,i,t} = \begin{cases} 1, & if\ u = i = t \\ 0, & else \end{cases} \tag{5.1}$$

The final tensor is approximated according the formula 2.21 in the section 2.4.2. The linear time complexity of the CD is caused by the construction of the core tensor and therefore there is only need for one loop which iterates through the diagonal entries of the tensor. The NTF is the special case of factorization when values in the approximated tensor can be only non-negative. We could observe negative values in the resulted tensor during the experiments and these cases were confusing, hardly interpretable. Therefore, the NTF could remove this problem by the non-negative constraints. The time complexity of the NTF depends how the core tensor is constructed – either linear when the core tensor is diagonal or cubic time when the core tensor contains entries not only on the diagonal positions. Our intention is to replace HOSVD factorization with the CD and NTF techniques and find out if the given methods will improve the time performance and quality of the recommendations.

Addressed problems

This proposal could decrease the impact of the following problems:

- time performance;
- negative values in the approximated tensor.

The proposed CD has linear time complexity, which clearly overcomes the HOSVD factorization. The only issue is the trade-off between the time performance and quality of the recommendations.
The usage of the NTF should solve the problem of the negative values in the approximated tensor.

Idea of the implementation

The mentioned techniques would replace the HOSVD factorization, therefore integration into our current architecture should be smooth and easy-going. The implementation of the factorization methods will be done according the pseudo code and formulas from the articles [18], [6] and the other available literature.

5.3 Conclusion

We proposed a number of the possible extensions which we will implement and evaluate during the next semester. The proposed techniques were developed and formulated in a such way that the recognized problems will be eliminated or the impact of them would be decreased. We assume they could benefit for achieving the improvements for the tensor based recommender systems. There is a number of the possibilities to combine the mentioned

techniques into the hybrid approaches, but a number of the experiments will have to be conducted to verify our assumptions.
In the following Figure 5.4 we present the ideas how our proposals could be combined:

- **Preparation of data**. Pre-filtering proposal could be applied no matter which tensor factorization technique will be chosen. More over, we could try to apply hash function to make the initial tensor smaller.
- The proposed **interpretation schemes** are independent from the other phases like: preparation of data, factorization techniques etc. Therefore, we will try to investigate which interpretation scheme is the most suitable and best performing within the all combinations of the other phases (i.e. different factorization techniques, pre-filtering etc). We expect that improved interpretation scheme should provide more precise and personalised results. It is possible that during the experiments we will observe and discover some unknown facts, aspects which will help us to propose a custom interpretation scheme.
- **Factorization techniques**. The techniques running in a linear time could greatly impact the time of the production of the recommendations. However, we were not able to test the quality of the factorization techniques running in the linear time.
- **Filtering results**. This step of the production of the recommendations is independent from the other proposals and could be applied freely no matter which factorization technique or interpretation scheme is used.

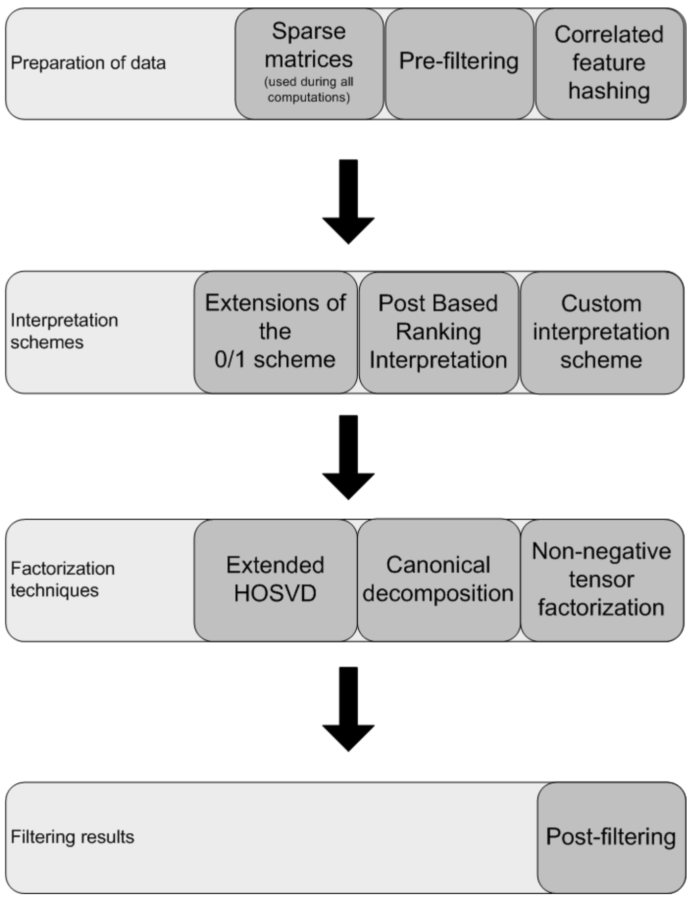

Preparation of data
Sparse matrices (used during all computations)
Pre-filtering
Correlated feature hashing
Interpretation schemes
Extensions of the 0/1 scheme
Post Based Ranking Interpretation
Custom interpretation scheme
Factorization techniques
Extended HOSVD
Canonical decomposition
Non-negative tensor factorization
Filtering results
Post-filtering

Figure 5.4: The combination of the proposed improvements

CHAPTER
6

Conclusion

The recommendation systems based on the tensor factorization techniques were the main concern of our studies during this semester. The systems based on the different factorization techniques like: Higher Order Singular Value, Non-Negative Tensor factorization and Canonical Decomposition were described.

During the analysis we discovered that these systems outperform the current state-of-the-art recommenders. However, they have the drawbacks that should be considered and resolved. Also, we investigated the hybrid recommendation system that is not based on the tensor factorization. This system is interesting because of the ideas how to calculate similarities between the syntactically different tags. We provided a comparison table that shows the main advantages and disadvantages of the different factorization methods from the various perspectives.

During the design phase we made the main decisions about the recommendation system. The recommender layer was designed according to the model described in the section 2.3.1. We described the main layers of the applications as Database, Data structures, Utilities, Recommender, Statistics and Graphical User Interface.

We conducted a series of the experiments using the implemented system. We have chosen to use two different data sets that are the samples of the real world data. According to the results of the experiments we discovered and proved the number of problems that our investigated recommendation system suffers from.

We proposed a number of possible solutions to the known problems from the analysis and the problems discovered during the experiments. We have grouped the proposals according to the problems they are addressing. Each of the idea can be implemented independently to the other ideas. Moreover, the combination of the ideas can be chosen to achieve the best results.

The main goals of our work during the next semester are as follows:

1. Investigate each of the discussed proposals deeply. Generate new ideas based on the most promising. Provide and extend the arguments why

the proposed ideas can work or can not.

2. Implement a prototype of the recommendation system that will embrace the best ideas how to improve the process of the recommendation. Conduct the experiments using the prototype and the same data sets to discover and verify the expected improvements.

3. Precisely define the advantages of the most promising factorization techniques. Provide an extensive overview of them, extended with the most contributing techniques from the other phases (techniques used within the interpretation schemes, preparation of data and filtering results phases).

4. Propose a recommendation system that embraces the currently known advantages and our proposals that minimize or eliminate the current problems.

BIBLIOGRAPHY

[1] 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.

[2] Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee. Linked data on the web (ldow2008). In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 1265–1266, New York, NY, USA, 2008. ACM.

[3] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12:331–370, November 2002.

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

[5] M. D. Choudhury. Datasets, 2010. [Online; accessed 01-December-2010].

[6] 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.

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

[8] Peter Dolog Frederico Durao. Extending a hybrid tag-based recommender system with personalization. Sierre, Switzerland, 21/03/2010 2010. ACM Press, ACM Press.

[9] 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.

[10] Hui Guo. Soap: Live recommendations through social agents, 1997.

[11] Anshul Gupta, George Karypis, and Vipin Kumar. Highly scalable parallel algorithms for sparse matrix factorization. *IEEE Trans. Parallel Distrib. Syst.*, 8:502–520, May 1997.

[12] Anshul Gupta, Seid Koric, and Thomas George. Sparse matrix factorization on massively parallel computers. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, SC '09, pages 1:1–1:12, New York, NY, USA, 2009. ACM.

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

[14] 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.

[15] Aleksandra Klasnja Milicevic, Alexandros Nanopoulos, and Mirjana Ivanovic. Social tagging in recommender systems: a survey of the state-of-the-art and possible extensions. *Artif. Intell. Rev.*, 33:187–209, March 2010.

[16] Michael J. Pazzani and Daniel Billsus. The adaptive web. chapter Content-based recommendation systems, pages 325–341. Springer-Verlag, Berlin, Heidelberg, 2007.

[17] 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.

[18] S. Rendle and L. Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 81–90. ACM, 2010.

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

[20] Edward Rothberg and Anoop Gupta. Techniques for improving the performance of sparse matrix factorization on multiprocessor workstations. In *Proceedings of the 1990 ACM/IEEE conference on Supercomputing*, Supercomputing '90, pages 232–241, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[21] J. Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. Collaborative filtering recommender systems. In Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, editors, *The Adaptive Web*, volume 4321

of *Lecture Notes in Computer Science*, pages 291–324. Springer Berlin / Heidelberg, 2007. $10.1007/978 - 3 - 540 - 72079 - 9_9$.

[22] Barry Smyth. The adaptive web. chapter Case-based recommendation, pages 342–376. Springer-Verlag, Berlin, Heidelberg, 2007.

[23] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009.

[24] 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.

[25] Panagiotis Symeonidis, Alexandros Nanopoulos, and Yannis Manolopoulos. A unified framework for providing recommendations in social tagging systems based on ternary semantic analysis. *IEEE Trans. on Knowl. and Data Eng.*, 22:179–192, February 2010.

[26] Thomas G. Tape. The area under an roc curve.

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

[28] Ellen M. Voorhees. Using wordnet to disambiguate word senses for text retrieval. In *SIGIR '93: Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 171–180, New York, NY, USA, 1993. ACM.

[29] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, pages 133–138, Morristown, NJ, USA, 1994. Association for Computational Linguistics.

[30] 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.