Incremental matrix factorization for simultaneous learning from parallel data streams

A dissertation presented by

Martin Jakomin

to
The Faculty of Computer and Information Science
in partial fulfilment of the requirements for the degree of
Doctor of Science
in
Computer and Information Science

Ljubljana, 2019
I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma of the university or other institute of higher learning, except where due acknowledgement has been made in the text.

— Martin Jakomin —
December 2019

The submission has been approved by

dr. Zoran Bosnić  
Professor of Computer and Information Science  
Advisor

dr. Matej Kristan  
Associate Professor of Computer and Information Science  
Examiner

dr. Sašo Džeroski  
Professor of Computer and Information Science  
Examiner  
Jožef Stefan Institute, Ljubljana

dr. Tomaž Curk  
Assistant Professor of Computer and Information Science  
Advisor

dr. Marinka Žitnik  
Assistant Professor of Computer and Information Science  
Examiner

dr. Mirjana Ivanović  
Professor of Computer Science  
External Examiner  
University of Novi Sad, Serbia
I hereby declare that the research reported herein was previously published/submitted for publication in peer reviewed journals or publicly presented at the following occasions:


I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Ljubljana.
POVZETEK

Matrična faktorizacija se je izkazala kot uporabna in zanesljiva metoda za implementacijo obsežnih aplikacij strojnega učenja, kot so na primer priporočilni sistemi. Težave z redkostjo podatkov in problem hladnega zagona se lahko posredno omilijo z uporabo več heterogenih virov podatkov, hkrati pa uspešna uporaba zlivanja podatkov doprinaša večjo točnost napovedi. Za vsakodnevnine aplikacije, na primer take s stalnimi povratnimi informacijami uporabnikov, ostaja inkrementalno posodabljanje modelov, naučenih na več podatkovnih tokih, ključen in le delno rešen problem.


Predlagana metodologija ponuja pomoč pri razvoju algoritmov za sočasno modeliranje podatkovnih tokov v realnem času. Poleg priporočilnih sistemov pa vsestranskost matrične faktorizacije omogoča njeno uporabnost za reševanje številnih drugih problemov strojnega učenja, kot so zmanjševanje dimenzionalnosti, gručenje in klasifikacija.

**Ključne besede** strojno učenje, matrična faktorizacija, podatkovni tokovi, zlivanje podatkov, inkrementalno učenje, priporočilni sistemi, generator sintetičnih podatkov
Matrix factorization techniques have proven to be useful and reliable for solving large-scale machine learning problems. The data sparsity and cold-start problems found in real-world applications, such as recommender systems, can be indirectly alleviated by considering multiple heterogeneous data sources, while at the same time the successful utilization of data fusion resolves in a higher predictive accuracy. However, incrementally handling models upon multiple data streams remains a crucial and only partially solved problem.

This work presents one way of fusing multiple data streams through matrix factorization. Our proposed method models heterogeneous and asynchronous data streams and provides predictions in real time. As a result of incremental updating, the proposed method successfully adapts to changes in data concepts, while application of data fusion improves prediction accuracy and reduces effects of the cold-start problem. Using the proposed methodology we develop a streaming recommender system and show how prediction accuracy can be substantially increased by considering multiple data sources. Nevertheless, evaluating data fusion, recommender and other incremental algorithms, such as our presented method, is inherently difficult due to the scarcity of obtainable data sources. In order to address this problem, we conjointly propose a synthetic data generator, capable of generating multiple temporal and inter-dependent data streams of relational data. Data streams generated this way successfully mimic real-life datasets in terms of statistical data properties and comparable performance of various machine learning models.

Proposed methodologies help in development of solutions for collective modeling of streaming data in real-time. Apart from recommender systems, the versatility of matrix factorization further allows for its use in solving several other machine learning problems, such as dimensionality reduction, clustering and classification.

**Keywords**  machine learning, matrix factorization, data streams, data fusion, incremental learning, recommender systems, synthetic data generator
ACKNOWLEDGEMENTS

First, I would like to thank my thesis advisors and mentors, prof. dr. Zoran Bosnić and dr. Tomaž Curk for their help and guidance during my research. Thank you for your patience and numerous helpful comments and remarks, which greatly improved my research skills and my work.

Next, I would like to thank all members of the Laboratory for Cognitive Modeling for creating a great and stimulative work environment.

I would also like to thank my family, my parents Irena and Bojan, and my brother Jernej for their continuous love and support.

This thesis is dedicated to my lovely wife Ana, who encouraged me through my years of study and whose endless love made this thesis possible. Thank you for making me a better person.

Lastly, I would like to thank our puppy Rubi for giving me the needed energy, enthusiasm and love to finish this journey.

# CONTENTS

- **Povzetek** vii
- **Abstract** ix
- **Acknowledgements** xi

## 1 Introduction
  1.1 Summary of the scientific contributions 3
  1.2 Availability 4
  1.3 Overview of thesis structure 4

## 2 Basic concepts and related work
  2.1 Matrix factorization 6
    - 2.1.1 Constrained matrix factorization 7
    - 2.1.2 Weighted matrix factorization 7
    - 2.1.3 Matrix tri-factorization 8
    - 2.1.4 Solving the optimization problem 8
  2.2 Matrix factorization in recommender systems 9
    - 2.2.1 Evaluating recommender systems 10
  2.3 Incremental matrix factorization 11
    - 2.3.1 Evaluation of incremental methods 13
  2.4 Data Fusion 14
    - 2.4.1 Collective matrix factorization 15
    - 2.4.2 Data Fusion in recommender systems 16
Generating inter-dependent data streams

3.1 Background

3.2 Generating multiple inter-dependent data streams

3.2.1 Cluster structure

3.2.2 Magnitudes of relations - generating rating values

3.2.3 Time dependency in data

3.2.4 Generating concept drifts and creating data streams

3.2.5 Generating multiple inter-dependent data streams simultaneously

3.2.6 Generating attributes and side information

3.2.7 GIDS dataset generator

3.3 Evaluation: Real-life dataset resemblance

3.3.1 GIDS parameter estimation and data generation

3.3.2 Comparison between GIDS and MovieLens descriptives in a static environment

3.3.3 Comparison between GIDS and MovieLens descriptives in a dynamic environment

3.4 Evaluation: Data modeling on synthetic datasets

3.4.1 Recommender system performance

3.4.2 Time-aware recommender system performance

3.4.3 Ranking

3.4.4 Data fusion algorithm performance

3.5 Summary

4 Simultaneous incremental matrix factorization

4.1 Background

4.2 Simultaneous Incremental Matrix Factorization

4.2.1 Matrix tri-factorization

4.2.2 Data fusion and collective matrix factorization

4.2.3 Collective factorization in the streaming environment

4.2.4 Tackling sparseness

4.2.5 Calculating derivatives

4.2.6 Incorporation of additional information

4.2.7 Final objective function
4.2.8 Initialization of factor matrices ...................................... 62
4.2.9 Updating the factorization ........................................... 62
4.2.10 Handling concept changes in multiple data streams ........ 66
4.2.11 Predictions ............................................................ 67
4.3 Deriving different update rules ......................................... 68
  4.3.1 SIMF with nonnegative multiplicative update rules ........ 68
  4.3.2 SIMF with gradient descent ...................................... 70
  4.3.3 SIMF with stochastic gradient descent ......................... 71
  4.3.4 The algorithm pseudocode ...................................... 72
4.4 Evaluation on synthetic data streams ................................ 73
  4.4.1 Robustness to concept drift ..................................... 75
4.5 Evaluation on real data streams ....................................... 77
  4.5.1 Evaluation under the cold-start .................................. 81
4.6 Summary ................................................................. 83
5 Conclusion ................................................................. 85
  5.0.1 Future work .......................................................... 87

Bibliography ................................................................. 89

Razširjeni povzetek .......................................................... 97
Introduction
The constant growth of obtainable data confronts machine learning algorithms with new challenges. Apart from the sheer amount of information (commonly known as the high “Volume” characteristic of the Big Data), machine learning algorithms must also handle the large heterogeneity of data (the high “Variety”). Finally, many real-world applications require processing and forecasting in real time, where new information comes in a shape of data streams (the high “Velocity”). Therefore, new problems call for scalable algorithms that can integrate different types of data in real time.

One such group of methods that inherits the above mentioned properties is matrix factorization. Matrix factorization is one of the fundamental and most popular building blocks of machine learning and can be found in dimensionality reduction, factor analysis, recommender systems, clustering, relational learning, ranking, classification and regression. Common algorithms in machine learning include: principal component analysis [1], singular value decomposition [2], latent semantic indexing [3], nonnegative matrix factorization [4], k-means clustering [5], maximum margin factorization [6], etc. A large part of this thesis will focus on recommender systems and latent collaborative filtering models, based on matrix factorization [7–9]. The scalability problem can be solved with appropriate dimensionality reduction (the main principle behind matrix factorization), while the integration of heterogeneous data can be achieved via data fusion [10–12]. For successful modeling of data streams in real time, matrix factorization techniques can utilize incremental updating, i.e. incremental matrix factorization [13, 14]. However, collective modeling of heterogeneous data streams in real time still remains an open problem.

This thesis focuses on solving the above mentioned problems by introducing a novel incremental and simultaneous matrix factorization technique. In this work we present Simultaneous Incremental Matrix Factorization (SIMF) that models multiple heterogeneous and asynchronous data streams and provides predictions in real time. SIMF dynamically refines its collective factorization model without storing any previously collected information. The incremental update also allows for quicker adaptation to new concepts in data, while fusion of multiple data streams results in a higher predictive accuracy. In recommendation problems SIMF incorporates additional implicit information, such as biases and regularization [8] and uses collective factorization to alleviate the cold-start problem and boost prediction accuracy.

In order to properly evaluate our proposed methodology, we conjointly designed a synthetic data stream generator (GIDS) for generating inter-dependent collections of
relational data. Data streams generated by GIDS mimic real-life datasets in terms of statistical data properties and can be tuned to test specific aspects of algorithms, such as robustness to changes in concept, etc. We use GIDS and data streams from real-life problems to extensively evaluate the proposed method in a streaming environment. Results confirm that predictions can be improved by extending the factorization process with additional data streams.

1.1 Summary of the scientific contributions

The following scientific contributions are presented in this dissertation:

1. **SIMF - a novel approach for incremental and simultaneous learning from multiple data streams (Chapter 4).**

   We propose a simultaneous and incremental matrix tri-factorization approach, capable of fusing multiple heterogeneous and asynchronous data streams of different sizes and densities. We provide derivation and mathematical analysis of the optimization algorithm, alongside with three distinct sets of update rules. Finally, we propose an application in a form of a streaming recommender system, which is evaluated on both, synthetic and real-life data streams.

2. **GIDS - a generator of inter-dependent data streams (Chapter 3).**

   We propose a synthetic generator of inter-dependent data streams. The proposed algorithm generates a collection of multiple inter-connected relational datasets in matrix form, alongside with a time dependency component that can be used for simulation of concept-changing data streams. A large set of tunable parameters allows for systematic generation of specific datasets for testing distinct algorithm properties. We provide a thorough evaluation and comparison to real-life datasets in terms of statistical properties through time and modeling capabilities of various machine learning algorithms.
1.2 Availability

We provide the following implementation of our proposed methods:

- SIMF - Simultaneous incremental matrix factorization
  https://github.com/MartinJakomin/SIMF

1.3 Overview of thesis structure

The thesis is split into three main parts. In the first part (Chapter 2), we make an overview of related work and describe the basic methods, such as matrix factorization, incremental learning, data fusion and recommender systems.

In the second part (Chapter 3), we describe the problem of data scarcity and present a solution – our proposed generator of inter-dependent data streams (GIDS). We present the basic structure and provide a pseudocode for its implementation. Afterwards, a thorough evaluation is held, where generated data is compared to real-life datasets in terms of statistical properties and modeling capabilities.

In the third and final part (Chapter 4) we present our main contribution, the methodology for simultaneous learning from multiple parallel data streams (SIMF). We present the method by gradually constructing the objective function, while explaining the underlying concepts along the way (collective matrix factorization, tackling sparsity, adding additional constraints, etc.). Next, we derive three sets of update rules (nonnegative multiplicative rules, gradient and stochastic gradient descent), followed by an incremental evaluation of the proposed methodology.

Chapter 5 holds conclusions with our final thoughts and directions for future work.
Basic concepts and related work
Our methodology is based on three major concepts: matrix factorization, data fusion and incremental (online) learning. In this chapter we present the basics of these concepts. We begin with matrix factorization and then continue with its role in recommender systems. Then, we discuss basic upgrades toward incremental matrix factorization, including a special mention of the evaluation of incremental and streaming algorithms. Finally, we present data fusion and collective matrix factorization, where “sharing” of factors between different data sources can lead to discovery of shared hidden latent properties.

2.1 Matrix factorization

Matrix factorization (matrix decomposition, matrix approximation) is an essential tool in machine learning. It is broadly used for various problems, mainly for dimensionality reduction and compression, collaborative filtering (recommending) clustering, ranking and classification.

In general, matrix factorization’s task is to factorize (decompose) an input matrix into two or more factor matrices. Let $X \in \mathbb{R}^{n \times m}$ be the input matrix (for instance $X$ can represent a matrix of measurements or matrix of object relations), then $X$ can be factorized into two matrices $U \in \mathbb{R}^{n \times k}$ and $V \in \mathbb{R}^{m \times k}$ so that:

$$X \approx \hat{X} = UV^T,$$

where $k$ is the rank. Usually, $k$ is selected to be relatively small compared to the matrix dimensions, i.e. $k << \min(n, m)$ (low-rank matrix approximation) which yields a powerful dimensionality reduction and compression tool.

The approximation of factors (the divergence between $X$ and $\hat{X}$) can be measured in various ways and is determined by the underlying problem. This loss function is usually expressed in a form of a generalized Bregman divergence, such as a squared loss (Frobenius norm) or Kullback-Leibler divergence \[15\]. The most common and used is the squared loss error of the difference between $X$ and $\hat{X}$:

$$||X - \hat{X}||_F^2 = ||X - UV^T||_F^2 = \sum_{i=1}^{n} \sum_{j=1}^{m} (x_{ij} - u_i v_j^T)^2.$$ \hspace{1cm} (2.1)
2.1.1 Constrained matrix factorization

Apart from different loss functions, we can apply additional constraints to the factorization process in order to tackle a specific task or to retain specific properties. These constraints can shape the factor spaces and introduce new interpretations of those factors and predicted values (unconstrained predictions can often lead to wrong interpretations of results). One of the most popular constraints is the nonnegativity constraint, on one or all factors. This special case of matrix factorization is called nonnegative matrix factorization or NMF. The use of only additive linear combinations in NMF imposes a parts-based representation of the input data and therefore provides interpretable solutions \[4\]. Furthermore, NMF has an inherent clustering property and is similar to k-means clustering, where the columns of the first factor represent the cluster centroids and rows of the second factor represent the cluster membership probabilities \[5\].

Other constraints include imposed sparseness (in order to constraint the number of linear combinations which can result in a better model), orthogonality (which guarantees uniqueness of a solution and provides clustering interpretations \[16\]) or any other intrinsic geometrical structure (which preserves local topological properties) \[17\]. Additionally, L$_1$ or L$_2$ regularization can be supplemented to the factorization process to avoid overfitting. Apart from constraints, other desirable properties can be achieved using non-linear combinations (interactions) between factors with kernel matrix factorization \[18, 19\].

2.1.2 Weighted matrix factorization

The factorization process can be weighted (weighted low-rank approximation) in order to emphasize the relative importance of different components in different manners. For example, zero-one weights can be used when dealing with missing values (when some measurements are not observed) in order to model only observed measurements, which leads to a better reconstruction of the underlying structures \[20\]. Weighted matrix factorization is defined as:

$$W \circ X \approx W \circ \hat{X} = W \circ (UV^T),$$

where $W$ represents the weighting matrix and $\circ$ represents the Hadamard product.
2.1.3 Matrix tri-factorization

In general, matrices can be factorized into any number of factors. Usually, the number of factors is not high, due to the increase in parameter count, model complexity and un-interpretable. Apart from the standard two-factorization, the tri-factorization models are commonly used in machine learning. In this three-factor factorization an input matrix $X \in \mathbb{R}^{n \times m}$ is factorized into three matrices $U \in \mathbb{R}^{n \times k_1}$, $S \in \mathbb{R}^{k_1 \times k_2}$ and $V \in \mathbb{R}^{m \times k_2}$ so that:

$$X \approx USV^T.$$ 

The factorization into three matrices provides an additional degree of freedom, which results in better clustering \cite{16} and better data fusion \cite{10}. Additionally, the middle factor $S$ can be used for controlling the sparsity, smoothness or filtering of the factorization process \cite{21}.

2.1.4 Solving the optimization problem

In order to compute the factors $U$ and $V$ one must solve the optimization problem from Equation 2.1 (or similar one, depending on constraints, weights and number of factors). These kinds of problems are usually solved by using the gradient descent methods. For different constraints, more general projected gradient descent can be used, which (after a gradient step) projects solutions to the feasible (constrained) regions \cite{22}. If the learning rate in gradient descent is made dynamic (not constant for all training points), we can construct a set of multiplicative update rules (MUR) for the factorization. Lee and Seung \cite{23} developed those for NMF, which guarantee a fast and monotonic convergence.

Another speedup is possible by using the stochastic gradient descent, where we minimize the objective function by moving into directions of gradients of individual instances of the training matrix. This approach was especially popular in recommender systems \cite{7,8} in which number of observed ratings is relatively low compared to the overall size of the input matrix.

The second common factorization method in recommender systems is the alternating least squares (ALS), where by fixing one factor the optimization problem becomes convex and can thus be solved optimally. The procedure then rotates between factors, while fixing one and optimizing the other. ALS is also naturally parallelizable and can
converge faster when input matrices are dense \cite{8, 24}.

The other standard distributed algorithm (apart from SGD \cite{25} and ALS \cite{24}) is the coordinate descent factorization \cite{26} and block-coordinate descent with block-wise updates \cite{27, 28}.

### 2.2 Matrix factorization in recommender systems

Recommender systems are tools that assist and augment our decision making in everyday life \cite{29}. They are large-scale machine learning and knowledge discovery tools aimed at providing personalized recommendations to customers based on their preferences and needs. Nowadays, they are essential both in business and the scientific world. They need to handle large quantities of diverse and very sparse data in a matter of seconds, which makes them a suitable problem or subject for various matrix factorization techniques. Matrix factorization features simplicity in modeling of explicit data and ability to incorporate additional information, such as implicit feedback, biases, temporal effects and confidence levels \cite{8}.

Recommender systems have emerged as an independent research area in the mid-1990’s \cite{30, 31} and have since been active both in the scientific world and in industry and commerce \cite{32}. Although there has been a lot of success and progress in the recent years \cite{9}, new types and larger amounts of obtainable information are leading an everlastign adaptation and evolution of recommender systems. The most common recommender systems fall into three categories \cite{30, 33}: Content-based where the user is recommended items similar to the ones he preferred in the past, Collaborative filtering where the user is recommended items that people with similar tastes and preferences liked in the past and Hybrid approaches that combine collaborative and content-based methods. The most successful and used are the collaborative filtering methods which are further classified into memory-based that make predictions on the entire stored collection of previous ratings and model-based that rather use this collection to train a model, which is then used to make new predictions \cite{34}. The most successful are the latter, the model-based collaborative filtering methods, based on matrix factorization (latent factor models). They try to explain ratings by characterizing items and users with factors inferred from the ratings patterns. These factors might represent obvious dimensions like movie or music genres, or something completely uninterpretable to us.

Matrix factorization has a rich history in recommender systems. Sarwar et al. \cite{35}...
proposed the usage of singular value decomposition for reducing the dimensionality of the data and thus making collaborative filtering more scalable and accurate. Their idea was based on latent semantic indexing [3], where SVD is used to tackle sparsity (and reduced coverage), scalability, synonymy and polysemy. The main feature of the SVD is the reduction of the dimensionality of the product space, making it more dense and less noisy from which latent associations between users and items can be found more easily.

However, the breakthrough happened in October 2006 within the Netflix Prize competition [36]. The competition has demonstrated [37] how matrix factorization can be remarkably efficient when dealing with large and sparse datasets, which is often the case in the real-world. A popular matrix factorization model was proposed (the regularized matrix factorization - RMF) [7], and used as basis for other further solutions [8, 9]. RMF uses stochastic gradient descent to directly model only the observed ratings and avoids overfitting through regularization. The Netflix Prize problem became a well-known benchmark dataset and also inspired many others in improving recommender systems [38, 39].

In contrast to latent factor models and SVD, where the recommendation task can be looked at as a matrix completion problem, matrix factorization methods can also be used as a neighborhood method, due to their powerful clustering features [16]. One such example is the Chen’s et al. [40] orthogonal nonnegative matrix tri-factorization for collaborative filtering.

### 2.2.1 Evaluating recommender systems

Evaluating and comparing recommender algorithms is inherently difficult [41], since algorithm performance depends on the properties of the datasets used for learning and evaluation. The datasets may contain a varying number of users and items, rating sparsity, different rating scales and other unique data properties. The second reason for difficult evaluation is that goals for evaluation can differ, although the bottom-line measure of recommender system success should be the user satisfaction (which is not trivial to determine and evaluate) [41].

In general, optimizing the optimization function (loss error) does not guarantee the optimal results for our problem (best predictions). In order to avoid overfitting and to choose the best model one must apply suitable factorization techniques with constraints and regularization alongside choosing suitable evaluation techniques and measures.

The most common goal of recommender systems is to predict ratings that particular
users would give to particular items. For evaluating these predictions we commonly use the predictive accuracy metrics such as root mean squared error (RMSE) or mean absolute error (MAE) on a hold out set (we keep a part of the training matrix as a test set).

The other goal is to present to users their personalized list of recommendations, typically in an ordered list. This becomes a problem of ranking, where we want to predict the correct order of a set of items for a particular user (usually we bind this problem to a small subset of N items, which becomes a problem of topN recommendations). Such problems can be evaluated using the coverage measures such as precision (defined as the ratio of selected relevant items to number of all selected items), recall (defined as the ratio of selected relevant items to total number of all known relevant items) and the $F_1$ score (the harmonic mean between precision and recall).

Alternative goals for recommender systems include novelty, diversity and serendipity [41, 42] which instead focus on items which the user might not have otherwise discovered if he was recommended only similar things he (or similar users) liked in the past.

### 2.3 Incremental matrix factorization

The tremendous growth of data in recent years poses many challenges for incremental algorithms, such as recommender systems, the most important being the ability to produce large numbers of high quality predictions for millions of customers in real-time [13]. Although matrix factorization techniques are highly accurate and scalable, some of them require a full batch training process on a static dataset, a precondition, that can be rarely satisfied in real-world applications [43]. Incremental models do not suffer from this requirement and are a natural choice for modeling modern e-commerce systems, where huge amounts of user feedback are collected constantly. An efficient incremental model that learns upon data streams should therefore require: a small constant time for updating (per data instance), a small and constant amount of memory, it could learn on a single scan over the data, should produce a usable model at every stage of learning and have an ability to deal with concept drifts and other temporal effects in the data [44].

One of the first uses of incremental matrix factorization was in latent semantic indexing [3], where models were incrementally computed using the singular value decomposition (SVD) while maintaining a fairly good approximation of the static model. Sarwar et al. [13] extended that idea to create a highly scalable recommender system where result-
ing SVD model sacrifices orthogonality for performance gain. However, orthogonality of the SVD can still be preserved using update [45] or hybrid [46] methods.

Nonnegative matrix factorization is one of those algorithms that require a batch training process. In most cases, the training matrix is very large, which leads to expensive computation. Furthermore, addition of new elements requires recomputation and repetitive learning, which greatly restricts the practical application of NMF [47]. That is why several incremental methods for nonnegative matrix factorization were proposed, such as INMF [47], which iteratively learns separate blocks of factor matrix and can thus update the factors only locally to avoid recomputation, or ONMF [48] where initial factors are computed the regular way and then updated using the specially derived rules. In similar fashion, Bucak and Günsel [14] proposed their version of the incremental nonnegative matrix factorization (INMF) for background modeling in video surveillance and clustering. INMF’s main principle assumes that new samples have a lesser effect on the optimality of the reconstruction of the original matrix. Their method is highly scalable, controls example contribution and preserves nonnegativity constraints which enables and preserves the intuitive parts-based representation. The INMF model is suitable for online processing of huge data sets and also features memorylessness (forgetting mechanism), which proves to be very convenient when modeling dynamic content. Chen and Candan [49] extended this approach to group incremental nonnegative matrix factorization, which speeds up the factorization and update process by partitioning the original data matrices into many submatrices (or multiple data streams) and then combines their factorized subfactors into the final factor model. Furthermore, maintaining and incrementally updating just partitioned subfactors is much easier and faster.

On the other hand, matrix factorization approaches that use gradient or stochastic gradient descent (SGD) can be easily modified to work in on-line fashion as they are naturally incrementable [43, 50–52]. The main idea behind it is to update the model with additional pass of the same algorithm using the newly acquired data. Further speed-ups can be achieved by considering the fact that new examples only change a few local features and thus do not drastically alter the global model and can sometimes even be omitted if the change is not significant enough. Rendle and Schmidt-Thieme [19] used this principle with kernel RMF for updating a large-scale non-linear recommender system.
2.3.1 Evaluation of incremental methods

While the research area of data streams and incremental methods is getting lots of notice, there are still many open issues. One such is the problematic evaluation of incremental methods [53, 54]. Not only is the data generated by non-stationary distributions and has a continuous flow, the models also evolve and change over time. Furthermore, with incremental learning (on data streams) new challenges arise that we need to address: infeasible amounts of data, concept drifts and other gradual changes in data distribution, uneven update frequency (of one or multiple data streams), etc.

Prequential analysis

In data stream context, where data is potentially infinite and models evolve over time, cross-validation and other sampling strategies are not applicable [53]. The train and the test set are ordered (by time) and have a specific structure. Typically, the predictor is trained on a small subset of the data stream, while the rest of the stream is used as a test set – predicting the future. Alongside, newly arrived data can be used for updating and adapting the predictor.

In order to evaluate the learning model in a streaming context, one can use either a (static or dynamic) holdout test set and test the learner on regular time intervals or use the predictive sequential (prequential) analysis, where the prediction error is computed as the accumulated sum of a loss function between the prediction and observed values along the data stream [53]. Because prequential estimator is pessimistic (it is over-influenced by the error history), forgetting mechanisms like fading factors can be added for more relevant evaluation. The prequential error using fading factors is defined as:

$$E_i = \frac{S_i}{N_i} = \frac{L_i + \alpha S_{i-1}}{n_i + \alpha N_{i-1}},$$

where $\alpha$ is the fading factor, $L_i$ is the loss function for the example $i$, $n_i$ is the number of examples used to compute the $L_i$ and $N_i$ is the fading increment (the number of examples seen thus far).

Comparison of streaming algorithms

Comparing two streaming algorithms is not trivial. The comparison (for example of a loss function) can differ in different parts of the stream, while the overall average may not be representative enough (for instance, one algorithm could experience a bad start, while being superior in latter stages). One solution is comparison using the $Q$ statistic [55]:

$$Q = \frac{\sum_{i=1}^{n} (L_i - \bar{L})^2}{\sum_{i=1}^{n} (L_i - \bar{L})^2 / (n-1)},$$
where $S^A_i$ and $S^B_i$ are the sequences of the prequential accumulated loss for algorithms A and B. The sequence of $Q_i(A, B)$ shows relative performance of both models through time alongside with the strength of that difference (value of the $Q_i(A, B)$ for a particular timestep $i$). However, $Q$ statistic experiences distortion from long term influences, which can be alleviated by considering fading factors:

$$Q^\alpha_i(A, B) = log\left( \frac{L_i(A) + \alpha S^A_{i-1}}{L_i(B) + \alpha S^B_{i-1}} \right).$$

### 2.4 Data Fusion

Additional knowledge can be extracted from heterogeneous datasets of different object types. Data fusion approaches try to discover the interaction information between different object types in data in order to boost prediction accuracy. Such collectively obtained information therefore has a greater benefit than what would have been derived from each of its contributing parts [56].

Pavlidis et al. [57] combined heterogeneous data using three conceptually different approaches:\textit{ early integration}, where data sources are firstly combined into a single data source which is then used to fit the model, \textit{intermediate integration}, where a model internally combines different data sources into a single prediction and \textit{late integration}, where multiple instances of the same or modified model are trained each on their own data source, while their results are combined into a single prediction (like an ensemble). Both early and late integration are model-unspecific and do not require additional modifications of the learning process, while intermediate integration requires a problem specific model, which can simultaneously learn from multiple data sources at once. This approach often results in a superior modeling of heterogeneous data [10, 57]. One such example of the intermediate data integration can be constructed using the SVM (support vector machines), if we sum up two kernels (each constructed from its own data source) and then use this combined kernel for predicting [57]. More generally, we can use a weighted linear combination of multiple kernels, each constructed from a different data source, which yields a powerful data integration tool, called multiple kernel learning [58]. The other example that utilizes the advantages of intermediate data integration is the collective (simultaneous) matrix factorization.
2.4.1 Collective matrix factorization

Collective matrix factorization addresses the problem of simultaneously factorizing multiple matrices from related data sources \([15, 59]\). Typically, these factorizations use shared factors for each object type that is shared among relations (input matrices). This type of factorization can either be multi-relational or multi-object type, based on the type of the association between input relations \([11]\).

In multi-relational collective factorization, one object type is shared (fixed) across all relations \([59, 60]\). Consider \(l\) object types \(\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_l\) and \(r\) data sources \(X_1, X_2, \ldots, X_r \in \mathbb{R}^{n_1 \times m_1}, \ldots, \mathbb{R}^{n_r \times m_r}\), where each data source relates two object types, object type \(\mathcal{E}_1\) with one of the rest. Matrices \(X_i\) are then factorized into two matrices \(U \in \mathbb{R}^{n \times k}\) and \(V_1, V_2, \ldots, V_r \in \mathbb{R}^{m_1 \times k}, \ldots, m_r \times k\) (respectively for each data source) so that:

\[
X_1, X_2, \ldots, X_r \approx UV_1^T, UV_2^T, \ldots, UV_r^T,
\]

where factor matrix \(U\) is shared across all factorizations and factor matrix \(V_i\) represents the latent factor space for the data source \(X_i\). Just like normal matrix factorization, this approach allows for additional constraints, such as nonnegativity \([61]\) or orthogonality \([62]\).

However, this basic collective factorization system is not flexible enough and cannot include data sources where the fixed object type (e.g. \(\mathcal{E}_1\) from the above example) is not present. In contrast, multi-object type collective factorization can model heterogeneous data sources of any combination of object types \([10, 12, 63]\) and model both inter-type relationships (shared relationships between objects of different type) and intra-type relationships (shared relationships between objects of the same type) between objects. This can be achieved using collective matrix tri-factorization:

\[
\forall X_{ij} \approx U_i S_{ij} U_j^T,
\]

where \(X_{ij}\) represents the data source which relates object types \(\mathcal{E}_i\) and \(\mathcal{E}_j\), while factor matrices \(U_i\) and \(U_j\) represent their latent representations. Same factors are always used for the same object type (\(U_i\) for object type \(\mathcal{E}_i\)). This kind of a system can be successfully co-clustered by solving multiple matrix factorization problems at once, which is equivalent to solving just one symmetric matrix tri-factorization problem \([12, 63]\):

\[
X \approx USU^T,
\]
where block matrix $X$ represents a collection of data sources $X_{ij}$:

$$X = \begin{bmatrix} 0 & X_{12} & \cdots & X_{1r} \\ X_{21} & 0 & \cdots & X_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ X_{r1} & X_{r2} & \cdots & 0 \end{bmatrix}$$

and block matrix $U = \text{Diag}(U_1, \ldots, U_r)$ represents the factors. Here, relations of the same type ($X_{ij}$) are modeled through constraint (penalty) matrices $\theta_{ij}$, which represent pairwise constraints on the same object type, i.e., must-link and cannot-link constraints. The objective function of this kind of factorization system is usually minimized using the quadratic loss:

$$\|X - USU^T\|_F^2 + \text{tr}(U^T\theta U).$$

Žitnik and Zupan [10, 11] further generalized this factorization framework, allowing for modeling of multi-relational data with missing or asymmetric relations ($X_{ij} \neq X_{ji}$). This is crucial for solving problems, where we do not have all information available as optimizing over non-existing relations can distort inferred factors (which leads to lower prediction accuracy). Their approach factorizes every relation separately, by minimizing the following objective function:

$$\sum_{X_{ij}} \|X_{ij} - U_{ij}S_{ij}U_{ij}^T\|_F^2 + \sum_r \text{tr}(U^T\theta^{(r)} U).$$

### 2.4.2 Data Fusion in recommender systems

The abundance of data and incorporation of data fusion can improve the quality of recommender systems which typically rely only on a single data source of user/item ratings. Apart from the explicit ratings, typical recommender problems also include many additional implicit information, such as user behavior and actions (clicks, watch times, etc.), additional item information (tags, genres, etc.) and contextual information (demographics, social relationships, time of watching, etc.) [64–67]. Integrating this “rich” side information can increase the prediction accuracy of the base relation of user/item ratings. Moreover, inclusion of additional data also alleviates the fundamental problem of recommender systems – the cold start [68, 69].
Implicit data, such as search patterns, purchase histories, etc., does not require users to explicitly show their preferences. Usually this kind of data is more abundant and can be mined either separately from explicit data [70] or jointly via data fusion in order to boost recommendation accuracy [71]. With collective matrix factorization, implicit data can be encoded into special feature-matrices (of user and item features) and then simultaneously factorized using the same latent factors for users and items (the multirelational learning) [71–73].

Apart from implicit data, context is often included in recommender systems. The group of algorithms that try to integrate context are usually referred to as context-aware recommender systems (CARS) [65, 74]. CARS use background information about the situation in which the rating event happened (e.g., demographic and social information, current location, time of a rating, current mood of the user, etc.) to increase the prediction accuracy. They are usually implemented with various matrix factorization and tensor decomposition techniques [75–78].

Yoo and Choi [69] showed that matrix tri-factorization approach outperforms the basic two-factorization approach in modeling multiple data sources in collaborative filtering, due to the ability to capture more complex relations. They simulated a (user and item) cold-start problem on a MovieLens dataset (a well known movie recommendation problem) and showed how integration of additional data (demographic information and genre information) boosts the prediction accuracy of movie ratings.

Matrix tri-factorization further allows for capturing heterogeneous user feedback and combining data of different scales and different types. Using probabilistic matrix factorization Pan and Yang [79] successfully combined a matrix with real values (movie ratings) with auxiliary matrices of binary values (likes/dislikes). Their method surpassed similar approaches that use the basic collective two-factorization [80, 81].

The other way of modeling heterogeneous recommendation data is to represent the whole collection with a tensor. Then, various tensor decomposition techniques can be applied to solve this problem [76, 77, 82]. However, tensor decomposition suffers from large model complexity and extreme sparsity, making it practically infeasible when dealing with high dimension spaces. In addition, many tensor factorization algorithms struggle with too heterogeneous data (with lots of different data distributions) and cannot model missing or asymmetric relation between dimensions.
The vast majority of current recommender systems are still designed for specific homogeneous problem domains. The successful implementation of data fusion in recommender systems would therefore make it possible to model rich heterogeneous data sources of different problem domains with abundant user information and context (e.g. user social network information, purchase history, web history, etc.). This way of combining and integrating multiple sources of information would mitigate the cold-start problem and lead to a higher prediction accuracy and better personalisation. Examples from real-life recommender problems with rich heterogeneous side information include: movie recommendations (e.g. Netflix\(^1\), MovieLens\(^2\), IMDb\(^3\)), music recommendations (e.g. Last.fm\(^4\), Yahoo music\(^5\), Amazon music\(^6\)), restaurant recommendations (e.g. Yelp\(^7\)) and many more. In addition to combining the ratings from different user groups (from different domains), many domains also include side information such as movie and song genres, artists and actors, film directors, year of release, restaurant location, etc., which can be used to further improve the quality of recommendations in various aspects (accuracy, novelty, serendipity and user personalization).

\(^1\)https://academictorrents.com/details/9b1383dc4d60676b773e2cd6de5542ce9a
\(^2\)https://grouplens.org/datasets/movielens/
\(^3\)https://www.imdb.com/interfaces/
\(^4\)https://www.last.fm/en/api/downloads
\(^5\)https://webscope.sandbox.yahoo.com/catalog.php?datatype=r
\(^6\)http://jmcauley.ucsd.edu/data/amazon/
\(^7\)https://www.yelp.com/dataset/challenge
Generating inter-dependent data streams
We have already mentioned how Big Data is bringing new challenges to the fields of data analytics and machine learning. However, the data availability is not universal. There exist several domains and problems where the amount of available and labeled data is scarce. This is due to several reasons, e.g. rarity of certain events (rare diseases, loan defaults, machine failures, etc.), expensive data collection (large physics or pharmacological experiments), privacy concerns (medical or genetic data) or business secrets (Amazon purchase histories, etc.).

Synthetic data generators (or artificial data generators) can help alleviate the problem of data scarcity by generating large amounts of data points in a controlled fashion. This additionally generated data helps us in different stages of the development of our algorithms. Additional data in the training phase can lead to better models and can help in parameter tuning, while having a larger test dataset can result in more rigorous evaluation and better comparison of our models (benchmark datasets). Moreover, synthetic data generators can generate specialized and test-specific datasets in order to test the algorithm’s robustness on specific edge cases or rare events (that almost never happens in real-life).

Lastly, synthetic data generators can help us create temporal effects in data, from cyclic and seasonal effects (for instance, seasonal weather or daily electricity consumption) to abrupt changes, i.e. concept drifts (sudden or gradual change in the data distribution). In this way we can create synthetic data streams that can be used to develop and evaluate various incremental and streaming algorithms.

In order to properly evaluate machine learning algorithms such as recommender systems, we need many different and specialized datasets, with varying numbers of users and items, rating sparsity, etc. However, the amount and diversity of publicly available data remains limited [83]. This problem gets worse when we try to collect multiple related datasets from the same domain with meaningful connections and temporal dependencies between them.

In this chapter we present a novel synthetic Generator of Inter-dependent Data Streams – GIDS, capable of generating multiple temporal and inter-dependent synthetic datasets of relational data. The generator is able to simulate a collection of time-changing data streams, helping to effectively evaluate a variety of recommender systems, data fusion algorithms and other incremental algorithms. GIDS works by simulating multiple sets of clusters (groups of objects) of different object types and connections (relations) between those clusters. In this way, GIDS mimics real-life problems where one can
Generating inter-dependent data streams

observe such structures; analogous to collaborative filtering, where users receive recommendations for items that users with similar taste and preferences (e.g., users within the same cluster) liked in the past. The evaluation using recommender and data fusion algorithms showed that our generator successfully mimics real datasets in terms of statistical data properties and achieved predictive performance.

3.1 Background

Machine learning problems often face a lack of sufficient data. This can arise due to the natural rarity of the data itself, difficult or expensive experiments, privacy concerns, uneven distribution of samples, etc. Several synthetic data generators have already been proposed for various problems in data mining and machine learning, such as: generating transaction data [84], filling databases [85], clustering [86], bioinformatics [87] and other general purpose data generators [88, 89].

For recommender system evaluation only a few synthetic data generators were proposed. Tso and Schmidt-Thieme [83, 90] introduced their SDG that generates data with additional content information for evaluating attribute-aware recommender systems. It works by assigning users and items into clusters and sampling attributes/ratings using that cluster information and connections between clusters. In contrast to our approach, their SDG allows only for one user/item relation and produces only static datasets with binary ratings (0/1). Similar to their method, many other approaches can simulate and generate additional side contextual information for evaluating context-aware recommender systems [91, 92]. Context-aware recommender systems [65] can boost their performance when using this additional information (context), which is often difficult to obtain in real-life problems. These approaches also produce only static datasets.

To generate time-dependent data Antulov-Fantulin et al. [93] proposed a generator for synthetic sequential data. Their approach produces clickstreams (sequences of items, such as visited web pages or liked movies) for evaluating recommender systems by simulating a random walk on a graph of all possible sequences. Similar to clickstreams are various sensor and time series data. Alzantot et al. [94] proposed a deep learning approach that uses recurrent neural networks to generate semi-artificial sensor data. Shamshad et al. [95] used Markov chain models for synthetic generation of wind speed time series.

Another way to incorporate multi-dimensional dependencies or time-changing behavior is to generate synthetic tensor data. Fanaee-T and Gama proposed a tensor gen-
erator SimTensor [96] capable of generating a large variety of synthetic tensors. The biggest advantage of SimTensor is its ability to generate temporal tensors and integrate several effects from periodic waves, seasonal events to streaming structures, alongside with anomalies, noise and sparsity. However, SimTensor works by randomly generating tensor decompositions and then backtracking to produce the “original” tensor data. In this way, the rating structure remains hidden and not controllable.

When dealing with time-dependent datasets or data streams special attention should be given to the problem of concept drifts. While it is very hard to determine the “ground truth” for the occurrence of concept drifts in real data, it is very easy to simulate them within artificial data. There exist several standard benchmarks like SEA concepts [97] and STAGGER [98] that can simulate different types of concept drifts.

3.2 Generating multiple inter-dependent data streams

In this section we present our novel synthetic data generator (GIDS), aimed at providing a collection of inter-connected data streams. GIDS generates data in a relational form (matrices with timestamps) suitable for evaluation of various streaming algorithms, such as on-line recommender systems.

When designing our data stream generator the main goal was to mimic relevant properties that are present in real datasets:

- **realistic relations** between groups of different entities (e.g., a realistic pattern of user ratings),
- **hidden correlations** between multiple datasets (a common dimension in a set of multiple relations, e.g., when same users rate different entities: movies, songs, restaurants, etc.),
- **real data distribution** (distributions of rating values and number of ratings per row and column match closely to those found in real datasets),
- **variable sparsity and rating scales** (some problems can have plentiful wide-ranging ratings, while other can have very few ratings that are binary),
- **temporal dependencies** in data and relevant data streams (produced datasets can be simulated as a meaningful data stream with a temporal dimension),
- **relevant concept changes** in data (ratings can reflect meaningful concept changes over time),
- **realistic growth** of new entities (e.g., the growth of new users is steady).

Our second goal was to make a generator with a variety of tunable parameters in order to generate distinct and domain-specific datasets to evaluate specific properties of algorithms.

In the next sections we describe the main parts of our algorithm, beginning with the generation of a single relational binary dataset, followed by formation of rating values, adding time dependency, and finally, generalization to multiple datasets and data streams.

### 3.2.1 Cluster structure

We begin by introducing the cluster structure of the data generator. At the start we will focus only on one relation that relates two sets (clusters) of objects of two different object types. Throughout this chapter we will refer to the first object type as “users”, the second object type as “items” and the magnitudes of relations as “ratings”. This serves as an intuitive example from the typical recommender systems’ domains (although this cluster structure can relate to various other domains).

The cluster structure of our synthetic data generator is inspired by the work of Tso and Schmidt-Thieme [83]. We want to simulate scenarios where groups of similar users rate groups of similar items in the same manner. The main principle behind our approach is to assign objects from the relating pair of object types (users and items) into different clusters. Afterwards, a fixed number of ratings is sampled (with no replacement), based on a probability that a cluster of a particular user is connected to a cluster of a particular item. This results in generated triplets of \{user, item, rating\} which can be represented with a rating matrix where each non-zero cell describes a relation between a particular user (row) and item (column).

We begin by choosing two probability distribution functions (PDF) for user and item clusters (e.g. $\beta$ distribution), denoted as $PDF_{users}$ and $PDF_{items}$, respectively. These PDFs determine how users and items are assigned to users and items clusters. Secondly, we chose one random PDF (among the same family of distributions) for each user cluster. These determine **inter-cluster weights** that define how likely a particular user cluster is
connected to item clusters. A schematic of such cluster structure is illustrated on Figure 3.1.

![Illustrative example of the cluster structure](Figure 3.1)

User and item cluster structure of the synthetic data. Users (represented with squares) and items (represented with circles) are sampled into clusters that form a bipartite weighted graph. Weights are determined by inter-cluster probability distribution functions.

After the cluster structure is determined, the sampling of users and items into clusters is performed using the chosen PDFs. For each user/item combination, i.e., each value of the final rating matrix, the probability of a relation (e.g., rating) is determined by the corresponding inter-cluster weight. These probabilities are then weighted with the size of the user cluster to ensure that clusters are being fairly represented. Finally, probabilities are normalized to sum up to 1, producing the “probability matrix” for this cluster structure. After obtaining this probability matrix, a sample of any size can be drawn (without replacement) to generate a data source. The product of this sampling is a binary relation matrix, where 1 represents a relation between some user (row) and some item (column).

**Illustrative example of the cluster structure**  
For example, let us choose two user clusters and two item clusters with uniform cluster PDF and inter-cluster weights \{0.3, 0.7\} for the first user cluster and \{0.6, 0.4\} for the second. The probability that user \(u_1\) rated item \(i_1\) in this scenario is determined by the sampling of users and items into clusters. With uniform PDFs for both clusters there are four possible cluster arrangements for \(u_1\) and \(i_1\) each with probability \(\frac{1}{4}\) as shown in Figure 3.2. If the user and item are in the first cluster arrangement, the probability of a relation is then 0.3, if they are in second arrangement the probability is 0.7 and so on. The final distribution of all possible samplings for this example is: \(P((u_1, i_1) = 1) \sim (\frac{3}{4} \frac{7}{4} \frac{6}{4} \frac{4}{4}).\)
3.2.2 Magnitudes of relations - generating rating values

In addition to determining relations between user and item pairs we also need to determine the magnitude of their relations, which defines final rating values. Firstly, for every user cluster we choose one PDF (over the domain of the rating scale) among the same family of distributions (e.g. the normal distribution). This PDF ensures that all users from the same clusters vote in the same manner. For example, if we were to generate movie ratings, we could choose a normal distribution model for every user cluster and vary the mean and variance. This could for instance correspond to one cluster of users that rate very generously (high mean and low variance) and maybe another, where users do not really prefer any movie and rate in a very random and unpredictable manner (high variance).

When all distribution models are chosen, we sample a rating for every user/item pair, always choosing the user's cluster rating PDF. This rating is then appended to the user/item pair, thus producing the final triplet \( \{\text{user}, \text{item}, \text{rating}\} \), that defines a cell in a real-number matrix.

3.2.3 Time dependency in data

To add time-dependency to sampled data we can supplement every rating with a timestamp \( t \) (a natural number that indicates when the rating occurred in a sequence of ratings). We begin by choosing a fixed timeframe of length \( T \), so that \( \forall t \in [0, T] \). Then we choose a timestamp \( o_u \in [0, T] \) for every user that has at least one rating in order to define the initial occurrence of the user (when the user first appeared in the system). The same process is afterwards repeated for every item (when the item first appeared in the system).
the system), producing timestamps \( o_i \in [0, T] \). These timestamps can be sampled using any distribution, mimicking different trends (high or low initial number of users, exponential growth of new items, etc.).

The final timestamp of the rating is then generated by sampling a random integer from \([M, T]\) (again using arbitrary distribution for simulating different trends), where \( M = \max(o_u, o_i) \) of the corresponding user and item. By choosing the maximum of the initial occurrences of the user and the item as the lower bound for the timestamp, we ensure that a particular user did not rate a particular item before either of them entered into our system. By doing this we want to realistically simulate scenarios where an abundance of new users and items enter into our system and become available only after a particular point in time.

It should be noted that with this general approach we are only able to produce numerical (and continuous) timestamps. However, categorical and contextual temporal variables may be induced from these timestamps by manually transforming them (e.g. dividing timestamps into days of the week).

### 3.2.4 Generating concept drifts and creating data streams

To generate artificial concept drifts in data we modify user ratings by applying different concept drift functions to selected parts of the stream. We apply these functions after all ratings are sampled and assigned their timestamps \( t \).

For example, to make a simple sudden/abrupt concept drift we can choose a certain point in time \( D \) and multiply every rating with timestamps \( t \geq D \) by some factor (or applying a simple linear function). Similarly, we can simulate a recurring concept drift by multiplying every rating with \( \sin(t) \) or \( \cos(t) \), or modifying every rating with a timestamp that is divisible by a selected number (e.g. a certain day in the week), etc. Examples of data streams with artificial concept drifts can be seen in Figure 3.3 (the figure shows four data streams, one with no concept drift and three with sudden and recurring concept drifts).

This approach further allows us to create complex concept drifts by simulating different user and item characteristics through time. For example, we can create a concept drift that simulates a scenario where newer users rate more conservatively than older users, or a scenario where every item that is older than some timestamp is obsolete and therefore rated very poorly. Additionally, we can use this concept drift function to simulate a predetermined flow of values in order to mimic selected real-life data streams. In general,
any function of a timestamp, rating or the object time occurrence can be used to generate the desired concept drift.

When all the above steps are complete, we are left with our final dataset. To further simulate a data stream, we only need to sort all ratings by their timestamps.

*Figure 3.3* Examples of different concept drift types in data streams. Data streams are represented as values – ratings (y-axis) through time (x-axis). Each figure also includes the information about the concept drift function.
3.2.5 Generating multiple inter-dependent data streams simultaneously

This approach can be easily generalized to generate multiple datasets that share mutual (hidden) information. These datasets then represent multi-relational problems and be used for effective evaluation of data fusion approaches.

Consider a system of \( k \) object types \( \varepsilon = \{ \varepsilon_1, ..., \varepsilon_k \} \) and \( l \) relations (which can be asymmetric or missing \( \Rightarrow l \leq k^2 \)), each relating a pair of object types \( \varepsilon_i \) and \( \varepsilon_j \), as shown on Figure 3.4.

Instead of having only two sets of clusters (for users and items), we generate clusters and their PDFs for each of the data object types \( \varepsilon \) that are in our system. In a similar fashion, instead of having one set of inter-cluster connectivity weights, we generate a set of PDFs for each of the \( l \) relations that are present in the system. Because we sample objects into clusters only once, all relations use exactly the same sets of objects and clusters, thus creating and enhancing hidden connections between different objects.

The procedure yields \( l \) probability matrices from which we can sample the final rating matrices - datasets. Each rating matrix can have a different size and sparsity, determined by the number of required samples for that particular relation. Exact rating values are sampled same as before using different rating distributions of different object clusters. Because relations that share the same object type (common dimension) have the same object clusters, they also share the same rating distributions, which further reinforces hidden connections and rating behaviors.
3.2.6 Generating attributes and side information

GIDS simulates relations between object types and does not model any side information, such as attributes or context. However, the generation of side information can still be achieved by artificially creating relations between objects and possible attributes.

For example, if we want to simulate features for users (categorical or numerical) we can create special users/features relations. We can construct the feature object type by creating an object for every possible feature value (binarization). Then, we can distribute these objects into clusters (either one feature per cluster, or multiple features per cluster). Finally, we can sample from this relation, which would result in user/feature-value pairs. Optionally, we can apply filters to ensure one value per individual feature.

Figure 3.4
Cluster structure of the synthetic data. This figure represents \( k \) object types \( (\varepsilon_1, \ldots, \varepsilon_k) \) each with its own set of clusters and objects in it (squares, circles and triangles), with connections between those clusters. Only non-zero weights (positive probabilities) are shown.
3.2.7 GIDS dataset generator

The input to our generator is a list of relations $R$. Each relation $r \in R$ is defined by two object types $\varepsilon_i$ and $\varepsilon_j$, each with its number of objects ($N_i$ or $N_j$) and own PDFs ($\phi_i$ or $\phi_j$), determining how objects are distributed into object clusters. All relations also contain a sample size $N_r$ and sets of PDFs $\phi_{i,j}$, determining inter-cluster weights, which connect clusters of $\varepsilon_i$ to clusters of $\varepsilon_j$. Additionally, all object types have defined PDFs $\phi_{O_i}$ and $\phi_{V_i}$, which determine the distribution of timestamps and rating values of objects within a particular cluster. The comprehensive list of all parameters along with some guidelines on how to select them can be found in Table 3.1.

The output of our algorithm is a set of datasets $D$, where each dataset is defined by four lists: list of rows (row), list of columns (col), list of values (val) and list of timestamps (ts). Alternatively, a dataset/data stream can be represented as a sorted list of quadruplets $\{\text{row}, \text{col}, \text{val}, \text{ts}\}$, sorted by timestamps $\text{ts}$. The pseudocode of the GIDS synthetic data generator is shown in Algorithms 1-3.

The computational complexity of the proposed generator is (due to the implementation) bounded with $O(nm)$, where $n$ and $m$ represent the number of objects of two largest object types. The bottleneck occurs during the generation and sampling from the probability matrix. However, reasonable sized data streams (such as those found in some real-life problems) can be generated in an order of minutes ($n$ and $m \approx 15000$).
### Table 3.1: Input Parameters for the GIDS Algorithm

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>Examples and Guidelines</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>For each object type $i$:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N_i$ number of objects</td>
<td>number of rows/columns in the rating matrix</td>
<td>domain-dependent</td>
</tr>
<tr>
<td>$N_C$ number of object clusters</td>
<td>indirectly creates different rating structures</td>
<td>similar to rank selection</td>
</tr>
<tr>
<td>$\varphi_i$ object cluster distribution</td>
<td>further controls the rating structure</td>
<td>e.g. $\mathcal{N}(1, 3)$</td>
</tr>
<tr>
<td>$\varphi_O$ object timestamp distribution</td>
<td>controls the growth of objects and cold-start</td>
<td>e.g. $\mathcal{N}(\frac{T}{2}, \sigma^2)$</td>
</tr>
<tr>
<td>$\varphi_V$ rating values distribution</td>
<td>controls the distribution of ratings</td>
<td>e.g. $\mathcal{N}(3, 1)$</td>
</tr>
<tr>
<td><strong>For each relation $r$:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$N_r$ sample size</td>
<td>controls sparsity</td>
<td>e.g. $N_i \times N_j \times 0.05$</td>
</tr>
<tr>
<td>$\varphi_{ij}$ inter-cluster weights</td>
<td>controls the rating structure</td>
<td>e.g. $\mathcal{N}(1, 3)$</td>
</tr>
<tr>
<td>$T$ maximum timestamp</td>
<td>controls the timestamp structure</td>
<td>e.g. 100</td>
</tr>
<tr>
<td>$\varphi_O$ rating timestamps distribution</td>
<td>controls temporal effects</td>
<td>e.g. uniform distribution</td>
</tr>
<tr>
<td>$\varepsilon_r$ value range</td>
<td>provides available rating values</td>
<td>e.g. $[0, 1, 2, 3, 4, 5]$</td>
</tr>
<tr>
<td>$cdf$ concept drift function</td>
<td>controls concept drifts</td>
<td>see Figure 3.3</td>
</tr>
</tbody>
</table>
Algorithm 1: GIDS

**input**: a list of relations R

**output**: a list of rating matrices (datasets) D

# Sample objects into clusters

for each \( \epsilon_i \) in \( \epsilon \) do

\[ C_i = \text{sample}(N_i, \phi_i) \]

end

# Compute probability matrices

for each \( r \) in \( R \) do

\[ \epsilon_i, \epsilon_j = r \]

\[ M = [ ] \]

for each \( ii \) in \( C_i \) do

for each \( jj \) in \( C_j \) do

\[ M[ii, jj] = \phi_{ij}[ii, jj] \* N_{ii}^{-1} \]

# \( N_{ii} \) - size of cluster \( ii \)

end

end

# Sample data matrix

\[ D_r[row, col] = \text{sample}(N_r, M) \]

\[ D_r[ts] = \text{SampleTimestamps()} \]

\[ D_r[val] = \text{SampleRatingValues()} \]

end

return D

Algorithm 2: SampleTimestamps

**input**: a parameter T

**output**: a list of rating timestamps ts

# Sample initial occurrences of objects

for each \( \epsilon_i \) in \( \epsilon \) do

\[ O_i = \text{sample}([0, ..., T], N_i, \phi_O) \]

end

# Sample a timestamp for each rating

for each \( rr, cc, vv \) in \( D_r \) do

\[ ts + = \text{sample}([\max(O_i[rr], O_j[cc]), ..., T], 1, \phi_O) \]

end

return ts
Algorithm 3: SampleRatingValues

input : a concept drift function \( cdf() \)
output : a list of rating values \( val \)

# Sample a value for each rating
for each \( r, c, v, t \) in \( D \), do
  \( v = cdf(t, \text{sample}(s_r, 1, \phi V_i, O_i, O_j)) \)  # \( s_r \) - ratings range of relation \( r \)
  \( \text{val} += v \)
end
return \( \text{val} \)

3.3 Evaluation: Real-life dataset resemblance

Our main goal was to create a synthetic data generator that is able to create multiple interdependent datasets that resemble real-life problems, in order to design a realistic benchmark for algorithm development and testing. To evaluate our approach we designed a special set of experiments that test our objectives (listed in Section 3.2). For comparison we included a state-of-the-art synthetic data generator SimTensor [96], while for the performance baseline we also included a basic synthetic data generator (denoted with Random) that randomly generates quadruplets \( \{user, item, rating, timestamp\} \) using uniform distribution. For the sake of simplicity, in the rest of this paper when we will be referring to GIDS, SimTensor and Random, we will actually be referring to datasets generated by the GIDS, SimTensor and Random data generator.

We begin by trying to generate artificial datasets that mimic properties of real datasets within the MovieLens problem domain [99]. The evaluation itself is done in two parts. In the first part we compare data statistics between GIDS, Random and MovieLens and in the second part we use those same datasets for modeling and observe the difference in recommender system performances. The second part is then further split into three separate evaluations, the basic and static performance test, the test with added temporal context and the test using multiple data sources via data fusion. For an additional comparison we also include two other real-life domains, Yahoo music and Yelp. On both additional domains we have created new versions of the GIDS, SimTensor and Random datasets, all of them optimized for the selected domain.

Our main assumption in data modeling is that recommender algorithms trained on GIDS have a similar prediction performance to those trained on the real datasets, while
algorithms trained on baseline (Random) perform worse.

In the first part of the evaluation we examine the GIDS’ capability to generate datasets with selected target properties. Firstly, we present a way of estimating GIDS’ parameters to generate similar datasets to MovieLens; secondly, we compare data statistics with the assumption of a static environment; and finally, we compare data statistics in the terms of a data stream in order to confirm a realistic time-dependency in data.

### 3.3.1 GIDS parameter estimation and data generation

We decide to imitate a temporally-contiguous subset of the MovieLens20M dataset. We chose the last 200,000 ratings to construct the final dataset (denoted MovieLens). It consists of 3,157 users, 12,914 movies and 200,000 ratings ranging from 0.5 to 5 and containing timestamps (99.5% sparsity). In order to generate data similar to MovieLens, we first need to choose suitable cluster sizes for two object types (users and movies), two object cluster PDFs, one PDF for inter-cluster weights and one family of PDFs for ratings.

Because of the way that data is generated there is no direct correlation between selected parameters (e.g. number of user clusters, mean and variance for the family of rating PDFs, etc.) and observed (evaluated) data descriptives. That is why the optimization of the input parameters is a very hard problem. However, there exist some indirect links between parameters and statistics which (for instance, inter-cluster weights and rating per column distribution). Therefore, we can apply domain knowledge in order to estimate some parameters by hand and narrow the search space for others. This proved to be an effective strategy, which was later demonstrated with our evaluations.

We decided to select 25 user clusters and 100 movie clusters to represent 25 different rating habits and 100 different groups of similarly rated movies. In general, the number of user and item clusters can be arbitrary, though it should be greatly lower than the number of users and items (similar to the problem of the rank selection in matrix factorization). For this experiment we chose these parameters accordingly to the problem domain in order to represent some meaningful structure.

Using spectral biclustering [100] on the MovieLens dataset, we co-clustered rows (users object type) and columns (movies object type) into meaningful clusters. Then, we observed cluster sizes for each object type separately and fit them to the $\beta$ distribution that represented the final object cluster distributions for users and movies. For inter-cluster weights we chose the same distribution as for the movies object cluster. Finally,
we took the distribution of movie ratings and fit it to the normal distribution (N), getting the base mean and variance parameters. Additionally, we have designed a simple concept-drift function that reduces rating values of ratings in the second third of the data stream in order to match the streaming average of MovieLens more closely. The whole procedure resulted in choosing 25 user clusters with $\beta(0.68, 4.8)$ cluster distribution, 100 movie clusters with $\beta(0.38, 3.0)$ cluster distribution, $\beta(0.38, 3.0)$ inter-cluster weights and $N(3.8 – 4.2, 0.5 – 1)$ rating distribution.

SimTensor works by simulating factors of latent spaces, which are then multiplied to generate the final data tensor. The latter is used to extract all of the data relations (by splicing on different dimensions). We used the number of objects and number of object clusters from GIDS as the factor shapes (inputs) for SimTensor. For the Random generator (the baseline SDG) we used the same number of users, movies and ratings as in MovieLens, while all data points $\{user, item, rating, timestamp\}$ were sampled using the uniform distribution.

3.3.2 Comparison between GIDS and MovieLens descriptives in a static environment

We compare GIDS with MovieLens by examining the distribution of rating values and distribution of number of ratings per individual row and column. Using the Kullback–Leibler (KL) divergence and Hellinger distance (HE) we measure how distributions of generated datasets diverge from those of MovieLens. Table 3.2 presents the results. It can be seen that in terms of distributions of the chosen data descriptives, GIDS diverges less from MovieLens than SimTensor and Random. It should be noted that KL divergence does not behave as a distance, so we cannot determine how closer GIDS is to MovieLens than data from other generators. However, we can clearly see that KL divergences of GIDS are always strictly lower (denoted with the boldface). On the other hand, we can use the Hellinger distance to show the small gap (almost zero for the distribution of ratings per column and distribution of values) between the GIDS dataset and MovieLens. Consistently with KL divergence, GIDS achieves the best results among all dataset generators in terms of Hellinger distance as well.

These results are further backed with graphical representations (histograms of the selected distributions) found in Figure 3.5.
Table 3.2
Kullback–Leibler divergences and Hellinger distances of data statistics between GIDS-MovieLens, SimTensor-MovieLens and Random-MovieLens. KL divergence of 0 indicates similarity between two distributions, while divergence of 1 and above indicates totally different distributions. Similarly, Hellinger distance of 0 indicates the similarity between two distributions, while the (maximum) distance of 1 indicates two different distributions.

<table>
<thead>
<tr>
<th>Distribution</th>
<th>GIDS-ML</th>
<th>SimTensor-ML</th>
<th>Random-ML</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>KL</td>
<td>HE</td>
<td>KL</td>
</tr>
<tr>
<td>ratings per row</td>
<td>0.512</td>
<td>0.194</td>
<td>1.672</td>
</tr>
<tr>
<td>ratings per column</td>
<td>0.084</td>
<td>0.035</td>
<td>0.787</td>
</tr>
<tr>
<td>values</td>
<td>0.081</td>
<td>0.0193</td>
<td>0.308</td>
</tr>
</tbody>
</table>

Figure 3.5
MovieLens, GIDS, SimTensor, and Random data descriptives - number of ratings per row, number of ratings per column and number of different rating values.

3.3.3 Comparison between GIDS and MovieLens descriptives in a dynamic environment

Next we evaluate the generated datasets from the data stream perspective and observe how data statistics change over time. Just like in the previous experiment, we firstly examine the number of ratings per row and column. In a real dataset we should expect that the growth of ratings per row and column should not be linear, meaning that same users give ratings in successions over a short period of time (they have a greater number of ratings per row than the linear average). Figures 3.6 and 3.7 confirm that assumption. GIDS has a similar pattern to MovieLens, while SimTensor and Random have linear growth. This is expected from Random, since every new rating is uniformly “assigned” into a row or a column.
Generating inter-dependent data streams

Similarly, we observe the new-user and new-item dynamics in datasets. Since our goal was to mimic a steady growth of new users and items over time, we expected that pattern in the data. Figures 3.8 and 3.9 show the results for GIDS, MovieLens, SimTensor and Random. We can observe the steady growth pattern in GIDS and MovieLens, while Random experiences a sudden jump in the start, considering there is a little chance of “choosing” the same row or column more than once.

Figure 3.6
Average number of ratings per row (user) through time for GIDS, MovieLens, SimTensor and Random.

Figure 3.7
Average number of ratings per column (movie) through time for GIDS, MovieLens, SimTensor and Random.
Finally, we examine the dynamics of rating values through time. Figures 3.10 and 3.11 show the normalized streaming average and variance for all four datasets. We can observe that the baseline dataset Random has a steady mean value and variance (normalized mean and variance are around 0), meaning that there is no change of the rating value distribution over time (no temporal dynamics in rating distribution). The opposite is true for GIDS, SimTensor and MovieLens, where average and variance changes over time. This indicates that the rating value distribution, for the dataset generated by GIDS, changes over time, therefore a temporal component to the rating values is present in the data.
3.4 Evaluation: Data modeling on synthetic datasets

We continue our evaluation of the proposed data generator by observing how recommender systems behave on our generated datasets in terms of the prediction performance.

Note that generating data with the same statistical properties as the desired (real-life) dataset is genuinely a hard problem. Often the optimization of one aspect of the desired properties can distort the others, while on the other hand, the fully optimized data properties would consequently yield the perfect predictor. In the above experiments, we therefore only wanted to show the closely matched patterns rather than the exact values.
In the first part of the experiments we took the MovieLens datasets and used them to train three matrix factorization models, the regularized matrix factorization (RMF) [7, 19], the nonnegative matrix factorization (NMF) [23, 101] and the probabilistic matrix factorization (PMF) [102] alongside a classic content-based model, the k nearest neighbor model (kNN). Additionally, we have generated two more datasets (for GIDS and Random) in order to compare recommender system performances also on two other domains, Yahoo music and Yelp.

In the second part of the experiments we observed if two time-aware recommender systems, kNN with time-decay [103] and kNN with contextual pre-filtering (PRF) [65], can enhance their recommender performance by gaining additional knowledge from time dependencies found in real and generated data.

Finally, we checked if GIDS can generate inter-dependent datasets with some hidden information in order to evaluate data fusion algorithms. For that reason we have generated a completely new set of three datasets for GIDS and Random and used a penalized matrix tri-factorization data fusion approach (DFMF) [10] and remote-average item-based kNN recommender system (RAkNN) [104] in order to observe a potential gain in the model accuracy.

### 3.4.1 Recommender system performance

For the MovieLens problem domain we took the same datasets that we used in previous experiments. These are MovieLens, GIDSML, SĭmTensorML and RandomML datasets, of which each contains 3,157 rows, 12,914 columns and 200,000 ratings ranging from 0.5 to 5 (99.5% sparsity). Next we took the Yahoo music dataset (the pre-sampled small dataset) and used the same procedure as explained in subsection 3.3.1 to generate the GIDS and Random datasets with the same data statistics. In this way we obtained Yahoo, GIDSYH, SĭmTensorYH and RandomYH, of which each dataset contains 15,400 rows, 1,000 columns and 311,704 integer ratings ranging from 1 to 5 (97.97% sparsity). Finally, we took the dataset from the Yelp challenge and manually created a new smaller dataset by choosing only the last 500,000 ratings of restaurants and further decreased the sample size by only choosing users that have rated at least 5 restaurants. On this final dataset we repeated the same process to generate GIDS and Random versions of the dataset. In this way we obtained Yelp, GIDSYL, SĭmTensorYL and RandomYL,
of which each dataset contains 18,153 rows, 27,842 columns and 182,648 integer ratings ranging from 1 to 5 (99.96% sparsity).

For every problem domain (MovieLens, Yahoo and Yelp) we took four corresponding datasets (the real-life dataset and datasets generated by GIDS, SimTensor and Random) and trained the above mentioned recommender systems on them: RMF, NMF, PMF and kNN. For control we added a baseline model (Average) that predicts every new rating with the overall train set average. A 5-fold cross validation was used with 4 performance metrics: root mean-squared error (RMSE), relative root mean-squared error (RRMSE), mean absolute error (MAE) and $F_1$ score. For every fold models were firstly trained on the train set and then evaluated on the test set in terms of the predictive accuracy metrics (RMSE, RRMSE and MAE) - how close can they predict rating values from the test set and in terms of the precision and recall measures (expressed in $F_1$ score) - how well can they predict the set of top 10 relevant items [41].

Tables 3.3 • 3.5 show the performance results of 5 recommender systems trained on the real-life dataset, GIDS, SimTensor and Random for three domains: MovieLens, Yahoo music and Yelp. It should be noted that our goal was not to optimize the parameters for selected models or to achieve as best results as possible, but to show the similarities and differences of the evaluated datasets.

On the MovieLens domain we observe that all models performed well on the real-life dataset, GIDS and SimTensor. We can see that the RMF model could learn more than the baseline (RRMSE < 1). This was also true for the kNN (GIDS) and NMF (SimTensor), which indicates that both data generators can produce datasets suitable for evaluating recommender systems. However, models that were trained on GIDS, experienced overall lower RMSE and MAE values and higher $F_1$ scores, which is closer to the results from the real-life problem. These results were in complete contrast to the baseline (Random), which is expected, due to the fact that the Random datasets are generated using the uniform distribution, making the average models predictions as “best” guesses as possible (every recommendation is therefore completely random). This further demonstrates that simple data generators should not be used for evaluating recommender systems. The similarities of performances between GIDS and MovieLens could indicate that datasets generated by GIDS have a relational structure that is similar to the real-life problems.

The same behavior can also be observed on the Yahoo music domain, where for both GIDS and Yahoo, the observed relative RMSE values were below 1 in all cases (note
that raw RMSE values were higher due to the difficulty of the prediction problem). For SimTensor only RMF and NMF performed better than the baseline, while in the case of the Random dataset performances of all recommender models were again worse than the performance of the Average model. We further draw the same conclusions yet again from the Yelp domain, where in some cases models performed better when trained on datasets generated by GIDS than on the real-life dataset (lower relative RMSE values). This is again the result of the higher difficulty of the real-life problem domain.

Table 3.3

MovieLens - Performance of 5 selected recommender systems on MovieLens, GIDS, SimTensor and Random. Results represent the mean value over 5 folds with standard deviation reported in the brackets. Note that the relative RMSE is normalized with the Average estimator for each dataset separately.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>RMSE</th>
<th>RRMSE</th>
<th>MAE</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovieLens</td>
<td>Average</td>
<td>1.0794 (0.003)</td>
<td>1.0</td>
<td>0.8292 (0.002)</td>
<td>0.6996 (0.0017)</td>
</tr>
<tr>
<td></td>
<td>RMF</td>
<td>0.8790 (0.002)</td>
<td>0.8144 (0.0041)</td>
<td>0.6639 (0.0012)</td>
<td>0.7446 (0.0019)</td>
</tr>
<tr>
<td></td>
<td>NMF</td>
<td>0.9386 (0.002)</td>
<td>0.8769 (0.0003)</td>
<td>0.6916 (0.0015)</td>
<td>0.7341 (0.0017)</td>
</tr>
<tr>
<td></td>
<td>PMF</td>
<td>0.8692 (0.0033)</td>
<td>0.8217 (0.0005)</td>
<td>0.6683 (0.0018)</td>
<td>0.7668 (0.0015)</td>
</tr>
<tr>
<td></td>
<td>kNN</td>
<td>0.9091 (0.0042)</td>
<td>0.8885 (0.0007)</td>
<td>0.7139 (0.0034)</td>
<td>0.7412 (0.0019)</td>
</tr>
<tr>
<td>GIDSML</td>
<td>Average</td>
<td>0.8982 (0.0018)</td>
<td>1.0</td>
<td>0.7029 (0.0031)</td>
<td>0.7800 (0.0018)</td>
</tr>
<tr>
<td></td>
<td>RMF</td>
<td>0.8742 (0.0045)</td>
<td>0.9772 (0.0005)</td>
<td>0.6601 (0.0028)</td>
<td>0.7672 (0.0018)</td>
</tr>
<tr>
<td></td>
<td>NMF</td>
<td>0.9019 (0.0035)</td>
<td>1.0063 (0.0007)</td>
<td>0.6928 (0.0011)</td>
<td>0.7821 (0.0009)</td>
</tr>
<tr>
<td></td>
<td>PMF</td>
<td>0.8982 (0.0033)</td>
<td>1.0002 (0.0030)</td>
<td>0.7029 (0.0024)</td>
<td>0.7795 (0.0010)</td>
</tr>
<tr>
<td></td>
<td>kNN</td>
<td>0.8800 (0.0034)</td>
<td>0.9804 (0.0024)</td>
<td>0.6519 (0.0032)</td>
<td>0.7164 (0.0027)</td>
</tr>
<tr>
<td>SimTensorML</td>
<td>Average</td>
<td>1.1153 (0.0142)</td>
<td>1.0</td>
<td>0.8897 (0.0096)</td>
<td>0.6513 (0.0098)</td>
</tr>
<tr>
<td></td>
<td>RMF</td>
<td>1.0672 (0.0163)</td>
<td>0.9571 (0.0012)</td>
<td>0.8421 (0.0019)</td>
<td>0.3396 (0.0019)</td>
</tr>
<tr>
<td></td>
<td>NMF</td>
<td>1.0814 (0.0164)</td>
<td>0.9707 (0.0009)</td>
<td>0.8207 (0.0149)</td>
<td>0.592 (0.0186)</td>
</tr>
<tr>
<td></td>
<td>PMF</td>
<td>1.151 (0.0066)</td>
<td>1.0000 (0.0200)</td>
<td>0.8869 (0.0045)</td>
<td>0.0431 (0.0014)</td>
</tr>
<tr>
<td></td>
<td>kNN</td>
<td>1.2941 (0.0218)</td>
<td>1.1619 (0.0345)</td>
<td>0.9993 (0.0227)</td>
<td>0.0999 (0.0099)</td>
</tr>
<tr>
<td>RandomML</td>
<td>Average</td>
<td>1.4358 (0.0011)</td>
<td>1.0</td>
<td>1.2496 (0.0021)</td>
<td>0.3220 (0.0021)</td>
</tr>
<tr>
<td></td>
<td>RMF</td>
<td>1.3960 (0.0012)</td>
<td>1.1214 (0.0012)</td>
<td>1.3428 (0.0039)</td>
<td>0.4196 (0.0031)</td>
</tr>
<tr>
<td></td>
<td>NMF</td>
<td>1.6734 (0.0009)</td>
<td>1.1664 (0.0041)</td>
<td>1.3943 (0.0045)</td>
<td>0.4574 (0.0039)</td>
</tr>
<tr>
<td></td>
<td>PMF</td>
<td>1.4848 (0.0010)</td>
<td>1.0144 (0.0039)</td>
<td>1.2744 (0.0047)</td>
<td>0.3464 (0.0020)</td>
</tr>
<tr>
<td></td>
<td>kNN</td>
<td>1.6136 (0.0048)</td>
<td>1.1380 (0.0035)</td>
<td>1.3666 (0.0001)</td>
<td>0.4313 (0.0019)</td>
</tr>
</tbody>
</table>
Table 3.4
Yelp music - Performance of 5 selected recommender systems on Yahoo, GIDS, SimTensor and Random. Results represent the mean value over 5 folds with standard deviation reported in the brackets. Note that the relative RMSE is normalized with the Average estimator for each dataset separately.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>RMSE</th>
<th>RRMSE</th>
<th>MAE</th>
<th>F_1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Yahoo</strong></td>
<td>Average</td>
<td>1.5842 (0.0037)</td>
<td>1.4147 (0.0019)</td>
<td>0.0233 (0.008)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RMF</td>
<td>1.501 (0.0021)</td>
<td>1.0070 (0.0014)</td>
<td>0.0016 (0.003)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NMF</td>
<td>1.2994 (0.0060)</td>
<td>0.8203 (0.0043)</td>
<td>0.9919 (0.0048)</td>
<td>0.5462 (0.0087)</td>
</tr>
<tr>
<td></td>
<td>PMF</td>
<td>1.1977 (0.013)</td>
<td>0.992 (0.0828)</td>
<td>1.1241 (0.0135)</td>
<td>0.1121 (0.2199)</td>
</tr>
<tr>
<td></td>
<td>kNN</td>
<td>1.2769 (0.0021)</td>
<td>0.8060 (0.0095)</td>
<td>0.9948 (0.0021)</td>
<td>0.6042 (0.0046)</td>
</tr>
</tbody>
</table>

| **GIDS**     | Average| 1.5356 (0.0018) | 1.569 (0.0023) | 0.0000 (0) |
|              | RMF    | 1.5720 (0.0041)  | 1.0984 (0.0026) | 0.9520 (0.0105) |
|              | NMF    | 1.3444 (0.0032)  | 0.8616 (0.0030) | 1.0417 (0.0035) | 0.5754 (0.0037) |
|              | PMF    | 1.4701 (0.009)   | 0.9617 (0.0741) | 1.1047 (0.116) | 0.0666 (0.1812) |
|              | kNN    | 1.2471 (0.0023)  | 0.8175 (0.0090) | 0.9971 (0.0009) | 0.9918 (0.0023) |

| **SimTensor**| Average| 1.6944 (0.009)  | 1.4616 (0.0021) | 0.4024 (0.008) |
|              | RMF    | 1.5147 (0.0019)  | 1.3249 (0.0144) | 0.4048 (0.0007) |
|              | NMF    | 1.8919 (0.008)   | 1.1121 (0.030)  | 0.4498 (0.0062) |
|              | PMF    | 1.6944 (0.012)   | 1.4616 (0.0024) | 0.4012 (0.0098) |
|              | kNN    | 1.6831 (0.020)   | 1.2799 (0.0243) | 0.4408 (0.0131) |

| **Random**   | Average| 1.4417 (0.0013) | 1.2008 (0.0014) | 0.0000 (0) |
|              | RMF    | 1.6883 (0.0012)  | 1.1987 (0.0016) | 0.334 (0.0007) |
|              | NMF    | 1.6269 (0.0016)  | 1.002 (0.0027)  | 0.3212 (0.0030) |
|              | PMF    | 1.4466 (0.0023)  | 1.1106 (0.0043) | 0.0001 (0.0002) |
|              | kNN    | 1.4624 (0.0016)  | 1.2999 (0.0002) | 0.1946 (0.0021) |

Table 3.5
Yelp - Performance of 5 selected recommender systems on Yelp, GIDS, SimTensor and Random. Results represent the mean value over 5 folds with standard deviation reported in the brackets. Note that the relative RMSE is normalized with the Average estimator for each dataset separately.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>RMSE</th>
<th>RRMSE</th>
<th>MAE</th>
<th>F_1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Yelp</strong></td>
<td>Average</td>
<td>1.1999 (0.0047)</td>
<td>0.9175 (0.0041)</td>
<td>0.7777 (0.0012)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RMF</td>
<td>1.1668 (0.0019)</td>
<td>0.868 (0.0021)</td>
<td>0.8031 (0.0001)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NMF</td>
<td>1.1452 (0.0014)</td>
<td>1.0412 (0.0049)</td>
<td>0.7759 (0.0026)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PMF</td>
<td>1.1983 (0.009)</td>
<td>0.9576 (0.0098)</td>
<td>0.7778 (0.0012)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>kNN</td>
<td>1.2906 (0.0060)</td>
<td>0.9772 (0.0044)</td>
<td>0.7726 (0.0027)</td>
<td></td>
</tr>
</tbody>
</table>

| **GIDS**     | Average| 1.1403 (0.0023) | 0.9082 (0.0016) | 0.8832 (0.0025) |
|              | RMF    | 1.0972 (0.0016)  | 0.8966 (0.0016) | 0.9047 (0.0017) |
|              | NMF    | 1.1166 (0.0041)  | 1.0000 (0.0048) | 0.7312 (0.0066) |
|              | PMF    | 1.1403 (0.0041)  | 0.9082 (0.0007) | 0.9048 (0.0023) |
|              | kNN    | 1.185 (0.0031)   | 0.904 (0.0013)  | 0.8876 (0.0018) |

| **SimTensor**| Average| 1.3043 (0.0013) | 1.1868 (0.0087) | 0.7086 (0.0099) |
|              | RMF    | 1.1316 (0.0016)  | 1.1398 (0.0109) | 0.7979 (0.0081) |
|              | NMF    | 1.3413 (0.0015)  | 1.1411 (0.0088) | 0.7064 (0.0061) |
|              | PMF    | 1.305 (0.0014)   | 1.186 (0.0126)  | 0.7066 (0.0046) |
|              | kNN    | 1.4012 (0.009)   | 1.0444 (0.0099) | 0.7068 (0.0041) |

| **Random**   | Average| 1.4479 (0.0028) | 1.2020 (0.0038) | 0.693 (0.0028) |
|              | RMF    | 1.523 (0.0031)   | 1.0745 (0.0018) | 1.2901 (0.009) | 0.6821 (0.0018) |
|              | NMF    | 1.7346 (0.0023)  | 1.1233 (0.0098) | 1.4310 (0.0034) | 0.6886 (0.0012) |
|              | PMF    | 1.4219 (0.0010)  | 1.002 (0.0009)  | 1.1200 (0.0005) | 0.6400 (0.0012) |
|              | kNN    | 1.4217 (0.0017)  | 1.002 (0.0011)  | 1.2071 (0.0023) | 0.6414 (0.0030) |
3.4.2 Time-aware recommender system performance

We took the same datasets that we used in the MovieLens experiment and observed the performance of time-aware recommender systems on that same datasets. Firstly, we have transformed MovieLens’s unix standard timestamps to days, starting at 0 for the first timestamp (up to 81) to match the timestamps of others that range from 0 to 100. Secondly, we arranged all datasets by their timestamp to form an ordered data stream. We split these data streams into a train set (80%) and a test set (remaining 20%) in order to resemble an online evaluation (therefore all test instances have a timestamp that is equal or greater to timestamps of instances from the train set). Similar to [78] we used 3 kNN variations: base kNN model that uses no time dependency, Time Decay (TD) approach and Contextual pre-filtering (PRF) that uses a context \{weekday, weekend\} to pre-filter instances. For MovieLens we obtained this context from the unix timestamps and for GIDS and Random we presumed that weekends are all timestamps that are divisible by either 6 or 7.

Table 3.6 shows the RMSE, Precision@10, Recall@10 and $F_1$ score (for the top10 ranking problem) for Base, TD and PRF models on MovieLens, GIDS, SimTensor and Random datasets. Results show that time-decay kNN performs better than the base model for both MovieLens and GIDS, while this is not the case for SimTensor and Random. Additionally, we observe that MovieLens and GIDS using contextual pre-filtering (though having slightly higher RMSE) both have a higher recall, resulting in an increased overall $F_1$ score. However, we did not observe this same behavior on SimTensor and the baseline.

3.4.3 Ranking

Next we observed how rankings of recommender systems models trained on the real-life datasets compare with rankings of models on the simulated datasets. In this way we wanted to “measure” the benchmarking capabilities of individual data generators. In addition, we have also calculated a normalized Damerau–Levenshtein distance (NDL) between rankings on the real-life datasets and rankings on the generated data. The normalized Damerau–Levenshtein distance is defined as an edit distance between two sequences (in our case this relates to the minimum number of required swaps to transform one sequence into the other) normalized by the length of the sequence. With NDL we therefore wanted to measure the distance between two ranks permutations, with respect
Table 3.6
Performance results of time-aware recommender systems on MovieLens, GIDS, SimTensor and Random. Best results for each metric on the same dataset are denoted with boldface.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>RMSE</th>
<th>P@10</th>
<th>R@10</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>MovieLens</td>
<td>Base</td>
<td>1.0299</td>
<td>0.9619</td>
<td>0.2570</td>
<td>0.4013</td>
</tr>
<tr>
<td></td>
<td>TD</td>
<td>1.0319</td>
<td>0.9360</td>
<td>0.2474</td>
<td>0.3913</td>
</tr>
<tr>
<td></td>
<td>PRF</td>
<td>1.0497</td>
<td>0.7723</td>
<td>0.4699</td>
<td>0.5843</td>
</tr>
<tr>
<td>GIDS</td>
<td>Base</td>
<td>0.9419</td>
<td>0.7457</td>
<td>0.3217</td>
<td>0.6214</td>
</tr>
<tr>
<td>ML</td>
<td>TD</td>
<td>0.8982</td>
<td>0.7479</td>
<td>0.3421</td>
<td>0.6286</td>
</tr>
<tr>
<td></td>
<td>PRF</td>
<td>0.9643</td>
<td>0.7433</td>
<td>0.5523</td>
<td>0.6537</td>
</tr>
<tr>
<td>SimTensor</td>
<td>Base</td>
<td>1.5030</td>
<td>0.5524</td>
<td>0.7365</td>
<td>0.6586</td>
</tr>
<tr>
<td>ML</td>
<td>TD</td>
<td>1.5118</td>
<td>0.5476</td>
<td>0.7456</td>
<td>0.6513</td>
</tr>
<tr>
<td></td>
<td>PRF</td>
<td>1.4894</td>
<td>0.4009</td>
<td>0.3947</td>
<td>0.3997</td>
</tr>
<tr>
<td>Random</td>
<td>Base</td>
<td>1.5481</td>
<td>0.6321</td>
<td>0.0640</td>
<td>0.1162</td>
</tr>
<tr>
<td>ML</td>
<td>TD</td>
<td>1.5426</td>
<td>0.6418</td>
<td>0.0696</td>
<td>0.1105</td>
</tr>
<tr>
<td></td>
<td>PRF</td>
<td>1.5312</td>
<td>0.7312</td>
<td>0.0380</td>
<td>0.0723</td>
</tr>
</tbody>
</table>

to the minimum number of required element swaps (distance 0 – no swaps required as rankings are equal, 1 – all elements must be swapped).

Tables 3.7 to 3.10 show the rankings of RMF, NMF, PMF, kNN and the baseline (Average) for all domains that were used in the previous evaluations (MovieLens, Yahoo music and Yelp). For each domain we present model rankings on the real dataset and three synthetic datasets (GIDS, SimTensor and Random), alongside with the normalized Damerau–Levenshtein distance between model rankings on real and simulated datasets.

Although there are certainly differences in the rankings, we can conclude that model rankings on GIDS follow more closely the model rankings on real-life datasets, compared to model rankings on SimTensor or Random. This applies particularly for the Yahoo domain and the time-aware MovieLens domain, where rankings of GIDS and real-life dataset are almost identical for every error measure. As expected, the rankings on Random are completely random, and often the baseline algorithm (Average) performs the best. We can confirm these results when we look at the NDL measure, as GIDS outperformed both data generators on the Yahoo and MovieLens (time-aware models) domains, while the SimTensor generator was slightly better on the Yelp and MovieLens domains for the RMSE error measure.
Table 3.7
Rankings of the recommender system models on MovieLens, GIDSML, SimTensorML and RandomML for three error measures: RMSE, MAE and F1, alongside with normalized Damerau–Levenshtein distances between rankings on real and synthetic datasets.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Dataset</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>NDLML</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMSE</td>
<td>MovieLens</td>
<td>RMF</td>
<td>PMF</td>
<td>NMF</td>
<td>kNN</td>
<td>AVG</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>GIDSML</td>
<td>RMF</td>
<td>kNN</td>
<td>PMF</td>
<td>AVG</td>
<td>NMF</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>SimTensorML</td>
<td>RMF</td>
<td>NMF</td>
<td>PMF</td>
<td>AVG</td>
<td>kNN</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>RandomML</td>
<td>AVG</td>
<td>PMF</td>
<td>RMF</td>
<td>kNN</td>
<td>NMF</td>
<td>0.6</td>
</tr>
<tr>
<td>MAE</td>
<td>MovieLens</td>
<td>RMF</td>
<td>PMF</td>
<td>NMF</td>
<td>kNN</td>
<td>AVG</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>GIDSML</td>
<td>kNN</td>
<td>RMF</td>
<td>NMF</td>
<td>PMF</td>
<td>AVG</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>SimTensorML</td>
<td>NMF</td>
<td>RMF</td>
<td>PMF</td>
<td>AVG</td>
<td>kNN</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>RandomML</td>
<td>AVG</td>
<td>PMF</td>
<td>RMF</td>
<td>kNN</td>
<td>NMF</td>
<td>0.6</td>
</tr>
<tr>
<td>F1</td>
<td>MovieLens</td>
<td>RMF</td>
<td>NMF</td>
<td>kNN</td>
<td>PMF</td>
<td>AVG</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>GIDSML</td>
<td>PMF</td>
<td>RMF</td>
<td>NMF</td>
<td>kNN</td>
<td>AVG</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>SimTensorML</td>
<td>NMF</td>
<td>kNN</td>
<td>RMF</td>
<td>PMF</td>
<td>AVG</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>RandomML</td>
<td>NMF</td>
<td>kNN</td>
<td>RMF</td>
<td>PMF</td>
<td>AVG</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Table 3.8
Rankings of the recommender system models on Yahoo, GIDSYH, SimTensorYH and RandomYH for three error measures: RMSE, MAE and F1, alongside with normalized Damerau–Levenshtein distances between rankings on real and synthetic datasets.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Dataset</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>NDLYH</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMSE</td>
<td>Yahoo</td>
<td>kNN</td>
<td>NMF</td>
<td>RMF</td>
<td>PMF</td>
<td>AVG</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>GIDSYH</td>
<td>kNN</td>
<td>NMF</td>
<td>RMF</td>
<td>PMF</td>
<td>AVG</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>SimTensorYH</td>
<td>NMF</td>
<td>RMF</td>
<td>PMF</td>
<td>AVG</td>
<td>kNN</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>RandomYH</td>
<td>AVG</td>
<td>PMF</td>
<td>kNN</td>
<td>NMF</td>
<td>RMF</td>
<td>0.8</td>
</tr>
<tr>
<td>MAE</td>
<td>Yahoo</td>
<td>NMF</td>
<td>kNN</td>
<td>RMF</td>
<td>PMF</td>
<td>AVG</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>GIDSYH</td>
<td>kNN</td>
<td>NMF</td>
<td>RMF</td>
<td>PMF</td>
<td>AVG</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>SimTensorYH</td>
<td>NMF</td>
<td>kNN</td>
<td>AVG</td>
<td>RMF</td>
<td>PMF</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>RandomYH</td>
<td>NMF</td>
<td>RMF</td>
<td>kNN</td>
<td>AVG</td>
<td>PMF</td>
<td>0.6</td>
</tr>
<tr>
<td>F1</td>
<td>Yahoo</td>
<td>kNN</td>
<td>RMF</td>
<td>NMF</td>
<td>PMF</td>
<td>AVG</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>GIDSYH</td>
<td>kNN</td>
<td>NMF</td>
<td>RMF</td>
<td>PMF</td>
<td>AVG</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>SimTensorYH</td>
<td>NMF</td>
<td>kNN</td>
<td>AVG</td>
<td>RMF</td>
<td>PMF</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>RandomYH</td>
<td>NMF</td>
<td>RMF</td>
<td>kNN</td>
<td>AVG</td>
<td>PMF</td>
<td>0.6</td>
</tr>
</tbody>
</table>
Generating inter-dependent data streams

Table 3.9
Rankings of the recommender system models on YELP, GIDS\textsubscript{YL}, SimTensor\textsubscript{YL} and Random\textsubscript{YL} for three error measures: RMSE, MAE and $F_1$, alongside with normalized Damerau–Levenshtein distances between rankings on real and synthetic datasets.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Dataset</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>NDL\textsubscript{YL}</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMSE</td>
<td>Yelp</td>
<td>RMF</td>
<td>AVG</td>
<td>PMF</td>
<td>NMF</td>
<td>kNN</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>GIDS\textsubscript{YL}</td>
<td>RMF</td>
<td>kNN</td>
<td>AVG</td>
<td>PMF</td>
<td>NMF</td>
<td>0.2</td>
</tr>
<tr>
<td></td>
<td>SimTensor\textsubscript{YL}</td>
<td>RMF</td>
<td>PMF</td>
<td>AVG</td>
<td>NMF</td>
<td>kNN</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>Random\textsubscript{YL}</td>
<td>AVG</td>
<td>kNN</td>
<td>PMF</td>
<td>RMF</td>
<td>NMF</td>
<td>1.0</td>
</tr>
<tr>
<td>MAE</td>
<td>Yelp</td>
<td>RMF</td>
<td>NMF</td>
<td>AVG</td>
<td>PMF</td>
<td>kNN</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>GIDS\textsubscript{YL}</td>
<td>RMF</td>
<td>kNN</td>
<td>AVG</td>
<td>PMF</td>
<td>NMF</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>SimTensor\textsubscript{YL}</td>
<td>NMF</td>
<td>RMF</td>
<td>PMF</td>
<td>AVG</td>
<td>kNN</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>Random\textsubscript{YL}</td>
<td>AVG</td>
<td>kNN</td>
<td>PMF</td>
<td>RMF</td>
<td>NMF</td>
<td>1.0</td>
</tr>
<tr>
<td>$F_1$</td>
<td>Yelp</td>
<td>RMF</td>
<td>PMF</td>
<td>AVG</td>
<td>NMF</td>
<td>kNN</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>GIDS\textsubscript{YL}</td>
<td>RMF</td>
<td>PMF</td>
<td>lNN</td>
<td>AVG</td>
<td>NMF</td>
<td>0.4</td>
</tr>
<tr>
<td></td>
<td>SimTensor\textsubscript{YL}</td>
<td>RMF</td>
<td>AVG</td>
<td>lNN</td>
<td>NMF</td>
<td>PMF</td>
<td>0.6</td>
</tr>
<tr>
<td></td>
<td>Random\textsubscript{YL}</td>
<td>RMF</td>
<td>NMF</td>
<td>lNN</td>
<td>PMF</td>
<td>AVG</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Table 3.10
Rankings of the time-aware recommender system models on MovieLens, GIDS\textsubscript{ML}, SimTensor\textsubscript{ML} and Random\textsubscript{ML} for four error measures: RMSE, P@10, R@10 and $F_1$, alongside with normalized Damerau–Levenshtein distances between rankings on real and synthetic datasets.

<table>
<thead>
<tr>
<th>Measure</th>
<th>Dataset</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>NDL\textsubscript{ML}</th>
</tr>
</thead>
<tbody>
<tr>
<td>RMSE</td>
<td>MovieLens</td>
<td>TD</td>
<td>BASE</td>
<td>PRF</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GIDS\textsubscript{ML}</td>
<td>TD</td>
<td>BASE</td>
<td>PRF</td>
<td>0.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SimTensor\textsubscript{ML}</td>
<td>PRF</td>
<td>BASE</td>
<td>TD</td>
<td>0.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Random\textsubscript{ML}</td>
<td>PRF</td>
<td>TD</td>
<td>BASE</td>
<td>0.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P@10</td>
<td>MovieLens</td>
<td>TD</td>
<td>BASE</td>
<td>PRF</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GIDS\textsubscript{ML}</td>
<td>TD</td>
<td>BASE</td>
<td>PRF</td>
<td>0.33</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SimTensor\textsubscript{ML}</td>
<td>BASE</td>
<td>TD</td>
<td>PRF</td>
<td>0.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Random\textsubscript{ML}</td>
<td>PRF</td>
<td>TD</td>
<td>BASE</td>
<td>0.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R@10</td>
<td>MovieLens</td>
<td>PRF</td>
<td>BASE</td>
<td>TD</td>
<td>0.33</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GIDS\textsubscript{ML}</td>
<td>PRF</td>
<td>TD</td>
<td>BASE</td>
<td>0.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SimTensor\textsubscript{ML}</td>
<td>BASE</td>
<td>TD</td>
<td>PRF</td>
<td>0.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Random\textsubscript{ML}</td>
<td>BASE</td>
<td>TD</td>
<td>PRF</td>
<td>0.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$F_1$</td>
<td>MovieLens</td>
<td>PRF</td>
<td>BASE</td>
<td>TD</td>
<td>0.33</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>GIDS\textsubscript{ML}</td>
<td>PRF</td>
<td>TD</td>
<td>BASE</td>
<td>0.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SimTensor\textsubscript{ML}</td>
<td>BASE</td>
<td>TD</td>
<td>PRF</td>
<td>0.66</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Random\textsubscript{ML}</td>
<td>BASE</td>
<td>TD</td>
<td>PRF</td>
<td>0.66</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
3.4.4 Data fusion algorithm performance

In the final experiment we observed the GIDS’ ability to generate multiple inter-dependent datasets with hidden correlations. This ability would prove useful in generating problems for recommender systems that learn from multiple problem domains and can utilize the power of data fusion. For example, we could generate three datasets, where the first dataset would represent movie ratings of a selected group of users, the second dataset would represent song ratings of this same group of users, while the third dataset would represent some contextual information about the movies (e.g. actors that appear in selected movies). The data fusion capable recommender system could then increase its accuracy by learning from hidden interactions between users that rate movies and songs in the same manner, interactions between actors and movie ratings, or even from some new and unknown hidden features.

With this experiment we wanted to show that GIDS is capable of generating sets of distinct relations with mutual information. The potential gain in prediction accuracy would therefore not be result of solely the addition of new training instances of the same type (as with Random datasets), but the result of new information extracted from the neighbour relations.

Using the procedure explained in section 3.2.5 we have generated three $500 \times 500$ dense matrices relating four object types with ratings from 0.5 to 5. Alongside we have also used the Random SDG to generate three additional matrices with the same data characteristics (rating value distribution, dimensions and sparsity). Similar to the previous experiment, we performed the matrix completion task (prediction of the holdout set). At the beginning only the first datasets for GIDS and Random were split into the train set ($\sim 90\%$) and test/holdout set ($\sim 10\%$). Then, the DFMF and RAkNN models were trained on the first dataset and initial RMSE was calculated. Finally, the whole second and third datasets were gradually added to the train set of the first dataset, each time predicting the same instances of the holdout set (again from the first dataset). Our assumption was that the predictors should achieve better performance when using multiple datasets generated by GIDS in contrast to using just one, due to the shared hidden information. At the same time, this should not be noticed when using datasets generated by Random.

Figure 3.12 shows the results in terms of the percentage loss in RMSE while using one, two or three datasets, generated by GIDS and Random. When using datasets generated
by GIDS, the DFMF model achieved better performance with addition of the second dataset (more than 2%) and also the third dataset (almost 5%). As expected, this was not the case when using datasets generated by Random. Same is observed for the RAkNN model where addition of the second and third dataset resulted in a better performance (more than 2%). These results indicate that models can successfully combine information from additional datasets (generated by GIDS) in order to improve prediction on the main relation.

Figure 3.12
Normalized RMSE (percentage loss) of the DFMF and RAkNN models, when using one, two or three datasets for GIDS and Random. The RMSE is normalized to the initial value for the first dataset for GIDS and Random separately. The values above each bar represent the raw RMSE values.

3.5 Summary

Scarcity and lack in diversity of publicly available datasets make development and evaluation of machine learning algorithms, such as recommender systems, a difficult and challenging task. In this chapter we have presented a methodology for generating multiple inter-dependent data streams for evaluating various static, incremental and data fusion algorithms. Furthermore, it can provide a reliable benchmarking and simulation tool for numerous large-scale problems.

The proposed GIDS data generator works by creating object clusters that are interconnected with each other, which creates a clear structure that can be found in real-life datasets. The evaluation showed that generated datasets resemble real datasets both in terms of data statistics over time as well as the prediction accuracy of recommender and data fusion algorithms.
Simultaneous incremental matrix factorization
Matrix factorization techniques have proven reliable for implementing on-line solutions such as recommender systems. The data sparsity and the cold-start problem can be indirectly alleviated by considering multiple heterogeneous data sources. For real-world applications, e.g., such with continuous user feedback, incrementally handling machine learning models upon data streams remains a crucial and only partially solved problem.

In this chapter we propose Simultaneous Incremental Matrix Factorization (SIMF), a novel method for simultaneous modeling of multiple heterogeneous data streams. We use a collective matrix tri-factorization approach, where we factor relation matrices of two object types into three smaller latent representations of those entities (we only use one shared factor matrix for one type of entity). Then, this collective model is updated upon data streams, using only newly arrived data, without storing any previous information. This incremental update further allows for addition of new objects and quicker adaptation to new concepts in data. Results on synthetic and real-life data streams confirm that predictions can be improved by extending the factorization process with additional data streams.

Throughout this chapter we will be using recommender systems as an example of the SIMF’s use case. Nonetheless, we are presenting a general incremental and collective matrix factorization technique, which can be directly or indirectly used to solve several other problems.

4.1 Background

New sources of information, such as information from social networks and the Internet of Things, are confronting recommender systems with great challenges as the sheer amount of constantly arriving and heterogeneous information grows. As we explained in Sections 2.2 and 2.4.2, the majority of the current state-of-the-art recommender systems is based on various matrix factorization techniques, which feature simple and highly accurate modeling with the ability to collectively mine multiple data sources.

However, due to a high computational complexity, common matrix factorization methods are not scalable or suitable for real-world applications, where new ratings and other user-feedback are generated continuously. While several incremental matrix factorization methods for recommender systems are available [14, 19, 43, 50], they are not suitable for direct modeling of multiple heterogeneous data streams, as the majority of them is limited to factorizing only a single relation using the classic two-factorization model.
4.2 Simultaneous Incremental Matrix Factorization

Let $\mathcal{D}$ be the collection of $n$ data streams $d_1 = d_{ij}, \ldots, d_n = d_{kl}$ and $\mathcal{O}$ be the collection of $r$ object types $\mathcal{E}_1, \ldots, \mathcal{E}_r$. Then, each data stream $d_{ij}$ represents an infinite source of relations between two object types $\epsilon_i$ and $\epsilon_j$. Furthermore, we can express each data stream $d_{ij}$ with a relation matrix $R_{ij}(t)$. This “dynamic” matrix changes through time as new relations arrive or are updated in the data stream. If, for example, we consider a collaborative filtering problem, the object types could represent users, movies, songs, books etc., and data streams could represent the real-time ratings of movies or other items from those users. Relations can be asymmetric ($R_{ij}(t) \neq R_{ji}(t)$) or missing if we do not have any information or sufficient data on how to relate two object types (e.g. we can collectively factorize users, movies and actors without the need to relate users with actors).

Our proposed method, simultaneous incremental matrix factorization (SIMF), collectively factorizes multiple parallel data streams into a common representation (the factor model), which is then incrementally updated to co-align to possible temporal changes in data streams. SIMF minimizes the objective (loss) function that was deliberately constructed in a way to accomplish our goals. Through the following subsections we present our algorithm by constructing and iteratively upgrading the SIMF’s objective function. We start by introducing basic matrix tri-factorization and explain its benefits further by combining multiple data sources via data fusion. Next, we introduce our incremental setting and notation for the simultaneous incremental matrix factorization, followed by our approach to tackle sparseness of the rating matrices and incorporation of additional information, such as constraints, biases and regularization.

4.2.1 Matrix tri-factorization

A tri-factorization model can be used to reduce the dimensionality of data and to find its latent representation (e.g. latent factors for collaborative filtering). Let $R$ be the matrix relating two object types (e.g. users and movies) as a ratings matrix. The matrix $R \in \mathbb{R}^{n \times m}$ is factorized into three matrices, $G_1 \in \mathbb{R}^{n \times k_1}$, $S \in \mathbb{R}^{k_1 \times k_2}$ and $G_2 \in \mathbb{R}^{m \times k_2}$, so that $R \approx G_1 S G_2^T$. We represent the objective loss function in terms of the Frobenius norm minimization:

$$\min f(G_1, S, G_2) = \|R - G_1 S G_2^T\|_F^2.$$
This objective function can be minimized using the multiplicative update rules [40, 69], SVD [13], gradient and stochastic gradient descent [8], alternating least squares [8], coordinate descent [105], or some other optimization method. Most of these methods also allow for inclusion of additional constraints, such as nonnegativity or orthogonality (depending on the nature of the problem).

### 4.2.2 Data fusion and collective matrix factorization

One way of utilizing the advantages of data fusion is through matrix factorization. Using the principles from DFMF [10] we represent the collection of data streams (dynamic relation matrices $R_{ij}$) with a single block matrix $R$:

\[
R = \begin{bmatrix}
* & R_{12} & \ldots & R_{1r} \\
R_{21} & * & \ldots & R_{2r} \\
\vdots & \vdots & \ddots & \vdots \\
R_{r1} & R_{r2} & \ldots & * 
\end{bmatrix}
\]

We represent the factors in a similar fashion using block matrices $G$ and $S$:

\[
G = \begin{bmatrix}
G_1 \\
G_2 \\
\vdots \\
G_r
\end{bmatrix}, \quad 
S = \begin{bmatrix}
* & S_{12} & \ldots & S_{1r} \\
S_{21} & * & \ldots & S_{2r} \\
\vdots & \vdots & \ddots & \vdots \\
S_{r1} & S_{r2} & \ldots & * 
\end{bmatrix}
\]

\[
G^T = \begin{bmatrix}
G_1^T \\
G_2^T \\
\vdots \\
G_r^T
\end{bmatrix}
\]

where block matrix $R$ contains relation matrices $R_{ij}$ that relate object types $\mathcal{E}_i$ and $\mathcal{E}_j$, diagonal block matrix $G$ contains latent representation of those object types and block matrix $S$ contains latent connections between two object types. Just like in DFMF, any relation can be asymmetric or even left out, while the relations of the same object type ($*$) are modeled with special constraint matrices (we further explain this in Section 4.2.6).
Simultaneous incremental matrix factorization

We represent the collective factorization \( GSG^T \) with the block matrix \( \hat{R} \):

\[
\hat{R} = \begin{bmatrix}
* & G_1S_{12}G_2^T & \ldots & G_1S_{1k}G_k^T \\
G_2S_{21}G_1^T & * & \ldots & G_2S_{2k}G_k^T \\
\vdots & \vdots & \ddots & \vdots \\
G_kS_{k1}G_1^T & G_kS_{k2}G_2^T & \ldots & * 
\end{bmatrix}
\]

Now we update the basic objective function from the previous section to handle multiple data streams at once, via the collective matrix factorization:

\[
\min f(G, S) = \sum_{R_{ij} \in D} \|R_{ij} - G_iS_jG_j^T\|_F^2,
\]

where every relation is approximated in terms of Frobenius norm separately, while (collectively) using the same factors for the same object types.

4.2.3 Collective factorization in the streaming environment

While most collective matrix factorization techniques work in a static setting, they can be easily adapted to incremental learning. In the following, we present an overview and notation for simultaneous incremental matrix factorization upon data streams.

Consider the object types \( \mathcal{E}_1, \ldots, \mathcal{E}_r \) and data streams \( d_1, \ldots, d_n \) from before. Each data stream is characterized by a relation matrix \( R_{ij}(t) \) that relates objects from \( \mathcal{E}_i \) and \( \mathcal{E}_j \) at timestamp \( t \), where cell \( R_{ij}(t)(u, p) \) represents the magnitude of relation between object \( u \) of the type \( \mathcal{E}_i \) and object \( p \) of the type \( \mathcal{E}_j \). Using the tri-factorization technique we factorize the relation block matrix \( R(t) \) into smaller block matrices \( G(t) \) and \( S(t) \), so that \( R(t) \approx G(t)S(t)^T \).

The SIMF algorithm works in two steps, the initial factorization and the streaming update phase. In the first step the initial data is collected up to some specific timestamp and then used for the collective factorization into the initial state (the initial model). Generally, the more data we spare at the start, the better initial factorization we can expect. During the second phase the initial data can be completely discarded, while the model is incrementally updated with the newly arrived information (i.e. changes in matrices \( R_{ij}(t) \)). These updates represent changes in matrix values or additions of new rows and columns (e.g. this could represent new users and new movies that are added to the system). In our setting we do not consider row or column deletions, since deletions are not usually present in the real-world (recommendation) scenarios.
The schematics of the proposed factorization system can be seen in an example from Figure 4.1. This figure shows three data streams that are being factorized into a collection of submatrices at a given timestamp $t$. For example, we can see how matrices $R^{(t)}_{12}$ and $R^{(t)}_{13}$ are factorized into five smaller matrices, having a common latent representation of type $\mathcal{E}_1$ - the factor matrix $G^{(t)}_1$.

![Figure 4.1](image)

*Figure 4.1*

Example of the collective factorization process with 4 object types ($\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4$) and 3 relations/data streams ($R^{(t)}_{12}, R^{(t)}_{13}$ and $R^{(t)}_{42}$) at the timestamp $t$. This figure illustrates how factorization of $R^{(t)}_{12}$ and $R^{(t)}_{13}$ results in a set of submatrices, $G^{(t)}_1$ being common to both data streams.
4.2.4 Tackling sparseness

Apart from the cold-start problem, sparse input matrices can diverge the optimization process. This is especially a problem in recommender systems, where rating matrices are over 90% sparse. That is why we need to adjust the factorization process so it does not overfit to zeros (unknown values) in input matrices $R^{(t)}_{ij}$. To achieve this we need to change our objective function by introducing mask (weighting) matrices $W^{(t)}_{ij}$:

$$W^{(t)}_{ij}(\text{row, col}) = \begin{cases} 1 & \text{if } R^{(t)}_{ij}(\text{row, col}) \text{ is given at the timestamp } t \\ 0 & \text{if } R^{(t)}_{ij}(\text{row, col}) \text{ is missing at the timestamp } t \end{cases},$$

This helps us construct the objective function in such a way that minimization only affects the non-missing values (similar to the weighted matrix factorization [20, 69, 106, 107]):

$$\min f(G^{(t)}, S^{(t)}) = \sum_{R^{(t)}_{ij} \in \mathcal{D}} \|W^{(t)}_{ij} \odot (R^{(t)}_{ij} - G^{(t)}_i S^{(t)}_j G^{(t)}_j^T)\|_F^2, \quad (4.1)$$

where $\odot$ denotes the Hadamard product (the element-wise product), defined as:

$$A \in \mathbb{R}^{n \times m} = \{a_{ij}\}, \quad B \in \mathbb{R}^{n \times m} = \{b_{ij}\} \Rightarrow A \ast B = \{a_{ij}b_{ij}\}.$$

4.2.5 Calculating derivatives

Next we need to find the derivatives of the Equation (4.1) in order to construct the update rules through the gradient steps. Firstly, we introduce the Frobenius inner product of two matrices $A$ and $B$, defined as:

$$A, B \in \mathbb{R}^{n \times m} \quad (A, B) = tr(A^T B) = \sum_{i,j} a_{ij} b_{ij},$$

with which we express the Frobenius norm as:

$$\|A\|_F = \sqrt{(A, A)}.$$

Since linearity holds for the Frobenius inner product, we can rewrite the Frobenius inner product of the sums of two matrices ($A + C$ and $B + D$) as:

$$A, B, C, D \in \mathbb{R}^{n \times m} \quad (A + C, B + D) = (A, B) + (A, D) + (C, B) + (C, D).$$
Furthermore, since the Hadamard product and the Frobenius inner product are closely related, we can apply the following simplification:

\[(A \circ B, C \circ D) = \langle A \circ (C \circ D), B \rangle\]

\[
\sum_{ij} (a_{ij}b_{ij})(c_{ij}d_{ij}) = \sum_{ij} (a_{ij}(c_{ij}d_{ij}))(b_{ij}).
\]

If we replace matrix C with matrix A or with the unit matrix, we further simplify the previous property of the Frobenius inner product:

\[C = A \in \{0, 1\}^{n \times m} \Rightarrow \langle A \circ B, A \circ D \rangle = \langle A \circ (A \circ D), B \rangle = \langle A \circ D, B \rangle\]

\[C = 1^{n \times m} \Rightarrow \langle A \circ B, D \rangle = \langle A \circ D, B \rangle.
\]

With these rules we express the objective function using the Frobenius inner product (for clearer analysis we omit the timestamp):

\[
\begin{align*}
\frac{1}{2} \sum_{R_{ij} \in \mathcal{D}} \left\| W_{ij} \right\| &= \frac{1}{2} \sum_{R_{ij} \in \mathcal{D}} \left( R_{ij} - G_i S_j G_j^T \right) \right\|^2 \\
&= \frac{1}{2} \sum_{R_{ij} \in \mathcal{D}} \langle W_{ij} \circ \left( R_{ij} - G_i S_j G_j^T \right), W_{ij} \circ \left( R_{ij} - G_i S_j G_j^T \right) \rangle \\
&= \frac{1}{2} \sum_{R_{ij} \in \mathcal{D}} \langle W_{ij} \circ \left( R_{ij} - X_{ij} \right), W_{ij} \circ \left( R_{ij} - X_{ij} \right) \rangle \quad \text{[} X_{ij} = G_i S_j G_j^T \text{]} \\
&= \frac{1}{2} \sum_{R_{ij} \in \mathcal{D}} \langle W_{ij} \circ \left( R_{ij} - X_{ij} \right), R_{ij} - X_{ij} \rangle \\
&= \frac{1}{2} \sum_{R_{ij} \in \mathcal{D}} \langle W_{ij} \circ R_{ij}, R_{ij} \rangle - \langle W_{ij} \circ R_{ij}, X_{ij} \rangle - \langle W_{ij} \circ X_{ij}, R_{ij} \rangle + \langle W_{ij} \circ X_{ij}, X_{ij} \rangle \\
&= \frac{1}{2} \sum_{R_{ij} \in \mathcal{D}} -2\langle W_{ij} \circ R_{ij}, X_{ij} \rangle + \langle W_{ij} \circ X_{ij}, R_{ij} \rangle + \langle W_{ij} \circ R_{ij}, R_{ij} \rangle \\
&= \sum_{R_{ij} \in \mathcal{D}} \text{tr}( -(W_{ij} \circ R_{ij})G_j S_j^T G_i + \frac{1}{2}(W_{ij} \circ G_i S_j G_j^T)G_j S_j^T G_i) + \frac{1}{2}(W_{ij} \circ R_{ij}, R_{ij})
\end{align*}
\]
We can now calculate the derivatives for $G_i$ and $S_{ij}$ as:

\[
\begin{align*}
    f(G, S) &= \sum_{R_{ij} \in \mathcal{D}} tr( - (W_{ij} \circ R_{ij}) G_j S_{ij}^T G_i) + \frac{1}{2} (W_{ij} \circ G_i S_{ij} G_j^T) G_j S_{ij}^T G_i \\
    &\quad + \frac{1}{2} \langle W_{ij} \circ R_{ij}, R_{ij} \rangle \\
    \frac{\partial f}{\partial G_i} &= \sum_{j : R_{ij} \in \mathcal{D}} ( - (W_{ij} \circ R_{ij}) G_j S_{ij}^T G_i + (W_{ij} \circ G_i S_{ij} G_j^T) G_j S_{ij}^T G_i ) \\
    &\quad + \sum_{j : R_{ji} \in \mathcal{D}} ( - (W_{ji} \circ R_{ji})^T G_j S_{ji} + (W_{ij} \circ G_i S_{ji} G_j^T) G_j S_{ji} ) \\
    \frac{\partial f}{\partial S_{ij}} &= -G_i^T (W_{ij} \circ R_{ij}) G_j + G_j^T (W_{ij} \circ G_i S_{ij} G_j^T) G_j
\end{align*}
\]

4.2.6 Incorporation of additional information

The main strength of matrix factorization is its robustness and versatility, as it allows for a straightforward incorporation of additional knowledge and information. We already described a way to incorporate supplementary data streams (via data fusion), and now we continue with introducing additional information fusion: single-object constraints, regularization, biases and relation weights.

**Single-object constraints**  As mentioned, SIMF handles relations of the same type with incorporation of constraint matrices that represent the cannot-link and must-link constraints between the objects of the same type (the penalties and rewards in the objective function). We denote these constraint matrices with $\Theta_i$ for a given object type $\mathcal{E}_i$, and each object type can have any number of them (up to $s_i$). These constraints are just a special case of a data stream (relation), where both object types are the same. For example, they could represent a “friends relationship” between users (social network).

We include the constraint matrices into the factorization by extending the objective function:

\[
\begin{align*}
    \min f(G^{(t)}, S^{(t)}) &= \sum_{R_{ij}^{(t)} \in \mathcal{D}} \| W_{ij}^{(t)} \circ (R_{ij}^{(t)} - G_i^{(t)} S_{ij}^{(t)} G_j^{(t)T}) \|_F^2 \\
    &\quad + \sum_{j=1}^{s_i} \sum_{t=1}^{\max_j} tr(G_i^{(t)} \Theta_i^{(t)} G_i^{(t)})
\end{align*}
\]

where $s_i$ represents the number of constraint matrices for $\mathcal{E}_i$. 
**Regularization**

To achieve better convergence and prevent overfitting, we also include the $L_2$ norm regularization. With regularization parameter $\lambda$ we extend the basic objective function as follows:

$$
\min f(G^{(t)}, S^{(t)}) = \sum_{R_{ij} \in \mathcal{D}} \|W_{ij} \circ (R_{ij} - G_i^{(t)} s_j^{(t)} G_j^{(t)T})\|_F^2 + \lambda \|G_i^{(t)}\|_F^2 + \lambda \|G_j^{(t)}\|_F^2.
$$

(4.3)

We can observe that Equation (4.3) can be expressed using Equation (4.2) by replacing the constraint matrices $\Theta_i^{(1)(t)}$ and $\Theta_j^{(1)(t)}$ with $\lambda I$. Therefore we model regularization through constraint matrices $\Theta$.

**Bias**

In recommender systems biases of users (or items) have a large effect on their rating behavior. For example, we can observe some users that systematically give higher ratings than others, or some items that receive higher ratings on average. Considering this, we can create a much better factorization model of the original data if we normalize the input matrices with user and item biases. One way of doing this is by subtracting the global average and objects deviations from the original rating matrices. More precisely, each dataset $R_{ij}^{(t)}$ is normalized as $\tilde{R}_{ij}^{(t)} = R_{ij}^{(t)} - \mu^{(t)} \mathbf{1} - \mathbf{b}_i^{(t)} \mathbf{1} - \mathbf{b}_j^{(t)}$, where $\mu^{(t)}$ denotes the overall average at the timestamp $t$, while $\mathbf{b}_i^{(t)} \in \mathbb{R}^{n \times 1}$ and $\mathbf{b}_j^{(t)} \in \mathbb{R}^{m \times 1}$ denote the biases of the object types $i$ and $j$ respectively for the timestamp $t$ (each entry represents the corresponding object’s deviation from the overall average).

Starting biases are commonly set to 0 and then updated during the factorization process. Alternatively, they can be calculated in advance which can result in a faster convergence. In SIMF we implement the second approach and estimate the initial biases using the method from [108], which computes the initial biases by minimizing the error for the baseline predictor (predictor that uses only the mean rating and biases). We calculate the initial biases as (for a given timestamp $t$):

$$
\hat{b}_j = \frac{\sum_{p \in R_{ij}(p)} (r_{up} - \mu)}{\lambda_1 + |R_{ij}(p)|},
$$

$$
\hat{b}_i = \frac{\sum_{p \in R_{ij}(u)} (r_{up} - \mu - b_j)}{\lambda_2 + |R_{ij}(u)|},
$$

where $R_{ij}(p)$ is the set of users who rated item $p$, $R_{ij}(u)$ is the set of items that were
Simultaneous incremental matrix factorization

rated by a user $u$, $r_{up}$ is the rating of a user $u$ for an item $p$, $\mu$ is the overall average rating and $\lambda_1$, $\lambda_2$ are the regularization parameters (for instance, typical values for the Netflix dataset are $\lambda_1 = 25$, $\lambda_2 = 10$ [108]).

Biases of object types are computed separately for each relation. However, they can be combined (one combined bias vector for one object type) in cases of similar relations with the same rating scales. This creates another layer of data fusion (upon biases) and can further boost the prediction accuracy of the model, especially in cases of high sparsity, where we do not have enough data from a single data source to properly estimate the bias of a given object type. Combined biases are then normally updated using appropriate update rules (they are updated every time the corresponding object type is updated).

Relation weights For better control and balance of the data fusion we include relation weights to our factorization process. We can manually weigh each relation/data stream $d_{ij} = (\mathcal{E}_i, \mathcal{E}_j)$ with the weight $a^{(t)}_{ij}$. Higher relation weights produce a higher derivative of this relation’s factors, shifting the factor change more to its favor. Thus, we can control how big of an effect a particular data source has on the final factorization model and allow for more stable updating.

For example, in the case of one main relation (relation on which we predict) and multiple auxiliary relations, the initial relation weights could be simply set as the ratios between the data stream length (number of relations in a given auxiliary stream, $l_{ij}$) and the length of the main data stream $l_m$, so that: $a^{(1)}_{ij} = \frac{l_m}{l_{ij}}$. Such weights would ensure that a particular auxiliary data stream with twice the frequency of the main data stream would contribute only with half strength to the factorization or the update, ensuring that more frequent updates from one stream would not skew the model towards them.

4.2.7 Final objective function

With addition of relation weights, biases, regularization and constraints, the final objective function is constructed as:

$$\min f(G^{(i)}, S^{(i)}) = \sum_{R^{(i)}_{ij} \in \mathcal{D}} a_{ij}^{(i)} \|W^{(i)}_{ij} \circ (R^{(i)}_{ij} - \mu^{(i)}1 - \hat{b}_i^{(i)}1 - \hat{b}_j^{(i)}1 - G^{(i)}_i S^{(i)}_j G^{(i)T}_j)\|_F^2$$

$$+ \max_{i,j} \sum_{t=1}^T tr(G^{(i)T}_i \Theta_j^{(i)(t)} G^{(i)}_i) \quad \Theta_j^{(i)(t)} = \Theta_j^{(i)(t)} = \lambda$$

(4.4)
This objective function can now be used to factorize multiple data streams of different relations with constraints, regularization and biases.

### 4.2.8 Initialization of factor matrices

Initialization of factor matrices can have an impact on the convergence speed and can decrease the probability of ending up in a local minima. A more instructive initialization can thus have an advantage over the basic one (such as sampling factors from $N(0, 1)$).

To achieve this, SIMF uses a modified version of the random Acol [109] initialization algorithm. Initial matrices $G_i^{(0)}$ are constructed by sampling random columns from the ratings matrix $R_i^{(0)}$ (or from multiple rating matrices at once if they are sparse) and averaging them, considering only non-zero elements to properly manage sparsity. As a result, this procedure yields dense factor matrices that resemble the original rating matrices. Middle factor matrices $S_{ij}^{(0)}$ are constructed with random initialization, using the normal distribution $N(0, 1/k_i + 1/k_j)$, where $k_i$ and $k_j$ are the ranks for $e_i$ and $e_j$, while the mean 0.1 is deliberately chosen small as the rating values are already normalized (with subtracted mean rating and object biases).

### 4.2.9 Updating the factorization

The proposed algorithm can incrementally update its factorization model in order to adapt to new changes in the data streams. These changes can represent new or updated ratings (values of input matrices $R_i^{(t)}$) or new objects (new rows or columns of input matrices $R_i^{(t)}$).

The main principle in updating the factorization system is that the individual ratings (individual input instances) only change a single row in the factor matrices during the factorization (apart from the small middle factor $S_{ij}^{(t)}$). In order to incorporate a new rating we therefore only need to change a relatively small part of our factorization model. Therefore, we can develop an efficient and fast incremental system. Some examples of the proposed update scheme can be found in Figures 4.2, 4.3 (addition of a value of an existing cell) and 4.4 (addition of a new item/column).

The update is achieved by applying a special set of update rules (in the case of NMF) or applying the same update rules as in the initial factorization (SGD). Both approaches are presented in the Section 4.3.
Figure 4.2: Factorization system in the timestep $t$. 

Simultaneous incremental matrix factorization
Figure 4.3
Factorization system in the timestep $t+1$. A new rating $R_{12}^{(t+1)}(5, 5)$ arrived in the first data stream, which resulted in the update of the fifth row of the factor $G_1^{(t+1)}$, fifth column of the factor $G_2^{(t+1)}$ and the whole factor matrix $S_{12}^{(t+1)}$. Apart from changes in the factorization $\hat{R}_{13}^{(t+1)}$, the data fusion also caused alteration of $\hat{R}_{13}^{(t+1)}$ and $\hat{R}_{42}^{(t+1)}$. 

\[
\begin{align*}
\hat{R}_{12}^{(t+1)} & \approx (\hat{R}_{12}^{(t)} + 1)S_{12}^{(t+1)} \\
\hat{R}_{13}^{(t+1)} & \approx (\hat{R}_{13}^{(t)} + 1)S_{13}^{(t+1)} \\
\hat{R}_{42}^{(t+1)} & \approx (\hat{R}_{42}^{(t)} + 1)(1)S_{42}^{(t+1)} \\
\end{align*}
\]
Figure 4.4
Factorization system in the timestep $t+2$. A new item (eighth row of the matrix $R_{12}^{(t+2)}$) was rated by the second user, which resulted in the update of the second row of the factor $G_{1}^{(t+2)}$, addition of the new column of the factor $G_{2}^{(t+2)}$, and the update of the whole factor matrix $S_{12}^{(t+2)}$. Consequentially, the whole eighth column of the relation $R_{42}^{(t+2)}$ can now be predicted.
4.2.10 Handling concept changes in multiple data streams

Since we model multiple data streams at once through data fusion, the dynamics of one stream can indirectly influence the common representation (factor model). These effects can range from increased update frequency or noise (which can skew the model too much towards the representation of that data stream) to concept drifts, that can occur at any stream at any time (we further analyze this effect in the evaluation in Section 4.4.1).

Suppose our system models multiple data streams, where one data stream is selected as the primary stream (the only stream on which we want to make predictions and therefore optimize), while others are set as auxiliary streams (on which we do not measure the error and are only present in the system in order to infer additional knowledge from data fusion). For example, our goal is to predict new ratings of movies (primary data stream) for a group of users, who also rate songs, books and restaurants (auxiliary data streams). Ideally, in such a scenario all concept drifts would be global: present in the same form on all data streams at the same time (e.g. a group of users start rating movies, songs, books and restaurants with higher ratings than before). In this way, the model would quickly adapt to new changes, as this information about the increase in the ratings would also be reinforced from the auxiliary data streams as well.

However, most of the times this is not the case, and in order to mitigate the negative effects from auxiliary data streams, SIMF can utilize a selective update procedure. There, the possible changes of shared factors are committed only if they improve the overall prediction accuracy (on the primary data stream). More precisely, SIMF can keep a sliding window (of independent samples) of the most recent data instances for every stream separately (or only for the primary relation, depending on the problem). If a particular update instance would then cause a change in a shared matrix, this change is applied only if the error on these sliding windows does not increase. In the case that the prediction error increases, the update can be reversed and discarded. In this way, the negative effects, such as concept drifts, are updated into the factorization of the primary relation only when this new concept reaches the primary data stream as well (because the sliding window error stops increasing). Since this procedure can slow down the global model adaptation (by rejecting the relevant effects) the number of samples in the sliding window should be kept short in order to take these changes into account sooner in the data stream.

Another way of controlling the stability of the factorization process is to introduce...
dynamic relation weights. In cases of increased update frequency we have already presented a solution in the form of the normalization of the relation weights by their length. Similarly, we can “penalize” data streams that experience abrupt concept drifts (that are not relevant to the main relation) by dynamically reducing their relation weight. In addition, we can then also “reward” positive effects by increasing the relation weight in the same manner.

By controlling the above mentioned processes we can effectively control the trade-off between model stability and adaptation to new concepts. Note that these proposed strategies were left out in our experiments and therefore still need to be properly evaluated in our future work.

Furthermore, the incremental nature of SIMF’s updates grants an inherent forgetting mechanism. Therefore, through the sequential application of updates (in the order of rating timestamps), newer ratings have a higher weight on factorization and vice versa. This is in contrast to batch factorization (and batch recomputation), where all training instances have the same contribution in the factorization (usually, all training instances are shuffled before applying different update rules).

4.2.11 Predictions

After the initial factorization or after the desired length of updates, we can make predictions (recommendations), i.e. completing the particular cell of the rating matrix $R_{ij}$ at a given timestamp $t$, by completing the following equation:

$$\hat{R}_{ij}(t) = G_i(t)(row)S_{ij}(t)G_j^T(col) + \mu(t) + \hat{b}_i(t)(row) + \hat{b}_j(t)(col),$$

where $row$ and $col$ are indices of the queried prediction, $G_i, G_j$ and $S_{ij}$ are the factors and $\hat{b}_i(t), \hat{b}_j(t)$ and $\mu(t)$ are the biases. If $row$ or $col$ represent a new, never seen object (for new rows or columns inserted into the input matrix $R_{ij}(t)$), we first initialize a new row for the corresponding factor matrices and bias vectors and then proceed with the update.
4.3 Deriving different update rules

In this section we present three gradient methods for deriving update rules to minimize the optimization problem from the Equation (4.4). Firstly, we introduce SIMF with nonnegativity constraint by deriving nonnegative multiplicative update rules (MUR), followed by the modified rules for incremental updates. Next, we present the standard gradient descent method and update rules. These rules are naturally suitable for incremental learning, so no modifications are required. Finally, we derive update rules for stochastic gradient descent, which feature faster learning and convergence in sparse environments. These update rules are independent of the time, that is why we omit the timestamps from the following derivations.

4.3.1 SIMF with nonnegative multiplicative update rules

The derivatives of our objective function can be decomposed into two strictly nonnegative parts. For instance: \( \frac{\partial f}{\partial G_i} = (\frac{\partial f}{\partial G_i})^+ - (\frac{\partial f}{\partial G_i})^- \), where operator \( ^+ \) retains only positive elements of the selected matrix and operator \( ^- \) retains only negative elements. All other elements are set to zero. From this decomposition we derive the multiplicative update rules for nonnegative matrix factorization as:

- \( S_{ij} \leftarrow S_{ij} \circ \left( \frac{\partial f}{\partial S_{ij}}^- \right) \left( \frac{\partial f}{\partial S_{ij}}^+ \right) \)
- \( G_i \leftarrow G_i \circ \left( \frac{\partial f}{\partial G_i}^- \right) \left( \frac{\partial f}{\partial G_i}^+ \right) \)
- \( G_j \leftarrow G_j \circ \left( \frac{\partial f}{\partial G_j}^- \right) \left( \frac{\partial f}{\partial G_j}^+ \right) \)

or if we expand these update rules:
We modify the above rules for incremental updating, where we update only parts of $(r, c, v)$ for relation $g$ where $v$ represents the $r$-th row of the matrix $G_r$, while $x_r$ and $x_c$ represent the $r$-th row and $c$-th column of the modified input.
matrix \( R_{ij} \) with the new value \( v \) (so that \( R_{ij}(r,c) = v \)). Note that matrices \( R_{ij} \) are not stored in memory, so the vectors \( x_r \) and \( x_c \) are calculated on the fly, using the factor model from the previous step \( G^{(t-1)} S^{(t-1)} G^{(t-1)^T} \).

### 4.3.2 SIMF with gradient descent

The most straightforward approach of minimizing the objective function is by gradient descent. The update rules for the factorization process can be written as:

\[
\begin{align*}
S_{ij} &\leftarrow S_{ij} - \eta \cdot \frac{\partial f}{\partial S_{ij}} \\
G_i &\leftarrow G_i - \eta \cdot \frac{\partial f}{\partial G_i} \\
G_j &\leftarrow G_j - \eta \cdot \frac{\partial f}{\partial G_j} \\
b_i &\leftarrow b_i - \eta \cdot \frac{\partial f}{\partial b_i} \\
b_j &\leftarrow b_j - \eta \cdot \frac{\partial f}{\partial b_j}
\end{align*}
\]

where \( \eta \) represents the learning rate. We rewrite these update rules as:

\[
\begin{align*}
S_{ij} &\leftarrow S_{ij} - \eta \cdot \alpha_{ij} \left( G_i^T (W_{ij} \circ G_j S_j^T) G_j - G_j^T (W_{ij} \circ R_{ij}) G_j \right) \\
G_i &\leftarrow G_i - \eta \cdot \alpha_{ij} \left( (W_{ij} \circ G_j S_j^T G_i) G_j S_j + \sum_t \Theta_i^{(t)} G_i \right) \\
G_j &\leftarrow G_j - \eta \cdot \alpha_{ij} \left( (W_{ij} \circ R_{ij}) G_i S_j + \sum_t \Theta_j^{(t)} G_j \right) \\
b_i &\leftarrow b_i + \eta \cdot (e_{ij} 1) \\
b_j &\leftarrow b_j + \eta \cdot (1 e_{ij})
\end{align*}
\]

These update rules are used sequentially for each relation \( R_{ij} \) separately. Afterwards, the same procedure can be used for incremental updating. Because we discard the original data, we must construct new relation matrices \( R_{ij} \) with newly arrived data and update the weighting (mask) matrices \( W_{ij} \). In this case, updating in batches is preferred.
4.3.3 SIMF with stochastic gradient descent

Finally, we present the stochastic gradient descent for factorizing the SIMF model. We directly minimize the prediction error only on known (non-zero) ratings, thus we omit the weighting matrices $W_{ij}$:

$$
\min f(G,S) = \frac{1}{2} \sum_{(u,p) \in R_{ij}} \sum_{R_{ij} \in D} (r_{up} - \mu - b_{iu} - b_{jp} - G_{iu} S_{ij} G_{jp}^T)^2 + \lambda (\|G_{iu}\|^2 + \|G_{jp}\|^2 + \|S_{ij}\|^2 + b_{iu}^2 + b_{jp}^2),
$$

where the set of $(u, p)$ represents all known ratings for a given matrix $R_{ij}$. The derivatives of the upper objective function are equal to:

$$
e_{up} = r_{up} - \mu - b_{iu} - b_{jp} - G_{iu} S_{ij} G_{jp}^T
$$
$$
\frac{\partial f}{\partial G_{iu}} = -(e_{up} \cdot G_{jp} S_{ij}^T - \lambda \cdot G_{iu})
$$
$$
\frac{\partial f}{\partial G_{jp}} = -(e_{up} \cdot G_{iu} S_{ij} - \lambda \cdot G_{jp})
$$
$$
\frac{\partial f}{\partial S_{ij}} = -(e_{up} \cdot G_{iu}^T G_{jp} - \lambda \cdot S_{ij})
$$

Therefore, the update rules for SIMF using SGD are defined as:

$$
e_{up} \leftarrow r_{up} - \mu - b_{iu} - b_{jp} - G_{iu} S_{ij} G_{jp}^T
$$
$$
G_{iu} \leftarrow G_{iu} + \eta \cdot (e_{up} \cdot G_{jp} S_{ij}^T - \lambda \cdot G_{iu})
$$
$$
G_{jp} \leftarrow G_{jp} + \eta \cdot (e_{up} \cdot G_{iu} S_{ij} - \lambda \cdot G_{jp})
$$
$$
S_{ij} \leftarrow S_{ij} + \eta \cdot (e_{up} \cdot G_{iu}^T G_{jp} - \lambda \cdot S_{ij})
$$
$$
b_{iu} \leftarrow b_{iu} + \eta \cdot (e_{up} - \lambda \cdot b_{iu})
$$
$$
b_{jp} \leftarrow b_{jp} + \eta \cdot (e_{up} - \lambda \cdot b_{jp})
$$

where $\eta$ is the learning rate and $\lambda$ is the regularization parameter. Again, the incremental nature of the SGD allows us to use the same update rules for incremental updating, without any modifications.
Input to SIMF is a list $\mathcal{D}$ containing data streams (relations) that relate pairs of object types $(\epsilon_i, \epsilon_j)$ by relation matrices $R^{(t)}_{ij}$. Each object type must have defined its own rank for the factorization. The algorithm works by firstly factorizing the initial data (data up to some pre-defined timestamp $t = 0$) from all input matrices $R^{(0)}_{ij}$, into the initial factorization model. Then, SIMF continuously updates its model, using new instances from data streams (which can be collected in batches). These updates can incorporate and update existing matrix cells (e.g. ratings), new rows (e.g. users) or new columns (e.g. items). The pseudocode of the SIMF algorithm is shown in Algorithm 4.

**Algorithm 4:** Simultaneous incremental matrix factorization. Source code of SIMF is available at https://github.com/MartinJakomin/SIMF/

```
input : list of relations R
output : factor matrices $G^{(t)}$, $S^{(t)}$ and bias matrix $B^{(t)}$

$G^{(0)}$, $S^{(0)}$, $B^{(0)} = \text{model} = \text{random\_initialization}(R);$  
for $R_{ij}$ in R do
  model = $\text{factorize}(R^{(t)}_{ij})$;  # Initial factorization
end
while true do
  for new_data in R do
    model.expand_factors(new_data);
    model = $\text{factorize}(new\_data)$;  # Batch update
  end
end
return $G^{(t)}$, $S^{(t)}$, $B^{(t)}$
```
4.4 Evaluation on synthetic data streams

The evaluation of the proposed method was done in a way to highlight its data fusion capabilities in the streaming environment. Particularly, we focused on the problem of recommender systems (note that incremental matrix factorization can be used for a variety of different problems, such as clustering, classification, dimensionality reduction and much more). That is why we created a scenario of multiple (inter-related) relations of rating data that change through time. Changes include addition of new objects and changes in rating distribution.

To compare SIMF with other algorithms we used the regularized matrix two-factorization algorithm (RMF) [8], and the average predictor (Average) as a baseline. The Average model holds the running mean value of the stream and returns it for every query, taking into account all potential object biases. Therefore, this model provides fairly good recommendations and a dynamic baseline for other algorithms. Both SIMF and RMF were factorized using the stochastic gradient descent with the same parameters which were chosen and optimized empirically (rank, initialization, learning rate, regularization etc.). Initial biases were computed once and used in all models and all relation weights were set to 1. On account of that, we can assume that any possible gain in predictive performance between RMF and SIMF can be contributed to the difference in the factorization (two-factorization versus tri-factorization), while differences between multiple instances of the SIMF model trained with different number of data streams can be contributed to data fusion.

Evaluation was done in two phases. In the first phase we used synthetic data streams to analyze accuracy and robustness of our method and in the second phase we repeated the evaluation on a real recommendation problem (Section 4.5).

Firstly, we tested if our proposed method successfully factorizes multiple data streams and maintains high prediction accuracy in the dynamic environment (changes in the ratings distributions, addition of new objects, etc.). For this task we used synthetic data streams, as they provide a controlled evaluation environment. Using GIDS [110] (explained in Chapter 3) we created three artificial data streams that resemble rating structure found in real recommendation problems. Three datasets were created with the same structure as seen in example from Figure 4.1, one main relation and two auxiliary relations, each sharing different object types from the main relation.
The primary evaluation was done as follows. Firstly, all three datasets were split into the initial training set (20% of the data stream) and the streaming test set (the remaining 80% of the stream). The training sets were used to factorize the models and the streaming test sets were used to incrementally update them. Since SIMF can utilize data fusion, additional models were also trained simultaneously on multiple data streams (while RMF was trained only on the first data stream).

In the streaming phase we performed the prequential evaluation by evaluating accuracy for examples that will yet become future updates of the model (explained in Chapter 2.3.1). The remaining parts of the data streams were split by their timestamp (therefore, each update batch $C_i$ included all the instances with the timestamp $t_i$). In total there were 800 distinct update batches. Root mean squared error (RMSE) was calculated between model predictions and real values from the set $C_i$. After, this error was normalized with the error from the baseline algorithm (Average) to calculate the relative RMSE (RRMSE) and to determine the usefulness of the particular model (from 0 – perfect prediction accuracy, to 1 or above – equal or worse prediction accuracy than the baseline). Following the error estimation, all models were updated with the ratings from $C_i$ (models capable of data fusion used all data points in the batch, while RMF only used the updates from the first relation).

Figure 4.5 displays the prequential RRMSE through time for three models: SIMF (our proposed method, trained only on the main data stream), SIMF(3) (variation of the SIMF that was trained on all three datasets) and RMF. It should be noted that prequential analysis shows the (averaged) cumulative sum of the loss function with the fading factors, which means that in the beginning of the stream, this average is not well established and is distorted by the initial errors (either over or under estimated). Therefore one can expect sudden changes of the error in the beginning of the stream.
The results show a similar shape of prediction error curves for all three recommender models. However, the initial RRMSE is lower for SIMF models (0.831 for SIMF(3), 0.844 for SIMF in contrast to 0.883 for RMF). Throughout the stream the collective factorization model SIMF(3) achieved approximately 1% better results than its counterpart SIMF that uses only one data stream for updating.

4.4.1 Robustness to concept drift

To evaluate the robustness of SIMF we repeated the previous experiment with the addition of introduced artificial concept drifts. Using our generator we have simulated a “sudden” change in rating values distribution (concept) that can happen at any time in the data stream. We implemented this change in two directions, the positive (by raising the rating mean) and the negative (by lowering the rating mean). We expected that concept drifts in additional data streams can have a negative effect on predictions from the main relation, due to their participation in data fusion.

Because we have three data streams in total, we tested all possible combinations of concept drifts. Therefore, 27 different sets of data streams were constructed: {{0, 0, 0}, {0, 1, 0}, {0, 0, 1}, {−1, 0, 0},..., {−1, −1, −1}}, where each data stream is characterized with either 0 (no concept drift was present in a given stream), 1 (concept drift in the positive direction) or −1 (concept drift in the negative direction). One such example of a generated set of data streams with concept drifts in the direction of [1, 0, −1] can be seen in Figure 4.6.
Figure 4.6
Example of simultaneous concept drifts with scenario \([1, 0, -1]\). The blue lines represent rating values, the green line separates the initial training phase and the rest of the stream, while the red line represents the exact time of the sudden concept drift.

After 27 sub-experiments we chose the best and the worst result for every model separately (overall lowest and highest error through time). Different results (pairs of algorithm’s prequential errors) were compared to each other with Q-estimate (as explained in Chapter 2.3.1). Figure 4.7 shows the best and worst prequential RRMSE through time for SIMF, SIMF(3) and RMF.

Figure 4.7
The range of the best and the worst case of RRMSE, given different concept drift scenarios for SIMF, SIMF(3) and RMF. For every case, each model starts with the same initial factorization and the same first few updates, therefore the error remains the same until the concept drift at timestamp 200. In this scenario SIMF(3) achieves the lowest error.

The best results for all models were achieved when no concept drift was present in data streams \([0, 0, 0]\). In Figure 4.7, this corresponds to the lowest error curves of each
model. Although at the end of the experiment the upper error for SIMF and SIMF(3) indicates slightly worse performance than the best result of RMF, the overall performance of SIMF and SIMF(3) still exhibits robustness in terms of the achieved RRMSE range. Namely by fusion of multiple streams, each being influenced by its own concept drift, one could expect higher deterioration of the performance (higher than in the case of non data fusion algorithm, such as RMF), which is not the case in this experiment. Nevertheless, these results indicate that in presence of concept drifts, the accuracy of the predictive model would rely strongly on successful underlying concept drift detector.

In our further evaluation we included two baseline models that do not update their factors at all (denoted with SIMF control and RMF control), with the aim to compare the scale of error of non-updating models. These results can be seen in Figure 4.8. As expected, the error constantly rises for non-updating models and in the case of severe concept drifts the prediction error surpasses the error of the Average predictor (relative RMSE over 1).

4.5 Evaluation on real data streams

Next, we continued our evaluation on a real-life problem. We choose to model data streams from the Yelp challenge\(^1\) (the recommendation problem within the Yelp applic-

\(^1\)https://www.yelp.com/dataset/challenge
ation). Particularly, we focused on the ratings dataset, which contains information on how Yelp users rated (by giving “stars” from 1 to 5) different establishments/businesses (restaurants, bars, clubs, shops, etc.).

We constructed a subset of the Yelp datasets by taking the last year of ratings (from 2. 7. 2017 till 2. 7. 2018) for 3 different businesses: restaurants, bars and hotels. We chose restaurants as our main relation (since it was the densest), bars as the second relation, and hotels as a third parallel relation. Ratings from all these businesses are therefore (hiddenly) connected by the common preferences of similar users. One year of ratings resulted in over 275,000 unique users, rating over 37,000 restaurants, 2,400 bars and 5,100 hotels. Around 650,000 ratings were collected, the majority of which were from users rating restaurants (more than 550,000), making the data streams sparse and unbalanced.

In total, three data streams were created in order to utilize the data fusion upon the common dimension of users: $\text{users}/\text{restaurants}$, $\text{users}/\text{bars}$ and $\text{users}/\text{hotels}$ (therefore our method could model these data streams with a single common factor for users and 3 other factors for different businesses).

In order to properly evaluate our proposed method on real-life data streams, we have compared it to several single and multi-sourced methods from related work:

- For comparison with matrix two-factorization we have again included the RMF model. Once more we have matched all parameters with SIMF.

- For comparison with relational learning via matrix factorization, we have included the collective matrix factorization [59] (CMF). We implemented the CMF as the multi-relational extension of the RMF model.

- For comparison with state-of-the-art recommender systems we have included Factorization machines [75] (FM) and Deep AutoEncoders [111] (DAE). For DAE we chose two layers for encoder (both with 128 hidden units) and two layers for decoder (both with 128 hidden units), while the representation layer consisted of 256 hidden units with 0.8 dropout rate (to tackle overfitting in a very sparse environment). This architecture was chosen empirically, while using the guidelines from [111].

Since all datasets relate their corresponding businesses with the same object type (Yelp users), we were able to modify all models to utilize data fusion. SIMF(3), CMF(3) and
FM(3) can take advantage of the intermediate data integration and model multiple datasets directly. SIMF(3) and CMF(3) do it by using the collective matrix factorization, while FM(3) models the information from parallel data streams as an added context (appended features).

On the other hand, RMF(3*) and DAE(3*) can not model multiple relations at the same time. That is why we have created a special combined dataset of all relations by stacking all data matrices together (to form a relation users/business). Here, * denotes the early data integration. Note that this was not possible in the previous experiments with synthetic datasets, due to the different factorization system (there did not exist a single object type, that related all possible rating matrices).

We took approximately 10% of instances for initial factorization and saved the rest for the streaming phase, where the data streams were split into 1 day intervals (from 21. 10. 2017 to 2. 7. 2018). As usual, we have used the data instances from these intervals to first measure the relative RMSE and then to update the models. Figure 4.9 shows the results for SIMF, RMF, FM and DAE, trained on the relation users/restaurants, while Figure 4.10 additionally shows the results for SIMF(3), CMF(3), RMF(3*), FM(3) and DAE(3*), which were trained on all three data streams at once.

![Figure 4.9](image)

Prequential relative RMSE of Yelp predictions (restaurants) for single-sourced SIMF, RMF, DAE and FM. Results indicate that SIMF achieves the lowest overall error.
The results show similar behavior as with synthetic data streams. The SIMF method surpassed the RMF in terms of RRMSE, while the addition of two auxiliary data streams (user ratings of bars and hotels) further boosted the SIMF’s prediction accuracy and lowered the overall error. However, this was not the case with RMF, where addition of two new relations contributed to higher overall error. The two-factorization approach RMF could not benefit from data fusion, neither in the form of collective factorization CMF(3) or in the form of early data integration RMF(3*). Both FM and DAE successfully utilized the data stream fusion and increased their prediction accuracy. However, their performance was still not as good as the matrix factorization models, which performed the best in our evaluation scenario. This was expected, due to the extreme sparsity in data and overall low number of training instances.

Considering that Yelp is a hard problem, we can view the SIMF’s outcome as a favorable one. In terms of the raw RMSE value, SIMF experienced an error lower than 1.34, which can be seen in Figure 4.11 showing the raw (prequential) RMSE values for all models.

Although we can find lower RMSE values in the related work [112–115] (average RMSE between 1.4 and 1.1), we cannot directly compare these results due to the large differences in problem definition and evaluation framework. In particular, there are major differences in the problem selection (different features and targets), the sample (different sample size and data instances), the evaluation procedure (cross validation or time-
Simultaneous incremental matrix factorization, the train/test split (usually 90/10 or 80/20, in contrast to our 10/90), etc.

![Figure 4.11](image)

**Figure 4.11**
Prequential RMSE of Yelp predictions (restaurants) for all models. Our proposed model SIMF achieves the lowest RMSE among all other recommender systems (1.34).

### 4.5.1 Evaluation under the cold-start

To further evaluate the stream fusion capabilities of our proposed approach, we reversed the prediction problem in the next experiment. Now we focused on predicting user ratings from bars (sparse relation with users), while using restaurant (dense relation) and hotel ratings as an additional information. This problem seemed to be harder due to increased sparsity and the cold-start problem (only 2,000 ratings of bars were present in the initial training phase). Nevertheless, we expected that utilization of data fusion could alleviate the cold-start problem due to abundance of restaurants ratings.

In this experiment we enabled the bias fusion for our model SIMF(3). It now uses a single combined bias vector for every object type (as explained in Subsection 4.2.6). We assumed that this fusion of biases would further alleviate the cold-start problem (in contrast to other models that try to compute the bias from a set of sparse rating matrices separately).

The results of this experiment are shown in Figures 4.12 and 4.13. While single-sourced SIMF and RMF achieved similar average RRMSE through time, the multi-sourced variation of our proposed method SIMF(3), with addition of bias fusion, clearly surpassed both of the single-sourced models and achieved the overall best performance.
Figure 4.12
Frequent relative RMSE of Yelp predictions (bars) for single-sourced SIMF, RMF, DAE and FM. Results indicate that SIMF and RMF achieve the lowest overall error.

Figure 4.13
Frequent relative RMSE of Yelp predictions (bars) for single-sourced SIMF, RMF, DAE and FM, and multiple-sourced (highlighted lines) SIMF(3), CMF(3), RMF(3*), DAE(3*) and FM(3). Results indicate that SIMF better integrates multiple datasets and achieves the lowest overall error.

The collective matrix two-factorization CMF(3) achieved overall similar result as the single-sourced counterpart (RMF), while RMF(3*) experienced a large increase in the prediction error with addition of two new data streams. Similar to the previous experiment, FM and DAE lowered their prediction error with additional relations but were unable to match the accuracy of the matrix factorization.
4.6 Summary

In this chapter we presented a novel method (SIMF) for simultaneous learning from data streams. We proposed one application in the form of a streaming recommender system, capable of fusing multiple data streams in order to increase the recommendation accuracy and alleviate the cold-start problem.

SIMF uses an incremental, collective and weighted matrix tri-factorization to firstly factorize all data streams into smaller factor matrices and then updates them using only newly arrived data, without storing any previous information. SIMF also incorporates additional information in the form of object constraints, regularization, bias and relation weights.

The proposed approach was extensively evaluated on both synthetic and real-life problems where it was shown that our incremental approach surpasses the conventional methods, while adding more data streams further decreases the prediction error.

Obtained results indicate that SIMF can infer and combine the information from other auxiliary sources (relations) in order to augment and boost the prediction accuracy. Said differently, when there is not enough data to properly model the object factors (latent spaces and biases) of a particular relation, SIMF can determine them by learning from other data sources. This is particularly evident in the recommender system domain, where we often deal with highly sparse and cold-start problems.

SIMF’s ability of modeling real-time streaming data and versatility of matrix factorization allows for a simple adaptation to various other problems beyond the scope of this thesis (social networks, business, medicine, biology, etc.).
Conclusion
Abundance of data calls for highly scalable algorithms able to learn in an on-line fashion with continuous user feedback. Furthermore, we have shown that successful utilization of data fusion upon data streams brings higher predictive accuracy and helps relieve the cold-start problems. However, the data abundance is not universal as there exist many areas of machine learning where the data is scarce and hard to collect. Furthermore, incrementally handling models upon multiple heterogeneous data streams remains an open challenge.

In this work we presented a novel simultaneous incremental matrix tri-factorization technique (SIMF) for successful modeling of multiple data streams, and presented another innovative approach for generating synthetic inter-connected data streams (GIDS) for evaluating various incremental and data fusion algorithms.

In the first part we introduced GIDS, a generator of synthetic inter-dependent data streams. The proposed approach works by creating multiple object clusters (one for each object type) that are inter-connected, thus constructing a clear structure in the data, mimicking statistical and modeling properties of real data. In this way synthesized data streams offer fair and thorough evaluation of different incremental and data fusion algorithms, such as recommender systems, and can help to further develop and test those algorithms for specific tasks.

The evaluation showed that data streams generated by GIDS resemble real-life data streams (such as the MovieLens dataset) in terms of data statistics (through time) and in terms of modeling capabilities of numerous algorithms. We evaluated different recommender systems on three sets of data (MovieLens, Yahoo music and Yelp) and two sets of synthetically generated data (GIDS and baseline generator Random) and showed that recommender systems achieve comparable performance on real-life datasets and on those generated by GIDS, for static and on-line learning. In comparison, the same algorithms performed much worse on datasets generated by the baseline data generator. Further evaluations also presented data dependency found in data streams generated by GIDS, as data fusion algorithms could learn more (lowered their prediction error) from additional datasets that were added to the learning process.

Finally, we presented SIMF, a novel incremental method for fusing multiple data streams via the collective matrix tri-factorization. SIMF can model streaming data in real-time and adapt to new objects and new concepts and thus provides a convenient way to tackle various matrix factorization problems, such as on-line recommender systems, clustering, classification, modeling of social networks, etc. SIMF works by collectively
factorizing multiple weighted relations into a factor representation (model) and then uses newly arrived data in the stream to incrementally update its factor model.

We evaluated the proposed approach using synthetically generated data streams and showed that inclusion of auxiliary relations (additional data streams) improves prediction accuracy. Additionally, we demonstrated the method’s robustness to concept drifts that may occur in different parts of the factorization system. Evaluation on the Yelp data-sets confirmed the results from synthetic data streams. The recommendation accuracy of the densest relation (recommending restaurants to users) can be improved by collectively factorizing multiple data streams (rating of restaurants, bars and hotels), while the cold-start problem of recommending on the sparsest relation (recommending bars) can be greatly alleviated using the same procedure.

5.0.1 Future work

The future work is split into two parts, each for its own presented algorithm.

For SIMF, future work consists of solving problems related to incremental learning. One such goal is a better adaption to concept drifts in terms of an early detection by observing the error directly on the objective function or on a holdout set. The second objective is a dynamic model adaptation by utilizing selective updates and dynamic relation weights on data streams that experience concept drifts or loss in accuracy. The third objective is to implement a better forgetting mechanism (by applying special forgetting functions at every incremental step) and ability to model seasonal effects. The fourth objective concerns with adaptation to multi-target prediction across multiple relations (predicting on multiple data streams at once). Further goals consist of better model sustainability and allowing for complete reconstruction and recomputation of the model or switching between several independent and parallel instances. Finally, more comprehensive evaluation is needed (several real-life recommendation problems with more heterogeneous data sources) and an adaptation of the proposed methodology to tackle other matrix factorization problems.

As regards to the GIDS data generator, future work consist of creating more realistic time dependency and more realistic rating patterns. We plan to additionally impose the time dependency to the object cluster level, thus creating scenarios where object clusters appear at a certain point in time and then gradually grow or shrink. Our second objective
is to modify rating function to consider both object clusters of corresponding object pair as it is currently only dependent on the origin object’s rating PDF and the concept drift function.
BIBLIOGRAPHY


Bibliography


Inkrementalna matrična faktorizacija za hkratno učenje iz vzporednih podatkovnih tokov

Martin Jakomin

doktorska disertacija
predana
Fakulteti za računalništvo in informatiko
kot del izpolnjevanja pogojev za pridobitev naziva
doktor znanosti
s področja
računalništva in informatike

razširjeni povzetek

Ljubljana, 2019
Matrična faktorizacija je izkazala kot uporabna in zanesljiva metoda za implementacijo obsežnih aplikacij strojnega učenja, kot so na primer priporočilni sistemi. Težave z redkostjo podatkov in problem hladnega zagona se lahko posredno omilijo z uporabo več heterogenih virov podatkov, hkrati pa uspešna uporaba zlivanja podatkov doprinosi večjo točnost napovedi. Za vsakodnevne aplikacije, na primer take s stalnimi povratnimi informacijami uporabnikov, ostaja inkrementalno posodabljanje modelov, naučenih na več podatkovnih tokovih, ključen in le delno rešen problem.


Predlagana metodologija ponuja pomoč pri razvoju algoritmov za sočasno modeliranje podatkovnih tokov v realnem času. Poleg priporočilnih sistemov pa vsestransko matrične faktorizacije omogoča njeno uporabnost za reševanje številnih drugih problemov strojnega učenja, kot so zmanjševanje dimenzionalnosti, gručenje in klasifikacija.

Generiranje sintetičnih podatkovnih tokov

Obstaja več področij in problemov, kjer je količina dostopnih in označenih podatkov redka. To je posledica več razlogov, npr. redkosti nekaterih dogodkov (redke bolezni, okvare strojev itd.), dragega zbiranja podatkov (veliki fizikalni ali farmakološki poskusi), težav z zasebnostjo (medicinski ali genetski podatki) ali poslovnih skrivnosti (zgodovina nakupov na Amazonu itd.).

Generatorji sintetičnih podatkov nam lahko pomagajo ublažiti problem pomanjkanja podatkov z nadzorovanim ustvarjanjem velikih količin podatkovnih primerov. Ti
Dodatno generirani podatki nam lahko pomagajo v različnih fazah razvoja naših algoritmov. Dodatni podatki v fazi učenja lahko vodijo do boljših modelov in lahko pomagajo pri učenju parametrov, medtem ko lahko večji nabor testnih podatkov vodi v strožje ocenjevanje in boljše primerjavo in izbiro naših modelov. Poleg tega lahko generatorji sintetični podatkov generirajo specializirane in specifične podatkovne množice. Tako lahko preizkusimo robustnost algoritma na točno določenih robnih primerih ali redkih dogodkih, ki se skoraj nikoli ne zgodijo v realnih podatkih.

V disertaciji predstavimo nov sintetični generator za medsebojno odvisne podatkovne tokove (GIDS), ki lahko generira več časovnih in medsebojno odvisnih podatkovnih relacij. GIDS deluje tako, da simulira več sklopov gruč (skupine objektov) različnih vrst in povezave (relacije) med temi gručami. Na ta način GIDS posnema resnične probleme, kjer lahko takšno strukturo tudi opazimo; kot na primer pri izbiranju s sodelovanjem (collaborative filtering), kjer uporabniki prejmejo priporočila za izdelke, ki so bili v preteklosti všeč ostalim uporabnikom s podobnim okusom (na primer uporabniki znotraj iste gruče objektov).

Pri razvijanju našega generatorja podatkovnih tokov je bil glavni cilj posnemati ustrezne lastnosti, ki so prisotne v dejanskih podatkih iz resničnega sveta:

- realni odnosi – relacije med skupinami različnih objektov (npr. realni vzorec uporabniških ocen),
- skrite korelace med več podatkovnimi tokovi (skupna dimenzija v množici več relacij, npr. kadar isti uporabniki ocenjujejo različne entitete: filme, pesmi, restavracije itd.),
- realna porazdelitev podatkov (porazdelitev vrednosti ocen, število ocen na posamezno vrstico in stolpec se tesno ujema s tistimi, ki jih najdemo v dejanskih podatkih),
- spremenljiva redkost in ocenjevanje (nekateri problemi imajo veliko ocen različnih velikosti, spet drugi imajo malo ocen, ki so binarne),
- časovne odvisnosti v podatkih in ustreznih podatkovnih tokov (generirane podatke je mogoče simulirati kot realni podatkovni tok),
- realistične spremenljive koncepte v podatkih (ocene lahko odražajo smiselne spremenljive koncepte skozi čas),
- realistična rast novih entitet (npr. stabilna rast novih uporabnikov).
Glavna ideja predstavljenega generatorja sintetičnih podatkovnih tokov je gručna struktura, podobna tisti, predstavljena v delu [83]. Generator simulira scenarije, pri katerih skupine podobnih uporabnikov ocenjujejo skupine podobnih predmetov na enak način. To naredi tako, da dodeli objekte določenega tipa v različne gruče, nato pa te gruče poveže z gručami nasprotnega tipa objektov. Pozneje vzorči (brez ponavljanja) določeno število ocen glede na verjetnost, da je gruča točno določenega uporabnika povezana z gručo točno določenega predmeta. Rezultat postopka so generirani trojčki \{uporabnik, predmet, ocena\}, ki jih predstavimo z matriko ocen, kjer vsaka neničelna celica opisuje odnos med določenim uporabnikom (vrstico) in predmetom (stolpec).

Predpostavimo, da imamo sistem s \(k\) tipov objektov \(\varepsilon = \{\varepsilon_1, \ldots, \varepsilon_k\}\) in z \(l\) relacij (ki so lahko asimetrične ali manjkajoče \(\Rightarrow l \leq k^2\)), ki vsaka povezuje en par tipov objektov \(\varepsilon_i\) in \(\varepsilon_j\). Primer take strukture prikazuje Slika 1.

\[\text{Slika 1}\]
Gručna struktura sintetičnih podatkov. Slika prikazuje \(k\) tipov objektov \(\varepsilon_i\), od katerih ima vsak svojo skupino gruč (in objektov v njih, predstawljenih s kvadratki, krogci in trikotniki) in povezavami med različnimi relacijami (med različnimi pari gruč). Prikazane so samo povezave z neničelnimi utežmi (pozitivne verjetnosti povezav).

Slika prikazuje predlagano gručno strukturo, kjer generator zgradi skupine gruč za vsak tip objektov posebej glede na vnaprej določeno porazdelitev (funkcija gostote verjetnosti). Nato se določijo med-gručne uteži (inter-cluster connectivity weights) glede na vnaprej podano porazdelitev. Ker objekte v gruče vzorčimo samo enkrat, vse relacije uporabljajo enake postavitve objektov v gruče ter tako ustvarjajo in krepijo skrite povezave.
med različnimi objekti.

Po določitvi opisane strukture lahko za vsako kombinacijo uporabnik/predmet določimo verjetnost povezave (verjetnost, da je nek uporabnik ocenil določen predmet) z vzorčenjem glede na podane med-gručne uteži ter določimo vrednost končne ocene (moč re- lacije) z vzorčenjem iz vnaprej generirane ocenjevalne funkcije. Ta ocenjevalna funkcija je določena za vsako gručo posebej in določa, kako pripadniki gruč ocenjujejo predmete. Ker je ta funkcija enaka za vse relacije, ki vsebujejo to določeno gručo, to dodatno krepi skrite povezave oziroma skrito strukturo v končnih podatkih.

Za dodano časovno odvisnost vzorčenim podatkom lahko vsako oceno dopolnimo s časovno značko \( t \) (naravno število, ki označuje, kdaj se je posamezna ocena pojavila v sistemu). Te časovne značke vzorčimo glede na vnaprej podano porazdelitev, ki lahko po- snema različne trendy (visoko ali nizko začetno število uporabnikov, eksponentno rast novih predmetov itd.). Opcijsko lahko v sistem dodamo tudi funkcijo spremnjanja kon- ceptov, kar doda novo razsežnost podatkom (na primer, simulacija padanja ocen starim predmetom ali porast ocen trendnim predmetom).

Naša glavna naloga je bila generiranje sintetičnih podatkov, ki so kar se da podobni tistim iz realnega sveta, kar omogoča oblikovanje realističnega okolja za ocenjevanje in izpold- njevanje različnih inkrementalnih algoritmov in algoritmov za zlivanje podatkov.

Predlagani pristop smo evalvirali v dveh delih. V prvem delu evalvacije smo primerja- li statistične lastnosti generiranih podatkov (skozi čas) s podatki iz podatkovne množice MovieLens (problem priporočanja). Kot primerjavo smo v evalvacijo vključili enostavni generator sintetičnih podatkov (imenovan Random), ki generira množice \( \{ \text{uporabnik, predmet, ocena, časovna točka} \} \) z vzorčenjem iz enakomerne porazdelitve. Evalvacija je pokazala, da naš predlagani pristop uspešno generira podatke, ki so podobni tistim iz množice MovieLens, glede na porazdelitev vrednosti ocen in porazdelitev števila ocen glede na posamezno vrstico ali stolpec. Prav tako pa se te porazdelitve obnašajo enako skozi čas, kar posredno ponazarja podobno rast števila novih uporabnikov oziroma novih predmetov, ter podoben trend ocenjevanja (povprečje in varianca vrednosti ocen) skozi čas.

V drugem delu evalvacije smo preverili, kako se različni priporočilni sistemi obna- jo (glede na točnost napovedi) na sintetičnih podatkih in kako na podatkih iz realne- ga sveta. Uporabili smo tri znane množice podatkov, MovieLens, Yahoo music in Yelp in pet priporočilnih sistemov, enostavno povprečje \( (\text{Average}) \), regularizirano matrično
faktorizacijo, nenegativno matrično faktorizacijo, verjetnostno matrično faktorizacijo in metodo $k$ najbližjih sosedov. Z uporabo prečnega preverjanja smo pokazali, da testirani priporočilni sistemi dosegajo boljše rezultate (večja točnost napovedi) na podatkovnih množicah iz realnega sveta in na podatkovnih množicah, ki so bile generirane z uporabo predlaganega pristopa GIDS. To se je pokazalo na vseh statičnih domenah in tudi pri naslednjem eksperimentu, kjer smo ocenjevali priporočilne sisteme v dinamičnem okolju (na podatkovnih tokovih). Hkrati smo pokazali, da generiranje podatkov z enostavnim generatorjem (Random) ni primerno, saj tam priporočilni sistemi delujejo še slabše kot zelo enostavni modeli (ki ocenijo vse relacije s povprečno vrednostjo).

V tretjem delu evalvacije smo pokazali, da GIDS generira podatke s skrito strukturo (informacijo), saj algoritmi zlivanja podatkov zmanjšajo svojo napak na podatkovnih zbirk naenkrat.

**Sočasna in inkrementalna matrična faktorizacija**

V disertaciji predlagamo novo metodo za hkratno modeliranje več heterogenih podatkovnih tokov (SIMF). Z uporabo kolektivne matrične tri-faktorizacije razgradimo podane relacijske matrike (ki povezujejo do dva različna tipa objektov) v tri manjše faktorske matrike, ki predstavljajo latentne predstavitve teh tipov objektov (uporabimo samo eno deljeno faktorsko matriko za vsak tip objekta). Ta faktorski model nato postopoma (inkrementalno) posodabljamo z novimi podatki v podatkovnem toku, ne da bi pri tem hranili stare podatke. Ta inkrementalna posodobitev omogoča dodajanje novih objektov in hitrejše prilagajanje spremembam in novim konceptom v podatkih, medtem ko zlivanje več podatkovnih tokov omogoča večjo napovedno točnost.

Cilj matrične tri-faktorizacije je poiskati tri matrike $G_1$, $G_2$ in $S$ tako, da:

$$R \approx G_1 S G_2^T$$

SIMF reši ta problem z minimizacijo posebne funkcije izgube (objective loss function), ki smo jo skonstruirali za potrebe sočasne inkrementalne matrične faktorizacije. Na začetku definiramo matrično tri-faktorizacijo kot minimizacijo Frobeniusove norme:

$$\min f(G_1, S, G_2) = \|R - G_1 S G_2^T\|_F^2,$$
kjer $R \in \mathbb{R}^{n \times m}$ predstavlja matriko ocen, ki povezuje dva tipa objektov (na primer uporabnike in filme), $G_1 \in \mathbb{R}^{n \times k_1}$ in $G_2 \in \mathbb{R}^{m \times k_2}$ predstavljata skrita koncepta za ta tipa objektov in matrika $S \in \mathbb{R}^{k_1 \times k_2}$, ki predstavlja skrite povezave med tema faktorskima prostoroma. To funkcijo lahko nato minimiziramo z uporabo multiplikativnih pravil \[40\, 69\], s pomočjo SVD-ja \[13\], gradientnega in stohastičnega spusta \[8\], ALS-ja \[8\], koordinatnega spusta \[105\] ali kakšne druge metode optimizacije. Večina teh metod omogoča tudi vključitev dodatnih omejitev, kot so nenegativnost ali ortogonalnost (odvisno od narave problema).

Naj bo $\mathcal{D}$ zbirka $n$ podatkovnih tokov $d_1, \ldots, d_n$ in $\mathcal{O}$ zbirka $r$ različnih tipov objektov $\mathcal{E}_1, \ldots, \mathcal{E}_r$. Potem vsak podatkovni tok $d_{ij}$ predstavlja neskončni vir relacij (odnosov) med dvema tipoma objektov $\varepsilon_i$ in $\varepsilon_j$. Poleg tega lahko vsak podatkovni tok $d_{ij}$ izraziemo z relacijsko matriko $R_{ij}$. Ta “dinamična” matrika se spreminja skozi čas, ko se v podatkovnem toku pojavljajo ali posodabljajo nove relacije. Na primer, pri priporočilnih sistemih lahko tipi objektov predstavljajo uporabnike, filme, pesmi, knjige itd., podatkovni tokovi pa lahko predstavljajo ocene filmov ali drugih objektov teh uporabnikov v realnem času. Relacije so lahko asimetrične ($R_{ij} \neq R_{ji}$) ali manjkajoče, če nimamo nobenih informacij ali zadostnih podatkov o tem, kako povezati dva tipa objektov. Primer predlagane faktorizacije (skozi čas) lahko vidimo na primeru s Slike 2:
Z uporabo hkratne (kolektivne) faktorizacije nadgradimo našo minimizacijsko funkcijo:

\[ \min f(G, S) = \sum_{R_{ij} \in \mathcal{D}} \| R_{ij} - G_i S_{ij} G_j^T \|_F^2, \]

kjer vsako relacijo aproksimiramo (z uporabo Frobeniusove norme) ločeno, a uporabimo iste faktorske matrike za iste tipe objektov.
Ker so relacijske matrike tipično zelo redke (na primer, pri običajnih problemih pripočetanja prek 90%), moramo faktorizacijo prilagoditi, da se ta prekomerno ne prilagodi na ničle (neznanje vrednosti) v vhodnih matrikah $R_{ij}$. Za ta namen uvedemo matrike uteži $W_{ij}$:

$$\min f(G, S) = \sum_{R_{ij} \in \mathcal{D}} \|W_{ij} \circ (R_{ij} - G_i S_j G_j^T)\|_F^2,$$

kjer $W_{ij} > 0$ za znane vrednosti.

Hkrati v naš sistem dodamo dodatne omejitve za posamezni tip objekta, regularizacijo, pristranost in uteži relacij (več opisano v podpoglavju 4.2.6). Končno izgubno funkcijo nato predstavimo kot:

$$\min f(G, S) = \sum_{R_{ij} \in \mathcal{D}} \alpha_{ij} \|W_{ij} \circ (R_{ij} - \mu_1 - \vec{b}_i 1 - 1 \vec{b}_j - G_i S_j G_j^T)\|_F^2$$

$$+ \sum_{t=1}^{\max t_i} \sum_{i=1}^r tr(G_t^\top \Theta_i^{(t)} G_t), \quad \Theta_i^{(1)} = \Theta_j^{(1)} = \lambda I$$

kjer $\alpha_{ij}$ predstavljajo uteži relacij, $\mu_1, \vec{b}_i, \vec{b}_j$ prestavljajo pristranost, $\lambda$ regularizacijski parameter in $\Theta_t$ omejitve posameznih tipov objekov.

Algoritem SIMF deluje v dveh korakih: po fazi začetne faktorizacije sledi faza posodabljanja na podatkovnih tokov. V prvem koraku se začetni podatki zberejo do neke časovne točke in se nato uporabijo za kolektivno faktorizacijo v začetno stanje (v začetni model). Na splošno velja, da čim več podatkov prihramimo za začetek, tem boljšo začetno faktorizacijo lahko pričakujemo. V drugi fazi se lahko ti začetni podatki popolnoma zavržejo, model pa se postopoma posodablja z novo prispelimi primeri iz podatkovnih tokov (to so spremembe $R_{ij}$ skozi čas). Samo faktorizacijo oziroma posodobitve pa lahko dosežemo z različnimi posodobitvenimi pravili. V tej disertaciji predstavimo tri različne pristope: nenegativna multiplikativna pravila (MUR), gradientni spust in stohastični gradientni spust (podrobneje opisano v podpoglavju 4.3).

Metodo smo evalvirali na način, s katerim smo želeli poudariti zmožnosti zlivanja podatkov v inkrementalnem okolju. Osredotočili smo se na problem priporočilnih sistemov in za primerjavo uporabili algoritem regularizirane matrične faktorizacije (RMF) in enostavni model povprečja (Average). Z uporabo generatorja sintetičnih podatkovnih tokov
GIDS smo pokazali, da predlagana metoda uspešno izboljša napovedno točnost z uporabo zlivanja tokov, hkrati pa je robustna na morebitne spremembe konceptov v podatkih.

V drugem delu evalvacije smo to tezo potrdili z eksperimenti na realni domeni. Izbrali smo podatkovno množico Yelp in modelirali tri različne podatkovne tokove: ocene uporabnikov za posamezne restavracije, bare in hotele za obdobje enega leta. Relacija, ki povezuje uporabnike z restavracijami, je bila najbolj gosta, vendar se je napovedna točnost predlaganega modela SIMF kljub temu dvignila, ko smo v model dodali dve dodatni a redki relaciji (bari in hoteli). Hkrati smo pokazali, da “enostavno” zlivanje podatkov (model RMF smo naučili s zlepljeno matriko vseh podatkov) ne prinese želenih rezultatov, saj se napaka kvečjemu poveča.

V zadnjem eksperimentu smo obrnili problem napovedovanja, tako da smo se osredotočili na napovedovanje uporabniških ocen barov (redka relacija). Ta problem se je izkazal za težjega zaradi povečane redkosti in težave s hladnim zagonom (v fazi začetne faktorizacije je bilo prisotnih le 2000 ocen barov). Kljub temu pa je naš predlagani pristop dosegel najvišjo točnost. Dodatno smo v eksperiment vključili še različico metode, ki dodatno združuje tudi pristranost objektov (več v razdelku 4.2.6), ki se je izkazala za najboljšo. Z eksperimenti smo pokazali, da predlagani inkrementalni pristopi presegajo običajne metode ter z dodajanjem več podatkovnih tokov uspešno zmanjšujejo napovedno napako in negativne učinke hladnega zagona.