Selective Learning for Recommender Systems

DISSERTATION

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von M.Sc. Pawel Matuszyk

geb. am 14.08.1987 in Walbrzych

Gutachterinnen/Gutachter

Prof. Myra Spiliopoulou
Prof. Alípio Mário Guedes Jorge
Prof. Ernesto William De Luca

Magdeburg, den 13.09.2017

SELECTIVE LEARNING FOR RECOMMENDER SYSTEMS

PAWEL MATUSZYK

DISSERTATION ZUR ERLANGUNG DES AKADEMISCHEN GRADES
DOKTORINGENIEUR (DR.-ING.)

First Supervisor:	Prof. Myra Spiliopoulou
Second Supervisor:	Prof. Alípio Mário Guedes Jorge
Third Supervisor:	Prof. Ernesto William De Luca

Knowledge Management and Discovery
Faculty of Computer Science
Otto von Guericke University

September 13, 2017

Pawel Matuszyk: *Selective Learning for Recommender Systems,* Dissertation zur Erlangung des akademischen Grades Doktoringenieur (Dr.-Ing.), © September 13, 2017

ABSTRACT

Recommender systems learn models of users' preferences towards items and use those models to predict future items of interest. They aim to solve the problem of information overload that users are confronted with. In the age of digital information users are especially often overwhelmed by the number of available products (e.g. books, websites, music, medications) from which only a few are relevant for them. The task of recommender systems is to find and recommend the relevant items to users in a personalized way.

To learn models of preferences recommender systems use users' feedback. Since the feedback is, typically, extremely sparse, learning algorithms use all available data for learning. However, we argue that it is beneficial to be selective about what information is used for learning of preference models. Thus, we introduce the term of *selective learning for recommender systems*. We propose three types of selective learning approaches for both: stream-based and batch-based learning.

In this work we focus on the stream-based learning, since it has several advantages over the conventional, batch-based learning. One of the advantage is the ability to immediately incorporate new information into a preference model without relearning the entire model. This is an essential feature for real-life recommender systems, as their application scenarios are highly dynamic. Therefore, new information typically appears at a high rate.

Our first type are forgetting methods for stream-based recommender systems. Selecting what information to forget is equivalent to selecting which information to learn from. We propose eleven different forgetting strategies that select the obsolete information to be forgotten and three different algorithms that enforce forgetting on a stream of ratings. We stress that obsolete information is not necessarily old. Next to incorporating new information into a preference model, our forgetting techniques are the second way of adapting to concept drift or shift.

In our second type of selective learning we introduce *selective neighbourhood* for collaborative filtering methods. It encompasses a novel selection criterion based on the Hoeffding Bound for removing unreliable users from a neighbourhood. Our criterion considers both, the number of common ratings between users and the value of their similarity.

Our last selective approach is based on semi-supervised learning (SSL) for stream-based recommender systems. In this approach a recommender system exploits the abundant unlabelled information (user-

item-pairs without ratings), which, typically, reaches 99% of all information. To exploit this information for training of preference models we propose the first stream-based semi-supervised recommendation framework. In semi-supervised learning, predictions are used as labels. However, not all predictions are equally reliable. We propose components that selectively and incrementally estimate reliability of predictions and filter out the unreliable ones. Only highly reliable predictions are used for training in one of two SSL approaches: co-training and self-learning.

Our evaluation on real-world datasets shows that selective learning yields a substantial improvement in quality of recommendations as compared to recommender systems without selective learning.

In the process of evaluation, many recommender systems, including our methods, require setting of hyperparameters. To make reliable conclusions about the performance of the algorithms, it is necessary to optimize their hyperparameters and compare their optima. However, hyperparameter optimization is not a trivial task and it has not been researched thoroughly in recommender systems. Therefore, as our last contribution, we conduct the first comparative study on hyperparameter optimization for recommender systems. Furthermore, we propose a distributed experimentation framework using Apache Hadoop.

ZUSAMMENFASSUNG

Empfehlungsmaschinen (recommender systems) lernen Modelle der Nutzerpräferenzen und verwenden sie, um zukünftigt relevante Produkte vorherzusagen. Das Ziel der Empfehlungsmaschinen ist es, das Problem der Informationsüberladung zu lösen. Im Zeitalter der digitalen Information sind zahlreiche Nutzer mit diesem Problem konfrontiert. Dieses Problem entsteht bei einer großen Zahl von Produkten (z.B. bei Büchern, Filmen, Musik, oder Medikamenten), von denen nur Wenige relevant sind. Die Aufgabe der Empfehlungsmaschinen besteht darin, die relevanten Produkte zu finden und sie den Nutzern auf eine personalisierte Art und Weise zu empfehlen.

Um Präferenzmodelle zu lernen, nutzen Empfehlungsmaschinen Daten zum Nutzefeedback aus der Vergangenheit. Da dieses Feedback typischerweise extrem rar ist, werden alle verfügbaren Daten verwendet. In dieser Dissertationsschrift argumentieren wir jedoch, dass es vorteilhaft ist, bei der Wahl der Trainingsdaten selektiv zu sein. Deswegen führen wir den Begriff vom *selektiven Lernen für Empfehlungsmaschinen* ein. Wir entwickeln drei Typen von Ansätzen zum selektiven Lernen, sowohl für strombasierte, als auch für batch-basierte Algorithmen.

Der Fokus dieser Arbeit liegt bei den strombasierten Algorithmen, da sie gegenüber den batch-basierten Methoden zahlreiche Vorteile aufweisen. Einer der wichtigsten Vorteile ist die Fähigkeit, neue Information sofort in ein Präferenzmodell zu integrieren, ohne das Modell neu lernen zu müssen. Da die Anwendungsszenarien von Empfehlungsmaschinen typischerweise dynamisch und volatil sind, ist dies eine essenzielle Eigenschaft.

Unser erste Ansatz zum selektiven Lernen sind Vergessensmethoden für strombasierte Empfehlungsmaschinen. Die Selektion von Daten, die vergessen werden sollten, ist äquivalent zur Selektion von Trainingsdaten. Wir schlagen elf unterschiedliche Vergessensstrategien vor, die entscheiden, welche Daten obsolet sind und vergessen werden sollten. Weiterhin entwickeln wir drei Algorithmen, die das Vergessen der selektierten Daten auf einem Strom von Nutzerfeedback umsetzen. Wir betonen hierbei, dass obsolete Daten nicht notwendigerweise alte Daten sind. Unsere Vergessensmethoden stellen, neben der Berücksichtigung neuer Information, eine weitere Möglichkeit dar, wie Modelle inkrementell an Veränderungen über die Zeit angepasst werden können.

In unserem zweiten Ansatz zum selektiven Lernen führen wir ein neuartiges Kriterium zum selektiven Entfernen von Nutzern aus einer Nachbarschaft in nachbarschaftsbasiertem Collaborative Filtering ein.

Dieses Kriterium basiert auf der Hoeffding-Ungleichung und berücksichtigt sowohl die Anzahl der gemeinsamen Ratings, als auch den Wert der Ähnlichkeit zwischen Nutzern.

Unser letzter selektiver Ansatz basiert auf teilüberwachten Techniken (SSL). In diesem Ansatz nutzt eine Empfehlungsmaschine die ungelabelte, im großen Umfang vorhandene Information. Die Menge der ungelabelten Information erreicht oft 99% aller Information und ist extrem groß im Vergleich zur gelabelten Information. Um das Potenzial dieser ungelabelten Information auszunutzen, schlagen wir das erste strombasierte Framework für teilüberwachtes Lernen für Empfehlungsmaschinen vor. In diesem Framework werden Vorhersagen für die ungelabelte Information zum Lernen verwendet. Nicht alle Vorhersagen sind jedoch im gleichen Maß vertrauenswürdig. Deswegen führen wir Komponenten ein, welche die Zuverlässigkeit der Vorhersagen inkrementell schätzen und nur die zuverlässigsten Vorhersagen selektieren. Nur diese Vorhersagen werden in einem von zwei SSL-Ansätzen (Co-Training und Self-Learning) zum Lernen verwendet.

Unsere Evaluierung auf reellen Daten zeigt, dass selektives Lernen eine wesentliche Verbesserung der Qualität der Empfehlungen im Vergleich zu Systemen ohne selektives Lernen mit sich bringt.

Viele Empfehlungsalgorithmen, einschließlich unserer Methoden, sind parametrisch. Um zwei parametrische Algorithmen zuverlässig vergleichen zu können, ist es erforderlich, ihre Hyperparameter zu optimieren. Nach der Optimierungsphase können dann die Optima der Algorithmen verglichen werden. Die Optimierung der Hyperparameter ist allerdings eine komplexe Aufgabe, die im Kontext von Empfehlungsmaschinen nicht ausführlich erforscht wurde. Deswegen ist unser letzter Beitrag in dieser Dissertationsschrift eine vergleichende Studie zu Hyperparameteroptimierung im Kontext von Empfehlungsmaschinen. Darüber hinaus schlagen wir ein experimentelles Framework zur verteilten Berechnung von Experimenten im Prozess der Hyperparameteroptimierung unter Nutzung von Apache Hadoop vor.

ACKNOWLEDGMENTS

Throughout my work on this thesis I received support from many people, to all of whom I am deeply thankful.

First of all, I would like to thank my mentor and first supervisor, Prof. Myra Spiliopoulou, for the opportunity to work under her supervision. During this time I have learned a lot from her and greatly improved my professional skills. Many fruitful discussions and valuable feedback helped me progress with my research and with the work on this thesis.

My cordial thanks also go to my second supervisor, Prof. Alípio Mário Guedes Jorge. Discussions with him, his feedback and insightful questions helped me to critically assess my research and to see it from a different perspective.

I would also like to express my gratitude to Prof. Ernesto William De Luca, who agreed to review my thesis and gave me many valuable advices regarding writing of the thesis.

During my time at the KMD lab I had the opportunity to work with remarkable colleagues. I am very grateful too all of them for providing a great working atmosphere. In particular, I thank Dr. Georg Krempl and Daniel Kottke for many enriching discussions that inspired some of our research.

Furthermore, I am very grateful to my co-authors for their contributions to our research, as well as for inviting me to contribute to their research. In particular, I am very happy to have collaborated with Dr. João Vinagre, who shares my enthusiasm for recommender systems. We exchanged many ideas that helped me immensely with my research and resulted in common publications.

Finally, I give my greatest thanks to my beloved wife, who encouraged and supported me unconditionally throughout the entire thesis and my research work.

Thank you all!

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACRONYMS

CF	Collaborative Filtering
MF	Matrix Factorization
RS	Recommender System
HB	Hoeffding Bound
SSL	Semi-supervised Learning
SL	Self-learning
USL	Unsupervised Learning
NN	Nearest Neighbours
SGD	Stochastic Gradient Descent
ALS	Alternating Least Squares
BRISMF	Biased Regularized Incremental Simultaneous Matrix Factorization
RMSE	Root Mean Square Error
SVD	Singular Value Decomposition
RQ	Research Question
HPO	Hyperparameter Optimization
RW	Random Walk
GA	Genetic Algorithm
SMAC	Sequential Model-based Algorithm Configuration
SMBO	Sequential Model-Based Optimization
SA	Simulated Annealing
PSO	Particle Swarm Optimization

SYMBOLS

A	learning algorithm ... 47
I	set of items ... 25, 34, 64
M_t	model at the time point t ... 49
P	latent user matrix ... 38
Q	latent item matrix ... 38
S_u	a stream of ratings $S_u = (r^{t_1}_{u,i_1}, r^{t_2}_{u,i_2}, \ldots, r^{t_n}_{u,i_3})$ ... 22
Te	test data set ... 50, 153
Tr	training data set ... 47–49, 153
Tr_t	training data set at the time point t ... 47, 48
U	set of users ... 25, 34, 36, 64
η	learning rate in SGD ... 39, 62, 70, 123, 133, 147
λ	regularization parameter in matrix factorization 39, 62, 70, 116, 123, 133, 147
τ	similarity threshold defining neighbourhood of an active user ... 36
ε	deviation of an observed mean from the true mean according to the Hoeffding Bound ... 88–90
$\widehat{r}_{u,i}$	prediction for a rating $r_{u,i}$ ... 39
k	number of latent dimensions in matrix factorization 38, 39, 62, 70, 123, 133, 146, 147
p_u	latent user vector ... 38, 57, 59, 63, 65, 114
q_i	latent item vector ... 38, 59
$r_{u,i}$	rating by the user u towards the item i ... 25, 34, 35, 49, 50, 57, 63
$r^t_{u,i}$	rating by the user u to the item i at the time point t 22
u_a	active user ... 36, 87, 88, 90, 91

Part I

INTRODUCTION AND PRELIMINARIES

1 INTRODUCTION

Recommender systems alleviate the problem of information overload that occurs in presence of abundant choices, from which only a few are relevant. This problem is characteristic, but not limited to, digital information. Considering the continuous growth of digital information, it can be expected that the information overload problem will be aggravated even further in the future. Therefore, it is essential to address this problem in research.

A challenge in finding the relevant information lies in its relative nature. Relevance depends, among others, on such factors as the preferences of the person in need of information, context in wide sense (social, temporal, geo-spacial, etc.), historical interests and many more. Therefore, recommender systems (RS) aim to find the relevant information in a personalized way, i.e. in a way that is tailored to its recipient and often to her/his context.

Possible application scenarios for recommender systems are manifold. They encompass e-commerce [Lin14; SKR99; PNH15], recommending friends in social networks [NGL11; Che+09; Hsu+06], recommending medication for patients [WP14; Che+12], tags in information systems [SVR09], educational materials [Man+11; WLZ15], news articles [LW15; Plu+11] and many more. Recommender systems have shown to be indispensable in many applications. They benefit not only their users, who search for relevant information, but also information providers, as recent studies suggest [GH16; Dia+08; LH14].

Recommender systems operate with high-dimensional data. Typical dimensions reach many millions of users and items. Since human perception is limited, each user can perceive only a few items, from which only a subset is relevant. Possibly, for each of the users it is a different subset of items. Considering the large dimensions typical to the RS domain, it is clear that recommendations cannot be created manually by human experts. This signifies the necessity for automated recommender systems. To provide accurate recommendations those systems build models of users' preferences based on past data. Those models are then used to predict future items of interest to selected users.

Since data in the recommender systems domain is typically sparse, state-of-the-art algorithms use all available data for building models. We argue that this is not beneficial and propose *selective learning* for recommender systems. In the next section we further motivate the need for selective learning.

1.1 MOTIVATION FOR SELECTIVE LEARNING

Learning users' preferences poses numerous challenges to learning algorithms. Those challenges encompass high volatility of preferences, concept drift or shift, sparse data, outliers, etc. We address them by proposing three types of *selective learning methods*:

- stream-based learning with *selective forgetting*
- selective neighbourhood
- stream-based *semi-supervised learning*

All these methods focus on selecting different aspects of data or models. In the first type of methods we select the information to learn from, instead of using all available information. This is equivalent to forgetting of selected information. Therefore, this type is called *selective forgetting*.

In the *selective neighbourhood* we focus on neighbourhood-based collaborative filtering methods. Usually, those methods rely on a definition of a neighbourhood based merely on a similarity threshold. We propose a more selective criterion for deciding, whether a neighbour belongs to a neighbourhood.

The last type of selective learning algorithms are *semi-supervised algorithms*. In those algorithms several learners provide predictions to each other. Those predictions are then used for training preference models. Since not all predictions can be trusted, we propose methods for selecting only the reliable ones for the purpose of learning.

The motivation behind each of the selective learning methods is explained below in more detail.

1.1.1 *Selective Forgetting*

Users' preferences are volatile - they are subject to change over time. Ideally, a recommender system should adapt to those changes. The adaptation can be implemented in three ways:

- by relearning an entire model from the scratch
- by incorporating new information into a model
- by forgetting obsolete information

The first way of adapting to changes is often implemented in so-called *batch learning*. In this scenario an algorithm considers all data to be available at once, i.e. all data is provided to the algorithm in form of a batch. The learning is carried out only on the provided batch of data. New data that accumulates after the last time point of learning

is temporarily disregarded. To incorporate the new data into a model, a new batch containing the new data is provided to the algorithm and the learning procedure is repeated from the beginning. The old model is completely discarded and substituted by a new model from the updated batch.

Clearly, this type of adapting to changes carries several disadvantages with it. It is computationally expensive to relearn the entire model periodically. Considering that, typically, only a part of the model needs to be updated when new data is collected, discarding the entire model is not efficient. Furthermore, in real-world applications new data constantly comes in after a model has been retrained. Consequently, the model becomes gradually out of date with respect to the new data, until the next retraining time point. In dynamic domains, such as recommending news articles, the period between retraining time points must be short to adapt to new events and changing users' interests. With a growing database of news articles and users the relearning time might become longer than the period, in which a model is considered up to date. In this case, the algorithm in the batch learning mode becomes infeasible. Also storing all data in the main memory of a computer might become infeasible when the numbers of users and items continuously grow.

To alleviate those problems researchers proposed stream-based learning algorithms. Unlike in the batch mode, here, algorithms only require a sequential, often one-pass access to the data. Changes are incorporated into models by performing incremental updates using only the new data. Consequently, the old model is not entirely discarded, but only adjusted to be up-to-date with respect to the new data.

Stream-based learning algorithms use the second way of adaptation to changes, i.e. incorporating new data into a model. If the newly acquired data contains a change of the learned concept, then through the incremental update this change will be incorporated into a model. A side effect of incorporating new data is that the importance of the old data will decrease over time, as the total amount of data increases. This way of adaptation has been investigated thoroughly in the recent research on stream-based recommender systems [Cha+11; LW15; Cha+17].

However, we argue that incorporating new information is not sufficient as an adaptation mechanism. Even, if the importance of the old data diminishes over time, its impact onto a model always remains to some extent in the the model. Its impact tends to go to zero as time goes to infinity. However, in real-life applications this consideration is not applicable.

Therefore, to remove the impact of the obsolete data from a model we propose *selective forgetting methods*. Selective forgetting is equiva-

lent to selective learning, since by selecting what to forget, we select what to learn from. The obsolete data is candidate to be forgotten by our selective learning algorithms.

The data used for model learning in the past can become obsolete for several reasons[1]. In data stream mining, common terms used to describe this phenomenon are concept drift or shift, i.e. a gradual or sudden change in concept (user's preferences in our case). Further reasons that are typical to recommender systems are purchasing items for other people, multiple persons sharing one account, noise, etc.

Motivating example 1 presents a hypothetical user u, who is subject to a concept drift.

Motivating Example 1 (Concept Drift). — Let u be a user, who provides a stream of ratings $S_u = (r^{t_1}_{u,i_1}, r^{t_2}_{u,i_2}, \ldots, r^{t_n}_{u,i_3})$. Where $r^t_{u,i}$ is a rating that the user u gave to the item i at the time point t. If the time interval between t_1 and t_n spans several years, then it is likely that at the time point t_n the rating $r^{t_1}_{u,i_1}$ is obsolete, since it does not reflect the current preferences any more. Possibly, considering this rating in the model is even counterproductive, as in the course of years the user u gradually changed preferences. In a hypothetical scenario of recommending movies, this happens e.g. when a user stops watching animated movies and starts watching thriller movies as she/he grows up.

In the medical scenario, of e.g. recommending medication to patients [Che+12], concept drift can occur as the condition of a patient worsens and new symptoms start occurring.

The change in users' preferences does not have to be gradual. Often external events trigger a sudden change of preferences that make considerable parts of a model obsolete. Motivating example 2 illustrates such an event.

Motivating Example 2 (Concept Shift). — Consider the user u with a stream of ratings S_u from the previous example. An external event at a time point t_i, $0 < i < n$, where i is typically unknown to the system, might change the preferences of the user u drastically. Such a change makes large parts of a preference model obsolete, since the preferences they reflect are not valid any more. Consequently, the information obtained before the time point t_i, i.e. $S'_u = (r^{t_1}_{u,i_1}, \ldots, r^{t_i}_{u,i_x})$,should be forgotten by the system.

Such an external event might be, for example, birth of a child, which results in a sudden change in the type of relevant products. A new parent, probably, focusses mostly on baby products, while previous interests become secondary.

1 note that obsolete does not necessarily mean old

The examples 1 and 2 motivate that past preferences are often subject to change and the data that becomes obsolete due to this change should be removed from a preference model. However, forgetting applies not only to old or outdated data. Often a newly obtained piece of information (e.g. in form of a rating $r^t_{u,i}$) is obsolete as well. Example 3 motivates a possible scenario of this type.

Motivating Example 3 (Outliers). — Similarly to the example 2 at the time point t_i a user might provide a rating that is not consistent with the past behaviour of this user. In the e-commerce scenario this happens frequently, when a user buys an item as a present for someone else. Also, users often share their account with other people, where next to the main user there are further, non-identified users - e.g. in a household, where a whole family shares an account. Such a rating $r^{t_i}_{u,i}$ is an outlier to the main user's profile and should, therefore, not be considered for learning this user's preferences. Note that in the previous example all ratings before t_i would be forgotten. Here, only the rating $r^{t_i}_{u,i}$ is affected. Ratings of this type can occur several times for each user. Despite the name "outlier" this phenomenon is not uncommon in the recommender system domain. Repeated occurrence of such outliers results in a selective, non-consecutive forgetting, where the forgotten data instances do not need to be close to each other with respect to the time dimension.

Examples 1 - 3 focus on the motivation from the machine learning perspective. They depict a situation, where it is beneficial for the quality of a model to forget certain data. Apart from this type of motivation, there is also a motivation from the user's perspective.

Recommender systems build user profiles, which often contain sensitive data. A well-designed system should consider privacy concerns of its users. One way of giving the users more control over their privacy is implementing their right to forget selected information. Note that forgetting is more than just deleting an information from a database. While deleting information might prevent unwanted access, it does not remove the impact of this information from a preference model (e.g. latent matrices). Consequently, a model would still contain the removed information implicitly and it would continue to give recommendations to a user based on the removed information. Example 4 illustrates such a scenario.

Motivating Example 4 (Privacy). — Consider an example of an online pharmacy, where users buy medications and receive recommendations based on their historical purchases. A user who does not need a certain medication any more might be interested in removing it from the recommendation model. Some users might be afraid of an accidental disclosure of their preferences (drugs used in the past) by receiving unwanted recommendations in a non-private context

(e.g. at work). Implementing forgetting mechanisms that remove the selected information not only from a database but also from a preference model provides a solution to this problem.

In the example 4 a user decides which information should be forgotten. In all other cases, it is a challenge to correctly identify such information. Typically, there is no ground truth that defines, which information should be forgotten. Therefore, no supervised methods are applicable here. Thus, we propose 11 unsupervised forgetting strategies that determine obsolete information. We then evaluate experimental results of a recommender system with forgetting strategies and of a system without forgetting. From our results we conclude that selective forgetting significantly improves the predictive power of a recommender system (cf. Section 4.5.1 for more results). Furthermore, we address more research questions regarding forgetting methods, such as, how to identify obsolete information (cf. Sec. 1.2).

1.1.2 *Selective Neighbourhood*

Our second type of selective methods is *selective neighbourhood*. This type of selective learning focuses on the neighbourhood-based collaborative filtering (CF). Those methods use a similarity measure to find a neighbourhood for the active user (i.e. the user, for whom recommendations are to be made). To define which users belong to the neighbourhood of the active user, a threshold value for the similarity can be defined. All users, who are above this threshold are considered neighbours to the active user.

Alternatively, a fixed number n of users is defined. All users are then sorted with respect to the similarity to the active user and the top n most similar users are considered neighbours (cf. Sec. 2.2.1 for more details on those methods).

However, in any of those methods considering the similarity value alone is often problematic, since the calculation of similarity is often based on a few common ratings (the so-called "co-ratings"). Motivating example 5 illustrates this problem using cosine similarity.

Note that there is more than one way of calculating the cosine similarity between users. The main difference between them is the way they define user vectors. In our examples we use the cosine similarity that operates on vectors with co-ratings only (cf. Eq. 2.5 in [Agg16], Eq. 24.6 in [AMK11] and Fig. 2 (for item-based CF) in [Sar+01]). Nevertheless, the problem illustrated in this example is not specific to this way of calculating cosine similarity only. Pearson correlation coefficient, adjusted cosine similarity, Spearman Rank Correlation, etc. are also affected by this problem.

Motivating Example 5 (Unreliable Similarity). — Consider the following (simplified) user-item-rating matrix. The set of users is denoted as $U = \{u_1, u_2, u_3\}$ and the set of items as $I = \{i_1, i_2, i_3, i_4\}$. Entries in the matrix are ratings from the range $[1, 5]$, where 5 means a high preference and 1 a low preference. Hereafter, we use the notation $r_{u,i}$ for the rating of the user u towards the item i. The last column represents the cosine similarity to the user u_1.

Users \ Items	i_1	i_2	i_3	i_4	cosine sim. to u_1
u_1	4	5	5	5	–
u_2	4				1
u_3	4	5	5	4	0.9956

Table 1.: Example of an unreliable similarity between users u_1 and u_2 due to a small number of co-ratings.

The similarity measures in the last column of the figure are based on co-ratings only. The only co-rating those users share is the one for the item i_1. Therefore, the similarity between the users u_1 and u_2 is equal to 1.

The similarity between the users u_1 and u_3 is lower than 1, because they diverge in their ratings of the item i_4. Consequently, the user u_2 is considered more similar to u_1 than u_3 is. This is a non-reliable conclusion, since $similarity(u_1, u_2)$ could be this high only due to a chance. Because $similarity(u_1, u_3)$ is based on more observations, it should be considered more reliable.

Adjusting the similarity threshold does not solve the problem, as a higher threshold would still favour unreliable but high similarities (i.e. the similarity of u_1 to u_2 rather than to u_3 in the motivating example 5). To solve the problem of unreliable similarities in RS, researchers proposed several extensions to CF, including e.g. shrinkage [BKV07] and significance weighting [Her+99]. Those methods assign lower weights to similarities that are based on few observations only.

However, we argue that those extension are not sufficient to solve the reliability problem in collaborative filtering. In example 6 we present a scenario, where similarity between users is unreliable even though they have many co-ratings.

Motivating Example 6 (Unreliable Similarity despite Many Observations). — Consider an extension of the matrix from example 5. We assume that items $i_1 - i_4$ are highly popular, i.e. they are rated by nearly all users highly (marked in red). In contrast, items $i_5 - i_7$

are known only to few users, whose opinions on these items differ (marked in blue).

Users \ Items	i_1	i_2	i_3	i_4	i_5	i_6	i_7	cosine sim. to u_1
u_1	4	5	5	5	2	4	5	–
u_2	4							1
u_3	4	5	5	4				0.9956
u_4					2	3	5	0.9915

Table 2.: Example of an unreliable similarity between users u_1 and u_3 despite many co-ratings.

As in the previous example the $similarity(u_1, u_2)$ is considered unreliable due to the low number of co-ratings. This can be achieved by applying e.g. the shrinkage method [BKV07] or significance weighting [Her+99]. Using those extensions, the $similarity(u_1, u_3)$ would be considered the most reliable. However, we claim that similar ratings upon items that are uncommon and controversial are more informative for a recommender system than ratings upon items liked by everyone. Therefore, a reliable similarity measure should not only consider the number of observations, but also the characteristics of items those observations relate to.

To solve the problem shown in example 6 we propose a method for a reliable *selection of neighbours* based on the Hoeffding Bound [Hoe63] and the notion of a baseline user. Informally, a baseline user is, for instance, an average user (cf. Sec. 5.2 for a formal definition). By this definition a baseline user rates popular items highly. Using the Hoeffding Bound (HB) we test if a potential neighbour is significantly more similar to the active user than the baseline user (e.g. an average user) is. By doing so we consider the additional information about popular items and typical user behaviour and penalize the neighbours, who have a high similarity just due to the typical behaviour. If the test using the HB is positive, then a user is included into the neighbourhood of the active user. Therefore, our method selectively learns the neighbourhood of an active user.

We again show that selective usage of information, also in neighbourhood based CF methods, improves the quality of recommendations.

1.1.3 *Stream-based Semi-supervised Learning*

The last of our selective learning methods is semi-supervised learning (SSL). With this method we address the problem of data sparsity. Sparsity in recommender systems often reaches an extreme value of 99%, i.e. there is only 1% of labelled data. We consider a triplet $(u, i, r_{u,i})$ as labelled information, where $r_{u,i}$ is considered a label (in analogy to data mining and machine learning literature). Unlabelled information is then a triplet $(u, i, NULL)$, where the corresponding rating is not known.

Sparsity is inherent to recommender systems due to large dimensions in typical application scenarios. Motivating example 7 illustrates such a scenario.

Motivating Example 7 (SSL). — Consider an e-commerce application of an online shop that has 10,000 items in its assortment and 1 million users. Typically, a user can provide ratings to only few items. Assuming that an average user buys 20 items, but rates only 10 of them, then we obtain a sparsity rate of 99.9%.

An additional challenge in this scenario lies in the non-uniform distribution of the rated items. While some items are popular and frequently rated, the majority of items receive ratings only sporadically. Next to sparsity, this also leads to the problem of recommending items from the co-called "long tail" of the distribution (cf. e.g. [Yin+12] for more information on the long tail problem).

As shown in the example 7, recommender systems face the problem of inferring users' preferences about all items given only few labels that are non-uniformly distributed in the item space. To alleviate this problem we propose two *stream-based* SSL methods: co-training and self-learning. Both of those methods utilize the abundant unlabelled information.

In the co-training approach there are multiple learners that run in parallel and learn not only using the provided ratings (labelled information), but also they train each other using their predictions (unlabelled information). The predictions of one learner are provided to another as labels.

In the self-learning approach a learner provides labels to itself by making predictions about the unlabelled information, i.e. by predicting the value of $r_{u,i}$ in the $(u, i, NULL)$ triplet.

Since the number of possible unlabelled triplets in recommender systems is large, it is essential to perform a well-informed selection of predictions $r_{u,i}$ that can be used as label information. Therefore, in both approaches we propose unsupervised methods to *select reliable predictions* that can be used as labels.

Since SSL in recommender systems is a new field, we propose a novel stream-based framework with several components and we test

their influence onto the quality of recommendations. An important aspect of our framework is being able to work on streams of ratings, which yields several advantages. One of them is incorporating the unlabelled information incrementally without relearning the entire model. Consequently, the trained models can benefit from the unlabelled information nearly immediately. Furthermore, a method that incorporates new information as the stream goes on can adapt to potential changes without the need of periodical retraining.

Finally, also in this aspect of selective learning, we show that *selective usage of unlabelled information* in recommender systems significantly improves their predictive power.

1.2 RESEARCH QUESTIONS

The motivation described in this chapter leads us to the following research questions, including the core research question (RQ).

Core research question:
Does selective learning improve the quality of predictions in recommender systems?

This question is central for the thesis, as it focusses on selective learning. It can be answered positively, if any of the following three research questions (**RQ 1** - **RQ 3**) is answered positively.

RQ 1 Does forgetting improve the quality of recommendations?

Our first type of selective learning is based on forgetting information selectively. To answer this question we compare recommender systems that apply our forgetting mechanisms to ones without forgetting. If the performance in terms of recommendation quality of the former is significantly higher than the performance of the latter, then the answer is positive. However, to answer this question, we first need to address the following subquestions:

RQ 1.1 How to select information to be forgotten?

RQ 1.2 How can forgetting be implemented in a state-of-the-art recommendation algorithm?

RQ 2 Does selective removal of users from a neighbourhood improve the quality of neighbourhood-based CF?

To answer this question we measure the predictive performance of a CF algorithm with and without our method for selective removal of users from a neighbourhood. The result of this comparison determines the answer to this research question. To make the comparison possible we, first, answer the following subquestion:

RQ 2.1 How to select users to be removed from a neighbourhood?

RQ 3 Does selective learning from predictions (semi-supervised learning) improve the quality of recommendations?

The answer to this question depends on the result of the comparison between our SSL method and a conventional method without SSL. If the SSL method performs significantly better in terms of recommendation quality, then the answer is positive. However, to be able to use SSL methods for recommender systems in a stream setting, we, first, answer the following subquestions:

RQ 3.1 How to select unlabelled instances for SSL?

In the SSL setting recommender systems learn from unlabelled instances (user-item-pairs without ratings). As discussed in the motivation, the number of unlabelled instances is typically large. Therefore, only a subset of them can be used for training. To answer this question we propose methods for defining this subset.

RQ 3.2 How to select reliable predictions to learn from?

Ratings of unlabelled instances, which are selected as specified in the previous answer, are predicted using models trained on a stream of ratings. Those predicted rating values can be used for training in SSL. However, not all of the predictions are reliable. To answer this question we propose measures that determine if a prediction can be considered reliable.

RQ 3.3 How to assemble predictions from an SSL system using co-training into a single prediction?

In the co-training approach there are multiple learners that run in parallel on a stream. Assuming that we use n learners, for each unlabelled instance we obtain n predictions. For this prediction to be used in real-world application, i.e. to rank items with respect to user's preferences, these n predictions have to be aggregated into a single value. To answer this question we propose several aggregation mechanisms for rating predictions.

RQ 3.4 How to divide labelled instances among multiple learners in an SSL system?

In the co-training approach the labelled information is used to train multiple learners. However, to gain an advantage from such a system, the learners have to differ from each other. Otherwise, they would provide same

or similar (for non-deterministic algorithms) predictions given the same input. Difference in the learners is beneficial, since it allows for specialization of learners on different aspects of a dataset. To achieve this specialization of learners, they receive different views onto the labelled instances that they learn from. To answer this question we propose several mechanisms that divide the labelled information among the learners, i.e. create different views for the learners.

1.3 SUMMARY OF SCIENTIFIC CONTRIBUTIONS

Our contributions span three different types of selective learning, as discussed in Sec. 1.1, a formal definition that unites them and auxiliary contributions, such as hyperparameter optimization for recommender systems, which helped achieving reliable results in the remaining fields discussed in this thesis.

To summarize, in this thesis, we make the following contributions:

- We propose methods for selective forgetting on streams of ratings that encompass:
  - eleven forgetting strategies of two types
  - three alternative algorithms enforcing forgetting, i.e. removing impact of selected data from a preference model
  - an evaluation protocol including significance testing and incremental recall
- We provide insights on forgetting with positive-only feedback and with rating feedback.
- We propose a method for selective removal of users from a neighbourhood in CF that includes the following sub-contributions:
  - We introduce the notion of baseline users.
  - We define a reliable similarity measure for CF based on the Hoeffding Bound.
  - We point out problems in the usage of the Hoeffding Bound in data stream mining and propose a correction.
- We design semi-supervised learning methods for stream-based recommender systems. To achieve that we provide the following sub-contributions:
  - We propose a novel framework for stream-based recommender systems with two approaches:
    - co-training

    - * self-learning
  - We propose several components in the framework, such as:
    - * reliability measures
    - * aggregation mechanisms for multiple predictions in co-training
    - * methods for creating views onto ratings for co-trainers
    - * methods for selecting unlabelled instances as prediction candidates
  - We introduce a new evaluation protocol with significance testing for stream-based recommender systems.
- We provide a unified definition of selective learning for recommender systems.
- We implement an experimental framework including parallel processing with Apache Hadoop and hyperparameter optimization.
- We conduct the first comparative survey on hyperparameter optimization for Recommender System (RS) (as a part of our experimental framework) that includes comparison of nine optimization algorithms on four real-world, public, benchmark datasets.

1.4 OUTLINE OF THE THESIS

This thesis is divided into three parts and an appendix. The first part consists of the introduction, including a summary of our scientific contributions and research questions, and of preliminaries on recommender systems in Ch. 2. In the preliminaries we shortly introduce the topic of recommender systems and provide knowledge and literature necessary to obtain a basic understanding of the research field. In this chapter we also position this thesis in the broader context of related work. The related work on each aspect of selective learning, however, is discussed in the corresponding chapters separately.

The second part of the thesis describes our main contributions. In this part we describe our three types of selective learning. We start in Ch. 3 with a formal definition of selective learning. In this chapter, we also provide a theoretical framework for evaluation of our core research question.

In Chapt. 4 we propose our first type of selective learning - forgetting methods. This chapter contains parts of our papers [Mat+17][2] and [MS14b; Mat+15]. The reused parts in this thesis are always indicated at the beginning of the corresponding chapter with a reference

2 This paper is currently under review. The submitted version (draft) can be found under: http://pawelmatuszyk.com/download/Forgetting-techniques-for-RS.pdf

to the used source. Their reuse in this dissertation is in accordance with guidelines regarding "Self-plagiarism and good scientific practice" by Prof. Christoph Meinel (ombudsman of University Regensburg) [Mei13].

The second type of selective learning, selective neighbourhood, is described in Ch. 5. This chapter is based on our publications [MS14a; MKS13]. Semi-supervised learning, as the last type of selective learning methods, is addressed in Ch. 6, where we present results published in the following papers [MS17; MS15].

Our last contribution encompasses the experimental framework that involves distributed computing of experiments using Apache Hadoop and a study on hyperparameter optimization for recommender systems [Mat+16].

The third part of the thesis concludes our work with a summary of results and answers to our research questions. We also discuss possible limitations of our methods and potential future work.

In the appendix we discuss our work related to the Hoeffding Bound. We used the results of this work in Ch. 5, where we present our selective neighbourhood method that is also based on the Hoeffding Bound. The appending is followed by a bibliography.

2 PRELIMINARIES ON RECOMMENDER SYSTEMS

In this section we discuss preliminary knowledge necessary to understand our contributions in the further sections. Furthermore, we position our research in the broader context of existing work. The related work to topics of selective learning will be discussed in more detail in the respective sections (cf. Sections 4.1, 5.1, 6.1, 7.1.2).

2.1 OVERVIEW ON TYPES OF RECOMMENDATION ALGORITHMS

One of the first recommender systems was Tapestry, a system for personalized filtering of electronic documents [Gol+92]. Already in 1992 the authors recognized that the amount of documents (including emails) often overwhelms users. Therefore, they proposed a system for automatic filtering of relevant information. The relevance of information in Tapestry is determined using content-based features and annotations from users. Thus, this systems also has a collaborative filtering component, because users collaborate indirectly to help filtering relevant documents for other users by providing annotation to items they know.

Since then, a plethora of different methods for personalized information filtering has been created and it is still an active and developing research area. Those methods encompass several types of algorithms. In Fig. 1 we provide a simplified overview over those types. The sub-disciplines that we focus on in this thesis are marked in red. This overview is not exhaustive, as it only serves the purpose of positioning our research in the broader context and giving introductory knowledge necessary to understand the following sections.

In the following subsections we provide a detailed description of each type of RS algorithms, focussing on the CF algorithms.

2.2 COLLABORATIVE FILTERING

Collaborative filtering algorithms use user feedback to make predictions about future items of interest without the need of analysing the content of items. They use structured feedback that can be represented as a matrix. The example in Tab. 3 illustrates such a matrix.

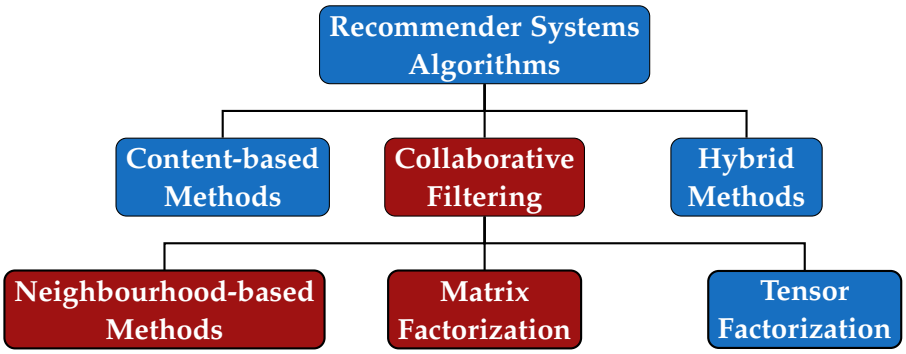

Figure 1.: A simplified classification of main types of recommendation algorithms. In this thesis we focus on the algorithm types marked in red.

Users \ Items	i_1	i_2	i_3	i_4	i_5
u_1	4		5	5	3
u_2	4		2		1
u_3	4	2		1	2

Table 3.: Exemplary matrix in collaborative filtering with explicit rating feedback

In the matrix, we denote users from a set U as u_i with $i \in \{1, ..., |U|\}$ and items from an item set I as i_j with $j \in \{1, ..., |I|\}$. The cells in the matrix stand for the relevance feedback of user u_i to the item i_j, which is denoted as $r_{u,i}$. The user feedback can be of different types. Depending on the way the feedback was acquired we distinguish between:

- explicit feedback
- implicit feedback

The explicit feedback is acquired from users, who specify directly how relevant an item is to them, e.g. by answering a questionnaire or rating an item on a ordinal scale. The implicit feedback is acquired indirectly by analysing users' behaviour and inferring a relevance score from it, e.g. a value of 1, if a user purchased an item and 0 otherwise.

With respect to the range of values the relevance feedback can assume, we distinguish between:

- rating feedback
- binary feedback
- positive-only (unary) feedback

Rating feedback involves a discrete multi-level or a real-numbered ordinal scale, over which a total order can be defined. The ordinal scale expresses the level of preference. Typically, the higher the value of $r_{u,i}$, the more relevant is the item i to user u. The example in Tab. 3 uses such rating feedback with the range $\{1, ..., 5\}$, where 5 means a high preference and 1 means a low preference.

The rating feedback can be of both types: explicit or implicit. For the explicit type users are often asked to rate an item with a score corresponding to their level of preference (e.g. 5 stars). The rating feedback can also be inferred from the user behaviour by mapping it to a real-valued number (e.g. a high percentage of a watched video implies a high relevance).

The binary feedback allows users to rate items with only two values (e.g preference vs. no preference), whereas the positive-only feedback allows only for one value (i.e. expressing preference only). Those two types of relevance feedback can also be implicit or explicit.

In this thesis we conduct experiments with rating and positive-only feedback of both explicit and implicit types.

All collaborative filtering algorithms share this type of data. However, they differ in the way they predict the ratings $r_{u,i}$, where those ratings are missing in the user-item-rating matrix. In the following subsections, we describe main types of CF algorithms, starting with neighbourhood-based methods.

2.2.1 *Neighbourhood-based Methods*

Neighbourhood-based CF methods can be divided into user-based and item-based variants. We focus on the explanation of the user-based variant, as they are more intuitive and, otherwise, nearly equivalent to the item-based CF.

According to Sarwar et al. the work of neighbourhood-based algorithms can divided into two phases [Sar+01]:

- prediction phase
- recommendation phase

In the prediction phase the missing rating values of the active user (i.e. the user, for whom the recommendations are made) are predicted. Once this phase is completed, then the predicted values are ranked with most relevant items first. From this ranking, top n items are recommended to the active user, where n is specified either by an expert or it is derived from domain-specific constraints (e.g. screen real estate). These tasks belong to the recommendation phase.

While the recommendation phase is straightforward and can be found in the same form in many recommendation algorithms, the pre-

diction phase is what defines a recommendation algorithm and poses more challenges.

To predict missing ratings of an active user (the user for whom recommendations are made), the user-based method works with the assumption that similar users have similar preferences. Therefore, the first step of this method is to determine a neighbourhood of the active user, i.e. a set of similar users with respect to their rating behaviour. For this purpose users are modelled as rows in the matrix (cf. the matrix in Tab.3). Consequently, each user $u_i \in U$ is a rating vector. To determine the neighbourhood of the active user, denoted hereafter as u_a, a similarity measure to all remaining users is calculated. This results in the following set of $|U| - 1$ user similarities:

$$\{\forall u \in U \wedge u \neq u_a : \ sim(u_a, u_i)\} \tag{1}$$

This set is then sorted in descending order and only top k users with highest similarity form a neighbourhood. Alternatively, instead of the parameter k, a threshold value τ for the similarity measure can be used. In this case, all users with $sim(u_a, u_i) \geqslant \tau$ form a neighbourhood, i.e. the following user set (NN for nearest neighbours):

$$NN(u_a) = \{u \in U \mid sim(u_a, u) \geqslant \tau \wedge u \neq u_a\} \tag{2}$$

In both cases the parameters k or τ are set by a domain expert. However, even for domain experts, it is a difficult task to set them well, which motivates our research on hyperparameter optimization (cf. Sec. 7.1).

In the literature there are several similarity measures $sim(u_a, u_b)$ that can be used in neighbourhood-based CF. Widely used measures include e.g. the cosine similarity [Agg16] (notation adjusted):

$$cos(u, v) = \frac{\sum\limits_{i \in I_{uv}} r_{u,i} \cdot r_{v,i}}{\sqrt{\sum\limits_{i \in I_{uv}} r_{u,i}^2} \cdot \sqrt{\sum\limits_{i \in I_{uv}} r_{v,i}^2}} \tag{3}$$

and the Pearson-correlation coefficient [DK11]:

$$PCC(u, v) = \frac{\sum\limits_{i \in I_{uv}} (r_{u,i} - \overline{r_u})(r_{v,i} - \overline{r_v})}{\sqrt{\sum\limits_{i \in I_{uv}} (r_{u,i} - \overline{r_u})^2 \sum\limits_{i \in I_{uv}} (r_{v,i} - \overline{r_v})^2}} \tag{4}$$

with $u, v \in U$ and $I_{uv} \subseteq I$ containing items rated both by u and v, i.e. a set of their co-ratings. $\overline{r_u}$ denotes an average rating by user u and I_u a set of items rated by the user u [DK11].

Once a neighbourhood of the active user is defined using a similarity measure, the ratings for items not rated by u_a are predicted by

aggregating ratings of neighbours. A commonly used way of aggregation is a weighted average of neighbours' ratings using their similarities as weights. Accordingly, a rating prediction is calculated using the following formula [DK11]:

$$\widehat{r}_{ui} = \frac{\sum_{v \in NN(u_a)} w_{uv} \cdot r_{v,i}}{\sum_{v \in NN(u_a)} |w_{uv}|} \tag{5}$$

Different extensions and variations to this formula are possible, as e.g. in case of significance weighting [Her+99] and shrinkage [BKV07] (cf. Sec.5.1 for more details) or in case of different normalization schemas [DK11].

Once the predictions for all missing ratings of user u_a have been calculated, we obtain the following rating set:

$$\{\forall i \in I : \widehat{r}_{u_a,i} \,|\, i \notin I_{u_a}\} \tag{6}$$

At this stage, the prediction phase of the algorithm is completed. It is followed by the recommendation phase, in which the items with highest predicted preference score are recommended. Usually, the number of recommendations is restricted by the application domain to only top-n recommendations.

2.2.1.1 *Stream-based Neighbourhood methods*

So far, we have discussed only batch-based algorithms that suffer from several disadvantages, as mentioned in the introduction. The most important ones are problems with scalability, missing adaptivity and need for regular relearning of the entire model. Also batch-based CF suffers from these problems.

To alleviate them, Papagelis et al. proposed a method called "incremental collaborative filtering" (ICF) [Pap+05]. According to their method, there is no need for recalculating the entire user similarity matrix, when new ratings occur in the system. Their method performs incremental updates of similarity measures between users. They provide incremental update operations for the Pearson correlation coefficient in user-based CF. Those update operations replace vector operations with a scalar operation, which are more efficient in terms of computation time.

Miranda and Jorge proposed an incremental algorithm for CF with binary feedback [MJ08] [MJ09]. To update the similarity matrix in an incremental manner they maintain an auxiliary matrix containing counts of item pairs that have been rated within the same user session. Updating the auxiliary matrix can be performed with little computational effort. Miranda and Jorge show that the similarity matrix can be derived directly from the auxiliary matrix for the cosine similarity

[MJ09]. Compared to the approach by Papagelis et al., here, there is no need for rescanning the rating database, which further saves computation time.

2.2.2 *Matrix Factorization*

Despite simplicity and a good predictive performance of neighbourhood-based methods, a different type of collaborative filtering algorithms established their position as state-of-the-art. They are based on matrix factorization (MF). Similarly to neighbourhood-based methods the input to MF is a rating matrix, as e.g. shown in Example 3. However, the process of predicting missing ratings for an active user is different from the neighbourhood-based methods and will be explained in this section.

MF works by decomposing the original rating matrix, denoted hereafter as R, into a product of, typically two, other matrices. This can be expressed by the following formula:

$$R_{m\times n} \approx P_{m\times k} \cdot Q_{k\times n} \tag{7}$$

The original rating matrix R has dimensions $m = |U|$ and $n = |I|$. Rows of this matrix are user vectors $u \in \{\mathbb{F} \cup NULL\}^n$ and items are column vectors $i \in \{\mathbb{F} \cup NULL\}^m$. $\mathbb{F}$ stands for the range of possible values of the rating feedback. Often the feedback is real-valued, then $\mathbb{F} = \mathbb{R}$, or it is a finite subset of natural numbers, e.g. $\mathbb{F} = \{1, 2, ..., 5\}$. $NULL$ represents a missing value.

P and Q denote a matrix with latent user features and a matrix with latent items features respectively. The matrix P contains m latent user vectors $p_u \in P$. The vectors p_u lie in a real-valued, latent space of dimensionality k, i.e. $p_u \in \mathbb{R}^k$. Similarly, the item matrix Q contains n latent, real-valued item vectors also of the same dimensionality as the user vectors $q_i \in \mathbb{R}^k$. The dimensionality of the latent space k is an exogenous parameter defined by a domain expert.

As indicated by the dimensionality k, the latent user and item vectors share the same latent space. The dimensions of the latent vectors are latent features of, usually, unknown meaning. In case of latent item vectors, a value in a cell of a vector, denoted hereafter as q_{if} with $f \in \{1, ..., k\}$, expresses to what degree the item i has the feature f. Analogously, the values in the cells of the user vectors, denoted as p_{uf}, express to what degree a feature is relevant to a user.

As shown in Eq. 7, the decomposition is approximative. It is obtained by solving a minimization problem. To define this problem formally, we first introduce a formula used for rating prediction [Tak+09]:

$$\widehat{r}_{u,i} \approx p_u \cdot q_i^\intercal = \sum_{f=1}^{k} p_{u,f} \cdot q_{i,f} \tag{8}$$

A prediction of a rating by the user u towards item i, denoted hereafter as $\widehat{r}_{u,i}$, is a product of the corresponding latent vectors, i.e. the latent user vector p_u and the latent item vector q_i. This rating prediction formula, when carried out for all cells in the matrix R, is equivalent to the multiplication of the latent matrices as in Eq. 7.

Using this formula the minimization problem can be defined in the following way [Tak+09]:

$$(P^*, Q^*) = \arg\min_{(P,Q)} \sum_{(u,i)\in T} (r_{u,i} - \widehat{r}_{u,i})^2 \tag{9}$$

P^* and Q^* denote the optimal user and item matrices and T stands for a test set, which is a subset of all available ratings. Therefore, the goal of this optimization problem is to find matrices P^* and Q^* that minimize the squared rating prediction error.

By using the Eq. 8 we obtain the following formula, as proposed by Takács et al. [Tak+09]:

$$(P^*, Q^*) = \arg\min_{(P,Q)} \sum_{(u,i)\in T} (r_{u,i} - \sum_{f=1}^{k} p_{u,f} \cdot q_{i,f})^2 \tag{10}$$

This problem can be solved using different optimization methods. In the literature on recommender systems two methods are prevalent in this context: stochastic gradient descent (SGD) (used e.g. in [Tak+09]) and alternating least squares (ALS) (used e.g. in [HKV08]). In SGD, which is the most frequently used method, the latent matrices are initialized randomly and they are improved incrementally using the iterative gradient formulas. For the derivation of these formulas we refer to [Tak+09]. Usage of SGD introduces the parameter η into the system, which controls the learning rate of SGD. Same as other parameters, such as k and λ (defined below), η also needs to be optimized, which further motivates our research in Sec. 7.1.

This basic prediction model from Eq. 8 suffers from a few problems. One of them is overfitting. Without countermeasures the aforementioned minimization problem result in unnecessarily long latent vectors that do not generalize well. To alleviate this problem, several MF methods incorporate regularization.

Takács et al. proposed the following implementation of regularization (notation was adapted) [Tak+09]:

$$(P^*, Q^*) = \arg\min_{(P,Q)} \sum_{(u,i)\in T} (r_{u,i} - \sum_{f=1}^{k} p_{u,f} \cdot q_{i,f})^2 + \lambda \cdot p_u \cdot p_u^\intercal + \lambda \cdot q_i \cdot q_i^\intercal \tag{11}$$

Adding the scalar products of the latent vectors to the minimization formula automatically penalizes solutions with high values in the vectors. Additionally, to control the impact of this penalty, Takács et al. introduced a control parameter λ.

Koren implemented regularization in a similar way. However, instead of adding scalar products of latent vectors, the Frobenius norm of these vectors [Koro9] is used. This translates to the following minimization problem:

$$(P^*, Q^*) = \arg\min_{(P,Q)} \sum_{(u,i)\in T} (r_{u,i} - \sum_{f=1}^{k} p_{u,f} \cdot q_{i,f})^2 + \lambda(\|p_u\|^2 + \|q_i\|^2) \quad (12)$$

To further improve the predictive performance of matrix factorization models, they can be extended by user biases. Koren proposed to extend the rating prediction formula in the following way [Koro9]:

$$\widehat{r}_{u,i} \approx \mu + b_u + b_i + p_u \cdot q_i^\intercal \quad (13)$$

The added components stand for biases at different levels. μ stands for a global bias, i.e. an average tendency of all users towards a given value, b_u is a user-specific bias, and b_i is specific for an item. Considering those biases allows matrix factorization models to focus on the part of users' preferences that cannot be explained by the different level of neutral ratings.

Takács et al. also implemented user and item biases in the BRISMF algorithm (biased regularized incremental simultaneous matrix factorization), however, they did so in a different way [Tak+09]. In BRISMF, the baselines are implemented by fixating one latent dimension in the matrices P and Q (a different one in each matrix). This means that those latent dimensions are kept at a constant value of 1 and are not updated in the training phase of the SGD algorithm. This results in the corresponding dimension of the remaining matrix assuming the values of the baselines. For instance, if the latent dimension f_1 (the first column) in the user feature matrix, P, is kept equal to an identity vector, then the corresponding latent dimension f_1 (the first row) in the item feature matrix, Q, will assume values of the user biases. The same applies for item biases, but with a different latent dimension. Therefore, BRISMF implements user and item biases without the need of explicitly modelling them in the optimization formula (cf. Eq. 12).

In this thesis, we use the BRISMF algorithm frequently, as it serves as a representative of the class of matrix factorization algorithms. While there is a plethora of different extensions to MF algorithms, in their core many of them work in the same way as the BRISMF algorithm does. Therefore, to maximize the generalization potential of our methods and transferability of our results, we apply them to extend the BRISMF algorithm. This shows that our methods do not depend on a specific extension to MF, but can be applied to any algorithm that works in a similar way to BRISMF.

A further extension to the basic matrix factorization model has been proposed by Koren in his algorithm timeSVD++ [Koro9]. Koren declared model variables as functions of time. Those variables

include user and item biases and the latent user vectors. To model changes over time Koren used different method for different aspects of a model. Those methods encompass linear regression, splines and of parameter estimation for discretized time intervals. Despite considering the time dimension, timeSVD++ is not a stream-based method and it is not adaptive. For more methods that exploit the time information we refer to recent studies by Vinagre et al. [VJG15a] and by Campos et al. [CDC14].

2.2.2.1 *Stream-based Matrix Factorization*

The MF algorithms discussed so far are batch-based, i.e. they have access to any rating information also from the past and they assume that the concept they model (i.e. users' preferences) is stationary. Since those algorithms do not adapt and do not incorporate new information, as it appears in an online system, they need regular retraining. That leads to problems with scalability, because retraining an entire model from scratch is computationally expensive. Therefore, in this work we focus on stream-based methods that incorporate new information incrementally without the expensive retraining.

Incremental matrix factorization was first used in recommender systems in [Sar+00], where the fold-in method for Singular Value Decomposition (SVD) [BDO95] was used. However, due to the reasons of computational efficiency and due to a high degree of missing data, state-of-the-art methods mostly use SGD [Pat07]. Incremental matrix factorization for positive-only feedback was investigated by Vinagre et al. in [VJG14]. Here however, we focus more on the ratings feedback. Therefore, we use an incremental MF algorithm proposed by Takács et al. [Tak+09]. Next to a batch-based version, Takács et al. also proposed an incremental version of BRISMF (cf. Algorithm 2 in [Tak+09]).

The incremental BRISMF algorithm preforms iterations of gradient descent using new rating information and, therefore, updating the affected latent user vectors in the matrix P. However, the incremental BRISMF algorithm does not update the latent item features in the matrix Q. Thus, it requires a sporadic retraining to update latent item features. We lift this limitation in our extension described in Chapter 6.

2.2.3 *Tensor Factorization*

In matrix factorization algorithms, users and items are considered dimensions in the preference learning problem. However, in real-world scenarios these are not the only dimensions that affect recommendations. Tensor factorization algorithms offer an additional flexibility in modelling those dimensions. Additionally to the user and item dimensions, tensors allow to add an arbitrary number of further dimen-

sions such as users' context [HT12; Kar+10], time dimensions [Xio+10], topics of items [Zhe+16], or tags in social tagging systems [SNM08; SNM10].

However, typically tensors of third order (i.e. with three dimensions) are used, as higher orders aggravate the problem of data sparsity and the factorization algorithms often suffer from scalability problems.

Often tensor factorization is considered a generalization of the matrix factorization task. A matrix can be considered a tensor of second order. Both matrix and tensor factorization share a common goal, which is predicting missing values in a matrix or in a tensor respectively. Also a commonly used method in tensor factorization, HOSVD (Higher Order SVD), is a generalization of the SVD method (Singular Value Decomposition) [LMV00].

As tensor factorization is not essential to this thesis, we refer to recent surveys for more details on HOSVD and other tensor factorization algorithms [FO16; RGS14].

2.3 CONTENT-BASED METHODS

The focus of this thesis is not on content-based methods. However, to make the fundamental differences between content-based and CF methods clear, we also describe them shortly in this section.

Unlike collaborative filtering algorithms, which only analyse users' feedback, content-based methods also analyse content of items. The content can be of manifold types, typically represented by unstructured or semi-structured data. Depending on the application domain the content can have form of music, videos, text, etc. Therefore, one of the challenges in content-based recommenders is transforming the data into a structured form.

Lops et al. propose an abstract framework shared by all content-based recommender systems [LGS11]. One of the components, the content analyser, is responsible for transforming unstructured data into a structured representation. For textual data a common model used in this transformation is the word-vector-model. Some of the transformation also include e.g. semantic information [Gem+15]. This transformation is domain-specific and needs to be designed differently for different types of data (e.g. music and videos). Therefore, we do not cover this topic in detail here.

Once the data in a structured form is available, a further component from the framework by Lops et al. [LGS11], the profile learner, takes them as an input. The task of profile learner is often a classification or a regression task. To train a model, the profile learner relates the content-related features of items to a relevance class or score provided by a given user in the past.

In case of a classification task, the profile learner trains the following function:

$$f(\Theta) : \mathbb{F} \rightarrow C \tag{14}$$

where $\mathbb{F} = \{f_1 \times f_2 \times ... \times f_n\}$ is an n-dimensional item feature space created by the content analyser, $C = \{c_1, ..., c_m\}$ is a set of relevance classes (e.g. $\{relevant, irrelevant\}$) and Θ denotes parameters of the classifier.

In case of a regression task this function changes as follows:

$$f(\Theta) : \mathbb{F} \rightarrow \mathbb{S} \tag{15}$$

where the set $\mathbb{S}$ determines the range of relevance scores. Typically, it is a subset of real numbers $\mathbb{S} \subseteq \mathbb{R}$.

For training the function $f(\Theta)$, next to the features from the feature space $\mathbb{F}$, the profile learner additionally uses users' feedback that plays the role of a target variable. A profile trained this way is a user-specific predictive model trained on users' preferences.

Typically, conventional data and stream mining algorithms are applied as a profile learners. Pazzani et al. applied decision trees, nearest neighbours classifier, Rocchio's algorithm, Naive Bayes, etc. as a learning component [PB07].

Once each user has a personal profile, this profile can be employed by a filtering component to predict the relevance of an item unknown to the user. Finally, users' feedback is collected and used for further training.

2.4 HYBRID METHODS

The last type of methods from Fig. 1 are hybrid methods. While we do not focus on them in this thesis, we still explain the basic concept behind them to clarify the differences to the methods within the scope of the thesis.

Hybrid recommender systems combine two or more methods to potentially achieve a better quality of recommendations than any of these methods alone. Since all methods have specific weaknesses, it is possible to compensate them by using a hybrid RS in combination with a method that does not share this weakness.

Ghazanfar et al., for instance, proposed a cascading hybrid RS that combines rating, feature, and demographic information and showed that it outperforms other state-of-the-art algorithms [GP10].

Furthermore, using hybrid recommender systems allows for employing different sources of information in one system. A hybrid system encompassing e.g. a matrix factorization algorithm and a content-based algorithm can use both rating feedback and textual features of items. While a matrix factorization algorithm struggles with the new-item-problem, content-based algorithms easily provide a solution in

such a case, thus, improving the overall quality of recommendations by the system [Bur07].

Also in case, when one of the methods cannot provide a recommendation to a given user e.g. due to missing data in one of the information sources, a hybrid system that uses different sources of information can potentially still make a recommendation. Therefore, it can outperform a single method in terms of user space coverage [GP10].

Combining multiple algorithms into one hybrid system is a non-trivial task. Robin Burke, in his studies, defined and compared seven ways of hybridizing recommendation algorithms [Bur02; Bur07]. He concluded that cascading algorithms are very effective, particularly when combining algorithms with different strengths [Bur07].

In this chapter on preliminaries of recommender systems, we described main classes of recommendation algorithms. While there are further types, e.g. demographic or knowledge-based recommender systems, they are not directly relevant to this thesis. Therefore, for further information on those algorithms we refer to recent studies [Par+12; Lü+12; Bob+13; Bee+16].

Part II

SELECTIVE LEARNING METHODS

3

FORMAL DEFINITION OF SELECTIVE LEARNING

In this chapter, we define selective learning for recommender systems and provide a formal framework for evaluating our core research question.

> **Definition:** *Selective learning* in recommender systems encompasses methods for learning and predicting users' preferences not by using all available data and models, but by selecting data and aspects of models that maximise the quality of recommendations.

In the next section we define a general algorithm common for all of these methods. Afterwards, we define selective learning as an optimization problem and, finally, we use those definitions to formulate a logical criterion for evaluating our core research question.

3.1 GENERAL ALGORITHM FOR SELECTIVE LEARNING

We focus on selective learning for stream-based methods. Therefore, in Alg. 1 we present a generalized algorithm for incremental selective training. As inputs it takes a training set Tr_t at time point t, a storage of training instances Tr, a learning algorithm A, a model M_{t-1} from the previous time point and a selection strategy S_{t-1}, also from the previous time point, as some selections strategies are adaptive and learn over time. A *selection strategy can be one of the three types* mentioned in the introduction.

Note that this is an incremental algorithm that is called at every time point t of a stream, beginning from $t = 0$. Since it is a training algorithm, its sole purpose is to train the model (i.e. adjust their internal parameters and store them for the next iteration) and not to output predictions.

Tr_t contains only ratings that arrive in a stream of ratings at the time point t and it is not a cumulative training set. Usually, this set contains only the newest rating that needs to be incorporated into a preference model. Lines 1-4 initialize the necessary variables at the beginning of a stream. Tr stores training instances over a longer period of time, as opposed to Tr_t. The algorithm A is responsible for initialization of a model if the previous model M_{t-1} is not known. A can be any incremental collaborative filtering algorithm (including incremental matrix factorization).

Algorithm 1 Incremental Selective Training

Input: $Tr_t, Tr, M_{t-1}, A, S_{t-1}$
1: **if** $t = 0$ **then**
2: $\quad Tr = \emptyset$
3: $\quad M_{t-1} = A.initializeModel()$
4: **end if**
5: $Tr_{selection} := S_{t-1}.SelectTrainingInstances(Tr, M_{t-1})$
6: $Tr_{toForget} = Tr \setminus Tr_{selection}$
7: $Tr_{toAdd} = S_{t-1}.SelectTrainingInstances(Tr_t, M_{t-1})$
8: $M_t := A.forget(M_{t-1}, Tr_{toForget})$
9: $M_t := A.updateModel(Tr_{toAdd}, M_t, S_{t-1})$
10: $S_t = S_{t-1}.adapt(Tr_{toAdd}, M_t)$
11: $Tr := Tr_{selection} \cup Tr_t$
Output: M_t, S_t, Tr

In line 5 the selection strategy S decides which ratings should be used for learning by removing unwanted ratings from the set Tr. To decide upon this selection, the strategy also considers the currently available preference model M_{t-1}. Considering the model in this process is essential as it allows to decide upon a rating e.g. based on its utility to the model.

Different selection strategies employ different mechanisms to carry out this selection. Those strategies will be defined in more detail in the chapters on forgetting methods (cf. Ch. 4) and SSL (cf. Ch. 6). Despite the diametric differences in the selection strategies between forgetting methods and semi-supervised learning, Algorithm 1 unifies them into one common framework.

Formally, the function $SelectTrainingInstances$ of a selection strategy S, which is used in lines 5 and 7, maps a training set of ratings into another set of ratings

$$f : Tr \rightarrow Tr_{selection} \tag{16}$$

such that $Tr_{selection} \subseteq Tr$.

After the selection is completed, in line 6 the set of ratings to be forgotten, $Tr_{toForget}$, is determined. In line 7 the selection is also carried out for the set Tr_t. If the set contains only one new rating, the selection is equivalent to deciding, if the preference model should be updated using this rating or not. In case of SSL this set contains several candidate ratings that are filtered e.g. based on their reliability (cf. Ch. 6).

In lines 8 and 9 the preference model M is incrementally updated. Line 9 carries out an iterative update of the model using the set Tr_{toAdd}, determined before. This update operation is typical to stream-based algorithms. Additionally, this function takes a selection strategy as an

argument, because the strategy can also modify the model selectively by e.g. giving more importance to selected parts of the model.

Line 8 implements the forgetting of ratings from the set $Tr_{toForget}$. Forgetting a ratings is equivalent to reversing its impact that it had on the preference model in the learning process. We describe possible implementations in Chapt. 4.

In line 10 the selection strategy is adapted using the new data from the stream Tr_{toAdd} and the adjusted preference model M_t. Finally, in line 11 the training set Tr is adapted so that it contains the ratings currently reflected by the preference model.

In Algorithm 1 the selective strategies affect the way an algorithm trains its preference model. However, our selective strategies also change the way an algorithm calculates predictions by using an existing preference model M partially in a selective way. This is shown abstractly in Algorithm 2.

Algorithm 2 Selective Prediction

Input: $r_{u,i}, M, A, S$

1: $\widehat{r}_{u,i} = A.selectivePredicting(r_{u,i}, M, S)$

2: **return** $\widehat{r}_{u,i}$

While conventional algorithms calculate their predictions for a rating $r_{u,i}$ considering only a given preference model M, our methods additionally consider the selection strategy S, which, in this case, controls which parts of the model are used for calculating this prediction. This type of selection strategies are particularly useful in neighbourhood-based models, where a part of a model, e.g. a neighbourhood of a user, can be selectively modified, e.g. by removing unreliable users from the neighbourhood. The selective prediction shown in Algorithm 2 can also be used in static algorithms (not only in stream-based ones) as we present it in Chapt. 5. There, we also propose a possible definition and implementation of such strategy.

3.2 SELECTIVE LEARNING AS AN OPTIMIZATION PROBLEM

Learning preference models in recommender systems can be understood as an optimization problem. In this section we formally define the problems that our methods aim to solve. First, to show the difference between the conventional, non-selective learning and our methods, we define the optimization problems for the non-selective methods.

3.2.1 *Optimization Problem in Non-selective Learning*

Let Q be a quality measure that needs to be maximized. In the conventional batch learning we define the quality measure as the following function:

$$Q : (Te, M) \rightarrow \mathbb{R} \tag{17}$$

The quality function takes a test data set Te and a predictive model M as arguments. Then, this function internally uses the model M to calculate predictions for all ratings $r_{u,i} \in Te$. Based on those predictions a quality value for each rating is calculated and aggregated into one, real-valued quality measure.

For the non-selective, stream-based methods the quality measure is calculated for each time point. Therefore, we denote it as follows:

$$Q_t : (Te_t, M_t) \rightarrow \mathbb{R} \tag{18}$$

with Te_t begin a test set used at the time point t and M_t the current model at the time point t.

Accordingly, the goal of non-selective batch-based methods is to find an optimal model:

$$M^* = \arg\max_{M} Q(Te, M) \tag{19}$$

For the definition of the optimization problem for stream-based methods we, first, introduce the term of a model series $\mathcal{M}$:

$$\mathcal{M} = (M_0, ..., M_t) \tag{20}$$

The model series is composed of different versions of a model M for different time points. Since models in stream-based methods adapt over time, there is no single model M for all time points, but there is a series of models with one model for each time point, i.e. M_t.

Considering this definition, the goal of stream-based recommender systems is to find an optimal series of models $\mathcal{M}^*$:

$$\mathcal{M}^* = \arg\max_{\mathcal{M}} \sum_{t} Q_t(Te_t, M_t) \tag{21}$$

with $M_t \in \mathcal{M}$.

According to this definition, the optimal series of models is one that has the maximal sum of quality measures over time. Ideally, but not necessarily, it is a model series that dominates all other model series at all time points with respect to their quality.

3.2.2 *Optimization Problem in Selective Learning*

In selective learning we introduce the selection strategies both: in stream-based and batch-based learning. This additional component affects also the optimization problems our methods aim to solve. Therefore, in this section we introduce definitions and a specification of the optimization problem for selective learning.

To measure the quality of batch-based selective methods, we define a quality measure as follows:

$$Q_S : (Te, M, S) \rightarrow \mathbb{R} \tag{22}$$

This quality measures additionally considers a selection strategy S, when calculating the quality value. To visually differentiate it from the non-selective counterpart, this quality measure is denoted with S in index.

Analogously, we also define a quality measure for stream-based selective methods:

$$Q_{S,t} : (Te_t, M_t, S_t) \rightarrow \mathbb{R} \tag{23}$$

Additionally to its non-selective counterpart, this measure also considers the selection strategy S_t at the time point t. Since some selection strategies are adaptive and change over time, there is no single strategy S for all time points, but they are specific for each time point. Also here, we denote the selective variant with S in the index to ease the differentiation.

Since a selection strategy is time-point-specific, we introduce the definition of series of selection strategies:

$$\mathbb{S} = (S_0, ..., S_t) \tag{24}$$

Given those definitions, we define the goal of selective batch-based methods as finding the optimal pair $(M, S)^*$:

$$(M, S)^* = \underset{(M,S)}{\arg\max}\, Q_S(Te, M, S) \tag{25}$$

Accordingly, stream-based methods optimise the following pair of series of models and selection strategies:

$$(\mathcal{M}, \mathbb{S})^* = \underset{(\mathcal{M},\mathbb{S})}{\arg\max} \sum_t Q_{S,t}(Te_t, M_t, S_t) \tag{26}$$

with $S_t \in \mathbb{S}$ and $M_t \in \mathcal{M}$. The optimal pair of $(\mathcal{M}, \mathbb{S})$ has the maximal sum of prediction quality over all time points.

3.2.3 *Answering the Core Research Question*

The definitions given in the proceeding sections allow us to answer our *core research question* from Sec. 1.2 by evaluating the following logical expression:

$$[\exists(\mathcal{M}, \mathcal{S}) : \sum_{t} Q_{S,t}(Te_t, M_t, S_t) > \sum_{t} Q_t(Te_t, M_t^*)] \vee$$
$$[\exists(M, S) : Q_S(Te, M, S) > Q(Te, M^*)] \quad (27)$$

where $M_t \in \mathcal{M}$, $S_t \in \mathcal{S}$, $M_t^* \in \mathcal{M}^*$, $\mathcal{M}^*$ is the optimal solution from Eq. 21 and M^* is a solution from Eq. 19.

If the expression in Eq. 27 is true, then we answer the core research question positively, i.e. we conclude that selective learning improves the quality of predictions in recommender systems.

The first part of Eq. 27 (before $\vee$) is true if there exists a series of models and selections strategies used by our method that outperform *the optimal series of models* $\mathcal{M}^*$ of a corresponding non-selective method. The second part of the equation is true if our batch-based selective method outperforms the corresponding, *optimal*, non-selective batch-based method. Since those two parts are connected with a logical OR, the expression is true, if any of the parts is true.

In the following chapters we provide answers necessary to evaluate this logical expression.

4 FORGETTING METHODS

The first type of our selective learning techniques for recommender systems is *selective forgetting*, as motivated in Sec. 1.1.1. In this chapter we address the following research questions:

RQ 1: **Does forgetting improve the quality of recommendations?**

RQ 1.1: **How to select information to be forgotten?**

RQ 1.2: **How can forgetting be implemented in a state-of-the-art recommendation algorithm?**

We begin with work related to forgetting techniques. In Sec. 4.2 we present our forgetting techniques, i.e. how to select information which should be forgotten. In the following section we show how the forgetting of selected information can be enforced on a stream of ratings. In Sec. 4.4 we present our evaluation settings and in Sec. 4.5 our experimental results. Finally, in Sec. 4.6 we conclude this chapter and give answers to the aforementioned research questions. The sections 4.1 - 4.6 come (with modifications) from our previous publications on this topic [Mat+17; MS14b; Mat+15].

4.1 RELATED WORK ON FORGETTING METHODS

In real world systems, data used by recommender systems has all characteristics of a data stream. Data streams arrive on-line, at unpredictable order and rate, and they are potentially unbounded [Bab+02]. Once a data element is processed it must be discarded or archived, and subsequent access to past data gets increasingly expensive. Algorithms that learn from data streams should process data in one pass, at least as fast as data elements arrive, and memory consumption should to be independent from the number of data points [DH01].

Forgetting past data is a model maintenance strategy frequently used in data stream mining. The underlying assumption is that some data points are more representative than others of the concept(s) captured by the algorithm. Common forgetting strategies in stream mining encompass a sliding window and weighting with an exponential decay function [Gam+14] [Gam12].

Although recommendation is seldom approached as a data stream mining problem, the following contributions must be pointed out. Two of them apply generic data stream mining algorithms in recommender systems. In [LBD07], Li et al. propose an approach to drifting

preferences of individual users using the CVFDT algorithm [HSD01b]. This is a popular classification algorithm for high speed data streams. The CVFDT algorithm is used to build a decision tree for each item in the dataset, given the ratings of other highly correlated items. The ratings given by users to these correlated items are used to predict the ratings for the target item. The mechanics of CVFDT provides automatic adjustment to drifts in user interests, avoiding accuracy degradation. In [Nas+07] Nasraoui et al. use the TECHNO-STREAMS stream clustering algorithm [Nas+03], using a sliding window through user sessions to compute a clustering-based recommendation model. However, those two approaches do not use dedicated recommendation algorithms, but rather rely on conventional stream-mining techniques.

In recommender systems, forgetting was introduced by Koychev in a content-based algorithm [Koy00]. The technique assigns higher weights to recent observations, forgetting past data gradually. This way, the algorithm is able to recover faster from changes in the data, such as user preference drift. A similar approach is used with neighbourhood-based collaborative filtering by Ding and Li in [DL06]: the rating prediction for an item is calculated using a time decay function over the ratings given to similar items. In practice, recently rated items have a higher weight than those rated longer ago. In [Liu+10], Liu et al. use the same strategy, additionally introducing another time decay function in the similarity computation, causing items rated closely in time to be more similar than those rated far apart in time.

Another contribution is made by Vinagre and Jorge in [VJ12]. The authors use two different forgetting strategies with neighbourhood-based algorithms for positive-only data. Forgetting is achieved either abruptly, using a fixed-size sliding window over data and repeatedly training the algorithm with data in the window, or gradually, using a decay function in the similarity calculation causing older items to be less relevant. However, these techniques are only effective in the presence of sudden changes with a high magnitude global impact, i.e. for all users, and do not account for more subtle and gradual changes that occur on the individual users' level. In our experiments we also compare to those strategies.

In [Kor09] Koren modelled users in a dynamic way, i.e. he tried to capture changes in latent user vectors by using linear regression and splines. However, his method was not able to recognize and forget obsolete information. Additionally, unlike our method, Koren's method is not able to work on streams of ratings. In 2014 Sun et al. introduced collaborative Kalman filtering [SPV14]. This method also attempts to model changes in users' preferences, but similarly to Koren's method, it is not designed for streams of data and it does not forget any information. Chua et al. modelled temporal adoptions using dynamic matrix factorization [COL13]. Their approach is not stream-based, but

works on chunks of data and combines models of single chunks using a linear dynamic system. Similarly to Sun et al., in [COL13] the authors also used Kalman filters.

The focus of this chapter is on stream-based recommender systems due to their relevance to real-world applications. As a representative of stream-based recommender systems we use an incremental matrix factorization algorithm BRISMF [Tak+09] explained in Chap. 2.

Forgetting for this type of algorithms had not been studied before the work in [MS14b] and [Mat+15]. Since matrix factorization in recommender systems is an active research field, there are numerous recommendation algorithms based on it. Those algorithms encompass several extensions, e.g. for implicit feedback [HKV08], time aspects (different than forgetting) [KBV09; Kor09], semi-supervised learning [MS15], active learning [Kar+11], etc. Despite different extensions, many of those algorithms work in the core in the same or similar way to BRISMF. Therefore, we show the effectiveness of our forgetting methods using this typical representative of MF algorithms.

4.2 FORGETTING STRATEGIES

In this section we discuss how to select information that should be forgotten. We introduce the term of *forgetting strategies* as methods for selecting this obsolete information. They work in an unsupervised way, since there is no ground truth determining when a rating becomes obsolete.

While detailed explanation on how those forgetting strategies are used is provided in Sec. 4.3, in Fig. 2 we give a simplified overview over the function and components of our framework with selective forgetting. Fig. 2 illustrates how selecting forgetting is embedded into the learning process on a stream of ratings. The process in Fig. 2 is repeated for every rating in a stream. Forgetting strategies play an essential role in this framework.

In total we present 11 forgetting strategies. We divide them into two categories based on their output. The first category is *rating-based forgetting* (cf. Sec. 4.2.1). As the name suggests, all strategies of this type take a user-specific list of ratings as input and decide which ratings should be forgotten. Data returned by those strategies is a filtered list of ratings of a user.

Forgetting strategies of the second type are based on latent factors from the matrix factorization model. The *latent-factor-based strategies* (cf. Sec. 4.2.2) modify a latent factor of a user or of an item in a way that lowers the impact of past ratings.

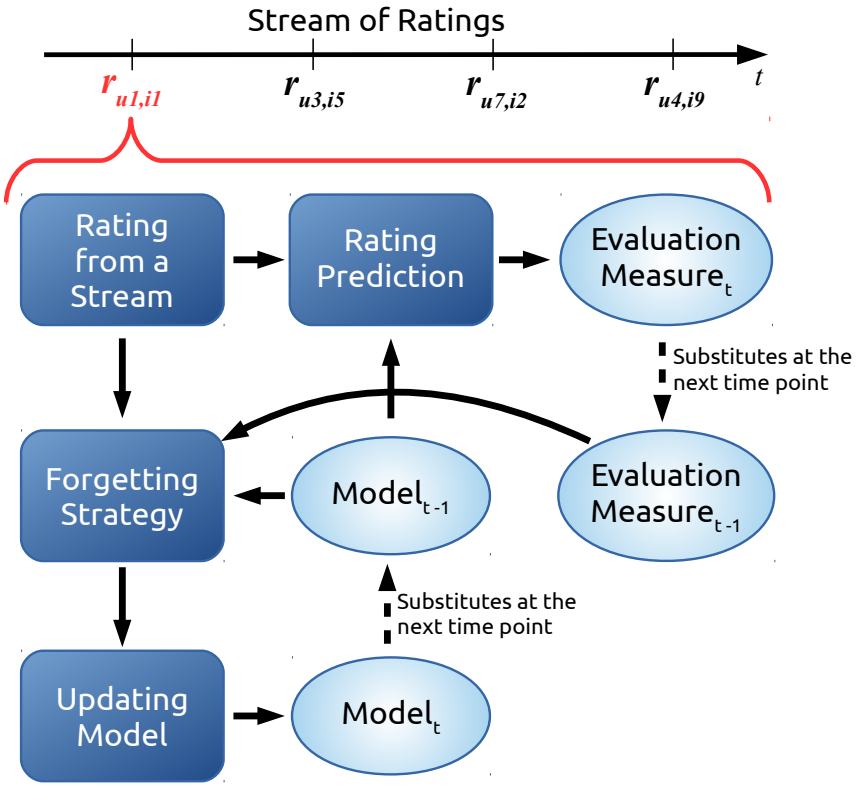

Figure 2.: Overview of the selective forgetting framework. The process presented here is repeated for every rating in the stream.

4.2.1 *Rating-based Forgetting*

This category of forgetting strategies operates on sets of users' ratings. We define $R(u)$ as a set of ratings provided by the user u. Formally, a rating-based forgetting strategy is a function $f(R(u))$:

$$f : R(u) \rightarrow R(u)' \tag{28}$$

where $R(u)' \subseteq R(u)$. Furthermore, for each user we define a threshold of n ratings a user must have, before forgetting is applied. This threshold ensures that no new users are affected by forgetting and that no users or items are forgotten completely. Therefore, it is not possible that a badly chosen forgetting strategy forgets everything and is unable to make recommendations. In our experiments we use $n = 5$.

In the context of our formal definition from Chapt. 3, those forgetting strategies are used in line 5 in Alg. 1, when the function $SelectTrainingInstances(Tr, M_{t-1})$ is called.

4.2.1.1 *Sensitivity-based Forgetting*

As the name suggests this strategy is based on sensitivity analysis. We analyse how much a latent user vector p_u changes after including a new rating of this user $r_{u,i}$ into a model. A latent user vector should always reflect user's preferences. With a new rating $r_{u,i}$ we gain more information about a user and we can adjust the model of his/her preferences (the latent vector p_u) accordingly. If the new rating is consistent with the preferences that we know so far of this particular user, the change of the latent vector should be minimal. However, if we observe that the latent vector of this user changed dramatically after training on the new rating, then this indicates that the new rating is not consistent with the past preferences of this user. Thus, we forget this rating, i.e. we do not update the user's latent vector using this rating, since it does not fit to the rest of user's preferences.

In real life recommenders this occurs frequently, e.g. when users buy items for other people as presents, or when multiple persons share one account. Those items are outliers with respect to preferences observed so far. Learning model based on those outliers distorts the image of user's preferences.

In order to identify those outlier ratings, we first store the latent user vector at time point t, denoted hereafter as p_u^t. Then, we simulate our incremental training on the new rating $r_{u,i}$ without making the changes permanent. We obtain an updated latent user vector p_u^{t+1}. Our next step is then calculating a squared difference between those two latent vectors using the following formula:

$$\Delta_{p_u} = \sum_{i=0}^{k} (p_{u,i}^{t+1} - p_{u,i}^{t})^2 \tag{29}$$

Furthermore, for each user we store and update incrementally at each time point the standard deviation of the squared difference. The notation $\overline{x}$ stands for a mean value of vector x and $SD(x)$ is standard deviation of this vector. If the following inequality is true, then it indicates that the new rating $r_{u,i}$ is an outlier:

$$\Delta_{p_u} > \overline{\Delta_{p_u}} + \alpha \cdot SD(\Delta_{p_u}) \tag{30}$$

Where α is a parameter that controls the sensitivity of the forgetting strategy. It is specific for every dataset and has to be determined experimentally.

4.2.1.2 *Global Sensitivity-based Forgetting*

Similarly to the previous strategy, Global Sensitivity-based Forgetting is also based on sensitivity analysis. However, in contrast to Sensitivity-based Forgetting, this strategy does not store the standard deviation

of squared differences of latent vectors separately for each user but rather globally. Formally, it means that the inequality (30) needs to be changed into:

$$\Delta_{p_u} > \overline{\Delta} + \alpha \cdot SD(\Delta) \tag{31}$$

Where Δ is a squared difference of latent user vectors before and after including a new rating for all users.

4.2.1.3 *Last N Retention*

To the category of rating-based forgetting we also count the Last N Retention strategy. This forgetting strategy uses sliding window, which often finds an application in stream mining and adaptive algorithms. Here however, a sliding window is defined for each user separately. *LastNRetention* means that the last N ratings in a stream of a user are retained and all remaining ratings are forgotten. If a user has fewer ratings than N, then forgetting will not be applied onto this user.

4.2.1.4 *Recent N Retention*

This forgetting strategy is also based on a user-specific sliding window [MS14b]. However, here the size of a window is not defined in terms of ratings, but in terms of time. Therefore, unlike *last N retention* it considers the time that has passed between providing ratings. If $N = 3days$, then only ratings from the last three days are retained and all the remaining ones will be forgotten. This strategy can be also set up to use only ratings from e.g. the last session. In contrast to the Last N Retention, this strategy allows to define a lifetime of a rating, which is especially beneficial in highly dynamic environments, such as news recommendations.

4.2.1.5 *Recall-based Change Detection*

In many applications a change of preferences takes place gradually. However, there are applications where a change can happen abruptly and without prior indication. For instance, a person who becomes a parent is likely to start preferring items suitable for children. A recommender system that adapts gradually is not able to capture this abrupt change. Consequently, the importance of these new preferences will be underestimated for a long period of time. Therefore, we propose a forgetting strategy that detects a change in preferences and forgets all ratings from before the change. This is equivalent to resetting a single profile of the affected user without discarding the entire model.

In recall-based change detection, incremental recall (or a different incremental quality measure) for each user is monitored. A high drop in the quality measure indicates a change of preferences.

In more quantitative terms, let incremental recall for user u at timepoint t be denoted as $incrRecall_u^t$. A change is detected, if the following inequality is true:

$$incrRecall_u^t < \overline{incrRecall_u} - \alpha \cdot SD(incrRecall_u) \tag{32}$$

i.e. when the current recall of a user is lower than this user's mean recall by at least $\alpha \cdot SD(incrRecall_u)$. The parameter α controls the sensitivity of the detector.

4.2.1.6 *Sensitivity-based Change Detection*

Similarly to the previous forgetting strategy, this one is also a change detector. We assume that, if incorporating a new rating from the stream provided by a user changes the underlying user's model dramatically, then this new rating does not fit into the old model. Consequently, we conclude that the user's preferences have changed.

To express the relative change of a user's model over time we use the notion of local model sensitivity and the formula 29. Let $\Delta_{p_u}^t$ be the change of user's model after incorporating a new rating at timepoint t. A change is detected, if the following inequality holds:

$$\Delta_{p_u}^t > \overline{\Delta_{p_u}} + \alpha \cdot SD(\Delta_{p_u}) \tag{33}$$

i.e. when the change to the current model is higher than the standard deviation of all changes of this user multiplied with a control parameter α.

This strategy is similar to *Sensitivity-based Forgetting*, because it also uses the idea of an outlier-rating that does not fit into the learnt model. However, the key difference is that, here, we conclude that it is the model learnt so far that should be forgotten due to a concept shift and not the single outlier-rating.

4.2.2 *Latent Factor Forgetting*

Latent factor forgetting is the second type of our forgetting strategies. Unlike rating-based forgetting, this type of strategies operates directly on preference models and not on the ratings. Therefore, in the context of our formal definition in Alg. 1, the usage of these strategies is equivalent to function $updateModel(Tr_{toAdd}, M_t, S_{t-1})$ (cf. line 9 in Alg. 1). In this function the model M_t is adjusted in a way controlled by the forgetting strategy.

In matrix factorization, preference models have form of latent user vectors, denoted as p_u, or latent item vectors, denoted as q_i. This type of forgetting is triggered when a new rating of the corresponding user u towards item i has been observed in a stream.

Formally, a latent factor forgetting strategy is a linear transformation of a latent user vector (cf. Eq. 34) or of a latent item vector (cf. Eq. 35):

$$\overrightarrow{p_u^{t+1}} = \gamma \cdot \overrightarrow{p_u^t} + \beta \tag{34}$$

$$\overrightarrow{q_i^{t+1}} = \gamma \cdot \overrightarrow{q_i^t} + \beta \tag{35}$$

The parameters γ and β are dictated by the strategies described in the following subsections. Since those strategies transform latent vectors, it is also not possible that users or items are forgotten completely. This could happen only if γ and β are equal to 0, which we do not allow in the following strategies.

4.2.2.1 *Forgetting Unpopular Items*

In this forgetting strategy unpopular items are penalized. Latent item vectors are multiplied with a value that is lower for unpopular items to decrease their importance in the prediction of interesting items. Formally, this strategy is expressed by the following formula:

$$\overrightarrow{q_i^{t+1}} = (-\alpha^{-|R(i)|} + 1) \cdot \overrightarrow{q_i^t} \tag{36}$$

$R(i)$ is the set of ratings for item i. α is a parameter that controls, how much the latent item vector is penalized. $(-\alpha^{-|R(i)|} + 1)$ is an exponential function that takes low values for items with few ratings (for α values > 1). Additionally, this function has an advantage of being limited to value range of $[0, 1)$, for $\alpha > 1$.

4.2.2.2 *User Factor Fading*

User Factor Fading is similar to using fading factors in stream mining. In this strategy latent user factors are multiplied by a constant $\alpha \in (0, 1]$

$$\overrightarrow{p_u^{t+1}} = \alpha \cdot \overrightarrow{p_u^t} \tag{37}$$

The lower is this constant, the higher is the effect of forgetting and the less important are the past user's preferences.

4.2.2.3 *SD-based User Factor Fading*

As in user factor fading, this strategy alters latent user vectors. However, the multiplier here is not a constant but it depends on the volatility of user's factors. The assumption behind this strategy is that highly

volatile latent vectors (the ones that change a lot), are unstable. Therefore, forgetting should be increased until the latent vectors stabilize.

Similarly to sensitivity-based forgetting, in this strategy we measure how much the latent user factor changed compared to the previous time point. We calculate again the squared difference between p_u^{t+1} and p_u^t and denote it as Δ_{p_u} (cf. Equation 29). Subsequently, we use the standard deviation of Δ_{p_u} in an exponential function:

$$\overrightarrow{p_u^{t+1}} = \alpha^{-SD(\Delta_{p_u})} \cdot \overrightarrow{p_u^t} \tag{38}$$

For high standard deviation of Δ_{p_u} the exponential function takes low values, penalizing unstable user vectors. The parameter α controls the extent of the penalty. For $\alpha > 1$ this function always takes values in the range $[0, 1)$.

4.2.2.4 *Recall-based User Factor Fading*

As in the previous strategy, users' latent vectors are also multiplied with a weight, which, here is based on user-specific recall. The idea is as follows: if a prediction model performs poorly for a user in terms of incremental recall, this forgetting strategy assumes that preferences of this user are changing. Therefore, forgetting should be amplified. In contrast, if the performance of the model is high, then forgetting is suppressed, so that a stable and well functioning model is not altered by the forgetting. To model this strategy we use the following formula:

$$\overrightarrow{p_u^{t+1}} = (-\alpha^{-incrRecall_u^t} + 1) \cdot \overrightarrow{p_u^t} \tag{39}$$

The exponential term $-\alpha^{-incrRecall_u^t} + 1$ takes high values for high recall values (for $\alpha > 1$). Therefore, if a model performs well, this term is close to 1, which makes the effect of the forgetting strategy low. Otherwise, a lower value of the exponential function increases the forgetting rate.

4.2.2.5 *Forgetting Popular Items*

This strategy is opposite to the one presented in Section 4.2.2.1. Here, popular items are penalized, so that their impact on the model is reduced. Forgetting popular items can be used to decrease the impact of mainstream products onto a preference model.

To achieve that, an exponential function is used that decreases the multiplier of the latent item vector (for $\alpha > 1$), when an item was rated by many users.

$$\overrightarrow{q_i^{t+1}} = \alpha^{-|R(i)|} \cdot \overrightarrow{q_i^t} \tag{40}$$

4.3 ENFORCING FORGETTING ON A STREAM OF RATINGS

Thus far, we have described how to select information that should be forgotten. In this section we discuss how the forgetting is implemented, i.e. how an impact of selected information can be removed from a model. We propose three alternative implementations. First however, we describe our baseline algorithm that does not use forgetting techniques. We use this algorithm in all our implementations and also in experiments as a comparison baseline.

4.3.1 *Baseline Algorithm*

We extend a state-of-the-art matrix factorization algorithm BRISMF by Takács et al. [Tak+09]. BRISMF is a batch method, however, Tákacs et al. also proposed an algorithm for retraining latent users features (cf. Algorithm 2. in [Tak+09]) that can be used as a stream-based algorithm. Latent user features are updated as new observations arrive in a stream, ideally in real time. Since item features are not updated online (as dictated by Tákacs et al. [Tak+09]), the method requires an initial phase, in which the latent item features are trained. We adopted this procedure also in our extension of this algorithm.

4.3.1.1 *Initial Phase*

Similarly to [Tak+09], in this phase we use stochastic gradient descent (SGD) to decompose the user-item-rating matrix R into two matrices of latent factors $R \approx P \cdot Q$, where P is the latent user matrix and Q the latent item matrix with elements $p_{u,f}$ and $q_{i,f}$ respectively. f is an index of the corresponding latent factor. In every iteration of SGD we use the following formulas to update latent factors [Tak+09] (notation adjusted):

$$p_{u,f} \leftarrow p_{u,f} + \eta \cdot (predictionError \cdot q_{i,f} - \lambda \cdot p_{u,f}) \tag{41}$$

$$q_{i,f} \leftarrow q_{i,f} + \eta \cdot (predictionError \cdot p_{u,f} - \lambda \cdot q_{i,f}) \tag{42}$$

Where η is a learning rate of the SGD and λ is a regularization parameter that prevents SGD from overfitting and $f \in \{1, 2, ..., k\}$ with k equal to the number of latent dimensions.

4.3.1.2 *Online Phase*

Once the initial phase is finished the online phase starts. Here, evaluation and prediction tasks run in parallel (cf. Section 4.4 for more information on the evaluation setting). With every new rating arriving in a data stream the corresponding latent user factor is updated.

The pseudo code showing our extended incremental training approach is presented in Algorithm 3. The algorithm takes as input the original rating matrix R, two factor matrices P and Q learnt in the initial phase and the aforementioned parameters required by the SGD. $r_{u,i}$ is most recent rating in a data stream that the algorithm uses to update the preference model, i.e. the latent user vector p_u. The update is carried out by performing further iterations of SGD using the new rating. $optimalNumberOfEpochs$ is determined during the initial phase. The complexity of updating a model with one rating is $O(E)$, where $E = optimalNumberOfEpochs$.

Algorithm 3 Incremental Learning - Baseline

Input: $r_{u,i}, R, P, Q, \eta, k, \lambda$, $optimalNumberOfEpochs$

1: $p_u \leftarrow$ getLatentUserVector(P, u)
2: $q_i \leftarrow$ getLatentItemVector(Q, i)
3: `Extend and initialize new dimensions`(p_u, q_i, P, Q, R)
4: $\widehat{r}_{u,i} = p_u \cdot q_i$ //predict a rating for $r_{u,i}$
5: evaluatePrequentially$(\widehat{r}_{u,i}, r_{u,i})$ //update evaluation measures
6: R.storeRating$(r_{u,i})$
7: $epoch = 0$
8: **while** $epoch < optimalNumberOfEpochs$ **do**
9: epoch++
10: $predictionError = r_{u,i} - p_u \cdot q_i$
11: **for all** latent dimensions $f \neq 1$ in p_u **do**
12: $p_{u,f} \leftarrow p_{u,f} + \eta \cdot (predictionError \cdot q_{i,f} - \lambda \cdot p_{u,f})$
13: **end for**
14: **end while**

In the pseudo code two of our extensions to the BRISMF algorithm are visible (cf. line 3 in Algorithm 3 and Algorithm 4):

- extending dimensions of the matrix
- different initialization of new dimensions

Extending dimensions of the matrix is an essential feature for stream-based algorithms. In a stream new users and items are introduced into the system frequently. Those users and items do not appear in the user/item matrix from the training phase of the algorithm. In order to make predictions for those users we extend the original matrix by new rows or columns. In experiments with offline datasets this extension allows to reduce the number of missing predictions considerably.

Our second extension regards initialization of new dimensions in the matrix. According to Takács et al. [Tak+09] latent matrices are initialized with small random values centered around zero. In batch algorithms this type of initialization is not problematic, since after the

initialization those values are overridden in multiple iterations of gradient descent that scans several times over a training set. In a data stream, however, scanning a training set multiple times is not possible. Ideally, stream-based algorithms are one-pass algorithms that scan all data only once. Therefore, we adjust the initialization of new dimensions in the matrix to be more meaningful from the start. We initialize new latent dimensions with average values over all latent users/items. After the initial phase of the algorithm, the latent matrices already contain meaningful values, therefore, their averages are different from just random values. Additionally, we add a uniformly distributed random component Z as in [Tak+09] from a small range of $(-0.02, 0.02)$. Thus, a new user vector p_{new_user} and new item vector q_{new_item} are initialized as follows:

$$p_{new\_user,f} = \frac{1}{|U|} \cdot \sum_{i=1}^{|U|} p_{i,f} + Z \sim \mathcal{U}(-0.02, 0.02) \tag{43}$$

$$q_{new\_item,f} = \frac{1}{|I|} \cdot \sum_{i=1}^{|I|} q_{i,f} + Z \sim \mathcal{U}(-0.02, 0.02) \tag{44}$$

Where f is the index of a latent dimension, U is a set of all users and I a set of all items. Accordingly, new user and items are treated as average users/items until there is enough learning examples to make them more specialized. The pseudo code presenting the initialization of new dimensions is shown in Algorithm 4.

Algorithm 4 Extend and initialize new dimensions

Input: p_u, q_i, P, Q, R

if p_u == null **then** {if u is a new user}
  R.addNewUserDimension(u)
  $\forall f : p_{u,f} = \frac{1}{|U|} \cdot \sum_{i=1}^{|U|} p_{i,f} + Z \sim \mathcal{U}(-0.02, 0.02)$
  P.addLatentUserVector(p_u)
end if
if q_i == null **then** {if i is a new item}
  R.addNewItemDimension(i)
  $\forall f : q_{i,f} = \frac{1}{|I|} \cdot \sum_{i=1}^{|I|} q_{i,f} + Z \sim \mathcal{U}(-0.02, 0.02)$
  Q.addLatentItemVector(q_i)
end if

Those two extensions are necessary adaptations of the BRISMF algorithm to the streaming scenario. However, our central contributions are the extensions to follow in the next subsections together with the forgetting strategies discussed before. Our goal is to investigate their impact.

Algorithm 5 Incremental Learning with Rating-based Forgetting

Input: $r_{u,i}, R, P, Q, \eta, k, \lambda$

1: $p_u \leftarrow$ getLatentUserVector(P, u)
2: $q_i \leftarrow$ getLatentItemVector(Q, i)
3: `Extend and initialize new dimensions`(p_u, q_i, P, Q, R)
4: $\widehat{r}_{u,i} = p_u \cdot q_i$ //predict a rating for $r_{u,i}$
5: evaluatePrequentially$(\widehat{r}_{u,i}, r_{u,i})$ //update evaluation measures
6: $\overrightarrow{r}_{u*} \leftarrow$ getUserRatings(R, u)
7: $\overrightarrow{(r}_{u*})$.addRating$(r_{u,i})$
8: **rating-basedForgetting**$\overrightarrow{(r}_{u*})$ **//obsolete ratings removed**
9: $epoch = 0$
10: **while** $epoch < optimalNumberOfEpochs$ **do**
11: epoch++
12: **for all** $r_{u,i}$ in $\overrightarrow{r}_{u*}$ **do**
13: $p_u \leftarrow$ getLatentUserVector(P, u)
14: $q_i \leftarrow$ getLatentItemVector(Q, i)
15: $predictionError = r_{u,i} - p_u \cdot q_i$
16: **for all** latent dimensions $f \neq 1$ in p_u **do**
17: $p_{u,f} \leftarrow p_{u,f} + \eta \cdot (predictionError \cdot q_{i,f} - \lambda \cdot p_{u,f})$
18: **end for**
19: **end for**
20: **end while**

4.3.2 *Matrix factorization for Rating-based Forgetting*

In this subsection we present incremental matrix factorization with rating-based forgetting in the online phase. This implementation extends the baseline algorithm from the previous subsection. Therefore, all differences in the performance between this algorithm and the baseline are due to our forgetting techniques. Those forgetting strategies can be also applied to different incremental matrix factorization methods analogously.

Algorithm 5 shows pseudo code of our rating-based forgetting. First, we introduce a new notation, where $\overrightarrow{r}_{u*}$ is a vector of all ratings of user u. In line 6 such a user vector is retrieved from the matrix and in line 8 a rating-based forgetting strategy is called. Consequently, all rating from the user's vector that have been deemed obsolete by the forgetting strategy are removed. Subsequently, the user's latent vector p_u is retrained using all remaining ratings from the $\overrightarrow{r}_{u*}$ rating vector.

This procedure introduces a further loop into the algorithm, due to which its complexity of a model update rises to $O(E \cdot \|\overrightarrow{r}_{u*}\|)$.

4.3.3 *Matrix factorization for Latent Factor Forgetting*

In Algorithm 6 we present an implementation that uses latent factor forgetting strategies. In lines 6 and 7 a forgetting strategy is invoked to modify the corresponding latent user or item vectors.

Algorithm 6 Incremental Learning with Latent Factor Forgetting

Input: $r_{u,i}, R, P, Q, \eta, k, \lambda$

1: $p_u \leftarrow$ getLatentUserVector(P, u)
2: $q_i \leftarrow$ getLatentItemVector(Q, i)
3: `Extend and initialize new dimensions`(p_u, q_i, P, Q, R)
4: $\widehat{r}_{u,i} = p_u \cdot q_i$ //predict a rating for $r_{u,i}$
5: evaluatePrequentially$(\widehat{r}_{u,i}, r_{u,i})$ //update evaluation measures
6: $p_u \leftarrow$ **latentForgetting(**p_u**)**
7: $q_i \leftarrow$ **latentForgetting(**q_i**)**
8: $epoch = 0$
9: **while** $epoch < optimalNumberOfEpochs$ **do**
10: epoch++
11: $predictionError = r_{u,i} - \widehat{r}_{u,i}$
12: **for all** latent dimensions $f \neq 1$ in p_u **do**
13: $p_{u,f} \leftarrow p_{u,f} + \eta \cdot (predictionError \cdot q_{i,f} - \lambda \cdot p_{u,f})$
14: **end for**
15: **end while**

Other than that, no further changes compared to the baseline algorithm are necessary. Latent factor forgetting does not require retraining on past ratings, therefore the complexity is here again at $O(E)$.

4.3.4 *Approximation of Rating-based Forgetting*

Since rating-based forgetting increased the complexity of updating a preference model to $O(E \cdot \|\vec{r}_{u*}\|)$, we propose a faster approximative method of implementing this type of forgetting.

The implementation in Algorithm 7 eliminates the need for the additional loop for retraining of the user latent vector on past ratings. Instead of this loop, impact of learning upon a rating is stored in a *deltaStorage* (cf. line 25). The impact in form of δ_{p_u} results from subtraction of a latent user vector before and after learning. If at a later time point this rating has to be forgotten, the impact of this rating is retrieved from the *deltaStorage* (cf. line 12) and subtracted from the corresponding latent user vector (cf. line 13).

Update of the preference model upon a new rating is done in the same way as in the baseline algorithm. The complexity of an update is again $O(E)$. However, the procedure requires a higher memory consumption due to the *deltaStorage*.

Algorithm 7 Incremental Learning with Approximative Rating-based Forgetting

Input: $r_{u,i}, R, P, Q, \eta, k, \lambda$

1: $p_u \leftarrow$ getLatentUserVector(P, u)
2: $q_i \leftarrow$ getLatentItemVector(Q, i)
3: `Extend and initialize new dimensions`(p_u, q_i, P, Q, R)
4: $\widehat{r}_{u,i} = p_u \cdot q_i$ //predict a rating for $r_{u,i}$
5: evaluatePrequentially$(\widehat{r}_{u,i}, r_{u,i})$ //update evaluation measures
6: $\overrightarrow{r}_{u*} \leftarrow$ getUserRatings(R, u)
7: $\overrightarrow{r}_{u*}$.addRating$(r_{u,i})$
8: $remainingRatings = ratingBasedForgetting(\overrightarrow{r}_{u*})$
9: $ratingsToBeForgotten = \overrightarrow{r}_{u*} - remainingRatings$
10: **for all** $rating_{u,i}$ in $ratingsToBeForgotten$ **do**
11: $p_u \leftarrow$ getLatentUserVector(P, u)
12: $\delta_{p_u} = deltaStorage$.getUserVectorImpact$(rating_{u,i})$
13: $p_u \leftarrow p_u - \delta_{p_u}$
14: **end for**
15: $epoch = 0$
16: $p_u^{beforeUpdate} = p_u$
17: **while** $epoch < optimalNumberOfEpochs$ **do**
18: epoch++
19: $predictionError = r_{u,i} - \overrightarrow{p_u} \cdot q_i$
20: **for all** latent dimensions $f \neq 1$ in p_u **do**
21: $p_{u,f} \leftarrow p_{u,f} + \eta \cdot (predictionError \cdot q_{i,f} - \lambda \cdot p_{u,f})$
22: **end for**
23: **end while**
24: $\delta_{p_u} = p_u^{beforeUpdate} - p_u$
25: $deltaStorage$.storeImpactOnUser(δ_{p_u})

4.4 EVALUATION SETTINGS

In this section we describe how we evaluate our methods. Our evaluation protocol encompasses

- a method for splitting datasets for incremental matrix factorization
- incremental recall measure by Cremonesi et al. [CKT10]
- parameter optimization
- significance testing

We applied this evaluation protocol to all 8 datasets used in our experiments (cf. Sec. 4.5).

4.4.1 *Dataset Splitting*

Our method operates on a stream of ratings. However, matrix factorization requires an initialization phase. According to the description of the BRISMF algorithms, which we adopted here with modifications (cf. baseline algorithm in Sec. 4.3.1), latent item features are trained only in the initial phase. Therefore, this phase is of even higher importance to the BRISMF algorithm.

Because our methods operate in two phases, we use the evaluation protocol from [Mat+15; MS15], briefly explained hereafter. Figure 3 represents an entire dataset with with three parts. Part 1) is used for initial training in the batch mode. To evaluate training of latent factors on part 1), we use the part 2) of the dataset ("batch testing"). After the initial training is finished, our method changes into the streaming mode, which is its main mode.

In this mode we use prequential evaluation, as proposed by Vinagre et al. [VJG15b] for recommender systems and by Gama et al. for general data stream mining [GSR09]. As Vinagre et al. described in their paper, the prequential evaluation offers several advantages compared to the conventional offline evaluation. Those advantages include continuous monitoring of selected metrics over time, usage of the real-time metrics in the logic of the algorithms, respecting the time dimension (unlike e.g. cross validation, which shuffles the data instances), etc. Due to those benefits of the prequential evaluation we use it in our streaming mode.

The main idea behind it is as follows. For each new rating, first, a prediction is made and evaluated. Only after that, the new rating is used for updating the corresponding preference model. Due to this temporal separation of prediction and update procedures, separation of the training and test datasets is guaranteed. In our experiments we use the following split ratios for the datasets: 20% for batch training, 30 % for batch testing and 50 % for the stream mode.

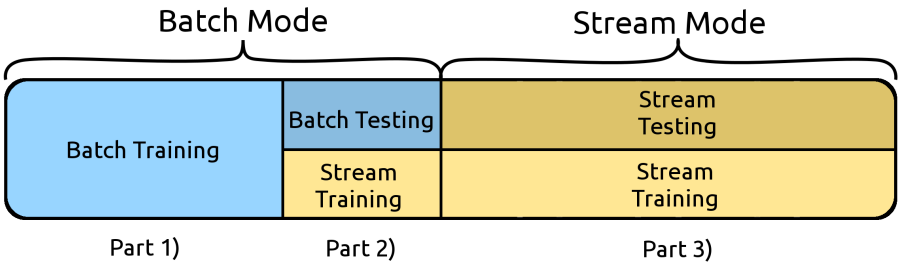

Figure 3.: Split of the dataset between the initialization and online phase (figure from [MS15]).

Since part 1) and 3) are used for training, part 2) of the dataset would represent a temporal gap in the training data. For stream-based

methods that rely heavily on time aspects it is highly beneficial to maintain time continuity in the model. Therefore, we also use part 2) of the dataset for stream training. However, since it was used for batch testing, we don't use it as test set for the streaming mode.

4.4.2 *Evaluation Measure*

As quality measure we use incremental recall, as proposed by Cremonesi et al. [CKT10]. It measures how often a recommender system can find a relevant item among random items in a stream. This measure should not be confused with the conventional recall. Also, while conventional precision and recall are complementary to each other and should always be considered together, it is not the case with the incremental recall and incremental precision. If the incremental recall was measured, then incremental precision can be derived from it and, therefore, it is redundant. For readers unfamiliar with this measure we refer to [CKT10] and summarise the process of measuring it briefly.

First, in the process of measuring incremental recall, a relevance threshold is defined, above which items are considered relevant e.g. more than 4 out of 5 stars. If a relevant item is encountered in a stream, 1000 further items are chosen randomly. Those additional 1000 items are assumed to be irrelevant. All those 1000 random items and the relevant item are put into one set without indication of relevance. Subsequently, a recommender systems ranks all 1001 items from this set. If the relevant item has been ranked as one of top N items, then a hit is counted. The final value of incremental recall for a given N is calculated using the following formula:

$$incrementallRecall@N = \frac{\#hits}{|Testset|} \tag{45}$$

In our experiments we use *incrementalRecall*@10. In experiments with explicit rating feedback, the ranking made by the recommender systems is sorted with respect to the relevance score (highest first). In experiments with positive-only feedback, the value of 1 indicates the existence of feedback. Therefore, the ranking in this case is sorted with respect to the proximity of a predicted rating to 1.

4.4.3 *Parameter Selection*

Each of our forgetting strategies uses an additional parameter that needs to be set in advance (e.g. the size of a sliding window). To find the optimal values of the parameters we used a grid search algorithm that optimizes the average incremental recall for each dataset separately.

To avoid overfitting, the grid search was performed on a small sample from our datasets, called optimization set. The size of the optimization sets, expressed as a percentage of all users, ranged between 0.01 and 0.1 depending on the dataset size (cf. Table 4, column "Ratio of Users for Parameter Optimization"). Using the optimization set, we determined an approximately optimal parameter value. This value was then applied onto the holdout dataset, which is the remaining, bigger part of a dataset. The results reported in the next section are results on the holdout sets.

The remaining parameters used by matrix factorization and SGD were set to the following, approximately optimal values: number of dimensions $k = 40$, learning rate $\eta = 0.003$ and regularization parameter $\lambda = 0.01$.

4.4.4 *Significance Testing*

To study the effect of our forgetting strategies we use significance testing. However, in the streaming scenario following all requirements of statistical tests is not a trivial task. One of the prerequisites of statistical tests is independence of observations of a random variable.

In our case the random variable is the average incremental recall. However, considering two measurements of incremental recall at timepoint t and $t+1$ as independent would be wrong, since incremental recall is a cumulative measure. Therefore, quality at timepoint t affects the measurement of quality at timepoint $t+1$. Consequently, the prerequisite of independent observations is violated.

As a solution to this problem we propose an alternative understanding of an observation. We partition every dataset into n disjoint, consecutive parts along the time dimension (imagine Fig. 3 n times along the time axis). Each of the parts is a sample of the entire dataset i.e. a sample from the same population. Since the samples are disjoint, they are also independent. As one observation we define the average incremental recall on one such sample. Consequently, on each dataset we observe n realisations of the random variable for each forgetting strategy. In our experiments we use $n = 10$, except for the ML100k dataset, where $n = 5$ due to its small size (cf. Table 4, column "Observations for Significance Testing").

Having n observations for each forgetting strategy and for our baseline, the "No Forgetting Strategy", we first test, if there is a significant difference among the strategies. Since we have several forgetting strategies and each one has n observations, we use the Friedman test for this purpose . It is more appropriate here than e.g. ANOVA, since it does not assume the normal distribution of the random variable and it is not parametric. If the null hypothesis is rejected, then it can

be assumed that there is a significant difference among the forgetting strategies.

If this is the case, we perform post-hoc tests to find out, which forgetting strategies are significantly better than our baseline, the "No Forgetting Strategy". Therefore, we use the Wilcoxon signed rank test (paired measurements with no assumption of normal distribution) with the following null hypothesis:

$$\overline{incr.Recall}_{StrategyX} = \overline{incr.Recall}_{NoForgettingStrategy} \tag{46}$$

and alternative hypothesis:

$$\overline{incr.Recall}_{StrategyX} > \overline{incr.Recall}_{NoForgettingStrategy} \tag{47}$$

where $\overline{x}$ denotes the median of the vector x.

If the null hypothesis is rejected, then the forgetting strategy X is significantly better than no forgetting. Since we test multiple hypothesis, we apply a correction for multiple testing according to the Hommel's method [Sha95] to avoid the alpha error accumulation. In the next section we report the corresponding adjusted p-values and a summary of significant improvements.

4.5 EXPERIMENTS

In this section we present results of our experiments on eight real-world datasets. We divide the results into ones showing the effects of the forgetting strategies (cf. Sec. 4.5.1) and ones showing the effects of our approximation of the rating-based implementation of forgetting (cf. Sec. 4.5.2). In total we conducted more than 1040 experiments on a cluster running a (Neuro)Debian operating system [HH12]. In those experiments we used datasets from Table 4.

The first type of datasets encompasses data with explicit rating feedback: MovieLens 1M and 100k[1] [HK16], a sample of 10 000 users from the extended Epinions [MA06] dataset and a sample of the same size from the Netfilx dataset. In this type of datasets our selection is limited, because many forgetting strategies require timestamp information.

The second type of datasets is based on positive-only feedback. Those datasets contain chronologically ordered user-item pairs in the form (u, i). Music-listen consists of music listening events, where each pair corresponds to a music track being played by a user. Music-playlist consists of a timestamped log of music track additions to personal playlists. Contrary to Music-listen, Music-playlist contains *only* unique (u, i) pairs – users are not allowed to add a music track twice to the same playlist. Both Music-listen and Music-playlist are extracted

1 http://www.movielens.org/

Dataset	Ratings	Users	Items	Sparsity	Ratio of Users for Parameter Optimization	Observations for Significance Testing
Music-listen	335,731	4,768	15,323	99.54%	0.1	10
Music-playlist	111,942	10,392	26,117	99.96%	0.1	10
LastFM-600k	493,063	164	65,013	95.38%	0.1	10
ML1M GTE5	226,310	6,014	3,232	98.84%	0.1	10
ML100k	100,000	943	1,682	93.7%	0.1	5
Netflix(10k users)	2,146,187	10,000	17,307	98.76%	0.01	10
Epinions (10k users)	1,016,915	10,000	365,248	99.97%	0.03	10
ML1M	1,000,209	6,040	3,706	95.53%	0.05	10

Table 4.: Description of datasets used in experiments. The column "Ratio of Users for Parameter Optimization" indicates what ratio of users was used to create an optimization dataset for the grid search. "Observations for Significance Testing" indicates the number of partitions of a dataset used as observations for significance testing. Table from [Mat+17]

from Palco Principal[2], an online community of portuguese-speaking musicians and fans. Furthermore, we also use a subset of the LasfFM[3] dataset [Cel10] – LastFM-600k – and a binarized version of MovieLens 1M dataset that we call ML1M-GTE5 hereafter. In ML1M-GTE5 we assume that a rating value of 5 indicates a positive feedback. All remaining ratings have been removed and considered negative.

4.5.1 *Impact of Forgetting Strategies*

To show the impact of our forgetting strategies we compare them with the baseline algorithm from Sec. 4.3.1. This baseline employs no forgetting strategy. Therefore, we call it "No Forgetting Strategy" hereafter. In this subsection no approximation from Alg. 7 was used (for results using the approximation, see the next subsection).

First, we tested if the application of forgetting strategies has a significant impact on the quality of recommendations measured in incremental recall. For this purpose we used the Friedman rank sum test for each dataset separately as an omnibus test. The null hypothesis of this test states that all recall values are equal, no matter what forgetting strategy or no forgetting strategy was used (cf. Sec. 4.4.4 for more details on the test and motivation behind it).

In Table 5 we present results of the test on each dataset. The null hypothesis was clearly rejected on all datasets, which is indicated by low

2 http://www.palcoprincipal.com
3 http://last.fm

p-values. Consequently, we conclude that forgetting strategies make a significant difference in incremental recall values. Further in this section, we use post-hoc tests to find out which forgetting strategies are significantly better than no forgetting.

Dataset	p-value (Friedman rank sum test)
Epinions Extended (10k users sample)	4.381e-16
Lastfm 600k	1.852e-04
ML1M GTE5	1.911e-09
ML1M	4.565e-11
ML100k	3.389e-09
Netflix (10k users sample)	1.735e-09
Music-listen	2.773e-05
Music-Playlist	7.246e-15

Table 5.: Results of the Friedman rank sum test as an omnibus test for each dataset. Very low p-values indicate that forgetting makes a significance difference in the quality of recommendations (table from [Mat+17]).

In Figure 4 we visualize the results of forgetting strategies on datasets with positive-only feedback. This figure contains box plots of incremental recall (higher values are better). Incremental precision in the streaming setting can be derived from the recall measure and is, therefore, redundant (cf. [CKT10]). Thus, we do not present the incremental precision.

Horizontal bars in each box in Fig. 4 represent medians of incremental recall from multiple partitions of a dataset (cf. Sec. 4.4). The hinges of each box represent the first and the third quartile of the distribution. Dots stand for outliers.

In the figure we, again, see that forgetting strategies have a great impact on the quality of recommendations as compared to the "No Forgetting Strategy" (leftmost in the plots). The latent factor forgetting strategies are particularly successful. On three out of four positive-only datasets the "SD-based User Factor Fading" was the best strategy. The "Forget Unpopular" strategy (also a latent factor forgetting strategy) performed the best on the MLGTE5 dataset only.

From this group of forgetting strategies the "Forget Popular" strategy is not recommendable. It performed better than the baseline on the Lastfm dataset only and otherwise considerably worse. The performance of rating-based strategies was generally worse than the one of the latent factor forgetting, often marginally different from the baseline.

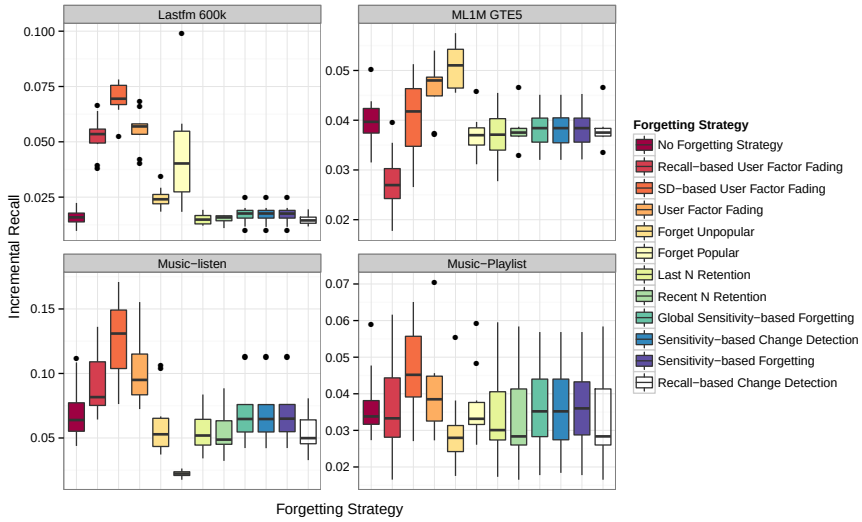

Figure 4.: Results of our forgetting strategies vs. "No Forgetting Strategy" (leftmost in the plots) on datasets with **positive-only feedback** (higher values are better). Latent factor based strategies are particularly successful (figure from [Mat+17]).

In Tabs. 6 and 7 we present the corresponding results of experiments with positive only feedback. The column "Param." describes the parameter setting for each forgetting strategy. The meaning of the parameter depends on the strategy itself (cf. Sec. 4.2). The parameter was determined by a grid search on a separate dataset not used for the final evaluation.

The column with mean incremental recall uses the following notation: $mean_{\pm std.\,deviation}$. The values of mean and standard deviation are based on multiple runs on different, consecutive parts of each dataset (cf. Sec. 4.4.4). The number of runs is indicated in Tab. 4 by column "Observations for Significance Testing". The forgetting strategy with the best value of mean incremental recall is marked in red.

P-values in the table refer to the Wilcoxon signed rank test, which we used as a post-hoc test (cf. Sec. 4.4.4). They are adjusted using Hommel's method to account for multiple testing. Values in bold font and a single asterisk indicate that the given strategy is better than the "No Forgetting Strategy" with significance level of 0.1. Values marked by two asterisks are significant at level of 0.05 and by three asterisks at level of 0.01. The column "Runtime" follows the same notation as the mean incremental recall. Values in each cell of this column represent a mean runtime in seconds on one partition of each dataset.

Tabs. 6 and 7 show that on all datasets with positive only feedback there is at least one forgetting strategy that is significantly better than no forgetting at the significance level better than 0.054. On three out of

Forgetting Strategy	Param.	Mean Incr. Recall	Adjusted P-value	Runtime (ms)
Lastfm 600k				
No Forgetting Strategy	0	$0.01583_{\pm 0.00361}$	-	$65.84_{\pm 5.23}$
Recall-based User Factor Fading	10^{12}	**0.0525**$_{\pm 0.00909}$	**0.00684*****	**74.11**$_{\pm 7.78}$
SD-based User Factor Fading	**1.2**	**0.06953**$_{\pm 0.00768}$	**0.00684*****	**64.21**$_{\pm 4.91}$
User Factor Fading	**0.5**	**0.05541**$_{\pm 0.00889}$	**0.00684*****	**64.31**$_{\pm 3.2}$
Forget Unpopular	**2**	**0.02458**$_{\pm 0.00465}$	**0.00684*****	**66.41**$_{\pm 4.87}$
Forget Popular	**1.005**	**0.04436**$_{\pm 0.02379}$	**0.00684*****	**66.24**$_{\pm 5.11}$
Last N Retention	10	$0.01496_{\pm 0.00241}$	0.88379	$88.7_{\pm 9.4}$
Recent N Retention	1h	$0.01519_{\pm 0.00183}$	0.88379	$86.04_{\pm 7.31}$
Global Sensitivity-based Forgetting	0.1	$0.01718_{\pm 0.00424}$	0.32032	$55.99_{\pm 5.77}$
Sensitivity-based Change Detection	0.5	$0.01716_{\pm 0.00426}$	0.32032	$58.29_{\pm 3.48}$
Sensitivity-based Forgetting	10	$0.01716_{\pm 0.00426}$	0.32032	$58.14_{\pm 5.61}$
Recall-based Change Detection	0.05	$0.01478_{\pm 0.00225}$	0.88379	$1284.42_{\pm 196.36}$
ML1M GTE5				
No Forgetting Strategy	0	$0.04005_{\pm 0.00501}$	-	$9.04_{\pm 1.36}$
Recall-based User Factor Fading	10^{12}	$0.02787_{\pm 0.00621}$	0.99903	$9.38_{\pm 1.28}$
SD-based User Factor Fading	1.08	$0.04075_{\pm 0.00795}$	0.99903	$9.78_{\pm 2.46}$
User Factor Fading	**0.99**	**0.04627**$_{\pm 0.00545}$	**0.00977*****	**8.94**$_{\pm 1.38}$
Forget Unpopular	**1.2**	**0.05094**$_{\pm 0.00452}$	**0.00977*****	**8.9**$_{\pm 0.69}$
Forget Popular	1.00001	$0.03706_{\pm 0.00403}$	0.99903	$9.74_{\pm 2.41}$
Last N Retention	5	$0.03692_{\pm 0.00517}$	0.99903	$13.67_{\pm 2.06}$
Recent N Retention	1h	$0.03803_{\pm 0.00342}$	0.99903	$43.77_{\pm 13.41}$
Global Sensitivity-based Forgetting	0.1	$0.03792_{\pm 0.00405}$	0.99903	$11.3_{\pm 1.92}$
Sensitivity-based Change Detection	0.5	$0.03789_{\pm 0.00408}$	0.99903	$10.59_{\pm 1.61}$
Sensitivity-based Forgetting	10	$0.03795_{\pm 0.00407}$	0.99903	$10.81_{\pm 1.53}$
Recall-based Change Detection	0.05	$0.03808_{\pm 0.00335}$	0.99903	$45.06_{\pm 10.74}$

Table 6.: Results on datasets with positive only feedback (Lastfm 600k and ML1M GTE5). Best value of incremental recall is marked in red. Asterisks indicate that a given strategy is significantly better than the no forgetting strategy (* at 0.1; ** at 0.05; *** at 0.01). Table from [Mat+17]

Forgetting Strategy	Param.	Mean Incr. Recall	Adjusted P-value	Runtime (ms)
Music-Listen				
No Forgetting Strategy	0	$0.07083_{\pm 0.02328}$	-	$68.42_{\pm 4.09}$
Recall-based User Factor Fading	10^{12}	$0.09178_{\pm 0.02308}$	0.58887	$61.36_{\pm 5.35}$
SD-based User Factor Fading	**1.2**	**0.12713**$_{\pm 0.02976}$	**0.01075****	**61.19**$_{\pm 5.7}$
User Factor Fading	0.5	$0.1033_{\pm 0.02649}$	0.18555	$67.83_{\pm 8.67}$
Forget Unpopular	2	$0.06078_{\pm 0.02505}$	1	$66.58_{\pm 9.02}$
Forget Popular	1.005	$0.02227_{\pm 0.00241}$	1	$73.01_{\pm 9.58}$
Last N Retention	40	$0.05588_{\pm 0.01745}$	1	$101.73_{\pm 14.09}$
Recent N Retention	1 week	$0.05527_{\pm 0.01885}$	1	$224.31_{\pm 108.35}$
Global Sensitivity-based Forgetting	0.1	$0.07082_{\pm 0.02473}$	1	$34.99_{\pm 1.88}$
Sensitivity-based Change Detection	0.5	$0.07082_{\pm 0.02476}$	1	$36.8_{\pm 4.57}$
Sensitivity-based Forgetting	10	$0.0709_{\pm 0.02473}$	1	$35.97_{\pm 4.59}$
Recall-based Change Detection	0.05	$0.05477_{\pm 0.01661}$	1	$249.34_{\pm 93.92}$
Music-Playlist				
No Forgetting Strategy	0	$0.03712_{\pm 0.00946}$	-	$4.12_{\pm 0.9}$
Recall-based User Factor Fading	10^{12}	$0.0366_{\pm 0.01358}$	0.99805	$4.16_{\pm 0.75}$
SD-based User Factor Fading	**1.2**	**0.04708**$_{\pm 0.01214}$	**0.05372***	**4.2**$_{\pm 0.49}$
User Factor Fading	0.99	$0.04058_{\pm 0.01232}$	0.99805	$3.85_{\pm 0.63}$
Forget Unpopular	1.5	$0.02972_{\pm 0.0109}$	0.99805	$4.09_{\pm 0.91}$
Forget Popular	1.00001	$0.03679_{\pm 0.00986}$	0.99805	$4.1_{\pm 0.67}$
Last N Retention	40	$0.03393_{\pm 0.01201}$	0.99805	$10.94_{\pm 1.5}$
Recent N Retention	1 year	$0.03333_{\pm 0.01231}$	0.99805	$39.64_{\pm 48.78}$
Global Sensitivity-based Forgetting	0.1	$0.03609_{\pm 0.01171}$	0.99805	$4.2_{\pm 0.66}$
Sensitivity-based Change Detection	0.5	$0.03604_{\pm 0.01171}$	0.99805	$4.06_{\pm 0.75}$
Sensitivity-based Forgetting	10	$0.03616_{\pm 0.0116}$	0.99805	$4_{\pm 0.56}$
Recall-based Change Detection	0.05	$0.03333_{\pm 0.01231}$	0.99805	$38.75_{\pm 48.41}$

Table 7.: Results on datasets with positive only feedback (Music-Listen and Music-Playlist). Best value of incremental recall is marked in red. Asterisks indicate that a given strategy is significantly better than the no forgetting strategy (* at 0.1; ** at 0.05; *** at 0.01). Table from [Mat+17]

four datasets the best strategy was the SD-based User Factor Fading, on the remaining dataset it was the Forget Unpopular strategy.

Forgetting strategies brought on this type of datasets an improvement in the incremental recall of 118.18 % on average (as a result of comparison of the best strategy with the "No Forgetting Strategy"). Especially on the Lastfm 600k dataset recall improved from 0.01583 to 0.06953 . The median of improvement is 53,34%.

Not only did quality improve, the computation time decreased for the best strategy by 3,21 % on average. However, considering the high variance of runtime, this decrease is not substantial. Some strategies, e.g. "Recall-based Change Detection" showed higher computation time.

In Fig. 5 and in Tabs. 8 and 9 we present analogous results of experiments with datasets with explicit rating feedback. Also here, the latent factor based strategies perform the best except for the "Forget Popular" strategy. In Tabs. 8 and 9 we show improvements in quality of recommendations over the baseline. The significance level on three out of four of the datasets is better than 0.01. Only on the ML100k dataset no significant improvement could be shown. ML100k is a small dataset with only five observations, therefore, it is difficult to show significance on this dataset.

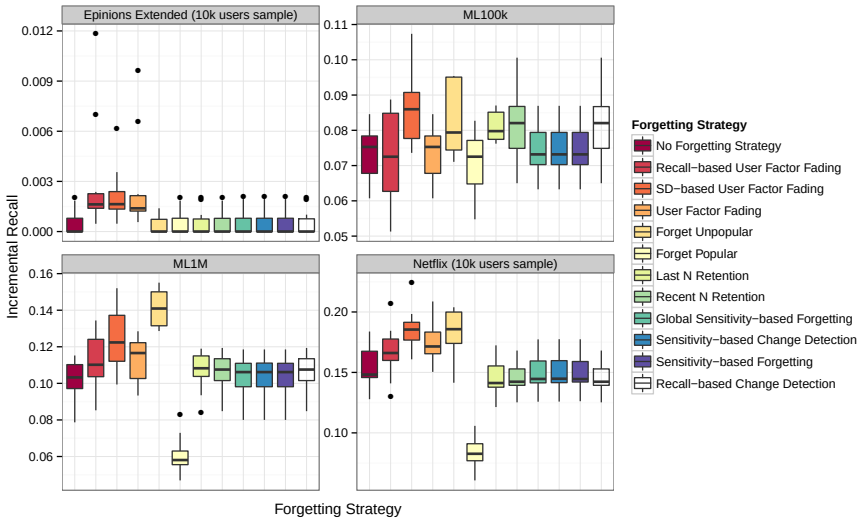

Figure 5.: Results of our forgetting strategies vs. "No Forgetting Strategy" (leftmost in the plots) on datasets with **explicit rating feedback** (higher values are better). Latent factor based strategies are particularly successful (figure from [Mat+17]).

The improvement of quality due to forgetting reached 147,83 % on average (best strategy vs. no forgetting). The percentage is so high because of the extreme improvement on the Epinions dataset. The median of the improvement amounts to 19,67%. The runtime, when

using the best strategy, increased on average by 48.62%, the median of the percentual runtime increase is 9,63%, though.

4.5.2 *Impact of the Approximative Implementation*

Since the rating-based forgetting strategies have a higher complexity than the latent factor based ones, we implemented also an approximative way of using them (cf. Sec. 4.3.4). This implementation stores a past impact of a rating and undoes it in an approximative way when the rating should be forgotten.

In Figure 6 we present the incremental recall values achieved by this approximative implementation (dashed line) in comparison to the original implementation of rating-based forgetting (solid line). The approximation performed similarly to the original implementation. In few cases it performed better (e.g. for the "Last N Retention" strategy on the ML1M GTE5 dataset). However, those cases are rather exceptional and no significant improvement can be concluded based on them. There are also cases with a decrease of the performance, such as on ML100k dataset.

Nevertheless, the goal of the approximation is to maintain a similar level of quality while decreasing the runtime. We present such a runtime comparison of those two implementations in Table 10. Values in the table represent median of a runtime of multiple runs of our algorithms. The values are grouped by dataset and implementation (approximation vs. the original rating-based implementation).

The approximation decreased the computation time for the strategies Last N Retention, Recent N Retention, Recall-based Change Detection. For the remaining three strategies the approximation often took more time despite lower complexity. This is explained by the fact that the approximative implementation of forgetting changes the latent model in a different way than the rating-based implementation does. Therefore, a forgetting strategy can select different ratings to forget depending on which implementation is used (approximative vs. rating-based). Consequently, a given forgetting strategy can decide to forget more, if the approximative implementation is used. Then, due to more storing and retrieving operations from the delta storage, the approximative implementation can require a longer computation time.

Consequently, we recommend the usage of the approximation only after prior testing of its behaviour with a given forgetting strategy.

4.6 CONCLUSIONS FROM FORGETTING METHODS

Before our work, adaptation to changes in recommender systems was implemented only by incorporating new information from a stream

Forgetting Strategy	Param.	Mean Incr. Recall	Adjusted P-value	Runtime (ms)
Epinions Extended (10k users sample)				
No Forgetting Strategy	0	$0.0005_{\pm 0.00084}$	-	$160.03_{\pm 16.1}$
Recall-based User Factor Fading	10^{10}	**0.00307**$_{\pm 0.00357}$	**0.00879*****	**451.89**$_{\pm 38.52}$
SD-based User Factor Fading	**1.08**	**0.00219**$_{\pm 0.00163}$	**0.00879*****	**154.45**$_{\pm 11.68}$
User Factor Fading	**0.5**	**0.00274**$_{\pm 0.00297}$	**0.00879*****	**152.92**$_{\pm 14.87}$
Forget Unpopular	2	$0.00035_{\pm 0.00057}$	1	$156.5_{\pm 15.81}$
Forget Popular	1.00001	$0.0005_{\pm 0.00084}$	1	$161.76_{\pm 18.74}$
Last N Retention	10	$0.0005_{\pm 0.00084}$	1	$192.48_{\pm 24.29}$
Recent N Retention	4 weeks	$0.0005_{\pm 0.00084}$	1	$2923_{\pm 601.75}$
Global Sensitivity-based Forgetting	0.1	$0.00052_{\pm 0.00087}$	1	$148.81_{\pm 16.9}$
Sensitivity-based Change Detection	0.5	$0.00052_{\pm 0.00087}$	1	$143.67_{\pm 10.98}$
Sensitivity-based Forgetting	10	$0.00052_{\pm 0.00087}$	1	$144.04_{\pm 13.9}$
Recall-based Change Detection	0.05	$0.00051_{\pm 0.00085}$	1	$5857.13_{\pm 4815.21}$
ML100k				
No Forgetting Strategy	0	$0.07336_{\pm 0.0093}$	-	$3.22_{\pm 0.39}$
Recall-based User Factor Fading	10^{10}	$0.07201_{\pm 0.0155}$	1	$3.24_{\pm 0.19}$
SD-based User Factor Fading	1.04	$0.08708_{\pm 0.01319}$	0.1875	$1.72_{\pm 0.12}$
User Factor Fading	0.99999999	$0.07336_{\pm 0.0093}$	1	$1.63_{\pm 0.09}$
Forget Unpopular	1.5	$0.08307_{\pm 0.01151}$	0.1875	$1.8_{\pm 0.11}$
Forget Popular	1.00001	$0.07039_{\pm 0.01093}$	1	$1.96_{\pm 0.28}$
Last N Retention	10	$0.08112_{\pm 0.00476}$	0.1875	$9.42_{\pm 0.83}$
Recent N Retention	1h	$0.08186_{\pm 0.01331}$	0.1875	$70.55_{\pm 7.59}$
Global Sensitivity-based Forgetting	0.1	$0.07462_{\pm 0.00901}$	0.375	$3.47_{\pm 0.39}$
Sensitivity-based Change Detection	0.5	$0.07462_{\pm 0.00901}$	0.375	$3.5_{\pm 0.3}$
Sensitivity-based Forgetting	10	$0.07462_{\pm 0.00901}$	0.375	$3.51_{\pm 0.34}$
Recall-based Change Detection	0.05	$0.08185_{\pm 0.01331}$	0.1875	$72.89_{\pm 7.98}$

Table 8.: Results on datasets with explicit rating feedback (Epinions and ML100k). Best value of incremental recall is marked in red. Asterisks indicate that a given strategy is significantly better than the no forgetting strategy (* at 0.1; ** at 0.05; *** at 0.01). Table from [Mat+17]

Forgetting Strategy	Param.	Mean Incr. Recall	Adjusted P-value	Runtime (ms)
ML1M				
No Forgetting Strategy	0	0.10211$\pm$0.01104	-	32.41$\pm$7.35
Recall-based User Factor Fading	10^{12}	0.11249$\pm$0.01479	0.10547	63.63$\pm$16.63
SD-based User Factor Fading	**1.1**	**0.1244**$\pm$0.01697	**0.02051****	**33.65**$\pm$7.07
User Factor Fading	**0.99**	**0.11327**$\pm$0.01237	**0.00586*****	**33.41**$\pm$7.33
Forget Unpopular	**1.2**	**0.14088**$\pm$0.01014	**0.00586*****	**37.44**$\pm$9.3
Forget Popular	1.00001	0.06073$\pm$0.01053	1	37.52$\pm$8.69
Last N Retention	**10**	**0.10679**$\pm$0.01115	**0.00586*****	**64.18**$\pm$3.56
Recent N Retention	**1 week**	**0.10585**$\pm$0.01056	**0.00782*****	**939.05**$\pm$103.08
Global Sensitivity-based Forgetting	**0.1**	**0.1041**$\pm$0.01153	**0.01172****	**25.88**$\pm$2.03
Sensitivity-based Change Detection	**0.5**	**0.10412**$\pm$0.01154	**0.01172****	**25.95**$\pm$2.19
Sensitivity-based Forgetting	**10**	**0.10408**$\pm$0.01151	**0.01172****	**25.81**$\pm$2.07
Recall-based Change Detection	**0.05**	**0.10585**$\pm$0.01056	**0.00782*****	**951.31**$\pm$107.55
Netflix (10k users sample)				
No Forgetting Strategy	0	0.15455$\pm$0.0165	-	38.24$\pm$9.32
Recall-based User Factor Fading	10^{10}	0.16687$\pm$0.02184	0.64063	73.58$\pm$13.47
SD-based User Factor Fading	**1.02**	**0.18644**$\pm$0.01707	**0.00977*****	**39.67**$\pm$9.25
User Factor Fading	**0.99**	**0.17442**$\pm$0.0163	**0.00977*****	**38.16**$\pm$7.61
Forget Unpopular	**1.1**	**0.18341**$\pm$0.01988	**0.06153***	**47.64**$\pm$7.61
Forget Popular	1.00001	0.08305$\pm$0.0129	1	47.2$\pm$7.93
Last N Retention	5	0.1454$\pm$0.01463	1	78.61$\pm$12.89
Recent N Retention	1 year	0.14521$\pm$0.0125	1	1446.4$\pm$342.16
Global Sensitivity-based Forgetting	0.5	0.14935$\pm$0.01454	1	61.86$\pm$13.76
Sensitivity-based Change Detection	0.5	0.14933$\pm$0.01465	1	58.37$\pm$10.4
Sensitivity-based Forgetting	3	0.14942$\pm$0.01439	1	59.43$\pm$11.4
Recall-based Change Detection	0.05	0.14521$\pm$0.01251	1	1508.71$\pm$333.19

Table 9.: Results on datasets with explicit rating feedback (ML1M and Netflix). Best value of incremental recall is marked in red. Asterisks indicate that a given strategy is significantly better than the no forgetting strategy (* at 0.1; ** at 0.05; *** at 0.01). Table from [Mat+17]

Forgetting Strategy	Approx.	Rating-based	Approx.	Rating-based
	Epinions (10k users)		**Lastfm 600k**	
Last N Retention	106.22	185.60	74.74	91.48
Recent N Retention	235.61	3068.94	91.59	86.51
Global Sensitivity-based Forgetting	381.59	142.08	198.23	56.16
Sensitivity-based Change Detection	439.96	142.07	211.40	59.68
Sensitivity-based Forgetting	387.46	143.43	208.36	58.69
Recall-based Change Detection	375.18	3352.39	128.50	1255.23
	ML100k		**ML1M**	
Last N Retention	2.27	23 621.00	16.99	65.18
Recent N Retention	2.52	69.52	35.37	948.10
Global Sensitivity-based Forgetting	4.16	16 497.00	64.84	26.33
Sensitivity-based Change Detection	3.87	16 862.00	60.44	26.73
Sensitivity-based Forgetting	3.63	23 802.00	55.35	26.52
Recall-based Change Detection	4.30	72.60	46.72	955.87
	ML1M GTE5		**Music-listen**	
Last N Retention	9.41	42 627.00	49.13	100.29
Recent N Retention	9.22	45.27	51.63	187.98
Global Sensitivity-based Forgetting	9.66	33 909.00	47.86	35.11
Sensitivity-based Change Detection	10.22	42 411.00	55.00	36.17
Sensitivity-based Forgetting	9.58	34 608.00	55.75	35.06
Recall-based Change Detection	8.85	47.31	51.55	225.22
	Music-Playlist		**Netflix (10k users)**	
Last N Retention	3.61	43 405.00	63.72	77.04
Recent N Retention	3.82	42 571.00	81.58	1299.18
Global Sensitivity-based Forgetting	4.84	44 287.00	156.48	62.35
Sensitivity-based Change Detection	4.57	30 376.00	161.85	59.99
Sensitivity-based Forgetting	3.79	42 404.00	158.41	61.00
Recall-based Change Detection	3.40	19.22	153.33	1327.40

Table 10.: Median runtime (in seconds) of the approximative and rating-based implementation of forgetting (table from [Mat+17]).

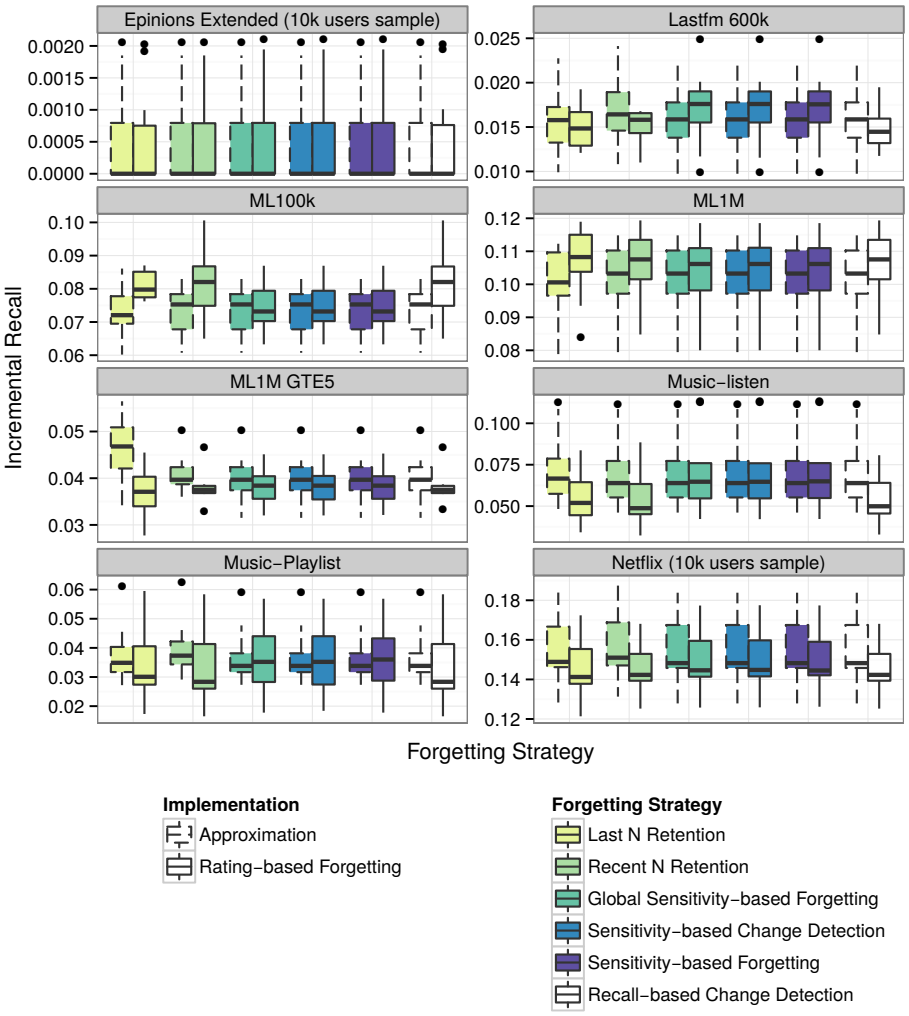

Figure 6.: Incremental recall of approximative rating-based forgetting is similar to the non-approximative variant (figure from [Mat+17]).

into a preference model. While this is a valid method of adaptation, it is not sufficient. In this chapter, we have shown that forgetting obsolete information (additionally to incorporating new one) significantly improves predictive power of recommender systems. Therefore we answer our research question **RQ 1** positively, as we have shown that the following expression from Ch. 3 is true:

$$\exists(\mathcal{M}, \mathcal{S}) : \sum_t Q_{S,t}(Te_t, M_t, S_t) > \sum_t Q_t(Te_t, M_t^*) \tag{48}$$

for Q being incremental recall. In our statistical tests we compared average incremental recall with forgetting (left side of the inequality) and without forgetting (right side of the inequality). As the time a recommender system observes is discrete, the average incremental recall over time is proportionate to its sum.

To answer the **RQ 1.1**, we proposed eleven unsupervised strategies to select the obsolete information and three algorithms to enforce forgetting. In our experiments we used a state-of-the-art incremental matrix factorization algorithm, BRISMF [Tak+09], and extended it by the ability to forget information and to add new dimensions to the matrix. Our forgetting strategies can be applied to any matrix factorization algorithm. Rating-based strategies are also applicable to neighbourhood-based methods.

Further, we proposed a new evaluation approach that includes significance testing. We conducted more than 1040 experiments on eight real world datasets with explicit rating feedback and positive only feedback. On seven out of eight of those datasets we observed a *significantly better performance* when using forgetting. On five of them the improvement was significant at level better than 0.01.

From all our forgetting strategies, the latent factor based ones were particularly successful in terms of quality of recommendations and in terms of computation time. On five out of eight datasets the *SD-based User Factor Fading* strategy achieved the best result, followed by the *Forget Unpopular* strategy with best result on two out of eight datasets. Therefore, we answer the **RQ 1.1** about how to select information to be forgotten with those two forgetting strategies.

Rating-based forgetting strategies also showed significant improvements over the "No Forgetting" strategy, however, their improvement was not as high, as in the case of latent-based strategies. We remark that not all of the forgetting strategies achieved a significant improvement. The strategy that achieved highly significant improvement on the most datasets (6 out of 8) is the *SD-based user factor fading* strategy.

Rating-based forgetting strategies have higher complexity and, therefore, a higher runtime than latent factor based strategies. Therefore, we proposed an approximative implementation that maintains a similar level of incremental recall as the original, rating-based implementation. For half of the strategies it reduced the computation time considerably. For the other half the computation often took longer. Therefore, we recommend to use this approximation only in time-critical applications and after prior testing.

Our recommendation to practitioners is to, first, use the *latent factor based forgetting strategies* (e.g. the *SD-based user factor fading* strategy) due to their outstanding performance in terms of incremental recall and only slightly increased computation time. This also answers our research question **RQ 1.2** about how to implement forgetting in recommender systems (cf. Alg. 6). The rating-based implementation forgetting is the next best alternative.

In the next chapter we address our research questions related to another form of selective learning - selective neighbourhood.

5 SELECTIVE NEIGHBOURHOOD

In this chapter we demonstrate that selective learning methods are also applicable in neighbourhood-based collaborative filtering and we answer the following two research questions:

RQ 2: **Does selective removal of users from a neighbourhood improve the quality of neighbourhood-based CF?**

RQ 2.1: **How to select users to be removed from a neighbourhood?**

As motivated in Sec. 1.1.2, we propose a method for selecting reliable users in a neighbourhood of an active user and evaluate its effect onto the quality of recommendations. In Sec. 5.1 we discuss related work on selective neighbourhood. In Sec. 5.2 we introduce our method, which is evaluated in Sec. A.5. We conclude the chapter on selctive neighbourhood in Sec. 5.4, where we also give answers to the above research questions. Sections 5.1 - 5.4 come (with modifications) from our publication on this topic [MS14a].

5.1 RELATED WORK ON RELIABLE NEIGHBOURHOOD

Neighbourhood-based collaborative filtering has been studied thoroughly in numerous publications (cf. studies on the most important aspects of them [DK11; Lü+12; Bob+13; Bee+16] and Ch. 2 for more details). In the existing work much emphasis is put on the predictive quality of a recommender's output. However, only little work focuses on selecting reliable neighbours. Reliability of neighbours should not be confused with the reliability of conclusions made about recommender systems, in the process of evaluation using hypothesis testing, as described in [SG11]. In contrast to statistical testing in the evaluation that aims to measure the significance of results, we investigate the significance of users' neighbourhood.

Herlocker et al. noticed that user similarity based on few co-ratings is unreliable [Her+99] (cf. example 5 in Ch. 1). Researchers have already proposed solutions to this problem, such as thresholds on the number of ratings two users should share to be considered similar, or assigning lower weights to users that have too few ratings in common [Her+99; BKV07; MKL07].

Herlocker et al. introduced the term of significance weighting [Her+99]. They recognized that similarity based on only few co-ratings is not representative and the amount of trust in this value should be lim-

ited. To limit the influence of those unreliable similarity values they weight them with a term $\frac{min(n,\gamma)}{\gamma}$, where n is the number of co-ratings and γ a control parameter that is fit to data (usually around 50). If n is greater than γ, then the weight is set to 1. Otherwise, the weight is always lower than 1, which gives lower importance to users with fewer co-ratings than γ.

Ma et al. use a similar but adjusted weighting schema [MKL07]. Bell et al. also define a different weighting schema, which they call "shrinkage" [BKV07]. They shrink the similarities towards zero to an extent that is inversely proportional to the number of co-ratings. The fewer co-ratings between users exist, the less influence does the particular similarity value have on the predicted rating value. Formally, they also use an additional weighting that is derived using the following term $\frac{n}{n+\beta}$. n is again the number of co-ratings and β is a control parameter, which is also fit to the data.

However, none of these methods answers the question "how many co-ratings are enough?", nor addresses the more important underlying question "whose co-ratings are *useful* enough?". They are based only on the number of co-ratings. We formalize the latter question and proposed a criterion for selecting neighbours. Instead of using weights, we can decide whether a similarity between two users can be relied upon. We stress that our method is not a solution to the cold start problem, where no enough information about users is known. Our goal is to quantify the reliability of the known information.

In our approach, we first formalize the concept of *baseline user* – informally, an average user for the population under observation. Then, we introduce the concept of *reliable similarity*: we use the Hoeffding Bound (HB), derived from Hoeffding's Inequality [Hoe63] to test whether a given user is more similar to the active user than the baseline user is. We then consider neighbours to the active user only those users whose similarity to the active user satisfies the bound. Hence, the recommender system decides on statistical grounds whether it can make a recommendation on neighbourhood-based similarity, no matter how small this neighbourhood is.

5.2 RELIABLE NEIGHBOURHOOD

Our approach consists of a formal model on *reliable similarity* of a user, an adjusted CF-based recommendation method and a mechanism that builds a user's neighbourhood by only considering users that are truly similar to the active user and ignoring all other users. We concentrate on user-user collaborative filtering, but our approach can be used for item-item CF analogously.

5.2.1 *Baseline Users*

To compute the *neighborhood* of the active user u_a, for whom recommendations must be formulated, we first introduce the notion of a "baseline user" u_B – a default, fictive user. Informally, a user x is *reliably* similar to u_a, if u_a is more similar to x than to u_B; then, the neighbourhood of u_a consists of the users who are *reliably similar* to her. Formally, u_B is a vector:

$$u_B = [ir_1, ir_2, ..., ir_{n-1}, ir_n] \tag{49}$$

where ir_j is a rating of the item j and n is the total number of items. We consider three types of baseline users: the *average user*, the *random Gaussian user* and the *random uniform user*. For the computation of the baseline users, we use an initial sample of ratings R_{train} for training.

AVERAGE USER: This baseline is computed by defining ir_j for an item j as the average rating of j in R_{train}:

$$ir_j = \frac{1}{|U(j)|} \sum_{x \in U(j)} r_{x,j} \tag{50}$$

where $r_{x,j} \in R_{train}$ is the rating of user x for item j and $U(j) = \{x | r_{x,j} \in R_{train}\}$ is the set of users who rated j.

RANDOM GAUSSIAN USER: Each item j in this baseline is assumed to follow the normal distribution with parameters μ and σ approximated on R_{train}. The value of ir_j for any j is drawn from this distribution:

$$ir_j \sim \mathcal{N}(\mu, \sigma^2) \tag{51}$$

RANDOM UNIFORM USER: Each item j in this baseline follows the discrete uniform distribution with r_{min} and r_{max} being the extreme rating values. Hence:

$$ir_j \sim \mathcal{U}\{r_{min}, ..., r_{max}\} \tag{52}$$

For example, if a rating can assume values between one and five stars, then $r_{min} = 1$ and $r_{max} = 5$.

We use the term of a baseline user to define the concept of *reliable similarity*, which is based on the Hoeffding Bound (HB).

5.2.2 *Reliable Similarity between Users*

To define *reliable similarity*, we begin with an arbitrary similarity function $sim()$. We will specify $sim()$ explicitly later.

Definition 1 (Reliable similarity)**.** *Let $sim()$ be a similarity function, and let u_B be the baseline user learned on R_{train}. We define the "reliable similarity" sim_{rel} between a user u_a , for whom recommendations must be formulated, and an arbitrary other user u_x as*

$$sim_{rel}(u_a, u_B, u_x) = \begin{cases} sim(u_a, u_x) & , \text{if } sim(u_a, u_x) \gg sim(u_a, u_B) \\ 0 & , \text{otherwise} \end{cases} \tag{53}$$

where we use the symbol $\gg$ for "significantly greater than". User u_x is "reliably similar" to u_a if $sim_{rel}(u_a, u_B, u_x) > 0$.

5.2.2.1 *Testing significance.*

We implement the "significantly greater than"-test of Def. 1 with help of the Hoeffding Inequality [Hoe63]:

$$Pr(\widehat{X} - \overline{X} \geqslant \varepsilon) \leqslant exp(\frac{-2n\varepsilon^2}{R^2}) \tag{54}$$

The Hoeffding Inequality quantifies the probability that the deviation of an observed average $\widehat{X}$ from the real average $\overline{X}$ of a random variable X is greater than or equal to ε. It takes the range R of the random variable and the number of observed instances n as inputs. The Hoeffding Inequality is independent of any probability distribution, however, it is thereby more conservative than other distribution-specific bounds [DHoo]. The inequality can be transformed into the Hoeffding Bound that specifies the maximal allowed deviation ε given a confidence level of $1 - \delta$:

$$\widehat{X} - \overline{X} < \varepsilon \text{ , where } \varepsilon = \sqrt{\frac{R^2 \cdot \ln(1/\delta)}{2n}} \tag{55}$$

We apply the Hoeffding Bound to ensure that the true similarity between two users is inside the ε-vicinity of the observed similarity. In particular, let u_1, u_2 be two users. Then, $\widehat{X}$ stands for the observed difference in similarity between them and $\overline{X}$ stands for the difference of their true similarities, thereby demanding that the similarity function is an average, as dictated in [Hoe63].

Definition 2 (Similarity Function for Significance Testing)**.** *Let u_1, u_2 be two users and let $I_{co\text{-}rated}(u_1, u_2)$ be the set of items that both have rated. Then, the similarity between u_1, u_2 is the following average (for a rating scale between 0 and 1, otherwise normalization is required):*

$$sim(u_1, u_2) = 1 - \frac{\sum_{j \in I_{co\text{-}rated}(u_1, u_2)} |r_{u_1 j} - r_{u_2 j}|}{|I_{co\text{-}rated}(u_1, u_2)|} \tag{56}$$

On the basis of this similarity function, we state with confidence $1-\delta$ that the non-observable true average similarity, denoted as $\overline{sim}(u_1, u_2)$, is within the ε-vicinity of the observed average similarity, denoted as $\widehat{sim}(u_1, u_2)$. The bound ε represents the uncertainty of the observed information. The fewer co-rated items we have for the two users, the larger is the possible deviation from the true unobserved values. This is captured by the number of observations n, which is here the cardinality of $I_{co\text{-}rated}(u_1, u_2)$. The smaller the value of n, the larger the bound ε (cf. Ineq.82) for a given confidence $1-\delta$.

The use the Hoeffding Bound in the significance test in Def. 1 means the following: when we observe that $\widehat{sim}(u_a, u_x) > \widehat{sim}(u_a, u_B)$, we want to test with confidence $1-\delta$ if $\overline{sim}(u_a, u_x) > \overline{sim}(u_a, u_B)$, subject to a bound ε.

To this purpose, we first need to ensure that the same number of observations is used for both the observed similarity $\widehat{sim}(u_a, u_x)$ and for the observed similarity $\widehat{sim}(u_a, u_B)$. Evidently, the set of co-rated items between u_a, u_B is the set of items rated by u_a, since the baseline user u_B has a rating for every item. Therefore, for each user u_x, whom we consider as potential neighbour of u_a, we compute $sim(u_a, u_B)$ on $I_{co\text{-}rated}(u_a, u_x)$ rather than on $I_{co\text{-}rated}(u_a, u_B)$. Thus, the number of observations is fixed to $n = |I_{co\text{-}rated}(u_a, u_x)|$.

Figures 7 and 8 explain this procedure visually. In Fig. 7, we depict the relative positions of $\widehat{sim}(u_a, u_x)$, $\overline{sim}(u_a, u_x)$, $\widehat{sim}(u_a, u_B)$, $\overline{sim}(u_a, u_B)$ in a case where both the observed and the true average similarity between u_a, u_x is larger than the corresponding values for u_a, u_B. In Fig. 8, we depict again the relative positions in a case where the observed average similarity between u_a, u_x is larger than the observed similarity between u_a, u_B, but the true similarity between u_a, u_x is smaller than the true similarity between u_a, u_B. Clearly, this is undesirable. Hence, we need a bound ρ such that it holds:

$$\text{if } \widehat{sim}(u_a, u_x) - \widehat{sim}(u_a, u_B) > \rho \text{ then } \overline{sim}(u_a, u_x) > \overline{sim}(u_a, u_B)$$

To ensure with confidence $1-\delta$ that $\overline{sim}(u_a, u_x) > \overline{sim}(u_a, u_B)$ for any values of $\widehat{sim}(u_a, u_x)$, $\widehat{sim}(u_a, u_B)$, we consider the extreme case, where $\widehat{sim}(u_a, u_x)$ is smallest and $\widehat{sim}(u_a, u_B)$ is largest, i.e. $\overline{sim}(u_a, u_x) = \widehat{sim}(u_a, u_x) - \varepsilon$ and $\overline{sim}(u_a, u_B) = \widehat{sim}(u_a, u_B) + \varepsilon$. Then, to ensure that $\overline{sim}(u_a, u_x) > \overline{sim}(u_a, u_B)$, following must hold:

$$\left(\widehat{sim}(u_a, u_x) - \varepsilon\right) - \left(\widehat{sim}(u_a, u_B) + \varepsilon\right) > 0$$

$$\text{i.e. } \widehat{sim}(u_a, u_x) - \widehat{sim}(u_a, u_B) > 2\varepsilon$$

This means that $\rho = 2\varepsilon$. Thus, we specify that:

$$sim(u_a, u_x) \gg sim(u_a, u_B) \iff \widehat{sim}(u_a, u_x) - \widehat{sim}(u_a, u_B) > 2\varepsilon \quad (57)$$

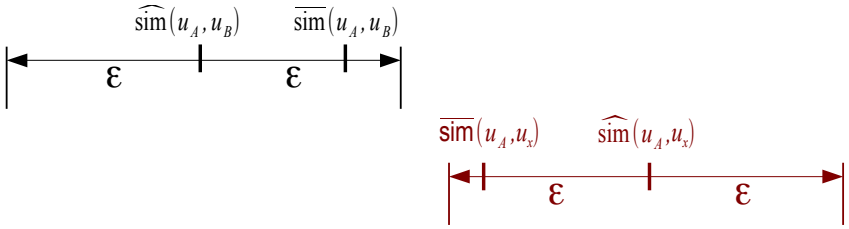

Figure 7.: Relative positions of the observed similarity between u_a, u_x and between u_a, u_B and true similarity within the ε-vicinity of the corresponding observed similarity; the observed similarities allow the conclusion that the true similarity between u_a, u_x is larger than the true similarity between u_a, u_B (figure from [MS14a]).

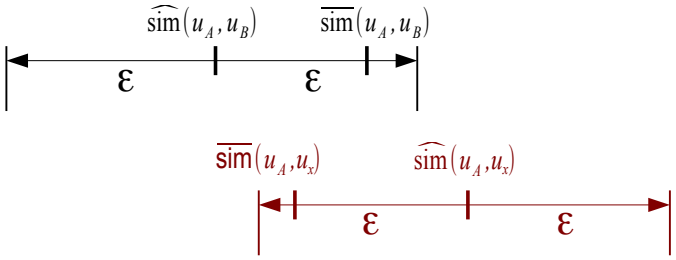

Figure 8.: Unlike in Fig. 7, the observed similarities, here, would lead to an erroneous conclusion, since their ε-regions overlap (figure from [MS14a]).

Due to our application of the Hoeffding Bound in recommender systems, which was inspired by numerous algorithms from the stream mining domain, we noticed that many algorithms in this domain use a border of only one ε for deciding between two means. Furthermore, more prerequisites for the application of the Hoeffding Bound are often not fulfilled. We investigated this problem and its effects in our work [MKS13]. We describe it in more detail in the Appendix A.

Definition 3 (Reliable Neighbourhood)**.** *Let u_a be an active user. Subject to Def. 1, the similarity function of Eq. 56 and the two invocations of the Hoeffding Bound, we define her reliable neighbourhood as:*

$$relNeighbourhood(u_a, \theta) = \{u_x \in U | sim_{rel}(u_a, u_B, u_x) > \theta\} \tag{58}$$

where U is a set of users and the similarity threshold θ is applied on reliable neighbours only. All unreliable neighbours are excluded, even if their similarity to u_a is larger than θ.

5.2.3 *Algorithms*

Algorithms 8 and 9 show a pseudocode of our extensions to collaborative filtering. Algorithm 8 computes a neighbourhood of an active user u_a using our method of checking the reliability of neighbours `isReliableNeighbour`, presented in Algorithm 9.

These algorithms require two hyperparameters: θ is a similarity threshold, also used in conventional CF, and δ that controls the confidence of the Hoeffding Bound used for checking the reliability. Since the criterion of the reliable similarity is much stricter than the conventional similarity, it can happen that no neighbours for an active user can be found at all. For this case we also adjusted the conventional CF algorithm. Our method can either abstain from recommending any items until more information about the given user is collected, or it provides non-personal recommendations e.g. the most popular items from the trainings dataset. We state that it is beneficial to make fewer, but reliable recommendations, than to recommend items that will cause a negative attitude or a distrust of users towards the recommender system.

5.3 EXPERIMENTS

We evaluate our method on the datasets MovieLens (100k), Flixter, Netflix and Epinions [MA06], comparing it to: a conventional user-based collaborative filtering recommender with cosine similarity, denoted as CF, to the method by Bell et al. called *shrinkage*[BKV07] and to *significance weighting* by Herlocker et al.[Her+99]. Since our goal is to compare different ways of building a neighbourhood, we implemented only the weighting schemas from the methods described in [BKV07] and [Her+99] and coupled them with the conventional CF algorithm. To ensure a fair comparison, all methods use the same core CF algorithm with no further extensions, so that only the way they build and weight their neighbourhoods differs.

We name our method *Hoeffding-CF*, abbreviated hereafter as H-CF. We consider one variant of our method per type of baseline user, denoted as H-CF_Gauss (Gaussian user), H-CF_Uniform (uniform user) and H-CF_Avg (average user). To optimise the parameters of the methods we run multiple experiments using a grid search over the parameter space. Since the number of experiments in the grid search is high, we chose a sample of users per dataset, taking over all their ratings. The evaluation settings are detailed below. Further information regarding datasets and our samples is summarized in Table 11.

Algorithm 8 Reliable CF

Input: θ similarity threshold
δ confidence in the Hoeffding Bound
u_a the active user
U set of users
R_{train} training set

```
reliableNeighbourhood(u_a) ← {}
u_B ← initializeBaseline(R_train, baseline_type)
for all {u_x ∈ U | u_x ≠ u_a} do
    u_x reliable ← isReliable(u_a, u_B, u_x, δ, θ)
    if u_x reliable then
        reliableNeighbourhood(u_a).add(u_x)
    end if
end for
if reliableNeighbourhood(u_A) == ∅ then
    abstain or recommend most popular items
else
    for all item i ∈ missingValues(u_a) do
        r̂_{u_a,i} =weightedAverage(reliableNeighbourhood(u_a))
        ranking.add(r̂_{u_a,i})
    end for
    ranking.sort()
    return top ranked items
end if
```

Algorithm 9 isReliable$(u_a, u_B, u_x, \delta, \theta)$

Input: u_a active user
u_B baseline user
u_a any other user
δ confidence in the Hoeffding Bound
θ similarity threshold

```
u_x_reliable ← true
if sim(u_a, u_x) ⩽ θ then
    u_x_reliable ← false
end if
ε ← computeHoeffdingBound(δ, Range, numberCoRatings(u_a, u_x))
(cf. Ineq. 82 and Eq. 58)
if ŝim(u_a, u_x) − ŝim(u_a, u_B) ⩽ 2ε then
    u_x_reliable ← false
end if
return u_x_reliable
```

Dataset	total Ratings	sampled ratings
Flixter	572531	59560
MovieLens 100k	100k	100k (no sampling)
Netflix	100 M	216329
Epinions	550823	165578

Table 11.: Samples of users on four datasets used in experiments (table from [MS14a]).

5.3.1 *Evaluation Settings*

As basis for our evaluation we use (a) the Root Mean Square Error (RMSE) of the predictions made by each method, and (b) the number of cases where the method encounters an empty neighbourhood and cannot make a neighbourhood-based prediction; this is denoted as *Missing Predictions*. However, a prediction is still provided using a fallback-strategy explained later. We further compute the *Average Neighbourhood Size*, the average size of non-empty neighbourhoods built by each method.

It is evident that the RMSE values for the three variants of our method are not directly comparable, because the value of *Missing Predictions* varies among the methods. Hence, we refine RMSE into following measures:

- *Neighbourhood-based RMSE*: the RMSE of the predictions made using the neighbourhoods of the users; limited to users with non-empty neighbourhoods (abbreviated hereafter as CF-RMSE)
- *Fallback-strategy RMSE*: the RMSE of the predictions made using the fallback strategy; limited to users with empty neighbourhoods
- *Global RMSE*: total RMSE by both *Neighbourhood-based RMSE* and *Fallback-strategy RMSE*

As fallback strategy we use the recommendation of the most popular items not rated by the active user. The impact of this strategy is encapsulated in *Fallback-strategy RMSE*.

For the variants of our method, we vary δ: the lower the value, the more restrictive is the confidence level of the Hoeffding Inequality and the less users are considered reliably similar to a given user. Hence, we expect that a decrease of δ will negatively affect the *Average Neighbourhood Size* and the *Missing Predictions*. For shrinkage and significance weighting we also optimize β and γ.

We further consider different similarity threshold values. Setting the threshold to a high value is not adequate for prohibiting recommendations on the basis of unreliable neighbourhoods. It must be noted that the CF may also fail to build neighbourhoods for some users, if the threshold is set very restrictively. In total, we performed more than 250 experiments, all of which were evaluated using 5-fold cross validation.

5.3.2 *Results*

In Tables 12 and 13, we present our results on each of the four datasets. For each of the methods we present only the best value found by the grid search in course of the optimization. The symbol "— " indicates that there are no applicable values for this position (e.g. δ is not applicable for the CF). The sizes of the neighbourhood in Tables 12 and 13 is seemingly high, however, these are the values found as approximatively optimal by the grid search.

The best result on on the Movie Lens 100k dataset was achieved by our method (1^{st} row in the Table) with a setting of $\delta = 0.999$, a uniform baseline user, and distance threshold of 0.25. The best value of global RMSE was 0.9864. The best result achieved by the conventional CF was 1.0207 (5th row in the table). This is a stable improvement verified using the 5-fold cross validation. Shrinkage and significance weighting yielded a result close to the conventional CF. When we compare our method with e.g. shrinkage with respect to the average neighbourhood size (row 1 and 3), then we notice an essential reduction from ca. 898 to 447 users. This means that our method reduced the neighbourhoods by 451 users on average and still performed better than the conventional CF. Regarding the baseline users on the Movie Lens dataset, the best results were achieved by the uniform random baseline. The average user baseline led to small neighbourhoods. This can be explained by the fact that many users in the MovieLens dataset are similar to the average user. Using this baseline makes the differences between user vectors insignificant and, consequently, many of the users are not considered as reliable neighbours to the active user.

If no reliable neighbours of an active user can be found, then is not possible to estimate the rating. We counted the occurrences of this case in our method (column "missing predictions"). In this situation a fallback-strategy (e.g. popular items) takes over the task of providing a recommendation (prediction error is included in global RMSE). We observed that those cases become more frequent when δ is low. This causes a more extensive pruning behaviour of our method, because more neighbourhoods are considered unreliable. If we allow our method to abstain from recommendation instead of using the fallback-strategy the improvement of RMSE is even higher (0.9683; row 1, col-

Row	Method	Distance Threshold	Setting	Missing Predictions	avgNeighborhoodSize	global RMSE	CF-RMSE	fallback-RMSE
MovieLens 100k								
1	H-CF_Uniform	0.25	$\delta = 0.999$	2905	447	0.9864	0.9683	1.4929
2	H-CF_Gauss	0.25	$\delta = 0.95$	3229	259.55	0.9875	0.9684	1.4693
3	Shrinkage	0.2	$\beta = 500$	215	898.08	1.0192	1.0192	—
4	Sig. Weighting	0.2	$\gamma = 200$	215	898.08	1.0192	1.0192	—
5	CF	0.2	—	215	898.08	1.0207	1.0207	—
6	H-CF_Avg	0.4	$\delta = 0.999$	13079	132.38	1.0321	1.0390	0.9839
Flixter (sample of 1000 users)								
7	H-CF_Gauss	0.8	$\delta = 0.95$	7047	78.14	1.0149	1.0133	1.0381
8	H-CF_Avg	0.8	$\delta = 0.95$	49918	5.84	1.0221	1.1355	0.9969
9	H-CF_Uniform	0.4	$\delta = 0.95$	4357	241.3580	1.0549	1.0532	1.1576
10	CF	0.7	—	3998	442.7564	1.0856	1.0856	—
11	Shrinkage	0.7	$\beta = 50$	3998	442.7564	1.0872	1.0872	—
12	Sig. Weighting	0.7	$\gamma = 50$	3998	442.7564	1.0889	1.0889	—

Table 12.: Results on Ml100k and Flixter datasets sorted with respect to global RMSE (lower values are better), grouped by the dataset (table from [MS14a]).

Row	Method	Distance Threshold	Setting	Missing Predictions	avgNeighborhoodSize	global RMSE	CF-RMSE	fallback-RMSE
Netflix (sample of 1000 users)								
13	H-CF_Gauss	0.2	$\delta = 0.95$	13601	199.66	0.9619	0.9511	1.1551
14	H-CF_Uniform	0.2	$\delta = 0.95$	11171	382.74	0.9622	0.9529	1.1849
15	H-CF_Avg	0.2	$\delta = 0.999$	60394	96.94	1.0075	1.0225	0.9669
16	Shrinkage	0.2	$\beta = 200$	4023	916.2519	1.0210	1.0210	—
17	Sig. Weighting	0.2	$\gamma = 100$	4023	916.2519	1.0214	1.0214	—
18	CF	0.2	—	4023	916.2519	1.0233	1.0233	—
Epinions (sample of 10 000 users)								
19	H-CF_Avg	0.3	$\delta = 0.5$	165578	0	1.0074	—	1.0074
20	H-CF_Gauss	0.8	$\delta = 0.5$	164948	0.2770	1.01100	1.3964	1.0106
21	H-CF_Uniform	0.4	$\delta = 0.5$	159842	1.5113	1.0279	1.3215	1.0109
22	CF	0.7	—	113117	461.39	1.2843	1.2843	—
23	Shrinkage	0.7	$\beta = 50$	113117	461.39	1.2894	1.2894	—
24	Sig. Weighting	0.7	$\gamma = 100$	113117	461.39	1.2907	1.2907	—

Table 13.: Results on Netflix and Epinions datasets sorted with respect to global RMSE (lower values are better), grouped by the dataset (table from [MS14a]).

umn CF-RMSE). Also the conventional CF, shrinkage and significance weighting exhibit some missing predictions. They are caused by either new users or new items that are not known from the training dataset.

We performed similar experiments on a random sample of 1000 users on the Flixter dataset. Also on this dataset our method achieved the best *globalRMSE* value of 1.0149 this time using a Gaussian baseline. The conventional CF (row 10) yielded a value of 1.0856 using neighbourhoods bigger by 365 users on average. Shrinkage and significance weighting were not able to outperform CF.

Also on a random sample of 1000 users from the Netflix dataset our method outperformed other approaches with respect to global RMSE, reaching the level of 0.9619 using the Gaussian user baseline. When abstention was allowed, the improvement was even more substantial and reached the level of 0.9511, compared to e.g. shrinkage with 1.0210 (row 16). Again here, we observed an essential reduction of the neighbourhood cardinality from ca. 916 by the shrinkage method down to ca. 200 by our approach. This proves that our approach selects the reliable neighbours, who are more informative for the preferences of an active user than the competitive methods.

The last dataset we performed our experiments on is the (small) Epinions dataset (cf. Table 13). Here our method clearly dominated the conventional CF. Hoeffding-CF achieved an RMSE of 1.0074 compared to 1.2843 by the conventional CF. Significance weighting and shrinkage performed worse than CF. Our approach recognized unreliable neighbourhoods and switched from the neighbourhood-based recommendation to the fallback-strategy that performs better on this dataset (cf. the columns *CF-RMSE* and *fallback-RMSE*). The average number of neighbours in the first row shows that the neighbourhood was limited to the minimum and this yielded the best result. Differently than on the other datasets, here the average user baseline performed the best. The statement about its strictness in the significance testing still holds. This very strictness was beneficial on this dataset. In row 19 we see that the neighbourhood was reduced to 0 i.e. there was no neighbourhood-based recommendations. All recommendations were provided by the fallback-strategy that, in this case, performed better.

5.3.3 *Summary of Findings*

Our experiments show that Hoeffding-CF is capable of recognizing unreliable neighbourhoods and selecting neighbours that are informative for the preferences of an active user. It outperformed the conventional collaborative filtering, shrinkage and significance weighting on all datasets. When abstention from providing recommendations was allowed, the improvement in terms of RMSE was often even more substantial. All of the best results were achieved using a smaller neigh-

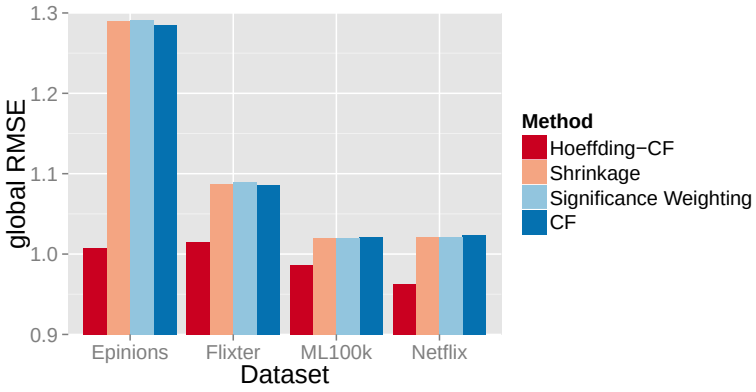

Figure 9.: Best results achieved by each method. Lower values of global RMSE are better. Our method, Hoeffding-CF, achieves best results on each dataset (figure from [MS14a]).

bourhood than in case of conventional CF and remaining approaches. A summary of the best results by each method is presented in Fig. 9.

We also observed that the parameter δ plays an important role in finding the optimal results. The lower its value, the stricter is the testing of the neighbourhood and the smaller is the average neighbourhood. Consequently, the number of predictions provided by the baseline method rises. The optimal value of δ varies across different dataset around 0.95. Cross-validation can be used for tuning on each dataset.

The choice of the baseline user has also an effect on the performance. We observed that the random-based user (Gaussian and uniform baseline) perform better than the average user baseline on most datasets. The reason for that is that many users are similar to the average user, so it is difficult to identify a user that is significantly more similar to a given user than the average. Hence, when the average user is the baseline, each user has only a few significant neighbours. On the Epinions dataset, however, this led to an improvement of accuracy.

5.4 CONCLUSIONS FROM SELECTIVE NEIGHBOURHOOD

We investigated the problem of neighbourhood-based recommendations when the similarity between users cannot be fully trusted. This problem does not emanate solely from data sparsity: even users with many ratings may be uninformative. We introduced the concepts of *baseline user* and of *reliable similarity*, and we use the Hoeffding Bound to select, for a given active user, those users who are informative, ignoring users that do not contribute more information than the baseline user.

Experiments on real datasets show that the use of reliable similarity improves recommendation quality. Our method outperforms the conventional CF, shrinkage and significance weighting on all datasets. However, the superior performance on the forth dataset is mainly owed to a good performance of the fallback-strategy rather than to neighbourhood-based recommendations. Our method outperforms other approaches despite using smaller neighbourhoods. This means that the reliability, rather than the size of a neighbourhood is decisive for good predictions.

By proposing our selective criterion for excluding of neighbours from a neighbourhood we answer the **RQ 2.1**. Our experimental results show that removing selected neighbours from a neighbourhood improves the predictive performance of CF algorithms. Therefore, we answer the **RQ 2** positively.

6 SEMI-SUPERVISED LEARNING

In this chapter we address the third type of selective learning for recommender systems - semi-supervised learning. Our SSL framework learns from selected unlabelled data instances as motivated in Sec. 1.1.3. Furthermore, we answer the following research questions:

RQ 3: Does selective learning from predictions (semi-supervised learning) improve the quality of recommendations?

RQ 3.1: How to select unlabelled instances for SSL?

RQ 3.2: How to select reliable predictions to learn from?

RQ 3.3: How to assemble predictions from an SSL system using co-training into a single prediction?

RQ 3.4: How to divide labelled instances among multiple learners in an SSL system?

First however, we discuss related work and the differences between our approach and existing work. In the following section we present an overview and components of our framework. Section 6.4 discusses the evaluation settings used in our experiments. In Sec. 6.5 we present our experimental results and we conclude this chapter in Sec. 6.6. Sections 6.1 - 6.6 come (with modifications) from our previous publications on this topic [MS17; MS15].

6.1 RELATED WORK ON SSL IN RECOMMENDER SYSTEMS

SSL has been investigated thoroughly in conventional data mining and machine learning [ZZQ07], also in the stream setting [DCP14; Sou+15]. A comprehensive survey of those techniques can be found in [Zhu05]. Those techniques encompass both co-training [SNB05] and self-learning techniques [RHS05]. Semi-supervised approaches for regression problems also have been proposed [ZL07]. However, the problem in recommender systems is inherently different from the conventional classification or regression. In recommender systems an entire matrix of real, binary or positive-only values is predicted. This matrix is extremely sparse (typically, around 99% of missing values) and there are no further features for a conventional regressor to train upon. Therefore, the methods from the conventional SSL are not applicable to recommender systems.

Dedicated SSL methods for recommender systems have been researched far less. Christakou et al. proposed in 2005 a model-based

recommender system using the k-means algorithm with SSL [Chr+05]. Nevertheless, this is not a dedicated recommender systems method, but clustering applied to the recommendation problem.

To decide which predictions can be used as labels, semi-supervised methods use reliability measures. A prediction with high estimated reliability can be then used for training. Hernando et al. proposed such a reliability measure, however, they did not use it in semi-supervised learning, but presented it to users to indicate certainty of the recommendation algorithm [Her+13]. Rodrigues et al. [RGB08] and Bosnić et al. [Bos+14] also proposed reliability measures, however, not for recommender systems, but for classification problems on streams. Nevertheless, we adopted their idea of reliability based on local sensitivity and adapted it to recommender systems (cf. Sec. 6.3.5).

Zhang et al. proposed a SSL method for batch-based recommender systems. In their approach they assess the reliability of a rating prediction based on frequency of occurrence of items and users [Zha+14]. They assume that popular items and active users are easier to predict, since there is more data about them. We implemented this reliability measure, that we call hereafter "popularity-based reliability measure", and we compare it to results of other measures. The method by Zhang et al. is batch-based. Once the model is trained, it cannot be changed incrementally. As a consequence, it is also not adaptive to changes and not responsive to new users and items. With our stream-based framework we lift those limitations.

Preisach et al. proposed a graph-based tag recommender system that employs untagged items [PMS10]. In this method the authors used a semi-supervised relational classification to find relevant tags. Therefore, this method is also not applicable to the typical rating prediction task in recommender systems.

Zhu et al. proposed a recommender system for web pages that uses conventional classification with self-learning on natural language data [Zhu+10]. Also this method is not applicable to the general collaborative filtering scenario in recommender systems.

6.2 SEMI-SUPERVISED FRAMEWORK FOR STREAM RECOMMENDERS

In this section we present our semi-supervised framework together with its components. We start with an incremental recommendation algorithm in Sec. 6.2.1 and then explain how it is applied in two alternative approaches: co-training (cf. Sec. 6.2.2) and self-learning (SL) (cf. Sec. 6.2.3). In Tab. 14 we present a summary of notation and abbreviations used in this work. Fig. 10 gives a simplified overview over the framework components and their interaction.

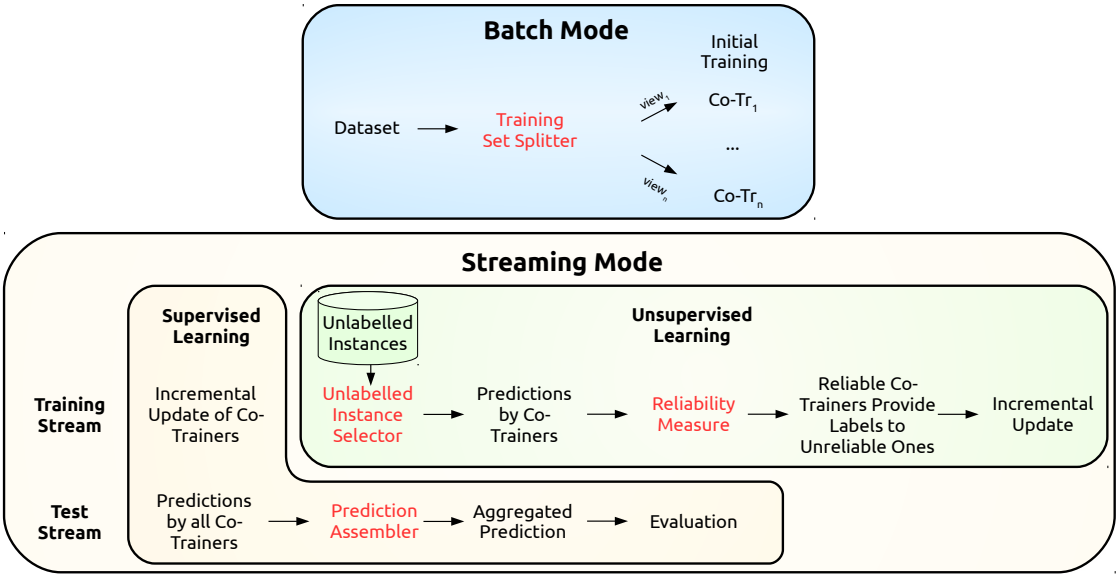

Figure 10.: A simplified overview of the framework visualizing the key components of the framework (in red) and their interplay (from [MS17] with modifications).

6.2.1 *Incremental Recommendation Algorithm*

The core of our framework is a recommendation system algorithm. Fig. 11 depicts two modes of a stream-based recommendation algorithm. The entire rectangle in the figure represents a dataset consisting of ratings. The dataset is split between a batch mode (blue part) and a stream mode (yellow part). The stream mode is the main mode of an algorithm, where information about new ratings is incorporated incrementally into the model, so that it can be used immediately in the next prediction. Semi-supervised learning takes place in this phase (green bars stand for Unsupervised Learning (USL)).

Before the stream mode can start, the algorithm performs an initial training in the batch mode. The batch mode data is, therefore, split again into training and test set. On the training dataset latent factors are initialized and trained. The corresponding prediction error is then calculated on the test dataset (second blue rectangle) and the latent factors are readjusted iteratively. Once the initial training is finished, the algorithm switches into the streaming mode, where learning and predicting take place simultaneously. Any incremental MF algorithm is applicable. We use our extended version of the BRISMF algorithm, etxBRISMF, as described in Section 6.3.1.

Notation	Meaning
SSL3	Semi-supervised Learning with co-training using 3 parallel learners
USL	Unsupervised Learning
noSSL	Algorithm without SSL (i.e. supervised learning only); it is used as a comparison baseline
SL	Self-Learning
IR@10	Incremental Recall at 10 (cf. [CKT10])
$CoTr_n$	The n-th Co-Trainer; one of incremental MF algorithms running in parallel
C	A set of all Co-Trainers
r_x	True value of rating x (ground truth)
$\widehat{r}_x$	A prediction of value of rating x
$\widehat{r}_{x\,CoTr_n}$	A prediction of value of rating x made by the Co-Trainer n
$\widehat{r}_{x\,Agg}$	An aggregate of all predictions of rating x made by all Co-Trainers from C
$rel(\widehat{r}_{i\,CoTr_a})$	Reliability of prediction $\widehat{r}_i$ by $CoTr_a$

Table 14.: Summary of notation and abbreviations used in this chapter (table partially from [MS17]).

6.2.2 *Stream Co-training Approach*

In semi-supervised learning we use two approaches: self-learning and co-training. The latter was proposed by Zhang et al. for batch recommender systems [Zha+14]. In this section we focus on the co-training approach. According to this approach we run in parallel multiple stream-based recommendation algorithms that are specialized on different aspects of a dataset and can teach each other. Due to this specialization an ensemble of co-trainers can outperform a single model that uses all available information.

6.2.2.1 *Initial Training.*

The specialization of the models takes place already in the initial training. In Figure 12 we present the batch mode from Figure 11. Here, the initial training set is divided between N co-trainers from the set $C = \{CoTr_1, ..., CoTr_N\}$, where $N \geqslant 2$.

The component that decides, how the initial training set is divided between the co-trainers is called *training set splitter* (marked in red in Fig. 12; cf. Sec. 6.3.2 for instances of this component). Formally, a

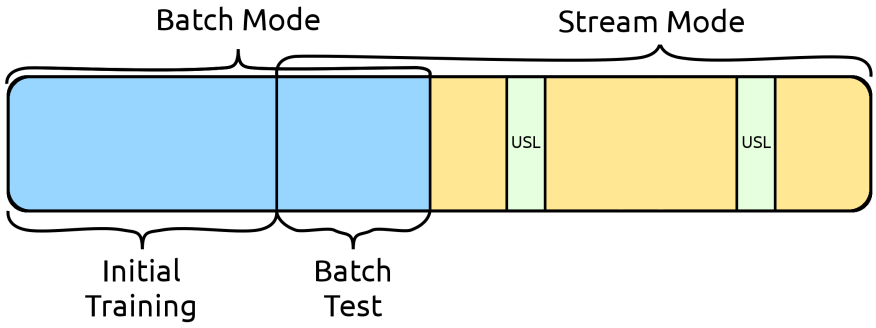

Figure 11.: Division of a dataset (entire rectangle) into batch (blue part) and stream mode (yellow part). The stream mode is the main part of an algorithm with incremental learning. Batch mode is used for initial training (figure from [MS15; MS17]).

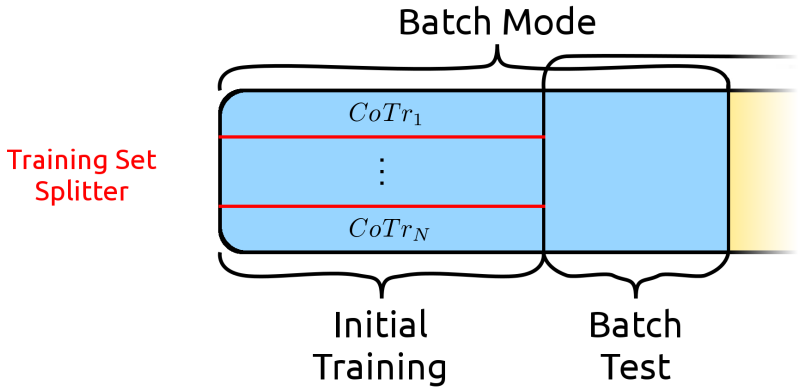

Figure 12.: Different co-trainers are trained on different parts of the initial training set. The component responsible for splitting the training set is *training set splitter* (figure from [MS15; MS17]).

training set splitter is a function that relates all co-trainers to subsets of all ratings in the initial training set $R_{initialTrain}$:

$$f : C \rightarrow P(R_{initialTrain}) \tag{59}$$

such that $f(CoTr) = R^{CoTr}_{initialTrain}$ and $P(X)$ is a power set of X. Therefore, $R^{CoTr_n}_{initialTrain} \subseteq R_{initialTrain}$. This function is not a partitioning function, since overlapping between different $R^{CoTr_n}_{initialTrain}$ is allowed and often beneficial. Implementations of this component are provided in Sec. 6.3.2.

6.2.2.2 *Streaming Mode - Supervised and Unsupervised Learning.*

After the initial training is finished, all co-trainers switch into the streaming mode. In this mode a stream of ratings r_t is processed incrementally. Figure 13 is a close-up of the stream mode from Figure

11. It represents a stream of ratings $r_1, r_2, \ldots$ [1]. The yellow part of the figure depicts the supervised learning, whereas the green part stands for the unsupervised learning (cf. next section).

In the supervised learning we distinguish between training and testing i.e. making recommendations. In the training all co-trainers calculate predictions for each rating r_x in the stream:

$$\forall n : CoTr_n(r_x) = \widehat{r}_{xCoTr_n} \tag{60}$$

Please, note that co-trainers are instances of the extBRISMF algorithm (cf. Sec. 6.3.1). Consequently, all extBRISMF instances calculate predictions for the rating in the stream. Once the predictions are made, all co-trainers receive the true value of the predicted rating. This value is then used to update the models of the co-trainers incrementally (cf. Algorithm 10).

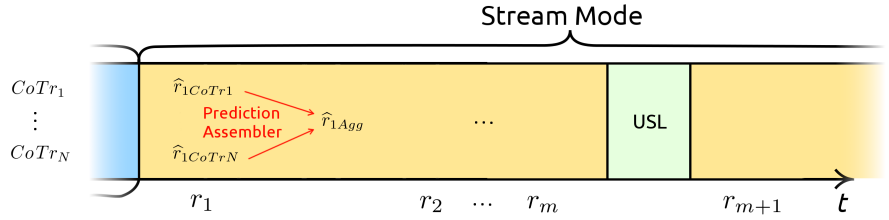

Figure 13.: A close-up of the stream mode from Figure 11. The yellow part represents the supervised learning and the green one unsupervised learning. Predictions made by co-trainers are aggregated by a *prediction assembler* (figure from [MS15; MS17]).

For the evaluation and for making recommendations, one more step is necessary. Since the co-trainers provide multiple predictions, they need to be aggregated into one common prediction of the entire system. Because the co-trainers had a different view of the training data in the batch mode, they can provide different predictions. In the stream mode all co-trainers receive the same ground truth.

In order to aggregate all predictions made by co-trainers into one prediction $\widehat{r}_{xAgg}$ we use a component called *prediction assembler*. The most simple implementation is arithmetical average (further implementations in Section 6.3.3). The function of prediction assembler is as follows:

$$predictionAssembler(r_x, C) = \widehat{r}_{xAgg} \tag{61}$$

In Fig. 13 this process is visualized only for the rating r_1 due to space constraints, however in a real application, it is repeated for all

1 Note that we diverge from the standard notation of $r_{u,i}$ for clarity. A rating r_x is still a triplet of user, item and a rating value, just the user u and item i are not relevant in this context. Different index x signifies different ratings.

ratings in the stream with known ground truth (supervised learning). For instances with no ground truth the procedure is different.

6.2.2.3 *Unsupervised Learning.*

Unsupervised Learning (USL) takes place periodically in the stream. After every m-th rating (m can be set to 1) our framework executes the following procedure. First, a component called *unlabelled instance selector* selects z unlabelled instances (cf. Fig. 14). Unlabelled instances in recommender systems are user-item-pairs that have no ratings. We indicate those instances with the purple colour in the following figures. The unlabelled instance selector is important, because the number of unsupervised instances is much larger then the number of supervised ones. Processing all unsupervised instances is not possible, therefore, with this component we propose several strategies of instance selection (cf. Sec. 6.3.4).

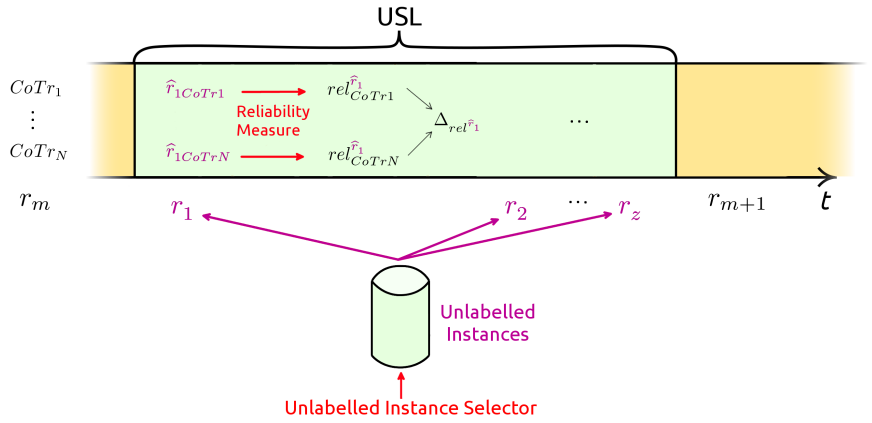

Figure 14.: The procedure of unsupervised learning. User-item-pair without ratings are selected using an *unlabelled instance selector*. Predictions and their *reliability values* are estimated. The most reliable predictions are used as labels for the least reliable co-trainers (figure from [MS15; MS17]).

Once the unlabelled instances $r_1, ..., r_z$ are selected, co-trainers are used again to make predictions:

$$\forall n, i : CoTr_n(r_i) = \hat{r}_{i\,CoTr_n} \tag{62}$$

where $i = 1, ..., z$ and $n = 1, ..., N$. After this step we use a *reliability measure* (cf. Sec. 6.3.5 for instantiation) to assess in an unsupervised way, how reliable is a prediction made by each co-trainer. Formally, a reliability measure is the following function:

$$reliability : (CoTr_n, \hat{r}_{i\,CoTr_n}) \rightarrow [0, 1] \tag{63}$$

This function takes a co-trainer and its prediction as arguments and maps them into a value range between 0 and 1, where 1 means the maximal and 0 the minimal reliability. Subsequently, we calculate pairwise differences of all reliability values of the predictions for r_i :

$$\Delta = |rel(\widehat{r}_{iCoTr_a}) - rel(\widehat{r}_{iCoTr_b})| \tag{64}$$

for all $a, b = 1, ..., N$ and $a \neq b$. All values of Δ are stored temporarily in a list, which is then sorted. From this list we extract the top-q highest differences of reliability i.e. cases, where one co-trainer was very reliable and the second one very unreliable. In such cases the reliable co-trainer provides a label to the unreliable co-trainer, who then trains incrementally using the provided label.

6.2.3 *Stream-based Self-learning*

Our second approach to semi-supervised learning on streams is self-learning (SL). According to this approach, a single learner is responsible for generating labels. Those labels are then scored with respect to their reliability values. The most reliable labels are used to train the learner. Our co-training framework described in the previous section is flexible, therefore it can be used for self-learning as well. In the following we describe the few changes that are necessary to adapt it to self-learning.

The first of those changes is in the initial training. While the co-training approach uses several learners and splits the initial training set among them, there is only one learner in the self-learning approach. Therefore, the entire initial training dataset is used by this learner. Consequently, there is no need for a co-training splitter.

Since there is only one learner, there is also no need for a prediction assembler that, otherwise, is responsible for aggregating predictions from several learners.

As a consequence of those changes the procedure shown in Fig. 14 changes as shown in Fig. 15. Unlabelled instances (user-item-pairs without a rating) are selected by the component called "unlabelled instance selector". Subsequently, the self-learner makes predictions for each of the selected instances $r_1, r_2, ..., r_z$. The reliability of this predictions is assessed using a reliability measure (cf. Sec. 6.3.5). Differently than in co-training, here the difference in reliability of learners, Δ from Equation 64, cannot be calculated. Therefore, the best label candidates are the predictions with highest reliability.

A further change affects the unlabelled instance selector. Its function remains the same, however, the possible implementations of this component are restricted to the ones working with a single learner. That criterion excludes, for example, implementations based on disagreement among multiple trainers (cf. Section 6.3.4 for details).

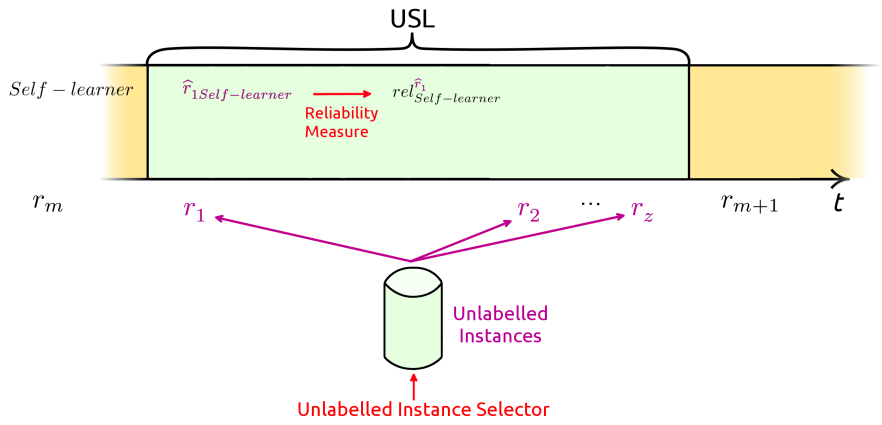

Figure 15.: Adjusted procedure of unsupervised learning from Fig. 14 for the self-learning approach. In this approach there is only one learner, whose predictions are assessed using a reliability measure (figure partially from [MS15; MS17]).

6.3 INSTANTIATION OF FRAMEWORK COMPONENTS

In the previous section we provided definitions of the components of our framework and explained their interplay. In this section we present several *instances* for each of the components. The reliability measure, for example, has many possible implementations (cf. Sec 6.3.5).

6.3.1 *Incremental Recommendation Algorithm - extBRISMF*

The core of our framework is a matrix factorization algorithm. We extended the BRISMF algorithm by Takács et al. [Tak+09] by the ability to deal with changing dimensions of the matrix over time. We named this new variant of the algorithm extBRISMF for dimensionality **ext**ending **BRISMF**. The original BRISMF keeps the dimensions of the matrix fixed and does not update latent item factors. In our algorithm we lift those limitations. This ability is important in SSL, because the algorithms often encounter items and users not seen before.

In Alg. 10 we present our extBRISMF. Apart from expanding dimensions of latent matrices, we also introduced a different type of initialization for new user/item vectors. Next to the initialization of the first column of P and second row of Q with a fixed constant value, which is typical for BRISMF, we initialize the vectors as an average vector of the corresponding matrix plus a small random component instead of just a random vector.

Algorithm 10 extBRISMF - trainIncrementally($r_{u,i}$)

Input: $r_{u,i}, P, Q, \eta, k, \lambda$

$p_u \leftarrow$ getLatentUserVector(P, u)
$q_i \leftarrow$ getLatentItemVector(Q, i)
if $p_u = null$ **then**
  $p_u \leftarrow$ getAverageVector(P) + randomVector
  $p_{u_1} \leftarrow 1$
  $P \leftarrow P.append(p_u)$
end if
if $q_i = null$ **then**
  $q_i \leftarrow$ getAverageVector(Q) + randomVector
  $q_{i_2} \leftarrow 1$
  $Q \leftarrow Q.append(q_i)$
end if
$\widehat{r}_{u,i} = p_u \cdot q_i$ //predict a rating for $r_{u,i}$
evaluatePrequentially($\widehat{r}_{u,i}, r_{u,i}$) //update evaluation measures
$epoch = 0$
for all $epoch \in \{1, ..., optimalNumberOfEpochs\}$ **do**
  $p_u \leftarrow$ getLatentUserVector(P, u)
  $q_i \leftarrow$ getLatentItemVector(Q, i)
  $predictionError = r_{u,i} - p_u \cdot q_i$
  for all latent dimensions k **do**
    if $k \neq 1$: $p_{u,k} \leftarrow p_{u,k} + \eta \cdot (predictionError \cdot q_{i,k} - \lambda \cdot p_{u,k})$
    if $k \neq 2$: $q_{i,k} \leftarrow q_{i,k} + \eta \cdot (predictionError \cdot p_{u,k} - \lambda \cdot q_{i,k})$
  end for
end for

6.3.2 *Training Set Splitter*

Training set splitters are used in the co-training approach to divide the initial training set among co-trainers. In the following we propose several types of training set splitters (cf. Fig. 11). All of them have one parameter p that controls the degree of overlapping between the co-trainers.

6.3.2.1 *User Size Splitter*

This splitter discriminates between users of different sizes. Size of a user is defined as the number of rating she/he has provided. Users are divided into segments based on their sizes and assigned to co-trainers. In case of only two co-trainers, for instance, one of them will be trained on so called "power users" and the other one on small users. This method is based on a histogram of user sizes. It creates N segments (N = number of co-trainers) using equal density binning

(each segment has the same number of users). Analogously, we also experiment with an *item size splitter*.

6.3.2.2 *Random Splitter*

Ratings are divided between co-trainers randomly. This method serves as a baseline for comparisons.

6.3.2.3 *Dimension Preserving Random Splitter*

This splitter also assigns ratings randomly to co-trainers, however, in contrast to the previous method, it guarantees that all co-trainers have a matrix with same dimensions. This means that all co-trainers have at least one rating from each user and item that appeared in the initial training set. This might be beneficial for methods not able to extend the dimensions of their matrices over time.

6.3.2.4 *User Variance Splitter*

As the name suggests, this splitter assigns users to different co-trainers based on their rating variance. For each users her/his rating variance is calculated. Using the histogram method and equal density binning, as in the user size splitter, different types of users are divided among co-trainers.

The rationale behind this splitter is that users with high variance tend to give differentiated ratings i.e. they rate both items they like and the ones they do not like. Users with low rating variance tend to give a standard rating to all items. This splitter utilizes this difference in users' behaviour and allows different co-trainers to specialize on separate groups of users.

6.3.2.5 *Item Variance Splitter*

Similarly to the user variance splitter, this splitter is also based on rating variance. However, here the variance is calculated for single items (within item). Based on this variance, ratings of items are divided among co-trainers.

The rationale behind this splitter is different than in the previous one. Items with a small rating variance are the ones that users agree upon. i.e. all users rate those items with approximately same value (e.g. 5 stars). Items with a high rating variance are not agreed upon by users. It means that there is a group of users rating a given item highly and a different group having an opposite opinion of it.

6.3.2.6 *Average Rating Splitter*

The division of ratings is performed by this splitter with respect to average rating of a user. Users with a high average rating are assigned

to a different co-trainer than the ones with a low average rating. This splitter can be used analogously for items.

6.3.3 *Prediction Assembler*

Prediction assembler aggregates rating predictions from all co-trainers into a single value. We propose several ways of calculating this aggregation that have the form of the following formula, but with different weights $w(\widehat{r}_{u,i}, CoTr_j)$:

$$\widehat{r}_{u,iAgg} = \frac{\sum_{j=0}^{N} w(\widehat{r}_{u,i}, CoTr_j) \cdot \widehat{r}_{u,iCoTr_j}}{\sum_{j=0}^{N} w(\widehat{r}_{u,i}, CoTr_j)} \tag{65}$$

Each of the following components defines the weight $w(\widehat{r}_{u,i}, CoTr_j)$ in a different way.

6.3.3.1 *Recall-based Prediction Assembler*

Recall-based prediction assembler aggregates predictions of N co-trainers using a weighted average with weights depending on their past recall values. Accordingly:

$$w(\widehat{r}_{u,i}, CoTr_j) = recall(CoTr_j) \tag{66}$$

In the above formula recall is measured globally for each co-trainer. Alternatively, recall can be measured also on user or item level. In this case $recall(CoTr_j)$ can be substituted with $recall(CoTr_j, u)$ or $recall(CoTr_j, i)$.

6.3.3.2 *RMSE-based Prediction Assembler*

Similarly to the previous method, this prediction assembler uses a weighted average, however, here the RMSE measures (root mean square error) serve as weights. Also here, measuring RMSE on user and item levels are possible.

$$w(\widehat{r}_{u,i}, CoTr_j) = RMSE(CoTr_j) \tag{67}$$

6.3.3.3 *Reliability-weighted Prediction Assembler.*

This prediction assembler uses a reliability measure to give more weight to more reliable co-trainers.

$$w(\widehat{r}_{u,i}, CoTr_j) = rel_{CoTr_j}^{\widehat{r}_{u,i}} \tag{68}$$

6.3.3.4 *Maximum Reliability Prediction Assembler*

Differently than in the previous prediction assembler, here only the prediction of the most reliable co-trainer is used. The aggregation is, therefore, performed using the following formula:

$$w(\widehat{r}_{u,i}, CoTr_j) = \begin{cases} 1, & \text{if } rel^{\widehat{r}_{u,i}}_{CoTr_j} = \max\limits_{k=1,\ldots,N} rel^{\widehat{r}_{u,i}}_{CoTr_k} \\ 0, & \text{otherwise} \end{cases} \tag{69}$$

6.3.4 *Selector of Unlabelled Instances*

This component is used in unsupervised learning to select unlabelled instances as candidates for training. Due to a large number of unlabelled instances a method for selecting them is needed. We propose such methods that as parameter take the number of instances to be selected.

6.3.4.1 *Random Selector*

Random combinations of known users and items are generated by selecting random users and random items independently. This method is used as a baseline for comparisons.

6.3.4.2 *Latent Disagreement Selector*

For each user each co-trainer stores a latent vector. We denote this vector as $p_u^{CoTr_n}$. In this method we search for those users, whose disagreement of the latent user vectors among the co-trainers is the highest. For each user u and a pair of co-trainers $CoTr_a$ and $CoTr_b$, we define the disagreement as follows:

$$disagreement(CoTr_a, CoTr_b, u) = |p_u^{CoTr_a} - p_u^{CoTr_b}| \tag{70}$$

This measure can be computed for all known users and all co-trainer pairs. Users with highest disagreement are then selected as candidates together with a random selection of items. The motivation behind this method is that the instances with highest disagreement can contribute the most to the learners. This method can be applied analogously onto latent item vectors.

6.3.4.3 *User-specific Incremental-recall-based Selector*

This selector chooses users with best incremental recall achieved by the framework. For those users it selects random items to generate unlabelled instances (user-item pairs without a rating). The rationale behind this selector is that users, for whom the past predictions were accurate, are good candidates for semi-supervised learning. Predictions

for those users should be reliable, assuming that the performance on the selected instances is consistent with the performance observed so far.

To avoid selecting instances from the user with highest incremental recall only, we create a list of best user candidates. From this list, the user on the first position is used for creating twice as many unlabelled instances as the second user, etc. This procedure is repeated, until the specified number of unlabelled instances is created.

Analogously to this selector, we experiment also with the *Item-specific Incremental-recall-based Selector*. The incremental measure of recall can be substituted by, e.g. the RMSE measure, creating *User-specific and Item-specific RMSE-based Selector*.

6.3.5 *Reliability Measure*

Reliability measures are used in our framework to assess the reliability of a rating prediction in an unsupervised way. Based on prediction reliability, decisions on which co-trainer teaches which one are made.

6.3.5.1 *Sensitivity-based Reliability Measure*

This is a novel measure of reliability for recommender systems that is based on local sensitivity of a matrix factorization model. As a user model in matrix factorization we understand a latent user vector p_u. This vector changes over time as new rating information from the stream is incorporated incrementally into the model. The changes of this vector can be captured using the following formula:

$$\Delta_{p_u} = \sum_{i=0}^{k} (p_{u,i}^{t+1} - p_{u,i}^{t})^2 \tag{71}$$

where $p_{u,i}^{t+1}$ and $p_{u,i}^{t}$ are latent feature values in the i-th position in the user vector p_u at different time points. If Δ_{p_u} is high, then it means that the user model is not stable and it changes considerably over time. Therefore, predictions made by this model can be trusted less. Similarly to the user sensitivity we can also measure a global sensitivity of the entire model as a different variant of this measure. Since Δ_{p_u} has a value range $[0, \infty)$ a normalization is needed (cf. Par. 6.3.5.5).

6.3.5.2 *Popularity-based Reliability Measure*

Zhang et al. proposed in [Zha+14] a reliability measure based on popularity. This measure uses the idea that the quality of recommendations increases as the recommender system accumulates more ratings. They used the absolute popularity of users and items normalized by a fixed term. We implemented this reliability measure in our framework for comparison, however, with a different normalization method.

Normalisation on streams is different and more challenging (cf. Par. 6.3.5.5).

6.3.5.3 *Random Reliability Measure*

A random number from the range $[0, 1]$ is generated and used as a reliability value. This measure is used as a baseline.

6.3.5.4 *RMSE-based Reliability Measure*

Another approach to assess the reliability of a prediction is to assume that the current performance of a prediction model will be consistent with its past performance. For instance, if a co-trainer performed better than others in the past, the reliability of the current prediction by this co-trainer can also be assumed higher then the reliability of the remaining co-trainers.

The reliability measure presented here uses the RMSE measure to evaluate the past performance of co-trainers. Other quality or error measures are also applicable. We also experiment with the *incremental-recall-based reliability measure.*

Furthermore, the performance of co-trainers can be measured on a finer level. For instance, on the level of single users or items. It could be that the past performance of a co-trainer is better for a specific user, even though on a global level, it performs worse than other co-trainers. In our experiments we use the reliability measures with different abstraction levels, e.g. the RMSE-based reliability measure on a user level is called "user-RMSE-based reliability measure". In our results we followed this naming convention.

6.3.5.5 *Normalization of Reliability Measures*

As defined in Section 6.2.2, a reliability measure is a function with value range of $[0, 1]$. With many aforementioned reliability measures this is not the case, therefore, a normalization is necessary. Normalization on a stream, however, is not trivial. Division by a maximal value is not sufficient, since this value can be exceeded in a stream and a retrospective re-normalization is not possible. Let rel be a reliability value returned by one of our reliability measures. In our framework we use the following sigmoid function for normalization:

$$f(rel) = \frac{1}{1 + e^{\alpha \cdot (rel - \mu)}} \tag{72}$$

where α controls the slope of the function and μ is the mean of the distribution. The parameters can be set either manually, or automatically and adaptively in a self-tuning approach. While the adaptive calculation of μ in a stream is trivial, the calculation of α requires more effort.

For that purpose we store 1000 most recent arguments of this function and determine their fifth percentile. We define that the value of the sigmoid function for this percentile should be equal to 0.9. From that, the optimal value of α can be derived. Note that α also controls if the function is monotonically increasing or decreasing. Reliability measures using this adaptive normalization can be recognized in our notation by the prefix "ST" (for self-tuning).

6.4 EVALUATION PROTOCOL

We propose a novel evaluation protocol for stream-based recommender systems that encompasses the following components:

- parameter optimization on a separate dataset
- a method for dataset splitting that allows for hypothesis testing
- an incremental recall measure by Cremonesi et al. [CKT10]
- significance testing

The incremental recall measure was explained in Sec. 4.4.2 in detail. In the following subsections we describe each of the remaining components.

6.4.1 *Parameter Optimization*

Our semi-supervised method consists of multiple components, each of which has several possible instantiations (e.g. a reliability measure can be instantiated as sensitivity-based, or popularity-based reliability measure, etc.). Additionally, matrix factorization itself requires setting of parameters, such as number of latent dimensions and the regularization constant λ. To find the optimal setting for the parameters and components, we perform an initial optimization step.

For that we hold out a small subset of an original dataset and run a grid search in the parameter space on it. The approximately optimal parameter settings from the grid search are then used in the final evaluation (cf. Ch. 7).

To create the holdout subsets we sample randomly a small percentage of users from the original dataset. Those percentages are listed in Tab. 16 for all datasets. Sampling users instead of ratings has the advantage of not artificially increasing the sparsity of the data.

Optimization of the parameters on a separate subset prevents favorizing methods with more parameters. Otherwise, such methods could be tuned more than methods with fewer parameters to perform best on the test dataset. This procedure ensures that all methods, no matter how many parameters they have, are run only once on the final evaluation set.

6.4.2 *Dataset Splitting*

Splitting the data set into training and testing in our evaluation is inspired by the prequential evaluation proposed by Vinagre et al. for recommender systems [VJG15b] and by Gama et al. for data stream mining [GSR09]. We described this evaluation protocol in detail in Sec. 4.4.1.

However, the prequential evaluation has a major disadvantage. Evaluation measures calculated on a stream at time point t and $t+1$ are statistically not independent from each other, even if the evaluation measure is not cumulative. This is due to the fact that the learner at the time point $t+1$ already trained on an instance from time point t.

In consequence, due to the lack of statistical independence, running of hypothesis tests is not possible on an instance level. For instance, let Q_t be a quality measure at time point t. We consider $Q_t, Q_{t+1}, ..., Q_{t+n}$ observations for a hypothesis test. The most basic prerequisite for a hypothesis test is the independence of those observations. In the prequential evaluation this prerequisite is violated and, therefore, hypothesis testing is not permitted.

To solve this problem, we propose to use two disjoint streams of ratings (one stream for training and one for evaluation). Because of this separation the observations $Q_t, Q_{t+1}, ..., Q_{t+n}$ for a hypothesis test are independent for non-cumulative quality measures. In section 6.4.3 we describe how to use a state-of-the-art evaluation measure in this setting.

A stream-based matrix factorization, the state-of-the-art in recommender systems, usually starts with a batch-based initialization phase. Before the algorithm switches into the streaming-mode, a short batch-training is performed, where the initial latent matrices are trained in a supervised way. While it is not strictly necessary to perform this initial phase, it is realistic to assume that in nearly all applications there is some historical data that can be used for this purpose. By using it, a bad initial performance at the beginning of the stream can be avoided.

Therefore, this initial phase also has to be considered, when splitting a dataset. Therefore, our method for splitting datasets incorporates all the following aspects:

- initial training and testing in batch mode
- a training stream for incremental updates of a model
- a disjoint test stream for evaluation and significance testing

A schematic representation of dataset splitting is in Fig. 16. Part 1) in the figure is used for batch training in the initialization phase. Since this is a supervised method, it also needs a test set in the batch

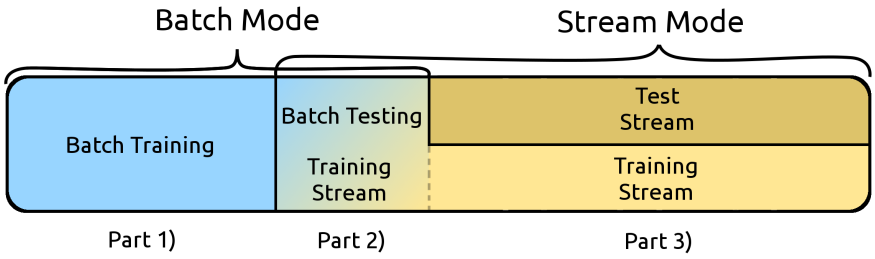

Figure 16.: Splitting of the dataset between the batch and streaming mode. Separation of training and test datasets in each of the modes (figure from [MS17]).

mode (part 2 in the figure). After the initialization phase the algorithm switches to the streaming mode, which is the main mode of this method.

In the streaming mode (part 3 of the dataset) there are two disjoint streams, one stream for training and one for testing. The results we present in the next section are calculated on the test stream.

If we consider the parts of the dataset used for training, so far it was part 1) and a subset of part 3). Part 2) would represent a temporal gap in the training data. Since many of methods in recommender systems rely heavily on the time aspect present in the data, such a gap would be problematic. Therefore, we include part 2) of the dataset into the training stream (represented by the colour gradient in the figure). Since this part was used once for batch testing already, we do not include it into the test stream.

The split ratios between the subsets of the dataset can be adjusted to the need of an application scenario. For our experiments we use the following ratios: 30% of a dataset are used for the batch training, 20% for batch testing. Those 20% are also included into the training stream. The remaining part of the dataset is used in the streaming mode, 30% of which is used as the test stream.

6.4.3 *Significance Testing*

To show that the improvement due to the application of semi-supervised learning is statistically significant, we incorporate hypothesis tests into our evaluation protocol. Hypothesis testing on instance level is possible, since we use two separate streams (one for evaluation and one for testing), which guarantees the independence of observations for non-cumulative quality measures.

As a quality measure for the hypothesis testing we use a binary hit count from the incremental recall@10. An example for observations of this quality measure is represented in Tab. 15.

	Timepoint (rating)				
Algorithm	t_0 (r_{u_a,i_w})	t_1 (r_{u_b,i_x})	t_2 (r_{u_c,i_y})	t_3 (r_{u_d,i_z})	...
Alg1	1	0	1	1	...
Alg2	1	0	0	1	...

Table 15.: An exemplary input to the McNemar's test. Rows indicate performance of algorithms over time. 1 means a hit in the sense of incremental recall. Therefore, an algorithm with significantly more hits is considered better (table from [MS17]).

The columns of the table indicate a time point in the stream with the corresponding rating (in the parenthesis). Rows represent the performance of algorithms over time. The performance is the binary representation of a hit. For instance, at the time point t_0 the rating r_{u_a,i_w} occurred in the stream. The Alg1 was able to rank the item i_w in top 10 (we measure the incremental recall at 10) among 1000 additional random items. Therefore, for this rating the Alg1 scores a hit (1 in binary notation). At time point t_1 none of the algorithm was able to rank the relevant item in top 10, therefore they both score zero in the table.

In this exemplary table we see that the Alg1 scored more hits than the Alg2. However, the evidence in this example is not sufficient to consider any of them superior. To test if the improvement of the Alg1 is statistically significant, we use the McNemar's test [McN47]. This test is used for paired, nominal, dichotomous data, same as presented here.

For this test, our exemplary input from Table 15 is transformed into a contingency table and odds ratio (OR) is calculated. The null hypothesis of the test is: $H_0 : OR = 1$ and the alternative hypothesis is $H_1 : OR > 1$. If the null hypothesis is rejected, we can say that Alg1 is significantly better than Alg2. All p-values reported in Section 6.5 result from this test (lower p-values are better).

In [GSR09] Gama et al. suggested to apply a sliding window or forgetting factors onto the test statistic in the McNemar test. By doing so, information about the dynamics of the learning process can be obtained. In this case, the test statistic reflects mostly the recent time interval and, therefore, allows to test hypotheses specific to a selected time period.

In this work, however, we are interested in the global effect of SSL, i.e., we test if there is a significant improvement due to SSL without restricting the time interval. Therefore, we do not apply sliding windows or forgetting factors and use the entire test stream for out significance testing.

Since we perform tests several times (e.g. SSL vs. noSSL and self-learning vs. noSSL, etc.), there is a risk of alpha error inflation. To account for this fact we correct the reported p-values for multiple testing. For this purpose we use the Hommel's method [Sha95]. All p-values in Section 6.5 have been corrected using this method.

6.5 EXPERIMENTS

In this section we report the results of empirical evaluation on five real world datasets (cf. next subsection). To show the improvements by our method we compare the semi-supervised framework to a single learner without semi-supervised learning (noSSL). In both cases the algorithm used is the extBRISMF (cf. Sec. 6.3.1), so that the only difference between the compared algorithms is the application of SSL.

Within the SSL framework we distinguish between three cases:

- self-learning (SL)
- co-training with two learners (SSL2)
- co-training with three learners (SSL3)
- comparison baseline without SSL (noSSL)

Our framework is also capable of using more that 3 co-trainers in an analogous way. However, the computation time rises with every additional co-trainer.

Results reported in this section are all calculated using the approximately optimal parameter and component settings from the grid search performed on hold-out datasets (cf. Sec. 6.4). Therefore, for each of the methods we have only one setting used in the final evaluation.

The grid search was performed on a cluster running the (Neuro)Debian operating system [HH12]. In total we conducted more than 700 experiments. In Section 6.5.3 we analyse the impact of different instances of framework components (e.g. which reliability measure performs the best).

6.5.1 *Datasets*

In Table 16 we present summary statistics of five real-world dataset that we used in our evaluation. Those datasets are: MovieLens 1M and 100k[2] [HK16], a sample of 5000 users from the extended Epinions [MA06] dataset, a sample of 10 000 users from the Netflix dataset[3] and a sample of the same size from the Flixster dataset[4]. From some of the

2 http://www.movielens.org
3 https://www.netflix.com
4 https://www.flixster.com

Dataset	Ratings	Users	Items	Sparsity	Ratio of Users for Parameter Optimization
ML1M	1,000,209	6,040	3,706	95.53%	0.05
ML100k	100,000	943	1,682	93.7%	0.1
Flixster (10k users)	569,623	10,000	18,108	99.69%	0.01
Epinions (5k users)	496,222	5000	250,488	99.96%	0.03
Netflix(10k users)	2,143,622	10,000	17,249	98.76%	0.01

Table 16.: Dataset statistics; "Ratio of Users for Parameter Optimization" indicates what percentage of users was used for parameter optimization using a grid search. Extreme sparsity values show the abundance of unlabelled information (table from [MS17]).

big dataset we took a sample of users because of a huge number of experiments we run in our grid search.

The last column in the table shows what percentage of users has been held out for parameter optimization. The sparsity values in the fifth column are extremely high. They show that more than 90% of the user/item combinations are unknown.Our semi-supervised framework exploits this abundantly available information.

6.5.2 *Performance of SSL*

In Figs. 17 - 21 we present the IncrementalReall@10 for each of the datasets. In Tab. 18 we report the significance of the differences in performance among the algorithms per dataset. and in Tab. 17 we summarize the results.

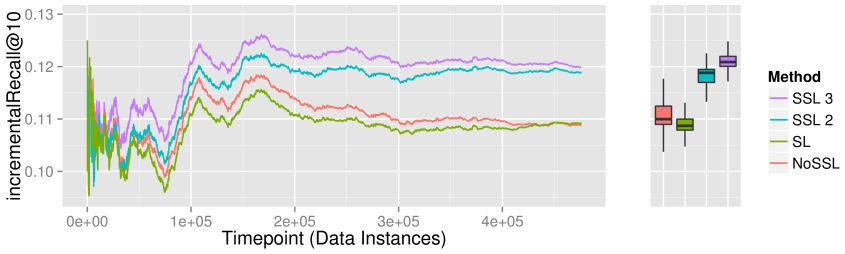

Figure 17.: Incremental Recall@10 over time on the *Movielens 1M* dataset achieved by different SSL methods as compared to noSSL (higher values are better). The box plot on the right visualizes an aggregated distribution of incremental recall. Figure from [MS17].

In Fig. 17 we present a comparison of those methods on the ML1M dataset. The figure presents incremental recall@10 over time (higher results are better). The right part of the figure shows a box plot with a simplified distribution of incremental recall. The middle bars of the boxes represent the median of recall values and the hinges stand for the first and third quartile of the distribution.

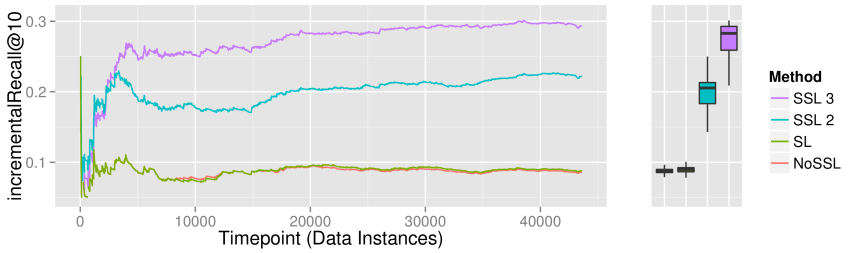

Figure 18.: Incremental Recall@10 over time on the *Movielens 100k* dataset achieved by different SSL methods as compared to noSSL (higher values are better). Figure from [MS17]

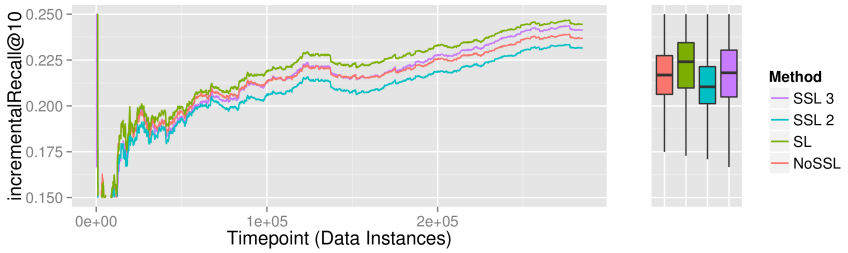

Figure 19.: Incremental Recall@10 over time on the *Flixster (10k users)* dataset achieved by different SSL methods as compared to noSSL (higher values are better). Figure from [MS17]

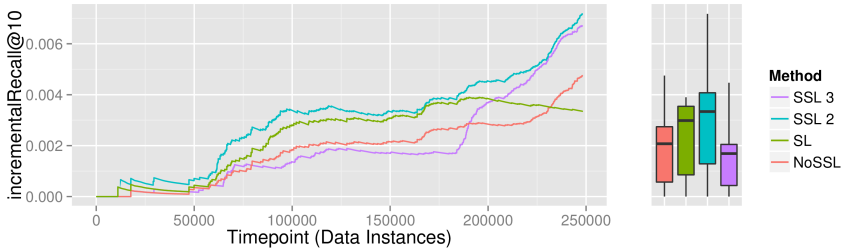

Figure 20.: Incremental Recall@10 over time on the *Epinions (5k users)* dataset achieved by different SSL methods as compared to noSSL (higher values are better). Figure from [MS17]

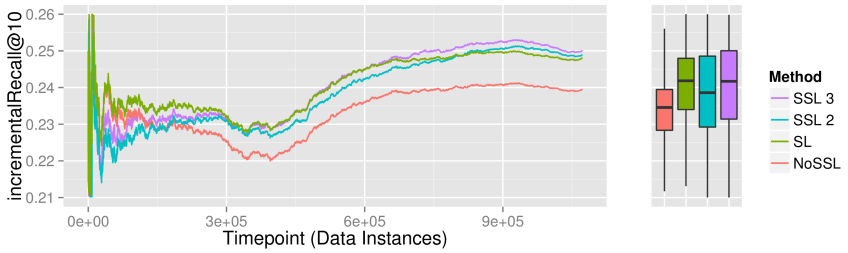

Figure 21.: Incremental Recall@10 over time on the *Netflix (10k users)* dataset achieved by different SSL methods as compared to noSSL (higher values are better). Figure from [MS17]

Fig. 17 shows that SSL3, i.e. co-training with three learners, performed the best on the Movielens 1M dataset, followed by the SSL2 method. Self-learning performed worse than noSSL and converged towards the noSSL-level towards the end of the dataset.

Tab. 17 shows the corresponding numerical values together with the parameter and component settings used in this experiment. This table is grouped with respect to datasets and methods. For instance, for the ML1M dataset and the SSL3 method the optimal number of dimensions k was 30, the regularization λ was 0.03 and the best reliability estimator was the global sensitivity estimator. In the table we do not present the learning rate parameter η, since its optimal value was 0.003 for all methods on all datasets. Also the periodicity of unsupervised learning (USL) m was set to 50 (USL every 50 rating), where $z = 100$ unlabelled instances were selected. SSL3 improved the incremental recall by ca. 8 % compared to noSSL on this dataset. However, the computation time was longer by ca. 18 milliseconds on average for each data instance.

To show that this improvement is statistically significant and not due to a chance, we performed the McNemar's test, as described in Sec. 6.4.3. In Table 18 we show the resulting p-values corrected for multiple testing using the Hommel's method. The columns in the table represent different hypothesis. The second column, for instance, indicates the p-values from the comparison of SSL3 to noSSL. For the ML1M dataset this value is extremely low indicating that the improvement due to SSL3 is highly significant (low p-values are better). Also SSL2 achieves a significant improvement compared to noSSL. Self-learning was not significantly better on the ML1M dataset.

While the improvement on ML1M dataset was 8%, on the ML100k dataset it reached a substantial improvement from 0.0872 (noSSL) to 0.2718 (SSL3). The comparison for this dataset is presented in Fig. 18. Similarly to ML1M, SSL3 is the best performing method, followed by SSL2. SL achieved no substantial improvement. Also here, the improvement is at cost of computation time that increased by 11.5% for SSL3 (cf. Tab. 17). Statistical significance was achieved by both SSL3 and SSL2 (cf. Tab. 18).

On the Flixster dataset (random sample of 10 000 users) the self-learning method showed the best performance (cf. Fig. 19). SSL3 improved the results towards the end of the dataset and SSL2 preformed lower than noSSL. Consequently, only SL and SSL3 achieved statistical significance in the McNemar's test.

On the Epinions dataset (random sample of 5 000 users), which shows the highest sparsity of all tested datasets, SSL2 performed the best. SSL3 achieved a similar performance at the end of the data stream, but it was dominated by the noSSL baseline on parts of the data stream. SL preformed well initially, but did not achieve a sig-

nificant improvement over noSSL (cf. Tab. 18). Both SSL2 and SSL3 yielded a significant improvement over noSSL. However, the computation time for a data instance rose from 18.3 ms with noSSL to 311.7 ms with SSL3.

On the Netflix dataset (random sample of 10 000 users) improvements of recommendation quality are clear for all SSL methods. This is also reflected by p-values in Table 18.

To summarise, our SSL framework achieved significant improvements in the recommendations quality on all datasets. However, the computation time rose, especially when SSL3 was used. Nevertheless, the average processing time of a single data instance remained in the range of milliseconds, ensuring that our approach can be used in real time.

6.5.3 *Analysing the Impact of Component Implementations*

In our framework we propose multiple components and several possible implementations for each of them. To find the best implementation for each of the components, in this section we present an analysis of impact of the implementations onto the quality of recommendations (IncrementalRecall@10).

In Fig. 22 we present the results of this analysis. Each sub-plot represents the impact analysis of one component. To quantify the impact an implementation has, we performed a series of experiments with the approximately optimal parameter setting from the grid search. Only the implementation of the analysed component varied between single experiments. Those experiments were run on all datasets. In the stacked bar plot in Fig. 22 we observe the cumulative performance of each implementation on all datasets.

Since not all datasets are equally difficult (e.g. incremental recall of 0.009 on Epinions dataset is a high result, while on other datasets it would be considered low), we normalized the bar height for each dataset separately. This gives the same importance to each dataset in the cumulative sum. The labels within the bars, however, indicate the incremental recall before normalization. This is the reason why in the first column, for instance, the bar with IncrementalRecall@10 of 0.006 on the Epinions dataset is higher that 0.058 on the ML1M dataset.

In the upper left subplot of Fig. 22 we see several instances of the *reliability estimator* component together with their cumulative performance on all datasets (cf. colour legend). The best cumulative performance was reached by the "STUser Popularity" reliability estimator with the stream-based normalization (ST for self-tuning). It is followed by the "Item Recall-based" estimator, "User Popularity" estimator and "User Recall-based" estimator with similar results. Those implementa-

Method	k	λ	Reliability Estim.	Prediction Assembler	Unlabelled Instance Selector	Training Set Splitter	Avg. IR@10	$\widehat{t}$ (ms)
ML1M								
noSSL	30	0.03	-	-	-	-	0.1103	0.2
SL	30	0.03	Random	Reliability-weighted	Item-RMSE-based	-	0.1082	1.2
SSL3	30	0.03	Global Sensitivity	Reliability-weighted	User-Recall-based	Dim.-preserving Random	**0.1190**	18.1
SSL2	30	0.03	Global Sensitivity	Global-Recall-based	Latent User Disagreement	Dim.-preserving Random	**0.1160**	9.1
ML100k								
noSSL	50	0.03	-	-	-	-	0.0872	0.2
SL	50	0.03	ST-User Popularity	Max. Reliability	Item-RMSE-based	-	**0.0884**	0.5
SSL3	30	0.01	ST-User Popularity	Max. Reliability	Random	Dim.-preserving Random	**0.2718**	2.3
SSL2	30	0.01	ST-User Popularity	Max. Reliability	Item-Recall-based	User Size Splitter	**0.2016**	2.6
Flixter 10k users								
noSSL	50	0.03	-	-	-	-	0.2147	0.3
SL	30	0.03	Global Sensitivity	Global-Recall-based	Random	-	**0.2205**	1.1
SSL3	30	0.03	Global Sensitivity	Global-Recall-based	Item-Recall-based	Dim.-preserving Random	0.2144	30.9
SSL2	30	0.03	Global Sensitivity	Global-Recall-based	Item-Recall-based	Item Size Splitter	0.2085	16.2
Epinions 5k users								
noSSL	50	0.01	-	-	-	-	0.0018	18.3
SL	30	0.03	Global Sensitivity	Global-Recall-based	Item-Recall-based	-	**0.0024**	41.9
SSL3	30	0.01	Global Sensitivity	Global-Recall-based	Item-Recall-based	User Size Splitter	**0.0020**	311.7
SSL2	30	0.01	Global Sensitivity	Global-Recall-based	Item-Recall-based	User Variance Splitter	**0.0031**	165.8
Netflix 10k users								
noSSL	50	0.01	-	-	-	-	0.2337	0.4
SL	50	0.01	Global Sensitivity	Global-Recall-based	Item-Recall-based	-	**0.2412**	3.1
SSL3	30	0.01	Global Sensitivity	Global-Recall-based	Item-Recall-based	Dim.-preserving Random	**0.2409**	57.8
SSL2	50	0.01	Global Sensitivity	Global-Recall-based	Item-Recall-based	Item Average Splitter	**0.2388**	34.2

Table 17.: Results of our SSL framework in comparison to the noSSL method on five datasets together with the corresponding parameter settings. Values of average incremental recall (Avg. IR@10) better than in noSSL are marked in bold. On all datasets our SSL framework achieved an improvement, however, at cost of average computation time for a data instance $\widehat{t}$. (table from [MS17])

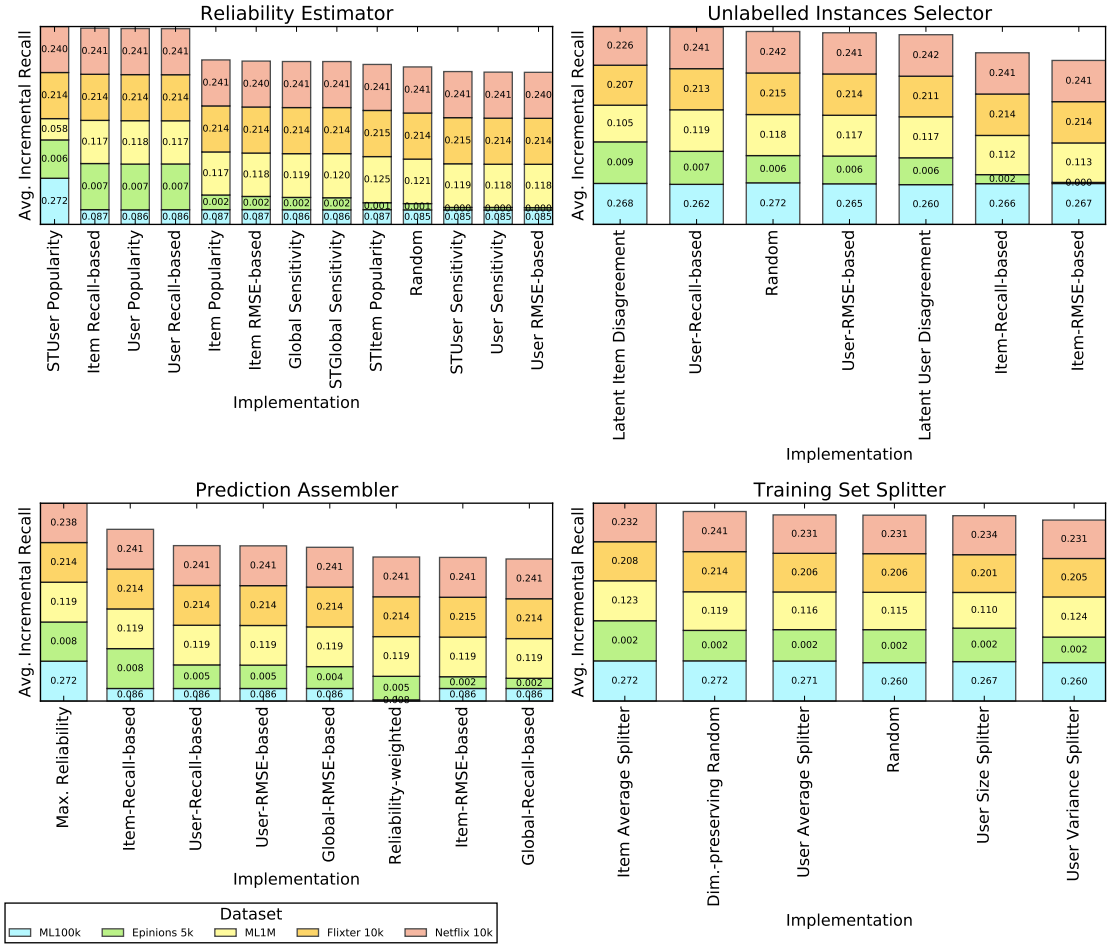

Figure 22.: Analysis of impact of component instances onto the quality of recommendations (avg. incremental recall). We conducted experiments with the optimal parameter setting, where only one component varied (e.g. reliability estimator in the upper left subplot). Component instances with the highest cumulative sum of performance on all dataset are the best (leftmost in all subplots). Figure from [MS17]

	P-values		
Dataset	**SSL3 vs. noSSL**	**SSL2 vs. noSSL**	**SL vs. noSSL**
ML1M	4.400e-16	4.400e-16	0.6058
ML100k	4.400e-16	4.400e-16	0.5722
Flixster (10k users)	0.006648	0.992	4.431e-10
Epinions (5k users)	1.9592e-08	6.6e-16	1
Netflix(10k users)	4.400e-16	2.612e-15	4.400e-16

Table 18.: P-values from the McNemar's test, corrected for multiple testing according to the Hommel's method (cf. Sec. 6.4.3). P-values lower than 0.05 are marked in red. They indicate a statistically significant improvement over the noSSL algorithm (lower values are better). Table from [MS17]

tions of the reliability estimator answer our research question about how to select reliable predictions to learn from (**RQ 3.2**).

The upper right subplot presents the same analysis for the unlabelled instance selector component. Here, the "Latent Item Disagreement" reached the best performance. However, the random selector achieved a similar result while being computationally less expensive. Therefore, in time-critical application scenarios we recommend the usage of the random unlabelled instance selector. This is also our answer to **RQ 3.1** about how to select unlabelled instances as candidates for semi-supervised learning.

The best implementation of the prediction assembler component is based on "Max.l Reliability", i.e. the prediction with maximal reliability serves as the final prediction of all co-trainers. This prediction assembler answer our question about how to aggregate multiple predictions of a co-training approach into one prediction of the entire system (**RQ 3.3**). Using reliability estimates as weights performed relatively poor (sixth place in the figure).

As a method of splitting the training dataset among co-trainers the "Item Average Splitter" works the best. It assigns items with different average ratings to different co-trainers (e.g. good items to one co-trainer, bad items to the other one), so that they can specialize on each subgroup. This implementation answers our research question regarding the division of labelled instances among different co-trainers in a semi-supervised setting (**RQ 3.4**).

6.6 CONCLUSIONS FROM SEMI-SUPERVISED LEARNING

Recommender systems suffer from extreme data sparsity. Only few items are labelled by users. Therefore, the number of unlabelled items is unproportionally higher than the number of labelled ones. We pro-

pose a novel framework for stream-based semi-supervised learning for recommender systems that selectively exploits this abundant unlabelled information and alleviates the sparsity problem.

We proposed the first such framework for stream-based recommender systems. We implemented two semi-supervised learning approaches: self-learning and co-training and evaluated them in a streaming setting on five real-world datasets. We have shown that our SSL framework achieves statistically significant improvements in the quality of recommendations. The best performing approach is co-training with three learners (SSL3). This approach achieved significant improvements compared to noSSL on all datasets. Co-Training with two learners (SSL2) was significantly better than noSSL on four out of five datasets. The improvements achieved by the self-learning method were not consistent on all datasets. Therefore, we recommend this technique only after prior testing.

The improvements due to the co-training approach show that selective learning from unlabelled information improves the quality of recommendations. Therefore, we answer the **RQ 3** positively.

Even though the computation time increased, especially with the SSL3 method, the resulting computation time for each data instance remained in the range of milliseconds (maximally 311.7 ms on the Epinions dataset). This proves the applicability of our framework to real-time applications.

In our experiments we used the BRISMF algorithm by Takács et al. [Tak+09], a state-of-the-art matrix factorization algorithm. We extended the it by the ability to add dimensions to a rating matrix during runtime, as new users and items appear in the stream. This is an important feature, especially for volatile applications.

We also introduced a new evaluation protocol for stream-based recommender systems that incorporates statistical testing, a correction for multiple tests and a sophisticated method of splitting datasets for an unbiased stream-based evaluation.

A limitation of our method is the computation time, which forced us to limit the number of co-trainers. While three co-trainers still showed to be applicable in real-time, their number cannot be much higher at the current state-of-the-art. This problem could be alleviated by parallelization and distributed computing.

Also, in our current framework, co-trainers use different views of the training data during the batch training phase. In the streaming mode, all co-trainers receive the same training instances. While it is not a problem for short streams, in potentially infinite streams the co-trainers can approximate each other (i.e. they can converge to predict the same values). In this case the advantage of SSL would slowly degrade and the performance of the algorithm would converge towards the performance of a noSSL algorithm. Once this happens, a retrain-

ing of the models with new data is needed. In our future work, we plan to extend our framework so that views are also applied online onto the stream instances. Thus, the potential retraining of models would not be necessary.

7 EXPERIMENTAL FRAMEWORK

To answer our research questions we have developed an experimental framework that made extensive experimentation possible. In this chapter, we describe two essential aspects of our framework:

- a component for automatic hyper-parameter optimization
- a mechanism for distributed computation of experiments

In Sec. 7.1 we explain the motivation behind hyperparameter optimization and describe our comparative study of different optimization algorithms in the context of recommendation algorithms.

To answer our research questions in a reliable way a large number of experiments is necessary. To conduct such a large number of experiments our framework includes a mechanism for distributing the computation. In Sec. 7.2 we explain this mechanism and describe how the numerous experiments can be distributed across multiple computers using Apache™ Hadoop ® [1]. Sec. 7.1 comes (with modifications) from our paper on hyperparameter optimization [Mat+16].

7.1 COMPARATIVE STUDY ON HYPERPARAMETER OPTIMIZATION IN RECOMMENDER SYSTEMS

In this section, we discuss hyperparameter optimization (HPO) in recommender systems. First, in Sec. 7.1.1, we motivate the need for HPO. Sec. 7.1.2 describes the related work on HPO techniques. In Sec. 7.1.3, we describe the implemented algorithms. In Sec. 7.1.13 we explain our evaluation protocol. Experiments and results are presented in Sec. 7.1.14. In Sec. 7.1.15 we draw conclusion from this study.

7.1.1 *Motivation for Hyperparameter Optimization*

Hyperparameter optimization is an indispensable tool for researchers and practitioners working with machine learning and data mining algorithms. Many of those algorithms are highly sensitive to parameter setting, which is considered a task of a human expert. An example is the k-means algorithm that requires setting of the parameter k, i.e. the number of centroids. Setting a non-optimal number results in clustering of a bad quality. Often human experts have no means to know in advance what a good hyperparameter setting is. Therefore, they try

1 http://hadoop.apache.org/

several settings blindly until an acceptable setting is found. Hyperparameter optimization algorithms offer a solution to this problem.

For researchers hyperparameter optimization is essential, for instance, when comparing two algorithms. Let A and B be two algorithms with different hyperparameters that need to be tuned. After experimental evaluation we obtain the error measure H of the two algorithms: H_A and H_B. Let $H_A < H_B$. When the tuning is done manually by a human, it is unknown if the difference in the measured error is due to the algorithm A being better than B, or it is because the human expert tuned A better than B. A reliable conclusion is not possible. To marginalize the influence of the human expert, an objective and fair tuning component is required. Hyperparameter optimization methods play the role of the objective component.

Formally, we define the hyperparameter optimization task as follows. Let A be the target algorithm with n number of parameters to be tuned. Each parameter θ_i can be a value taken from an interval $[a_i, b_i]$ in parameter configuration space $\Theta = [a_1, b_1] \times \ldots \times [a_n, b_n]$. Let the vector $\overrightarrow{\theta} = [\theta_1, \theta_2, \ldots, \theta_n]$ represent a parameter configuration and $H : \Theta \to \mathbb{R}$ be an error measure that maps $\overrightarrow{\theta}$ to a numeric score computed over a set of instances. Therefore, the optimization problem aims to find $\overrightarrow{\theta} \in \Theta$ that minimizes $H(\overrightarrow{\theta})$. In our problem definition we partially adopted the notation proposed by Lindawati et al. [LLL11].

As many other algorithms, RS are also sensitive to setting parameters correctly. We first investigate which hyperparameter optimization method is the most appropriate for them *compare nine optimization algorithms on four real-world, public, benchmark datasets*. As result of our research, we give clear recommendations to practitioners and researchers in the RS domain (cf. Sec. 7.1.15).

7.1.2 *Related Work on Hyperparameter Optimization*

Two main classes of algorithms in the hyper-parameter optimization are model based (MB) and derivative-free (DF) approaches. MB approaches approximate the response surface with another function by sampling points using the current model. DF approaches make use of heuristics in order to achieve the best parameter combination.

Recently, MB approaches are gaining popularity over DF approaches due to the fact that evaluating a surrogate model instead of the response surface is computationally cheaper. Jones et al. proposed the Efficient Global Optimization algorithm (EGO), which computes the expected improvement using a combination of a linear regression model and a noise-free stochastic Gaussian process model (also known as DACE model)[JSW98]. Many Sequential Model Based Optimization algorithms (SMBO) are based on EGO [Ber+11],[SLA12], among others also the SMAC algorithm [HHL11] (cf. Sec. 7.1.8). The main

competitor of SMAC is a meta-heuristic called genetic algorithm GGA [AST09], which combines the power of heuristics and model-based approaches. A recent direct search approach is CALIBRA. It uses a simple local search algorithm combined with fractional factorial design [AL09]. Frank Hutter et al. (the authors of SMAC, which we used in our experiments, cf. Sec. 7.1.8) implemented another prominent local search algorithm called PARAMILS[Hut+14], and proved its superiority over CALIBRA.

There is also a wide range of stochastic optimization algorithms. However, most of them depend on the availability of a gradient (e.g. stochastic gradient descent, alternating least squares, etc.), making them inapplicable to our problem. The gradient can be, however, finely approximated by heuristic algorithms or DF approaches. Commonly used heuristics encompass genetic algorithms (Sec. 7.1.7), simulated annealing and particle swarm optimization.

To our knowledge, this is the first such comparative study for recommender systems. While hyperparameter optimization has been investigated for classification or regression problems, recommender systems pose additional challenges that do not occur in conventional regression or classification problems. In contrast to regression problems, the challenge in RS is often to predict a real-valued sparse matrix of ratings that express the degree of preference of users towards items. This results in a different response surface in the optimization problem than in the case of regression or classification. Consequently, optimization methods known to perform well for the conventional regression or classification do not necessarily perform well in recommender systems.

We focus on matrix factorization as recommendation algorithms. MF algorithms are one of the most successful types of algorithms in recommender systems. In the current research they are considered state-of-the-art, since they have shown their superiority in terms of predictive performance and runtime in numerous publications [Tak+09] [KBV09]. Hence, in this work we use a generic representative of those algorithms, the BRISMF algorithm by Takács. et al. [Tak+09], which requires optimization of three parameters: learn rate η, regularization λ and the number of latent dimensions k. While there are numerous MF methods, most of them are based on the same principle that the BRISMF algorithm also uses. Since it is not possible to experiment with countless variants of MF algorithms, we focus on BRISMF, which is representative to most of them and, therefore, allows for generalization.

7.1.3 *Hyperparameter Optimization Algorithms*

In this section, we provide descriptions of the implemented hyperparameter optimization algorithms. In the following, we assume without

loss of generality that the goal of the algorithms is minimization of an error function (and not maximization of a quality function). For all of the described algorithms we use an the same stopping criterion, i.e. a predefined budget of E experiments. [2]

7.1.4 *Full Enumeration*

Full enumeration (also known as *grid search*) attempts to calculate all possible parameter settings. It works only with discrete parameters, since the number of possible combinations with continuous parameters is infinite. Therefore, it often requires a discretization of parameters. Even then, it is highly inefficient, because the number of possible combinations grows combinatorially with the number of parameters.

This method iterates over the whole multi-dimensional grid starting from a randomly selected dimension. The dimensions of the grid are the hyperparameters. A grid point is, therefore, a point in this hyperparameter space and is equivalent to a hyperparameter setting. For the sake of feasibility and of a fair comparison to other methods, we cap this process after 50 experiments (cf. Sec. 7.1.13). This capping is performed equally with all tested methods.

This method serves to us as a comparison baseline that simulates behaviour of a user, who starts an optimization, intending to test all combinations, but interrupts the process due to the excessive computation time and uses the best result found so far.

7.1.5 *Random Search*

We use two versions of random search. Random search with discretized parameters is equivalent to the random grid search. In our experiment we call it "Random discrete". The second version works in the continuous search space and is, therefore, called "Random continuous" hereafter.

Despite its simplicity the random method has several advantages. It is highly parallelizable, in contrast to sequential methods, such as Sequential Model-based Algorithm Configuration (SMAC). This feature of random search is particularly useful when a computational cluster or multi-core processors are available.

Furthermore, it is possible to specify theoretical guarantees on the goodness of the found optimum, as Bergstra and Bengio did [BB12]. They define a hyper-cube around the optimal point in the search space. Let v be volume of the hypercube and V volume of the search space.

2 All algorithms presented in this section (except for SMAC) have been implemented within our framework. In this context we thank our students: Renê Tatua Castillo, Ananth Murthy, Elson Serrao, Ajay Jason Andrade and Prashanth Siddagangaiah for their help with the implementation.

Then the likelihood of finding a point in this hypercube after T trials is equal to:

$$1 - (1 - \frac{v}{V})^T \tag{73}$$

If we define $\frac{v}{V}$ as 5%, then the likelihood of finding a setting within this hypercube after 50 trials amounts to 0.9231.

7.1.6 *Random Walk*

In random walk (RW), the entire search space can be seen as a (high-dimensional) grid, where each point represents a specific hyperparameter setting $\overrightarrow{\theta} \in \Theta$. The RW algorithm is an iterative method. First, it selects a random point on the grid and considers it a central grid point $\overrightarrow{\theta}_c^t$. Then, it computes the performance of each neighbour surrounding that specific grid point. Using a selection mechanism which is described later, we select one of the neighbours and make a step towards this neighbour. This neighbour then becomes the new central grid point $\overrightarrow{\theta}_c^{t+1}$. The whole process is repeated until a stopping criterion is satisfied which is either a maximum number of steps t (iterations) or a maximum number of experiments.

In each iteration of RW a selection of a neighbour is made. We apply a roulette selection algorithm. According to this algorithm, neighbours with high fitness values have a higher chance to be selected. The fitness value is computed as follows. We have K neighbours $\overrightarrow{\theta}_1, \ldots, \overrightarrow{\theta}_K$ and their error measures $H(\overrightarrow{\theta}_1), \ldots, H(\overrightarrow{\theta}_K)$. We sort them and obtain a ranking $H_1, \ldots, H_K$, so that $H_1 \leqslant \ldots \leqslant H_i \leqslant \ldots \leqslant H_K$. Then, the fitness value f_i, assigned for the neighbour at the position i in the ranking is $f_i = \frac{1}{i \cdot H_i}$.

The error measure H_i is multiplied with its ranking i to give more priority to higher ranked settings. Otherwise, the roulette selection would be nearly equivalent to a random selection, since $H_1, ..., H_n$ are very similar to each other. Consequently, the probability p_i, for selecting a neighbour at the position i is:

$$p_i = \frac{f_i}{\sum_{i=1}^{n} f_i} \tag{74}$$

The pseudo-code for RW is shown in Alg. 11. In our algorithms we used the naming convention proposed by Hutter et al. [HHL11].

7.1.7 *Genetic Algorithm*

Genetic algorithms are widely used for solving optimization problems by exploiting the process of natural selection present in evolution. Over the last decades, many GA have been proposed by the research community. One of the best known ones is the Simple Genetic

Algorithm 11 Random Walk Algorithm

Input: R storage of all runs and their performances;
N the current neighbours;
Θ the configuration space for target algorithm A;
S the max. number of steps;
E the max. number of experiments
Output: Optimized parameter configuration $\overrightarrow{\theta}_{inc}$

1: $[\overrightarrow{\theta}_{inc}, R] = Initialization(\Theta)$
2: **while** S and E are not exhausted **do**
3: $[N] = GetCurrentNeighbors(\overrightarrow{\theta}_{inc}, \Theta)$
4: **for** $i := 1, ..., length(N)$ **do**
5: $[R] = RunExperiment(N(i), R)$
6: **end for**
7: $[\overrightarrow{\theta}_{inc}] = GetCurrentGridPointbyRoulette(\overrightarrow{\theta}_{inc}, R)$
8: **end while**

Algorithm (SGA) by Holland [Hol93], which lays the foundations of genetic algorithm (GA). GA aims to optimize a function by evolving a population of candidate solutions (called phenotypes). The information of each candidate solution is encoded into an array (called chromosomes) which can be mutated and altered. The symbols that form the array are called genes.

The GA evolution process is iterative. Usually the first population is generated randomly, and at each generation the fitness of each member of the population is evaluated. Only those individuals with high fitness are likely to transfer their information to the next generation. The process continues until it reaches a stopping mechanism. For instance, when a maximum number of generations is reached or by obtaining a target value within a given threshold. More details about how GA works can be found in [Hol93]. In Alg. 12 we show the pseudo-code for the GA.

A selection mechanism is required to evaluate who are the fittest members in the population. Since the differences in the fitness values are not high we use selection by ordering. It sorts the candidate solutions in decreasing order of fitness value. The population is then trimmed to the original population size. Therefore, only the best candidate solutions (those with high fitness value) are kept and passed to the next generation. In our experiments we used the population size of 4. The pseudo-code for this process is shown in Alg. 13. This version of genetic we call "GA-ordering" in our experiments. Additionally, we use another version "GA-roulette", where the selection is performed by a roulette algorithm.

The recombination (also known as chromosomal crossover) is a convergence operation that is intended to pull the population towards a

Algorithm 12 Genetic Algorithm

Input: R storage of all runs and their performances;
P the current population;
si the original population size;
parameter configuration space Θ;
G maximum number of generations;
E maximum number of experiments (budget)
Output: Optimized parameter configuration $\overrightarrow{\theta}_{inc}$

1: $[P, R]$= $Initialization(\Theta)$
2: **while** G and E are not exhausted **do**
3: **for** $i := 1, ..., length(P)$ **do**
4: $[R]$=$RunExperiment(P(i), R)$
5: **end for**
6: SortPopulation(P)
7: TrimPopulation(si)
8: $[\overrightarrow{\theta}_{inc}]$=P(o)
9: $[P]$= $GetNextPopulation(P)$
10: **end while**

Algorithm 13 Get Next Population

Input: P a population;
C a chromosome
Output: New population P_{new}

1: **for** $i := 1, ..., length(P)$ **do**
2: $P_{new}add(P(i))$
3: **end for**
4: **for** $i := 1, ..., length(P)$ **do**
5: $C = GetChromosomebyIndex(i)$
6: $C_{mutated} = MutateChromosome(C)$
7: $C_{random} = GetRandomChromosome(P)$
8: $[C_{cross1}, C_{cross2}] = Crossover(C, C_{random})$
9: $P_{new}add(C_{mutated})$
10: $P_{new}add(C_{cross1})$
11: $P_{new}add(C_{cross2})$
12: **end for**
13: return P_{new}

local minimum/maximum by combining genes. On the other hand, mutation is a divergence operation that acts as a source of diversity. The GA requires a trade-off between exploitation and exploration similarly to other optimization algorithms.

In our application, a chromosome is the parameter vector $\overrightarrow{\theta}$ (cf. the definition in Sec. 7.1.1). Mutation in the chromosome $\overrightarrow{\theta}$ is equivalent to changing a parameter value at a random position m from this vector into a random value θ_m^{random} from the predefined interval $[a_m, b_m]$. The resulting, mutated chromosome is

$$\overrightarrow{\theta} = [\theta_1, \ldots, \theta_m^{random}, \ldots, \theta_n] \tag{75}$$

Crossover is a binary operation that swaps parts of two chromosomes with each other. Let $\overrightarrow{\theta}_y$ and $\overrightarrow{\theta}_z$ be chromosomes of length n. A result of crossover on $\overrightarrow{\theta}_y$ and $\overrightarrow{\theta}_z$ looks as follows:

$$\overrightarrow{\theta}_y = [\theta_{y_1}, \ldots, \theta_{y_m}, \theta_{z_{m+1}}, \ldots, \theta_{z_n}] \tag{76}$$

and

$$\overrightarrow{\theta}_z = [\theta_{z_1}, \ldots, \theta_{z_m}, \theta_{y_{m+1}}, \ldots, \theta_{y_n}] \tag{77}$$

where m is a random number from $[0, \ldots, n]$.

7.1.8 *Sequential Model-based Algorithm Configuration*

Sequential Model-based Algorithm Configuration (SMAC) [HHL11] is a state-of-the-art, model-based algorithm using Sequential Model-Based Optimization (SMBO). SMBO tackles optimization problems by using four components: an initialization mechanism, a surrogate model, a selection mechanism, and an intensification phase. As described in [HHL11], the first step in SMBO algorithm is the "initialization mechanism" which aims to find the best initial configuration $\overrightarrow{\theta}_{inc}$. Random or default selection is commonly used in this step. In the next step a model M is fitted in order to characterize the response surface H (the surface defined by the error measure). M uses all target algorithm runs performed so far as training data . In other words, a training set of K instances can be seen as

$$\{(\overrightarrow{\theta}_1, H(\overrightarrow{\theta}_1)), \ldots, (\overrightarrow{\theta}_K, H(\overrightarrow{\theta}_K))\} \tag{78}$$

where a parameter configuration is $\overrightarrow{\theta}_i$ and $H(\overrightarrow{\theta}_i)$ is the corresponding target algorithm's observed performance. The selection mechanism's main goal is to select the most promising parameter configurations based on M and the configuration's expected improvement EI. $EI(\overrightarrow{\theta})$ tells us how promising a configuration $\overrightarrow{\theta}$ could be by giving a trade-off

between exploration and exploitation. More details about the calculation of expected improvement can be found in [JSW98]. Finally, the intensification mechanism compares a list of promising configurations P against the current incumbent $\overrightarrow{\theta}_{inc}$ in order to select the new incumbent for the next iteration. The SMBO algorithm is shown in Alg. 14. This pseudo-code comes from [HHL11] and was minimally adjusted to our problem definition.

Algorithm 14 Sequential Model Based Algorithm [HHL11]

Input: R storage of all runs and their performances;
M a model;
P list of promising configurations;
t_{fit} the run times required to fit the;
E the maximum number of experiments (budget) model
t_{select} the run times required to select configurations;
c the cost metric.

Output: Optimized parameter configuration $\overrightarrow{\theta}_{inc}$

1: $[\overrightarrow{\theta}_{inc}, R] = Initialize(\Theta)$
2: **while** total time for configuration or budget not exhausted **do**
3: $[M, t_{fit}] = FitModel(R)$
4: $[P, t_{select}] = SelectConfigurations(M, \overrightarrow{\theta}_{inc}, \Theta)$
5: $[R, \overrightarrow{\theta}_{inc}] = Intensify(P, \overrightarrow{\theta}_{inc}, M, R, t_{fit}, t_{select}, c)$
6: reduce available time and budget
7: **end while**
8: $return(\overrightarrow{\theta}_{inc})$

Internally, SMAC models are based on random forests [Bre01], a machine learning technique for classification and regression tasks. Random forest models are, in this case, an ensemble of regression trees. SMAC uses random forest models to compute EI and implements its own simple multi-start local search for finding configuration $\overrightarrow{\theta}$ with large $EI(\overrightarrow{\theta})$. In [HHL11], SMAC is explained in more detail. In our experiments we used the SMAC software developed by Hutter et al.[3].

While SMAC can use two types of stopping criteria: a time budget or a maximal number of experiments, we use only the latter for the sake of comparison with other algorithms.

7.1.9 *Greedy Search*

Greedy search optimizes only one dimension/parameter at a time, while keeping other dimensions fixed. To optimize one dimension, it sets the value of the corresponding parameter to a random value. For

3 http://www.cs.ubc.ca/labs/beta/Projects/SMAC/

Algorithm 15 Greedy search algorithm

Input: R storage of all runs and their performances;
E the maximum number of experiments (budget)
Output: Optimized parameter configuration $\overrightarrow{\theta}_{inc}$

1: $[\overrightarrow{\theta}_{inc}, R]$= *RandomInitialization*$(\Theta)$
2: **while** E is not exhausted **do**
3: **for** $\forall \theta_i \in \overrightarrow{\theta}_{inc}$ **do**
4: **for** $i := 1, ..., \sqrt{\frac{E}{|\overrightarrow{\theta}_{inc}|}}$ **do**
5: $\theta_i^{new} = random \sim \mathcal{U}\{a_i, b_i\}$
6: $\overrightarrow{\theta}_{inc} = [\theta_1, ..., \theta_i^{new}, ..., \theta_n]$
7: $[R]$=*RunExperiment*$(\overrightarrow{\theta}_{inc}, R)$
8: **end for**
9: $\theta_i = R$.getBest$\overrightarrow{\theta}$.get(θ_i)
10: **end for**
11: **end while**

each parameter it selects m random samples. m is determined according to a heuristic rule $m = \sqrt{E/n}$, where E is the maximal number of experiments (so called budget) and n is the dimensionality of the search space. Then, the value of the parameter is fixated to the best setting from the random samples and the procedure is repeated with all remaining parameters. A pseudo-code explaining this procedure in detail in shown in Alg. 15.

7.1.10 *Simulated Annealing*

Simulated Annealing (SA) is a method motivated by the cooling processes observed in metals. SA uses temperature as a control variable that determines the probability of acceptance of worse solutions than the current one. As temperature goes down, the probability of accepting worse solutions also decreases.

SA systematically compares the current incumbent (parameter setting) with random configurations (so called neighbours). Then, a difference Δ in the error measure between the incumbent and each neighbour is computed. The probability of accepting a neighbour as new incumbent depends on the current temperature and Δ. It is calculated using the formula $e^{-\left(\frac{\Delta}{Temp}\right)}$. Finally, the temperature is updated using a cooling schedule (e.g. linear or geometric). The algorithm stops when it reaches a maximum number of experiments or when a temperature threshold is reached. The pseudo-code in Alg. 16 describes SA.

In our experiments we use two versions of SA with a different definition of a neighbourhood. The first version "SA-discrete" uses a discrete grid and considers adjacent nodes in the grid neighbours. The second

version "SA-Gaussian" uses a Gaussian distribution with mean equal to current parameter value and standard deviation adjusted so that 98% of points lay in the range of parameter values. According to this variant of SA, a neighbouring value is a sample from this distribution. For more details and the initial publication on SA we refer to [KJV83].

Algorithm 16 Simulated annealing algorithm

Input: R storage of all runs and their performances;
E the maximum number of experiments;
T_{ini} the initial temperature;
T_{min} the minimal temperature;
N current neighbours;
T_{rate} temperature cooling rate.
Output: Optimized parameter configuration $\overrightarrow{\theta}_{inc}$

1: $[\overrightarrow{\theta}_{inc}, R] = RandomInitialization(\Theta)$
2: $Temp = T_{ini}$
3: **while** E is not exhausted and $Temp > T_{min}$ **do**
4: $[N] = GetCurrentNeighbors(\Theta)$
5: **for** $i := 1, ..., lenght(N)$ **do**
6: $[R] = RunExperiment(N(i))$
7: $\Delta = R_{N(i)} - R_{\overrightarrow{\theta}_{inc}}$
8: $P = e^{-\left(\frac{\Delta}{Temp}\right)}$
9: **if** $P \geqslant random \sim \mathcal{U}\{a_i, b_i\}$ **then**
10: $\overrightarrow{\theta}_{inc} = N(i)$
11: **end if**
12: **end for**
13: $Temp = Temp * T_{rate}$
14: $return(\overrightarrow{\theta}_{inc})$
15: **end while**

7.1.11 *Nelder-Mead*

This algorithm, also called downhill simplex method, minimizes a function by spanning a simplex in the parameter space. This simplex has $k + 1$ vertices, where k is the number of dimensions of the parameter space. In two dimensional space, for instance, the simplex is a triangle.

The position of the vertices of the first simplex is determined randomly. After that, the simplex is transformed iteratively by using a set of operations: reflection, contraction and expansion described in [NM65]. For every new vertex determined by those operations the function value is calculated. Based on its value, further transforma-

Algorithm 17 Nelder-Mead algorithm [NM65]

Input: E the maximum number of experiments;
α reflection coefficient;
γ expansion coefficient;
ρ contraction coefficient;
σ shrink coefficient;
n number of parameters to be tuned.
Output: Optimized parameter configuration $\vec{\theta}_0$

1: initialize $n+1$ points randomly $\rightarrow \vec{\theta}_0, \vec{\theta}_1, ..., \vec{\theta}_n$
2: **while** E is not exhausted **do**
3: sort the points $\vec{\theta}_0, \vec{\theta}_1, ..., \vec{\theta}_n$ (best first)
4: calculate a centroid $\vec{\theta}_m$ of $\vec{\theta}_0, ..., \vec{\theta}_{n-1}$(without the worst point)
5: calculate the reflection $\vec{\theta}_r$ of the worst point $\vec{\theta}_n$ using the centroid $\vec{\theta}_r = (1+\alpha)\vec{\theta}_m - \alpha \cdot \vec{\theta}_n$
6: **if** $H(\vec{\theta}_0) \leqslant H(\vec{\theta}_r) < H(\vec{\theta}_{n-1})$ **then**
7: $\vec{\theta}_n := \vec{\theta}_r$
8: GoTo line 2
9: **end if**
10: **if** $H(\vec{\theta}_r) < H(\vec{\theta}_0)$ **then**
11: calculate the expanded point $\vec{\theta}_e = \vec{\theta}_m + \gamma \cdot (\vec{\theta}_r - \vec{\theta}_m)$
12: $\vec{\theta}_n := \arg\min_{x \in \{\vec{\theta}_e, \vec{\theta}_r\}} H(x)$
13: GoTo line 2
14: **end if**
15: **if** $H(\vec{\theta}_r) \geqslant H(\vec{\theta}_{n-1})$ **then**
16: $\vec{\theta}_c := \vec{\theta}_m + \rho(\vec{\theta}_n - \vec{\theta}_m)$
17: **if** $H(\vec{\theta}_c) < H(\vec{\theta}_n)$ **then**
18: $\vec{\theta}_n := \vec{\theta}_c$
19: GoTo line 2
20: **end if**
21: **end if**
22: **for** $i \in \{1, ..., n\}$ **do**
23: $\vec{\theta}_i = \vec{\theta}_0 + \sigma(\vec{\theta}_i - \vec{\theta}_0)$
24: **end for**
25: GoTo line 2
26: **end while**
27: Return $\vec{\theta}_0$

tions are carried out iteratively to minimize the values associated with vertices.

An important feature of the Nelder-Mead algorithm is that is does not require the function to be differentiable, in contrast to e.g. gradient descent. Its transformations are based solely on the function values evaluated at the vertices of the simplex. This advantage is essential when optimizing e.g. RMSE as a function of hyperparameters in recommender systems. In Alg. 17 we present the pseudo-code of this algorithm. For details on the transformations, we refer to the work by Nelder and Mead [NM65].

7.1.12 *Particle Swarm Optimization*

Particle Swarm Optimization (PSO) is an optimization method proposed originally by Eberhart and Kennedy [EK95]. It is inspired by the behaviour of swarms in nature. Accordingly, this optimization methods maintains a population of particles (parameter settings in our case) called "swarm".

First, the positions of the particles in the swarm are initialized randomly (cf. pseudo-code in Alg. 18). Each particle maintains additionally its velocity vector and its best position. The entire swarm also stores its best position. Subsequently, guided by the optimum of the swarm and their local optima, each particle in the swarm adjusts its position using the previous position and its velocity. The velocity is derived from the distance of a particle from its own optimum and from the swarm's optimum. Additionally, updating the velocity contains a random component.

7.1.13 *Evaluation Settings*

Our evaluation framework provides a fair and realistic comparison of different hyperparameter optimization methods for recommender systems. In particular:

- The hyperparameter optimizer cannot access information from a hold-out evaluation set (it provides a parameter configuration using only the given training data),
- We repeat the experiments several times to exclude random effects,
- In every repetition each algorithm uses the same permutation of data.

This evaluation framework is given in Alg. 19. As input, it uses a list of different hyperparameter optimization algorithms to be evaluated, a dataset, a recommender system algorithm, and the number of repetitions of experiments.

Algorithm 18 Particle Swarm Optimization

Input: E the maximum number of experiments;
S number of particles;
n number of parameters to optimize;
ω parameter controlling the impact of past particle speed;
φ_p parameter controlling the impact of the distance from the particle optimum;
φ_g parameter controlling the impact of the distance from the swarm optimum.

Output: Optimized parameter configuration $\overrightarrow{\theta}^*_G$

1: **for** $s \in \{1, ..., S\}$ **do**
2: the position of the particle $\overrightarrow{\theta}_s \in \mathbb{R}^n$ is initialized randomly
3: the velocity of the particle $\overrightarrow{v}_s \in \mathbb{R}^n$ is initialized randomly
4: particle's best position is initialized $\overrightarrow{\theta}^*_s := \overrightarrow{\theta}_s$
5: **end for**
6: swarm's best position is initialized $\overrightarrow{\theta}^*_G := \underset{x \in \{\overrightarrow{\theta}_1, ..., \overrightarrow{\theta}_S\}}{\arg\min}\ H(x)$
7: **while** E is not exhausted **do**
8: **for** $s \in \{1, ..., S\}$ **do**
9: random numbers a and b are generated
10: particle's $\overrightarrow{\theta}_s$ velocity is updated $\overrightarrow{v}_s := \omega \cdot \overrightarrow{v}_s + \varphi_p \cdot a \cdot (\overrightarrow{\theta}^*_s - \overrightarrow{\theta}_s) + \varphi_g \cdot b \cdot (\overrightarrow{\theta}^*_G - \overrightarrow{\theta}_s)$
11: $\overrightarrow{\theta}_s := \overrightarrow{\theta}_s + \overrightarrow{v}_s$
12: **if** $H(\overrightarrow{\theta}_s) < H(\overrightarrow{\theta}^*_s)$ **then**
13: $\overrightarrow{\theta}^*_s := \overrightarrow{\theta}_s$
14: **end if**
15: **if** $H(\overrightarrow{\theta}_s) < H(\overrightarrow{\theta}^*_G)$ **then**
16: $\overrightarrow{\theta}^*_G := \overrightarrow{\theta}_s$
17: **end if**
18: **end for**
19: **end while**
20: Return $\overrightarrow{\theta}^*_G$

Algorithm 19 Evaluation Framework

Input: $\mathcal{HO}$ (list of hyperparameter optimizers)
X (dataset)
RS (recommender system)
M (number of repetitions, e.g., 100)
N (number of experiments, e.g., 50)

for $i \in \{1, \ldots, M\}$ **do**
  $res_* \leftarrow \{\,\}$
  for $j \in \{1, \ldots, N\}$ **do**
    $(X_{RSTr}, X_{RSTe}, X_{eval}) \leftarrow \text{randSplit}(X)$
    for $HO \in \mathcal{HO}$ **do**
      Training phase
      $\theta \leftarrow \text{nextParam}(HO, res_{HO})$
      $rs_{tr} \leftarrow \text{trainRS}(RS, \theta, X_{RSTr})$
      $perf_{tr} \leftarrow \text{evalPerf}(rs_{tr}, X_{RSTe})$
      $res_{HO} \leftarrow res_{HO} \cup \{(\theta, perf_{tr})\}$
      Evaluation phase
      $\theta^* \leftarrow \text{getBestParam}(res_{HO})$
      $rs_{ev} \leftarrow \text{trainRS}(RS, \theta^*, X_{RSTr} \cup X_{RSTe})$
      $perf_{ev}^{(i,HO,j)} \leftarrow \text{evalPerf}(rs_{ev}, X_{eval})$
    end for
  end for
end for

The outer loop repeats the whole experiment a given number of times (in this work: $M = 100$). To decide on the best parameter setting, each optimizer has to store all tested parameter configurations and the corresponding performances. This is done in res_* which is initialized with an empty set ($*$ indicates that all cells are initialized accordingly). The optimizer is allowed to conduct a given number of experiments (in this work: $N = 50$) with different parameters.

In each experiment, we split the dataset into three non-overlapping subsets (40%, 27%, 33%). The first two of them (X_{RSTr}, X_{RSTe}) are training sets for the hyperparameter optimizer. More specific, the first one is used to train a recommender system with a chosen parameter setting and the second one to test the recommender. The evaluation set is not accessible to the optimizer to ensure unbiased results.

After splitting the dataset, each hyperparameter optimizer HO selects its next parameter setting θ. The recommender RS is trained on the training set X_{RSTr} with the chosen parameters, which are then evaluated on X_{RSTe} to obtain a performance score. Each parameter-performance-tuple $(\theta, perf_{tr})$ is added to the optimizer's result set res_{HO}.

In the evaluation phase, we determine the best parameter setting based on the calculated parameter-performance-tuples, i.e. the one with the best training performance. Using this parameters θ^*, we train a recommender on all available training data ($X_{RSTr} \cup X_{RSTe}$), and evaluate it on X_{eval} to obtain the evaluation performance. To evaluate the quality of a recommender system, we use the RMSE measure.

Since every optimization method can request RMSE values for a sequence of 50 parameter settings, the result is a learning curve that shows how this method reduces the RMSE with every further experiment performed. Example of such learning curves are shown in Fig. 23 (any blue curve). This figure shows the evaluation RMSE score of each repetition (blue) of the greedy algorithm w.r.t. the experiment number on the Netflix dataset. Since comparing 100 learning curves of one method to sets of curves of a different method would be impossible, we aggregate all those 100 curves into one median curve and plot its 25th and 75th percentiles using the shaded area. This means that half of all curves lies in the shaded area (cf. the red curve in Fig. 23). This enables a simple comparison of many methods. Consequently, all curves in the next section are the median curves.

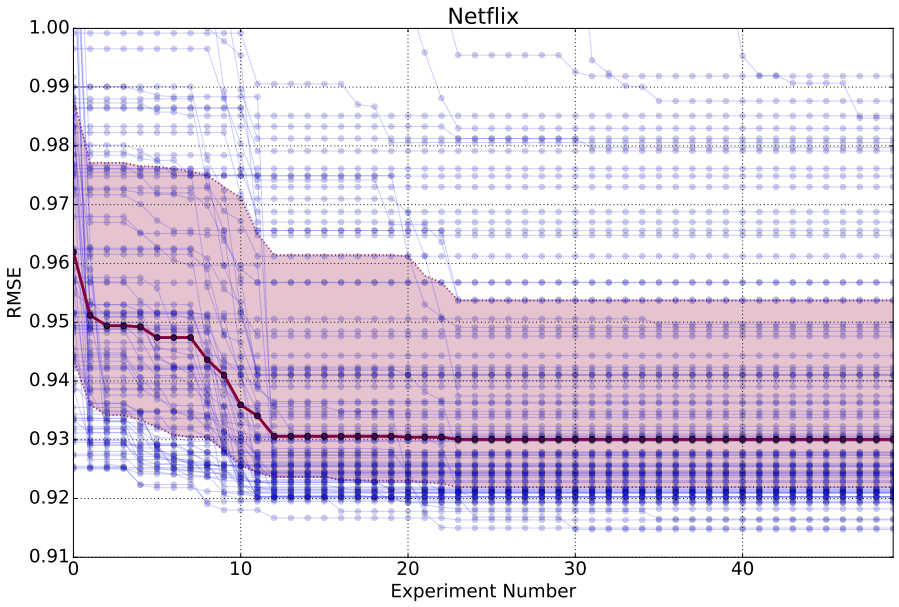

Figure 23.: An example of learning curves of the greedy algorithm on the Netflix dataset. To prevent random effects, every algorithm was run 100 times (blue curves). For comparison, we aggregate them into a median learning curve (in red) and indicate the 25th and 75th percentile by red shaded area (figure from [Mat+16]).

7.1.14 *Experiments*

To evaluate different hyperparameter optimization methods, we use four real-world datasets from the RS domain. The datasets are MovieLens 1M and 100k[4] [HK16], a sample of 2000 users from the Netflix dataset[5] and 2000 random users from the Flixster dataset[6]. A description of these datasets is given in Table 19. Since we performed more than 160,000 experiments in our evaluation, we took a random sample of 2000 users from the big datasets (i.e. Netflix and Flixter). Without sampling a comparative study of this size would be not feasible due to a long computation time. For our computations, we used a cluster running the (Neuro)Debian operating system [HH12].

Dataset	Ratings	Users	Items	Sparsity
ML1M	1,000,209	6,040	3,706	95.53%
ML100k	100,000	943	1,682	93.7%
Flixster (2k)	101,106	2,000	8,419	99.4%
Netflix (2k)	427,223	2,000	13,588	98.43%

Table 19.: Summary of datasets and samples used in hyperparameter optimization experiments (table from [Mat+16]).

For our experiments, we define a search space with three dimensions. The first dimension is k from the matrix factorization algorithm. k is the number of latent dimensions and it is an integer number in the range $[10, 200]$. The second parameter is η, which is a learn rate used by the stochastic gradient descent in the process of factorizing a rating matrix. For η we define a range of $[0.001, 0.1]$. The last parameter is λ. It is a regularization parameter used also by the gradient descent to prevent overly high latent factors in the matrix factorization. Its range is also set to $[0.001, 0.1]$.

Some of the hyperparameter optimization algorithms require discrete parameters. One of them is e.g. the full enumeration (cf. Sec. 7.1.4). Using this method in a continuous search space is not possible because of the infinite number of parameter values. Therefore, we discretize the parameters for those methods by determining 20 different, equidistant values in the aforementioned range. For λ this results in the following set of possible values: $\lambda \in \{0.0010, 0.0062, ..., 0.1\}$.

The same discretization procedure is also applied to the remaining parameters. According to this discrete definition of parameters, the

4 http://www.movielens.org
5 https://www.netflix.com
6 https://www.flixster.com

entire search space encompasses 8000 possible parameter combinations. Every optimization method used in our experiments is allowed to request computation of maximally 50 parameter combinations.

Since some hyperparameter optimization methods are also parametric, we specify the parameters that were set manually for this application in Tab. 20. Ideally, they should also be optimized, however, it is not feasible to also optimize the parameters of the optimizers.

Method	**Parameters**
GA	pop. size = 4; mutation perc. = 0.5
PSO	swarm size = 5; influence by local opt. = 1; influence by global opt. = 3;
Nelder-Mead	reflection coeff.=1; expansion coeff.=2; contraction coeff.=0.5; shrinking coeff.=0.5
SA	cooling schedule=geometric; cooling rate=0.8; iter. for equilibrium=10; init. temp.=100

Table 20.: Parameters of the hyperparameter optimization methods (table from [Mat+16]).

In Fig. 24, we present the median learning curves (cf. Sec. 7.1.13) on a sample of 2000 random users from the Netflix dataset. Every learning curve represents a median of cumulative minimum achieved over K experiments, where K is on the horizontal axis. The curves visualize the results of all methods described in Sec. 7.1.3 except for the full enumeration. The results of this method are considerably worse than the rest (full enumeration was capped after 50 experiments, same as all other methods). Therefore, plotting its curve would make the plot unreadable.

On the Netflix dataset only the random walk algorithm and the full enumeration worked considerably worse than other methods (lower values are better). Other optimization methods converged to a similar level as random methods. The Nelder-Mead algorithm performed the best most of the time, but only with slightly better results than random methods or simulated annealing. A high degree of overlapping in the shaded areas suggests that the differences between the algorithms are not substantial.

Fig. 25 shows the learning curves on the ML1M dataset. The random walk and the full enumeration (not plotted due to high error) were outperformed by the random methods. Also on this dataset the Nelder-Mead algorithm dominated other algorithms with only a marginal improvement as compared to random sampling. At the end

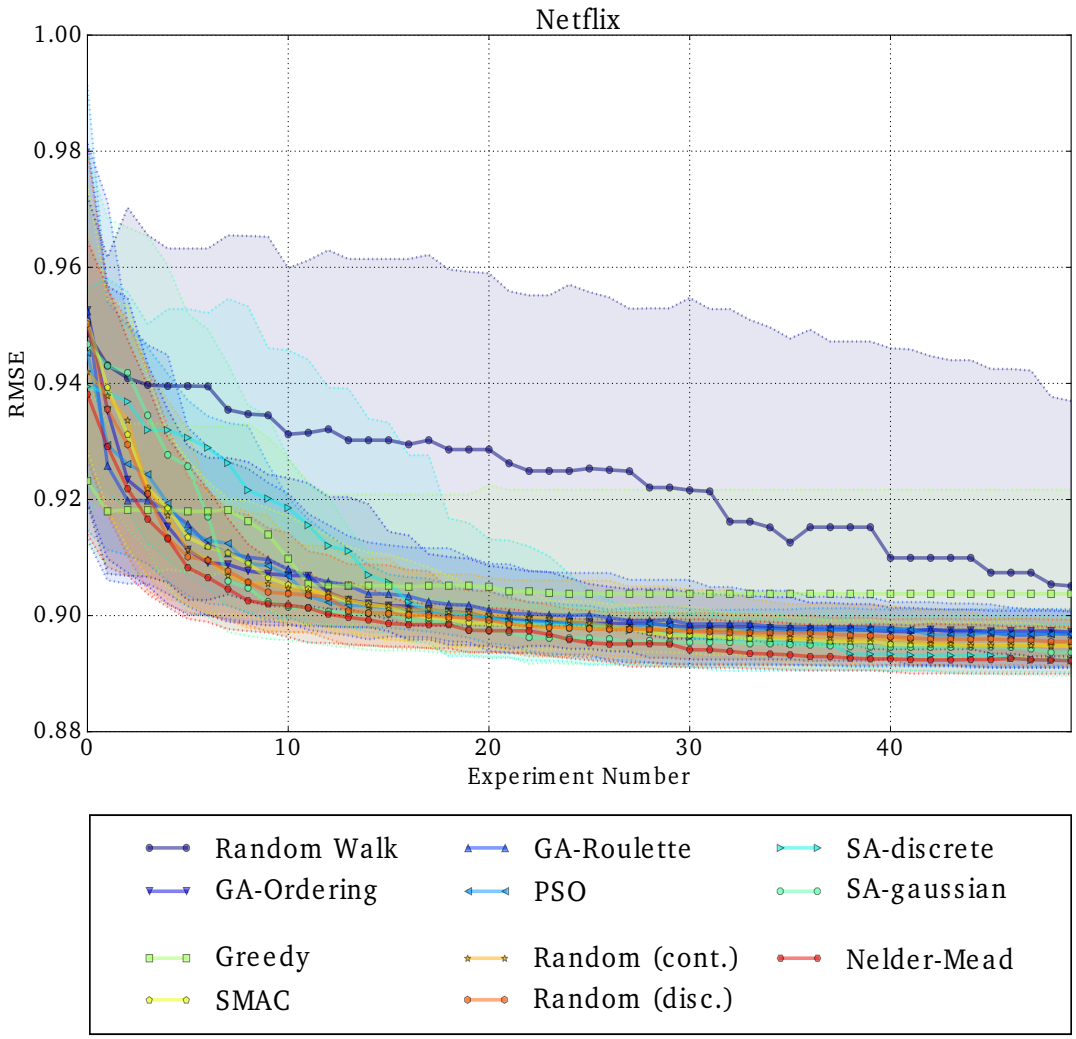

Figure 24.: Median learning curves on a random sample of 2000 users from the Netflix dataset (lower results are better). Figure from [Mat+16]

of the optimization, simulated annealing with Gaussian neighbourhood reached nearly the same result as the Nelder-Mead algorithm.

We observe similar results on a sample from the Flixster dataset (cf. Fig. 26). Here, also the Nelder-Mead algorithm achieves the best result by converging to a similar level as the random algorithms.

Results on the ML100k dataset in Fig. 27 are consistent with previous observations. The Nelder-Mead algorithm performs the best. Random walk and the greedy algorithm are considerably worse.

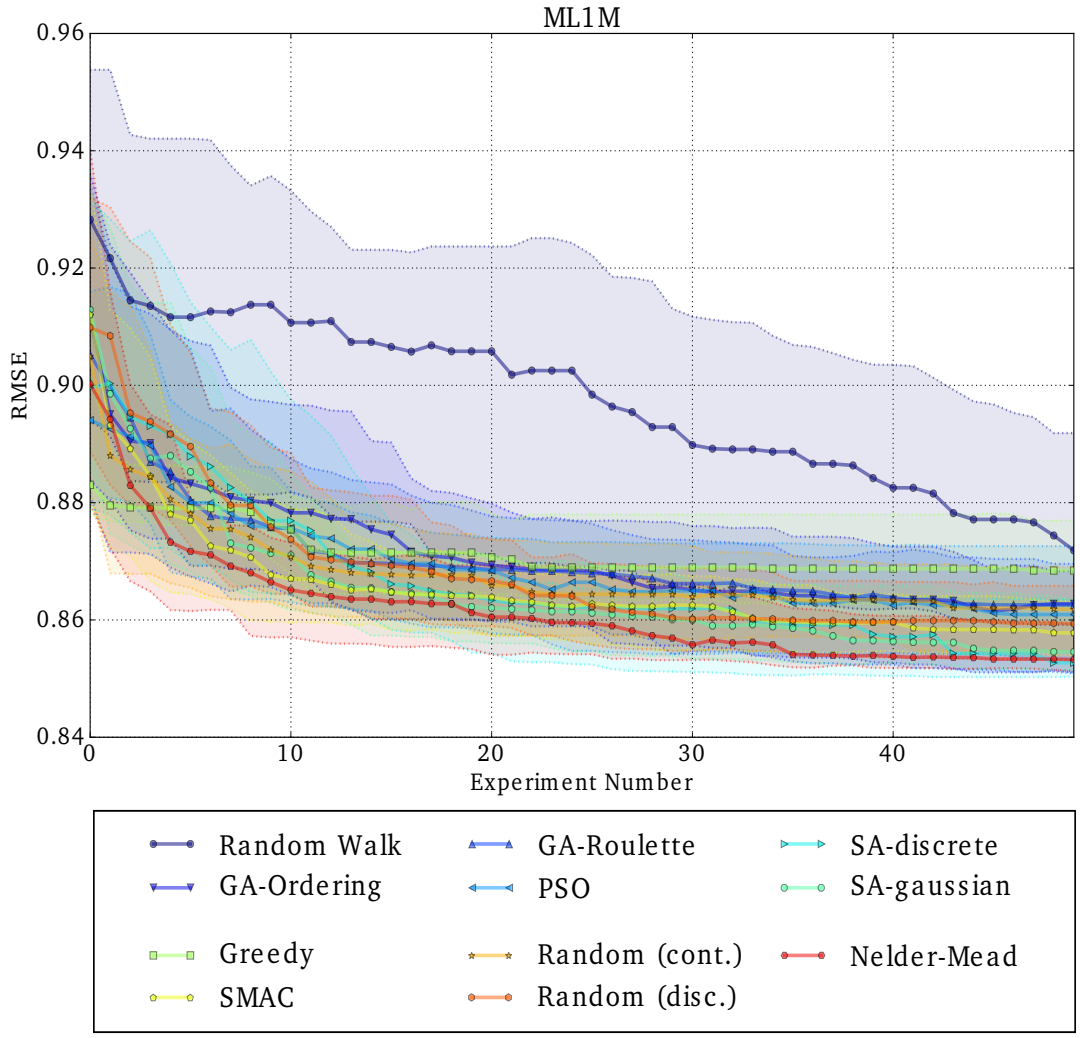

Figure 25.: Median learning curves on ML1M (figure from [Mat+16]).

7.1.15 *Conclusions on Hyperparameter Optimization*

In this study we compared nine hyperparameter optimization algorithms. To our knowledge, this is the first such study in the domain of recommender systems. We performed more than 160,000 experiments on four real-world datasets.

From our experiments, we conclude that random walk, greedy algorithm and full enumeration (also known as grid search) are certainly not recommendable. Those algorithms were outperformed by the random search on all datasets or, in the best case, converged to the level of random search.

SMAC, PSO and genetic algorithms performed similarly to the random search. The best results were achieved by the Nelder-Mead algo-

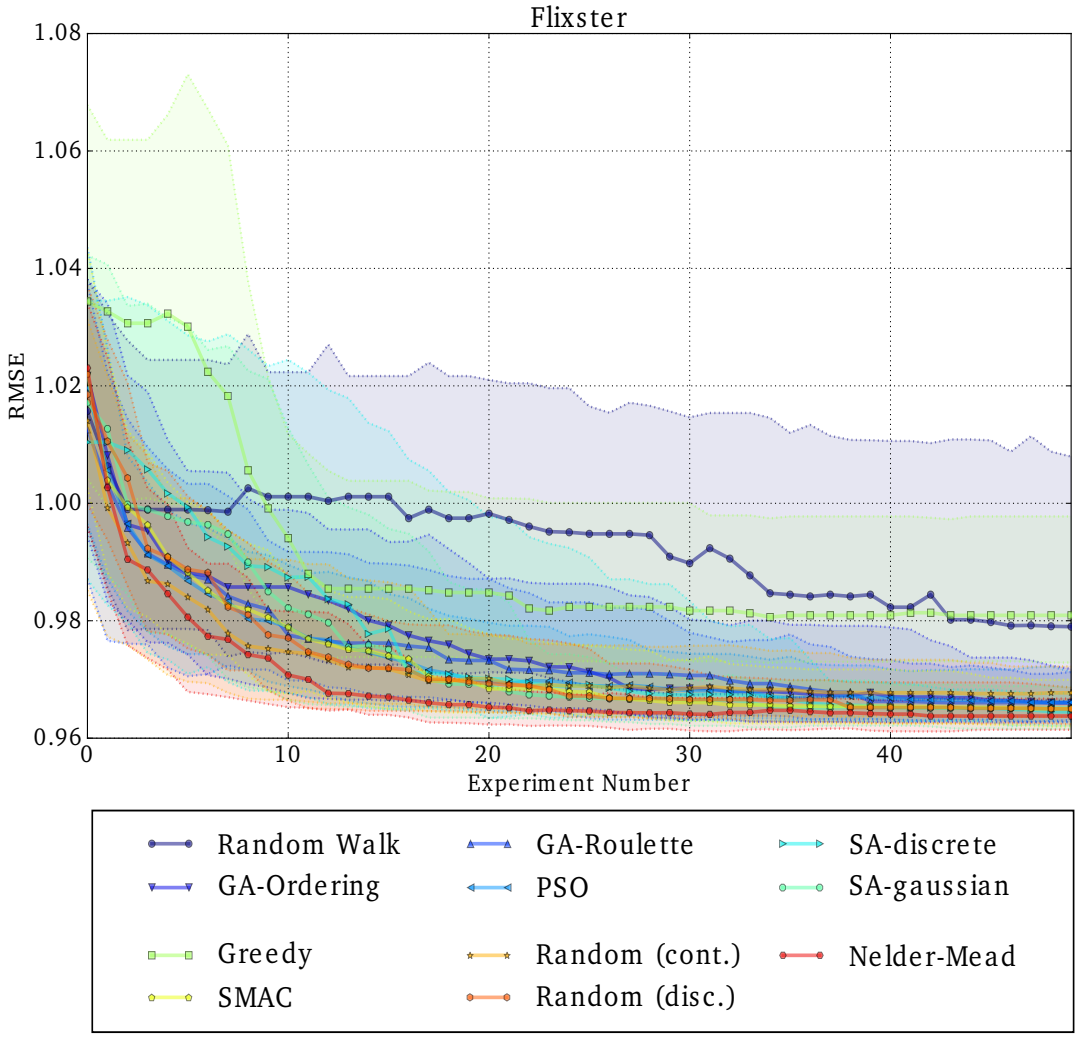

Figure 26.: Median learning curves on a random sample of 2000 users from the Flixster dataset (figure from [Mat+16]).

rithm on all datasets, followed by simulated annealing. Their improvement compared to the random search was, however, only marginal.

Considering the advantages of random search, such as full parallelization, simplicity, constant and nearly negligible computation time, we clearly recommend the random search for optimizing hyperparameters in the domain of recommender systems. Only in application scenarios, where marginal improvements play an important role, the parameters are numerical and paralleization is not necessary, we recommend the application of the Nelder-Mead algorithm or of simulated annealing.

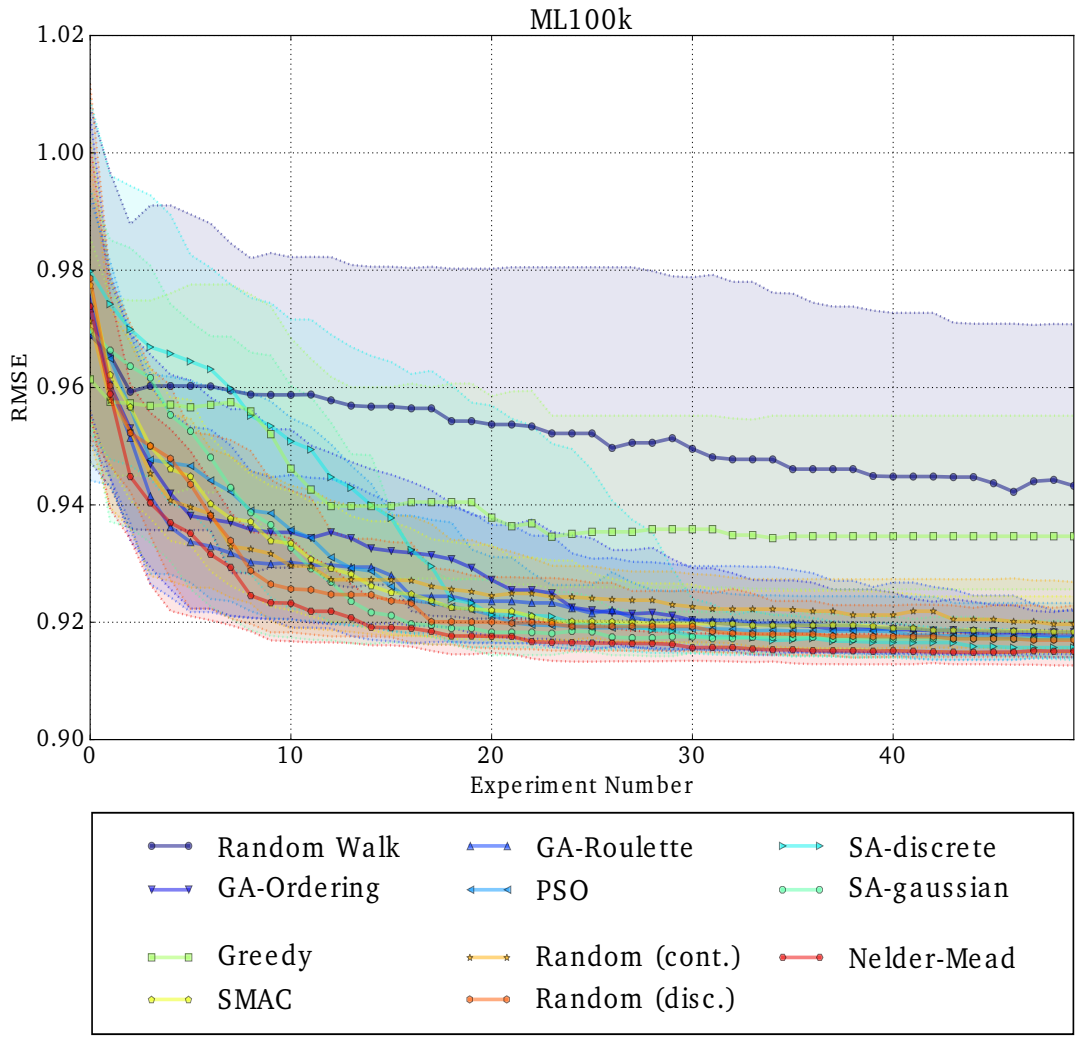

Figure 27.: Median learning curves on the ML100k dataset (figure from [Mat+16]).

7.2 DISTRIBUTION OF EXPERIMENTS

To conduct a large number of experiments, e.g. in the process of hyperparameter optimization, we extend our framework by the ability to run several experiments in parallel using the Apache™ Hadoop ® framework. In HPO we understand as experiment the computation of a single parameter setting, i.e. computation of a function f that transforms a parameter setting $\overrightarrow{\theta}$ into a real-valued quality or error measure:

$$f : \overrightarrow{\theta} \to \mathbb{R} \tag{79}$$

HPO is ideal for distributed computing, for the following reasons:

- experiments in the optimization process are independent from each other
- after distribution of the computation, only little communication between nodes is required
- networking overhead is minimal, since the transferred information contains only parameter settings and results (assuming existence of a distributed storage system for datasets)
- the ratio of networking time to computation time is low

To exploit this potential for parallelization, we express the execution of HPO in terms of *map-reduce* functions that the Hadoop ® framework can parallelize easily. Algs. 20 and 21 show these functions [7].

Algorithm 20 Map

Input: $\overrightarrow{\theta}_i$ a candidate configuration
A a learning algorithm
Tr training set (from a distributed file system)
Te test set (from a distributed file system)

1: $A.setParameters(\overrightarrow{\theta}_i)$
2: $M = A.trainModel(Tr)$
3: $RMSE_i = M.testModel(Te)$

Output: a tuple $< \overrightarrow{\theta}_i, RMSE_i >$

Algorithm 21 Reduce

Input: list L of $< \overrightarrow{\theta}_i, RMSE_i >$ pairs, $|L| = n$

1: $RMSE^* = \infty$
2: **for** $i := 1, ..., n$ **do**
3: **if** $RMSE_i < RMSE^*$ **then**
4: $\overrightarrow{\theta}^* = \overrightarrow{\theta}_i$
5: $RMSE^* = RMSE_i$
6: **end if**
7: **end for**

Output: the optimal tuple $< \overrightarrow{\theta}^*, RMSE^* >$

The process of hyperparameter optimization starts with an optimization algorithm that defines which parameter combinations are calculated. This algorithm, e.g. one of the algorithms discussed in the

7 These functions are based on results of a team projects by our students: Renê Tatua Castillo and Tugce Habip.

previous section, creates a list of n parameter configurations $\overrightarrow{\theta}_1, ..., \overrightarrow{\theta}_n$. This list of configurations serves as input to a splitter in the Hadoop framework. The splitter divides the configurations among several mappers. A mapper takes a configuration $\overrightarrow{\theta}_i$ as input and invokes the map function described in Alg. 20.

In the map function the main part of the computation takes place. The mapper runs an experiment with the given configuration $\overrightarrow{\theta}_i$. The result of this computation is a quality or an error measure. Once the computation of the given experiment is completed, the mapper issues a <key,value> pair. In this case it is a $< \overrightarrow{\theta}_i, RMSE_i >$ pair. This process is repeated by several mappers until the results of all n configurations are known. This results in a set of $< \overrightarrow{\theta}_i, RMSE_i >$ pairs for all n configurations. They are input to a reducer that calls the function described in Alg. 21. The goal of the reducer is to aggregate all given <key,value> pairs into one final result. In our case the reducer returns the optimal parameter setting together with the corresponding RMSE value: $< \overrightarrow{\theta}^*, RMSE^* >$. Generally, the Hadoop framework supports usage of several reducers, however, in our application scenario it is sufficient to use only one of them. Once the reducer returns the optimal configuration $\overrightarrow{\theta}^*$ the distributed hyperparameter optimization is completed.

For the distribution of the optimization process the choice of the optimization algorithm is essential. This algorithm creates the list of n candidate configurations. However, the number n is different for each algorithm. Genetic algorithms, for instance, can create only n equal to the size of a population. If the desired total number of experiments, the so-called budget, is higher than this n, then several iterations of the algorithm are necessary.

Algorithm	Number of parallel experiments
Genetic algorithm	Size of a population
SMAC	1 (fully sequential)
PSO	Number of particles
Random Walk	Number of neighbours
Nelder-Mead	1 (fully sequential)
Sim. Annealing	Number of exp. for the equilibrium state
Random	Size of a budget (fully parallel)

Table 21.: Number of parallel experiments by different optimization algorithms

In Tab. 21, we present the maximal number n for different optimization algorithms. Algorithms, such as SMAC or Nelder-Mead are fully

sequential, i.e. for them $n = 1$. Only after calculation of this single experiment, the optimization algorithm can determine the next candidate configuration.

Random algorithms, on the other hand, are fully parallelizable. For those algorithms $n = E$ (budget). They can create an arbitrary number of candidate configurations in a single iteration. Combined with a low computational effort and a good performance in finding the optimal configuration, it makes the random algorithms the preferred choice for the HPO in recommender systems.

Part III

CONCLUSIONS AND FUTURE WORK

8

CONCLUSIONS

In this thesis we proposed selective learning for recommender systems. Since recommender systems suffer from the data sparsity problem, it is commonly believed that all available data should be used to train preference models. We, however, argue that it is beneficial to select the data used for training. We proposed three types of selective learning and formulated corresponding research questions that we address in the subsections below.

Furthermore, in this chapter we answer our core research question using the formalism proposed in Ch. 3 and draw final conclusions from the combined contributions of this thesis. Finally, we discuss limitations of the proposed approaches and directions for future research.

8.1 SELECTIVE FORGETTING

First of our selective learning approaches is *selective forgetting*. Selecting what to forget is complementary to selecting what to learn from. Our first research question is related to this approach:

RQ 1: Does forgetting improve the quality of recommendations?

To answer this one and further questions we proposed 11 forgetting strategies and three algorithms that enforce forgetting in matrix factorization. We also proposed a new evaluation protocol for stream-based evaluation that includes significance testing.

In our evaluation we used a state-of-the-art matrix factorization algorithm, BRISMF, and compared its performance with and without forgetting strategies. Our experimental results on eight real-world datasets with positive-only and rating feedback show that the same algorithm with forgetting methods performs significantly better than the same algorithm without forgetting.

This enables us to validate the following inequality (cf. Ch. 3 for notation and details):

$$\exists(\mathcal{M}, \mathcal{S}) : \sum_t Q_{S,t}(Te_t, M_t, S_t) > \sum_t Q_t(Te_t, M_t^*) \tag{80}$$

Our results show that this inequality is true, as there is a pair of model series and selection strategy series $(\mathcal{M}, \mathcal{S})$ that, in sum, outperforms the optimal model without a selection strategy over time. Therefore we answer the **RQ 1** affirmatively.

This answer has consequences for practitioners and researchers in the recommender systems domain. Improvement of prediction quality due to forgetting proves the existence of concept drift in typical RS applications. Forgetting techniques, for instance, provide a further way of adaptation to such changes over time. With those techniques it is also possible to the providers of recommendations to support users' privacy. Recommendation providers can give their users a possibility to decide what should be forgotten.

8.2 SELECTIVE NEIGHBOURHOOD

Our second approach to selective learning is *selective neighbourhood* computation. We proposed a method that builds a reliable neighbourhood, i.e. it selectively excludes non-reliable neighbours from a neighbourhood of the active user. We introduced the notion of a baseline user and a selection criterion based on the Hoeffding Bound, in order to answer the following research question:

RQ 2: **Does selective removal of users from a neighbourhood improve the quality of neighbourhood-based CF?**

To address **RQ 2**, we have studied CF methods and have enhanced them with our new selective approach. We then compared them with two other variants of CF algorithms, *shrinkage* and *significance weighting* that also attempt to decrease the impact of unreliable users in the neighbourhood.

Our results on four datasets show that our method with selective learning outperforms the remaining ones with respect to the quality of recommendations. Therefore, we have shown that the following inequality from Ch. 3 is true:

$$\exists(M,S) : Q_S(Te,M,S) > Q(Te,M^*) \tag{81}$$

as there is a pair of a model and of a selection strategy, which outperforms the best model that can be built without selection of neighbours. Therefore, we answer the **RQ 2** positively.

By answering this question we have shown that selective learning improves the predictive power not only in the streaming setting, but also in frequently used and simple algorithms, such as neighbourhood-based CF.

8.3 SEMI-SUPERVISED LEARNING

Our last type of selective learning is SSL. We proposed a stream-based framework for SSL in recommender systems. In this type of selective learning a recommender system exploits some of the abundantly available unlabelled information (user-item-pairs without ratings). A semi-

supervised system makes predictions for those unrated pairs in a selective way and uses those predictions for training.

We proposed two approaches to stream-based SSL in recommender systems:

- co-training
- self-learning

In the co-training approach there are several different learners that teach each other by providing their predictions as labels. Not all predictions are used for training.We proposed criteria for the selection of the pairs and of the predicted ratings.

In the self-learning approach a learner provides labels to itself. Also in this case the labels are selected using our criteria.

We conducted a thorough evaluation to answer the following research question:

RQ 3: **Does selective learning from predictions (semi-supervised learning) improve the quality of recommendations?**

We compared the results of a representative of MF algorithms, the BRISMF algorithm, with our selective SSL techniques and without them. Our results show that the co-training approach significantly outperforms the method without SSL in terms of quality of recommendations. From our two approaches, the co-training approach showed better results. Results of the self-training method were not consistent across datasets.

Those results allow us to validate the Ineq. 80 in the SSL setting. As there is a series of models and selection strategies that outperform the best model without any selection, the inequality is true. Therefore, we answer the **RQ 3** affirmatively.

This answer has impact on numerous applications of recommender systems that suffer from an extreme data sparsity. Our research shows that it is possible to selectively exploit the abundant unlabelled information to improve the quality of recommendations, however, at cost of computation time.

8.4 CORE RESEARCH QUESTION

Our core research question, as defined in Sec. 1.2, is:

Does selective learning improve the quality of predictions in recommender systems?

In Ch. 3 we described a formalism to determine the answer to this question. According to this formalism, which combines Ineq. 80 and Ineq. 81, the answer is positive if any of selective learning types outperforms the optimal non-selective model (cf. Ineq. 27). Given that this

is the case and all aforementioned research questions were answered positively, we conclude that selective learning *does improve the quality of predictions in recommender systems*. This answer applies, obviously, to carefully designed and tuned selection strategies, as described in this thesis and to any thinkable selective learning.

This conclusion has many consequences for numerous application scenarios. We show that selective learning can be combined with many different methods, which makes it relevant to practitioners and researchers. For practitioners it is not strictly necessary to implement new recommendation methods. It is possible to combine selective learning of various types with existing methods to potentially benefit from them.

The fact that selective learning improves predictive performance of recommender systems, shows the challenges of real world data. This data is often affected by concept drift or shift. User behaviour is often unpredictable, e.g. when account is shared with other parties, or items are purchased for other people. This causes even new data instances to be obsolete. Therefore, instances used for training should be selected carefully. Also when learning from other algorithms, as in the SSL setting, not all of the available training instances are trustworthy or useful. In this thesis we investigated those challenges and proposed selective learning methods that tackle them.

Lastly, next to answering the aforementioned research questions we conducted the first comparative study on hyperparameter optimization for recommender system. To further aid researchers and practitioners we additionally proposed a distributed HPO framework using Apache Hadoop.

8.5 LIMITATIONS

In our experiments with matrix factorization algorithms we used a representative of this class of algorithms, the BRISMF algorithm by Takács et al. [Tak+09]. As recommender systems are an active research field, there are numerous extensions of matrix factorization and several different versions of it. The number of different algorithms is so high that it is not feasible to use all of them in experiments. Therefore, we decided to use an algorithm that is representative for all of them. Despite the abundance of different MF algorithms, most of them internally work in the same way as BRISMF. To ensure generalizability and transferability of results it is meaningful to use this basic version of MF without any extensions.

Similarly to a high number of algorithms, there is also a high number of possible application domains in recommender systems. Also in this case it is not feasible to test all of them. Therefore, we focus on domains with available benchmark datasets. Those include movie

recommendation (Movielens, Flixster and Netflix datasets), music recommendation (Lastfm, Palco Principal) and general product opinions (Epinions dataset). Those datasets include positive-only and rating feedback. While it is not clear how our results and findings would transfer to other domains, the diversity and number of the datasets used in our evaluation together with strict evaluation protocols imply that our results are general.

Even though selection strategies can be combined with different types of methods, they cannot be understood as a preprocessing step. For a successful application of these strategies, it is necessary to embed them into a learning algorithm. Many of them cannot be decoupled from the learning algorithm, as e.g. forgetting strategy that is based on local sensitivity of latent features, excluding neighbours from a neighbourhood or measuring a reliability of an SSL prediction based on past error measured on a stream.

Selective learning from predictions, i.e. SSL, also has a drawback of highly increased computation time. This issue, however, will be addressed in our future work (cf. Sec. 8.6).

8.6 FUTURE WORK

During our work on selective learning in recommender systems we recognized further domains where selective learning can be applied. Our techniques can be transferred to e.g. conventional regression problems and other data mining and machine learning algorithms. While there are approaches in data mining that use sliding window for adaptation, it can be investigated if the selective forgetting of parts of a model leads to improved predictive power.

Another problem, which has been addressed in the conventional data mining already, but needs more investigation in RS, is change detection. In recommender systems this problem is particularly challenging, as a sudden change in user's preferences is difficult to predict or detect. Unlike in conventional data mining, in recommender systems the change manifests itself in form of only few data instances. A successful change detector could be coupled with our forgetting strategies, so that information from before the change is forgotten.

Our definition of selective learning could be also extended to capture active learning methods. These methods let the learner select what labels / ratings it would like to use for training. Those selected labels are then requested from users. Active learning for recommender systems is already an active research field with work e.g. by Karimi et al. [Kar+12; Kar+15], Eliahi et al. [ERR16], etc.

Our selective forgetting enables providers of recommender systems to let their users decide what should be forgotten. The application of forgetting methods as means of privacy preservation should be

researched more in the future work. A relevant topic in this context is, for instance, the possibility to allow users to select parts / aspects of a model to be forgotten instead of single items only.

A further topic worth investigating is combining different selective learning methods e.g. forgetting strategies with SSL, different forgetting strategies with each other or active learning with one of our selective learning methods. The interplay and synergies in such combinations are not clear and require more research.

In this work we have shown that not all data instances should be used for learning. This can have many possible reasons, such as concept drift and shift, or several persons sharing one account. Especially the latter is an unresolved challenge in recommender systems. In such a case, a user profile is a compound of preferences of several persons. Ideally, a recommender system should recognize this phenomenon and create sub-profiles. The learning should then take place selectively with ratings corresponding to a sub-profile. As the detection of such sub-profiles is challenging and, probably, not possible in a clear way, probabilistic or fuzzy methods are promising in this setting.

Also within our selective learning methods there is potential for future work. Our framework is designed in a modular way, so that extending it by implementing more components is easily possible. One of the components worth extending in the future are dataset splitters in the SSL framework. The extension could encompass the capability to apply views onto the stream instances adaptively, so that no retraining is necessary. Especially for our SSL methods, reducing the computation time also remains a challenge. One of possible solutions can exploit distributed computing of models in the co-training approach.

Part IV

APPENDIX

A

CORRECTING THE USAGE OF THE HOEFFDING BOUND

Our work on selective neighbourhood (cf. Ch. 5) relies on the application of the Hoeffding Bound. To build selective neighbourhoods in Ch. 5 we use findings presented in this appendix. For instance, the test for significance of two users' similarities and the width of the applied bound (cf. Sec. 5.2.2.1) is based on results presented here.

The Hoeffding Inequality is often used to derive theoretical bounds. In particular, we have observed that the assumptions, on which the Hoeffding Inequality builds, are often not satisfied in stream mining. This leads to invalid bounds. In this chapter we investigate the impact of the problem and propose a correction. The following sections of the appendix come (with modifications) from our paper on this topic [MKS13].

A.1 RELATED WORK ON HOEFFDING BOUND

After the seminal work of Domingos and Hulten on a Very Fast Decision Tree for stream classification [DH00], several decision tree stream classifiers have been proposed, including CVFDT [HSD01a], Hoeffding Option Tree [PHK07], CFDTu [Xu+11], etc. All of them apply the Hoeffding Bound [Hoe63] to decide whether a tree node should be split and how. We show that the bounds derived from the Hoeffding Inequality do not hold due to not fulfilled prerequisites.

Concerns on the reliability of stream classifiers using the Hoeffding Bound have been raised in [PHK07]: Pfahringer et al. point out that "Despite this guarantee, decisions are still subject to limited lookahead and stability issues." In Section A.5, we show that the instability detected in [PHK07] is quantifiable.

Rutkowski et al. [Rut+12] state that the Hoeffding Inequality [Hoe63] is too restrictive, since it only operates on numerical variables and since it demands an input that can be expressed as a sum of the independent variables, which is not the case for Information Gain and Gini Index. They recommend McDiarmid's Inequality instead, and design a McDiarmid's Bound for Information Gain and Gini Index [Rut+12]. However, as we explain in Section A.2, the violation of the Hoeffding Inequality in stream classification concerns the independence of observations. This violation of the Inequality's assumptions is not peculiar to the Hoeffding Inequality. It would apply even if the McDiarmid's Bound would be used in this setting without further changes. Hence,

we rather replace the split criterion with one that satisfies the prerequisites.

We propose `correctedVFDT`, which uses the Hoeffding Bound with a new split criterion that satisfies the prerequisites. Thus, `correctedVFDT` provides the expected performance guarantees. We stress that our aim is not to propose a more accurate method, but one, whose theoretical bounds are *reliable* and can be interpreted.

A.2 HOEFFDING BOUND - PREREQUISITES AND PITFALLS

The Hoeffding Inequality proposed by Wassily Hoeffding [Hoe63] states that for a random variable Z with range R, the true average of Z, $\overline{Z}$, deviates from the observed average $\widehat{Z}$ not more than ε, subject to an error-likelihood δ:

$$|\overline{Z} - \widehat{Z}| < \varepsilon \text{ , where } \varepsilon = \sqrt{\frac{R^2 \cdot \ln(1/\delta)}{2n}} \tag{82}$$

where n is the number of instances. Inequality 82 has the following PREREQUISITES:

1. The random variables must be identically distributed and almost surely bounded; the variable ranges are used when computing the bound.
2. Random observations of the variables must be independent of each other.

In many stream classification algorithms, Z is the value returned by the function computing the 'goodness' of a split attribute. Given a significance level δ, the Hoeffding Inequality states whether the instances n seen thus far are enough for choosing the best split attribute. This is essential, since wrong splits (especially for nodes close to the root) affect the performance of the classifier negatively. In the presence of drift, this may also lead to uninformed decisions about discarding or replacing a subtree. We show that stream classification methods violate the prerequisites of the Hoeffding Inequality (subsection A.2.1) and that the decision bound is wrongly set (A.2.2).

A.2.1 *Violation of Prerequisites*

Domingos and Hulten [DH00] proposed Information Gain (IG) and Gini Index (GI) as exemplary split functions appropriate for the Hoeffding Bound: at each time point, the data instances in the tree node to be split are considered as the observations input to the Hoeffding Inequality, and the values computed upon them by IG/GI are assumed to be averages.

VIOLATION 1: The Hoeffding Inequality applies to *arithmetical* averages only [Hoe63]. IG and GI are clearly not arithmetical averages.

VIOLATION 2: The variables, i.e. the observations used for the computation of the split criterion must be independent (PREREQ. 2). However, consider a sliding window of length 4 and assume the window contents $w_1 = [x_1, x_2, x_3, x_4]$ and then $w_2 = [x_3, x_4, x_5, x_6]$, after the window has moved by two positions. Obviously, the window contents overlap. When a function like IG computes a value over the contents of each window, it considers some instances more than once. Thus, the computed values are not independent.

A.2.2 *Insufficient Separation between Attributes*

Domingos and Hulten specify that the Hoeffding Bound should be applied as follows. X_a is the best attribute and X_b be the second-best attribute with respect to a split function $\overline{G}$ after seeing n examples. $\Delta\overline{G} = \overline{G}(X_a) - \overline{G}(X_b) \geqslant 0$ is the difference between their observed heuristic values. The Hoeffding Bound guarantees with probability $1-\delta$ that X_a is indeed the best attribute if $\Delta\overline{G} > \epsilon$. [1] [DHoo]

Claim 1. *The Hoeffding Bound does* not *provide the guarantee expected in [DHoo].*

Proof. Assume that the split candidates are X, Y with IG values G_X and G_Y, observed averages $\widehat{G_X}, \widehat{G_Y}$ and real averages $\overline{G_X}, \overline{G_Y}$ (cf. Figure 28). Considering n observations in range R (of the split test), the probability that the real average $\overline{Z}$ deviates from the observed one $\widehat{Z}$ by more than ε is bounded by Ineq. 82 [Hoe63]:

$$Pr(\widehat{Z} - \overline{Z} \geqslant \varepsilon) \leqslant exp(\frac{-2n\varepsilon^2}{R^2}) \tag{83}$$

In Figure 28, we see that $\widehat{G_Y}$ is greater then $\widehat{G_X}$ by more than ε, but this does not hold for the real averages $\overline{G_Y}$ and $\overline{G_X}$. Hence, a span of one ε is not sufficient to guarantee separation of the gain values. □

This claim holds also when we consider $G_X - G_Y$ as a single random variable ΔG (as done in [DHoo]): the range of ΔG is the sum of ranges of G_X and G_Y, again requiring a change of the decision bound. We give the correct bound in A.3.1.

A.3 NEW METHOD FOR CORRECT USAGE OF THE HOEFFDING BOUND

Our new core `correctedVFDT` encompasses a correction on the decision bound, and a new split function that satisfies the prerequisites of [Hoe63] (cf. section A.2).

1 Note: we use ε instead of ϵ, $\overline{Z}$ for the true average and $\widehat{Z}$ for the observed one.

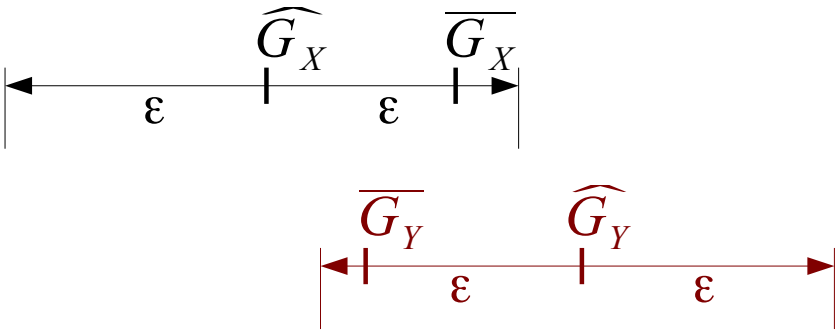

Figure 28.: Observed vs. real averages of two random variables: the observed averages differ by more than ε, but the Hoeffding Bound does not guarantee that G_Y is superior (figure from [MKS13]).

A.3.1 *Specifying a Correct Decision Bound*

Domingos and Hulten define $\Delta G = G_Y - G_X$ as a random variable with range $R = log(c)$ (for Information Gain IG, c is the number of classes) and check whether $\widehat{\Delta G} - \overline{\Delta G}$ exceeds ε [DHoo], where ε is a positive number. However, this definition of ΔG assumes that it is already non-negative, i.e. there exists some non-negative constant k, so that $|G_Y - G_X| \geqslant k$ holds.

Assume that there exists a $k > 0$ so that $E(|G_Y - G_X|) \geqslant k$, where we denote the true average of Z as $E(Z)$ instead of $\overline{Z}$ for better readability. The absolute value is a convex function and $|G_Y - G_X|$ does not follow a degenerate distribution, so Jensen's inequality [JZoo] holds in its strict form, i.e.:

$$E(|G_Y - G_X|) > |E(G_Y - G_X)| \equiv |E(G_Y) - E(G_X)| \tag{84}$$

So, we cannot conclude that $|\overline{G_Y} - \overline{G_X}| \geqslant k$, i.e. even if the true average of $|G_Y - G_X|$ exceeds some positive value, we cannot say that Y is superior to X.

We must thus perform *two* tests with the Hoeffding Inequality, (1) for $\Delta G_1 := G_Y - G_X$ under the assumption that $\Delta G_1 \geqslant 0$, *and* (2) for $-\Delta G_1 := G_X - G_Y$, assuming that $\Delta G_1 < 0$. Equivalently, we can perform a single *modified test* on a variable $\Delta G := G_Y - G_X$ that ranges over $[-\log c; +\log c]$, i.e. it may take negative values. Consequently, the new range of the variable ΔG that we denote as R' is twice as high as the original range R. To apply the Hoeffding Inequality on such a variable, we must reset the decision bound to:

$$\varepsilon' = \sqrt{\frac{R'^2 \cdot \ln(1/\delta)}{2n}} = \sqrt{4\frac{R^2 \cdot \ln(1/\delta)}{2n}} = 2 \cdot \sqrt{\frac{R^2 \cdot \ln(1/\delta)}{2n}} \tag{85}$$

i.e. to twice the bound dictated by Ineq. 82. Then, the correctness of the split decision is guaranteed given δ. Alternatively, we can keep the

original decision bound and adjust the error-likelihood to δ^4. Further, a larger number of instances is required to take a split decision. We study both effects in Section A.5.

A.3.2 *Specifying a HB-compatible Split Function*

Functions such as information gain cannot be used in combination with the Hoeffding Inequality, because they are not arithmetic averages [Rut+12]. We propose a split function that is an arithmetic average and satisfies the two prerequisites of the Hoeffding Bound (cf. Section A.2). We call it HB-compatible.

In the new split function we perform the computation of the expected quality of a node split on each element of the node independently. We propose *Quality Gain*, which we define as the improvement on predicting the target variable at a given node v in comparison to its parent $Parent(v)$, i.e.

$$QGain(v) = Q(v) - Q(Parent(v)) \tag{86}$$

where the quality function $Q()$ is the normalized sum:

$$Q(v) = \frac{1}{|v|} \sum_{o \in v} oq(o) \tag{87}$$

and $oq()$ is a function that can be computed for each instance o in v. Two possible implementations of $oq()$ are: $isCorrect()$ (Eq. 88), whereas $Q()$ corresponds to the conventional accuracy, and $lossReduction()$ (Eq. 89) that can capture the cost of misclassification in skewed distributions:

$$isCorrect(o) = \begin{cases} 1, \text{ if o is classified correctly} \\ 0, \text{ is misclassified} \end{cases} \tag{88}$$

$$lossReduction(o) = 1 - misclassificationCost(o) \tag{89}$$

We use $isCorrect()$ to implement $oq()$ hereafter, and term the so implemented $QGain()$ function as *AccuracyGain*. However, the validation in the next Section holds for all implementations of $oq()$. In the research regarding split measures the misclassification error has been indicated as a weaker metric than e.g. information gain [Has+09]. Our goal is, however, not to propose a metric that yields higher accuracy of a model, but one that can be used together with the Hoeffding Bound without violating its prerequisites and thus allowing for interpretation of the performance guarantees given by this bound. In Section A.5.2 we show that this metric is competitive to information gain in terms of accuracy and it reveals further positive features important for a streaming scenario.

A.4 VALIDATION

We first show that our new split function satisfies the prerequisites of the Hoeffding Inequality. Next, we show that no correction for multiple testing is needed.

A.4.1 *Satisfying the Assumptions of the Hoeffding Bound*

Quality Gain, as defined in Eq. 86 using a quality function as in Eq. 87, satisfies the PREREQUISITES of the Hoeffding Inequality. PREREQ 1 (cf. Section A.2) says that the random variable has to be almost surely bounded. The implementations of $oq()$ in Eq. 88, range in $[0, 1]$ and the same holds for the quality function $Q()$ in Eq. 87 by definition. Hence PREREQ 1 is satisfied.

PREREQ 2 (cf. Section A.2) demands independent observations. In stream mining, the arriving data instances are always assumed to be independent observations of an unknown distribution. However, as we have shown in subsection A.2.1, when Information Gain is computed over a sliding window, the content overlap and the combination of the instances for the computation of entropy lead to a violation of PREREQ 2. In contrast, our *Quality Gain* considers only one instance at each time point for the computation of $Q()$ and builds the arithmetical average incrementally, without considering past instances. This ensures that the instances are statistically independent from each other. The *Quality Gain* metric uses those independent instances to compute the goodness of a split. The result of this computation depends, however, on the performance of the classifier. Since, we consider a single node in a decision tree, the classifier and the entire path to the given node remains constant during the computation of the Hoeffding Bound. Consequently, all instances that fall into that node are conditionally independent given the classifier. This conditional independence of instances given the classifier allows us to use the Hoeffding Bound upon our split function.

A.4.2 *Is a Correction for Multiple Testing Necessary?*

As explained in subsection A.3.1, the split decision of `correctedVFDT` requires two tests on the same data sample: we compute ε for the best and second-best attributes. Since the likelihood of witnessing a rare event increases as the number of tests increases, it is possible that the α-errors (errors of first type) accumulate. To verify whether a correction for multiple tests (e.g. Bonferroni correction) is necessary, we consider the different possible areas of value combinations separately. The areas, enumerated as I-IV, are depicted in Figure 29.

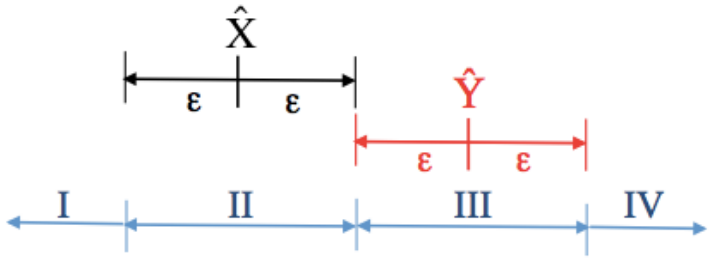

Figure 29.: When stating that Y is superior to X with confidence $1-\delta$, the error likelihood is δ; error and non-error areas are represented by numbers I - IV (figure from [MKS13]).

Figure 29 depicts a situation where the Hoeffding Bounds of attributes X and Y are separable, and allow us to state with confidence $1-\delta$ that Y is superior to X. There is a chance of δ that this statement is wrong. We distinguish three cases for variable X (and equivalently for Y):

CASE (1): the true average $\overline{X}$ is indeed in the ε-vicinity of $\widehat{X}$: $\widehat{X}-\varepsilon \leqslant \overline{X} \leqslant \widehat{X}+\varepsilon$ (area represented by II in Figure 29)

CASE (2): $\overline{X}$ is left to the ε-vicinity of $\widehat{X}$: $\overline{X} < \widehat{X}-\varepsilon$ (area I)

CASE (3): $\overline{X}$ is right to the ε-vicinity of $\widehat{X}$: $\overline{X} > \widehat{X}+\varepsilon$ (areas III and IV)

According to the Hoeffding Inequality, the likelihood of the Case (1) is $1-\delta$; we denote this case as `normal` or (n). We assume that the likelihood of error δ is distributed symmetrically around the ε-vicinity of $\widehat{X}$, hence the likelihood of Case (2) and of Case (3) is equal to $\delta/2$. In Case (2), the real average $\overline{X}$ is at the left of the ε-vicinity, hence the split decision would be the same as in Case (1). Therefore, we mark Case (2) as `not_harmful` (nh). In contrast, Case (3) for variable X may lead to a different split decision, because we would incorrectly assume that $\overline{X}$ is higher than it truly is. This is represented by areas III and IV in Figure 29. We mark Case (3) as `harmful` (h).

In Figure 30 we show all possible combinations of cases and their likelihoods. This tree depicts the likelihood of the outcome of each combination; the middle level corresponds to the first test, the leaf-level contains the outcomes after the first and the second test. For instance, the left node on the middle level denotes the `not_harmful` (nh) error of the first test. At its right we see the `normal` case (n) with likelihood $1-\delta$. The leaf nodes of the tree represent the likelihood of outcomes after performing two tests: green nodes correspond to the `not_harmful` outcomes (n), (nh); red ones are potentially `harmful` (h); the blue ones contain both `harmful` and `not_harmful` outcomes.

Even if we consider all blue solid nodes as `harmful`, the sum of the likelihoods of `harmful` outcomes (cf. Eq. 90) is still smaller than δ,

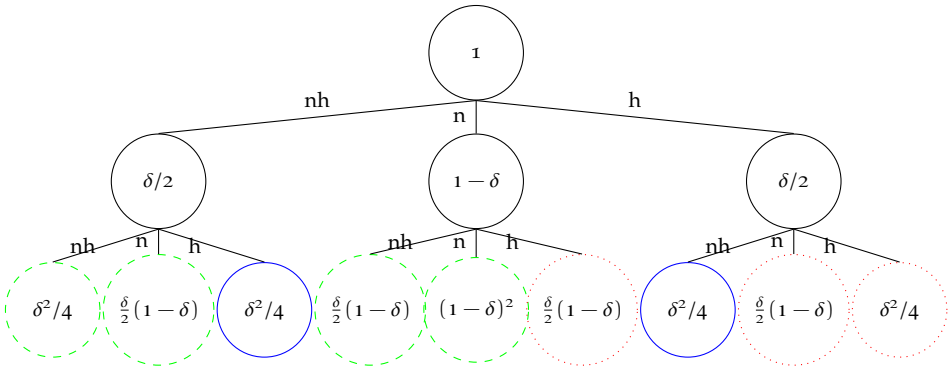

Figure 30.: Likelihood of all possible test outcomes. The middle level of the tree stands for outcomes of the first test. The leafs correspond to likelihood of outcomes after performing two tests. Green dashed leafs stand for `no error` (n) or `not_harmful` error (nh). Red dotted ones denote `harmful` error (h). Blue solid leafs combine `harmful` and `not_harmful` errors, so they have no label (figure from [MKS13]).

hence a correction for multiple tests (e.g. Bonferroni correction) is not necessary.

$$\frac{\delta^2}{4} + \frac{\delta}{2}(1-\delta) + \frac{\delta^2}{4} + \frac{\delta}{2}(1-\delta) + \frac{\delta^2}{4} = \delta - \frac{\delta^2}{4} \tag{90}$$

A.5 EXPERIMENTS

We evaluate our `correctedVFDT` with $oq()$ implemented as $isCorrect()$ (Eq. 88), i.e. with *AccuracyGain* as our split function (cf. end of Section 5.2). We measure the impact of the modifications to VFDT [DH00] on classifier performance.

To quantify the impact of the incorrect use of the Hoeffdingin Inequality we use two indicators: the number of *Incorrect Decisions* and the average number of instances (*Average n*) considered before taking a split decision. For this experiment, a dataset with known ground truth is necessary. The artificial dataset and the experiment are described in A.5.1.

When experimenting on real data, we quantify the performance of the stream classifier as *Avg. Accuracy* and the tree size as *Avg. # Nodes*. For this experiment, presented in subsection A.5.2, we use the Adult dataset from the UCI repository[FA10].

A.5.1 *Experimenting under Controlled Conditions*

For the experiment under controlled conditions, we generate a dataset with a discrete multivariate distribution, as described in Table 22. The dataset has two attributes: A_1 with three discrete values in $\{A, B, C\}$, and A_2 with two discrete values in $\{D, E\}$. The target variable takes values from $\{c_1, c_2\}$.

A_2		D		E	
A_1	A	$c_1 : 0.0675$	$c_2 : 0.1575$	$c_1 : 0.0675$	$c_2 : 0.1575$
	B	$c_1 : 0.1350$	$c_2 : 0.0900$	$c_1 : 0.1575$	$c_2 : 0.0675$
	C	$c_1 : 0.0450$	$c_2 : 0.0050$	$c_1 : 0.0450$	$c_2 : 0.0050$

Table 22.: Joint probability distribution of the synthetic dataset (table from [MKS13]).

In this experiment, we simulate a decision tree node and observe what split decision is taken in it. Since the distribution of the dataset is known , the attribute that each split function should choose at each moment is known. As we show in Table 23, we consider VFDT with IG - denoted as 'InfoGain' (cf. first two rows of Table 23 below the legend) for a decision bound of 1ε and 2ε, and we compare with our `correctedVFDT` with *Accuracy Gain* - denoted as 'AccuracyGain' (cf. last two rows of Table 23), again for 1ε and 2ε. This means that we consider both the erroneous decision bound 1ε and the corrected invocation of the Inequality with 2ε (cf. A.3.1) for both VFDT and `correctedVFDT`.

In Table 23 we show the results, aggregated over 100,000 runs. In the second column, we count the 'Incorrect Decisions' over a total of 100,000 decisions. The third column 'Average n' counts the number of instances seen before deciding to split a node. The confidence level of the Hoeffding Inequality was set to $1 - \delta = 0.95$, hence only 5,000 (5%) incorrect split decisions are theoretically permitted. We run a binomial test to check whether the difference between the observed error and the expected one is significant (column before last) and return the computed p-value (last column of Table 23).

The original VFDT (1^{st} row below legend in Table 23) exceeds the theoretical threshold of 5000 Incorrect Decisions by far. The corrected invocation of the Hoeffding Inequality (subsection A.3.1) reduces the number of incorrect decisions by 92.497% (cf. 2^{nd} row in Table 23), but at the cost of increasing the number of instances required to make a split from 117.31 to 1671.53. This means that the learner would wait approximatively 10 times longer to take a split decision and would

Setup	Incorrect Decisions	Average n	Alternative Hypothesis	p-value
InfoGain, 1ε	25738	117.31	P(incorrect decision) > 0.05	$< 2.2e-16$
InfoGain, 2ε	1931	1671.53	P(incorrect decision) < 0.05	$< 2.2e-16$
AccuracyGain, 1ε	3612	17.68	P(incorrect decision) < 0.05	$< 2.2e-16$
AccuracyGain, 2ε	22	37.45	P(incorrect decision) < 0.05	$< 2.2e-16$

Table 23.: Results of 100 000 repetitions of decision process on a split attribute at a node in a decision tree. We compare VFDT with 'InfoGain' to `correctedVFDT` with 'AccuracyGain' for the incorrect invocation of the Hoeffding Inequality (decision bound 1ε) and for the correct invocation (decision bound 2ε). For the performance indicators 'Incorrect Decisions' and 'Average n' lower values are better. The last column shows the results of the significance test on the deviation of the measured error from the theoretically permitted one, depicted in the 'Alternative Hypothesis' column, where the error-likelihood δ of the Hoeffding Bound is set to 0.05 (table from [MKS13]).

abstain from possibly good split decisions. In contrast, `correctedVFDT` makes less incorrect decisions and decides sooner, as seen in the last two rows of Table 23. The 3^{rd} row shows that even with the incorrect decision bound, `correctedVFDT` makes less incorrect decisions than the theoretic threshold. Best results are achieved for the correct decision bound of course (4^{th} row): only 22 of the total 100,000 decisions are wrong, corresponding to an improvement of 99.915 %. At the same time, our method for 2ε needs only 2.24% of the instances needed by VFDT, i.e. `correctedVFDT` converges much sooner than VFDT.

To ensure that these results are statistically significant we present the results of the binomial tests. The alternative hypothesis in the 4^{th} column in Table 23 differs from row to row. In the first row, the alternative hypothesis says that the number of incorrect decisions will be higher than the theoretic bound (by the Hoeffding Inequality); the p-value in the last column states that the alternative hypothesis should be accepted already at a confidence level lower then $2.2e-16$. Hence, the theoretical bound *is clearly* violated by the original VFDT. The alternative hypothesis in the other three rows states that the number of incorrect decisions will stay within bound; this hypothesis is accepted.

In Table 24, we compare VFDT to `correctedVFDT` at a confidence level $1-\delta$ =99%. The results are similar to Table 23, except for the `correctedVFDT` with incorrect decision bound: the theoretic bound is violated (significantly, see last column), i.e. even a good method will ultimatively fail if the Hoeffding Inequality is used erroneously: both the corrected decision bound and a HB-compatible split function are necessary for good performance (see last row).

Setup	Incorrect Decisions	Average n	Alternative Hypothesis	p-value
InfoGain, 1ε	14034	347.55	P(incorrect decision) > 0.01	$< 2.2e-16$
InfoGain, 2ε	339	2872.24	P(incorrect decision) < 0.01	$< 2.2e-16$
AccuracyGain, 1ε	1062	22.42	P(incorrect decision) < 0.01	0.9757
			P(incorrect decision) > 0.01	0.02617
AccuracyGain, 2ε	2	49.7	P(incorrect decision) < 0.01	$< 2.2e-16$

Table 24.: Results analogous to those in Table 23, but with a confidence level of 0.99 (table from [MKS13]).

A.5.2 *Experiments on a Real Dataset*

We have shown that the `correctedVFDT` with *Accuracy Gain* and correct decision bound (2ε) leads to an essential reduction of incorrect split decisions and that the decisions are taken much sooner. We now investigate how these new components influence the classification performance and size of created models on a real dataset. We use the dataset "Adults" from the UCI repository [FA10].

Stream mining algorithms are sensitive to the order of data instances and to concept drift. To minimize the effect of concept drift in the dataset, we created 10 permutations of it and repeated our tests on each permutation, setting the grace period of each run to 1. Therefore, the results presented in this section are averages over ten runs. This also increases the stability of the measures and lowers the effect of random anomalies.

For the two algorithms, we used the parameter settings that lead to best performance. For VFDT, these were $1-\delta = 0.97$ and decision bound $\varepsilon = 0.05$, i.e. the use of the Hoeffding Inequality is incorrect. According to subsection A.3.1, the true confidence is therefore much lower. For `correctedVFDT`, the correct decision bound 2ε was used, the confidence level was set to $1-\delta = 0.6$. The second column of Table 25 shows the average accuracy over the 10 runs, the third columns shows the average number of nodes of the models built in all runs.

Algorithm	Avg. Accuracy	Avg. # Nodes
VFDT	81,992	863.7
`correctedVFDT`	80,784	547.2

Table 25.: Performance of VFDT and `correctedVFDT` of it on the "Adult dataset". The columns "Avg. Accuracy" and "Avg. # Nodes" denote the accuracy and the number of nodes of the decision trees, as averaged over ten runs (table from [MKS13]).

According to the results in Table 25, VFDT reached a high accuracy, but it also created very large models with 863.7 nodes on average. That high amount of nodes not only consumes a lot of memory, but it also requires much computation time to create such models. Furthermore, such extensive models often tend to overfit the data distribution. In the second row of the table we see that `correctedVFDT` maintained almost the same accuracy, but needed only 63.36% of the nodes that were created by VFDT.

Our `correctedVFDT` does not only have the advantage of lower computation costs regarding time and memory usage, but also a split confidence that is interpretable. As we have shown in the previous subsection, the Hoeffding Bound of the VFDT cannot be trusted, for it does not bound the error the way it is expected. Consequently, setting the split confidence to 0.97 does not mean that the split decisions are correct with this level of confidence. In contrast to that, our method does not violate the requirements for using the Hoeffding Bound and thus, we can rely on the split decisions with the confidence that we have set.

For this particular amount of data and concept contained in this dataset (approximatively) optimal results have been achieved using the confidence of 0.6. This is much lower than 0.97 used with the VFDT, but this is only an illusory disproportion. In fact, the confidence guaranteed by the VFDT was much lower due to the violations of the requirements of the Hoeffding bound and it is probably not possible to estimate it. Usage of our method allows to interpret the results. We can see that it is necessary to give up the high confidence to achieve the best result on a so small dataset.

A.6 CONCLUSIONS ON THE USAGE OF THE HOEFFDING BOUND

We have shown that the prerequisites for the use of the Hoeffding Inequality in stream classification are not satisfied by the VFDT algorithm [DHoo] and its successors. In a controlled experiment, we have demonstrated that the prerequisite violations do have an impact in classifier performance.

To alleviate this problem, we have first shown that the Hoeffding Inequality must be applied differently, to cater for an input that may take negative values. We have adjusted the decision bound accordingly. We have further specified a family of split functions that satisfies the Inequality's prerequisites and incorporated them into our `correctedVFDT`.

Our experiments on synthetic data show that `correctedVFDT` achieves significantly more correct split decisions and needs less instances to make a decision than the original VFDT. Our experiments on real data show that `correctedVFDT` produces smaller models, converges

faster and maintains a similar level of accuracy. More importantly, the split decision of `correctedVFDT` are reliable given the predefined confidence level, while those of the original VFDT are not guaranteed by the Hoeffding Inequality.

Those findings allowed us to design our selective neighbourhood method in Ch. 5 in a way that satisfies the prerequisites of the Hoeffding Bound. The design decisions influenced by these findings encompass specifying a sufficient ε-distance between two mean similarities and applying a HB-compatible similarity measure that can be represented as a mean difference in independent observations (ratings). This allowed us to design a theoretically correct method for selecting neighbours (cf. Ch. 5) with a reliable confidence setting in the Hoeffding Bound.

BIBLIOGRAPHY

[AL09] Belarmino Adenso-Díaz and Manuel Laguna. "Fine-Tuning of Algorithms Using Fractional Experimental Designs and Local Search." In: *OR* (Sept. 7, 2009).

[AMK11] Gediminas Adomavicius, Nikos Manouselis, and YoungOk Kwon. "Multi-Criteria Recommender Systems." In: *Recommender Systems Handbook*. Ed. by Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. Springer, 2011, pp. 769–803. ISBN: 978-0-387-85819-7.

[Agg16] Charu C. Aggarwal. *Recommender Systems - The Textbook*. Springer, 2016. Chap. 2, pp. 1–498. ISBN: 978-3-319-29659-3.

[AST09] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. "A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms." In: LNCS. Oct. 1, 2009.

[Bab+02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. "Models and Issues in Data Stream Systems." In: *PODS*. Ed. by Lucian Popa, Serge Abiteboul, and Phokion G. Kolaitis. ACM, 2002, pp. 1–16. ISBN: 1-58113-507-6.

[Bee+16] Jöran Beel, Bela Gipp, Stefan Langer, and Corinna Breitinger. "Research-paper recommender systems: a literature survey." In: *Int. J. on Digital Libraries* 17.4 (2016), pp. 305–338.

[BKV07] Robert Bell, Yehuda Koren, and Chris Volinsky. "Modeling relationships at multiple scales to improve accuracy of large recommender systems." In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2007.

[Ber+11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. "Algorithms for Hyper-Parameter Optimization." In: *NIPS*. 2011.

[BB12] James Bergstra and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization." In: *JMLR* (2012).

[BDO95] M.W. Berry, S.T. Dumais, and G.W. O'Brien. "Using linear algebra for intelligent information retrieval." In: *SIAM review* (1995), pp. 573–595.

[Bob+13] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. "Recommender systems survey." In: *Knowl.-Based Syst.* 46 (2013), pp. 109–132.

[Bos+14] Zoran Bosnić, Jaka Demšar, Grega Kešpret, Pedro Pereira Rodrigues, João Gama, and Igor Kononenko. "Enhancing data stream predictions with reliability estimators and explanation." In: *Engineering Applications of Artificial Intelligence* 34 (2014), pp. 178–192. ISSN: 0952-1976.

[Bre01] Leo Breiman. "Random Forests." In: *Machine Learning* 45 (2001), pp. 5–32.

[Bur02] Robin Burke. "Hybrid Recommender Systems: Survey and Experiments." In: *User Modeling and User-Adapted Interaction* 12.4 (2002), pp. 331–370.

[Bur07] Robin Burke. "Hybrid Web Recommender Systems." In: *The Adaptive Web*. Ed. by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. Vol. 4321. Lecture Notes in Computer Science. Springer, 2007. ISBN: 978-3-540-72078-2.

[CDC14] Pedro G. Campos, Fernando Díez, and Iván Cantador. "Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols." In: *User Modeling and User-Adapted Interaction* 24.1 (2014), pp. 67–119. ISSN: 1573-1391.

[Cel10] O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

[Cha+11] Badrish Chandramouli, Justin J. Levandoski, Ahmed Eldawy, and Mohamed F. Mokbel. "StreamRec: a real-time recommender system." In: *SIGMOD Conference*. Ed. by Timos K. Sellis, Renée J. Miller, Anastasios Kementsietsidis, and Yannis Velegrakis. ACM, 2011, pp. 1243–1246. ISBN: 978-1-4503-0661-4.

[Cha+17] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A. Hasegawa-Johnson, and Thomas S. Huang. "Streaming Recommender Systems." In: *ACM International World Wide Web Conference*. WWW. 2017.

[Che+09] Jilin Chen, Werner Geyer, Casey Dugan, Michael Muller, and Ido Guy. "Make new friends, but keep the old: recommending people on social networking sites." In: *Proceedings of the 27th international conference on Human factors in computing systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 201–210. ISBN: 978-1-60558-246-7. DOI: 10.1145/1518701.1518735.

[Che+12] Rung-Ching Chen, Yun-Hou Huang, Cho-Tsan Bau, and Shyi-Ming Chen. "A recommendation system based on domain ontology and SWRL for anti-diabetic drugs selection." In: *Expert Systems with Applications* 39.4 (2012), pp. 3995–4006. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2011.09.061.

[Chr+05] C. Christakou, L. Lefakis, S. Vrettos, and A. Stafylopatis. "A Movie Recommender System Based on Semi-supervised Clustering." In: *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*. Vol. 2. Nov. 2005, pp. 897–903.

[COL13] Freddy Chong Tat Chua, Richard Jayadi Oentaryo, and Ee-Peng Lim. "Modeling Temporal Adoptions Using Dynamic Matrix Factorization." In: *ICDM*. Ed. by Hui Xiong, George Karypis, Bhavani M. Thuraisingham, Diane J. Cook, and Xindong Wu. IEEE Computer Society, 2013, pp. 91–100. ISBN: 978-0-7695-5108-1.

[CKT10] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. "Performance of Recommender Algorithms on Top-n Recommendation Tasks." In: *Proceedings of ACM RecSys*. RecSys '10. ACM, 2010, pp. 39–46. ISBN: 978-1-60558-906-0. DOI: 10.1145/1864708.1864721.

[DK11] Christian Desrosiers and George Karypis. "A Comprehensive Survey of Neighborhood-based Recommendation Methods." In: *Recommender Systems Handbook*. Ed. by Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. Springer US, 2011, pp. 107–144. ISBN: 978-0-387-85819-7.

[Dia+08] M. Benjamin Dias, Dominique Locher, Ming Li, Wael El-Deredy, and Paulo J. G. Lisboa. "The value of personalised recommender systems to e-business: a case study." In: *RecSys*. Ed. by Pearl Pu, Derek G. Bridge, Bamshad Mobasher, and Francesco Ricci. ACM, Oct. 27, 2008, pp. 291–294. ISBN: 978-1-60558-093-7.

[DL06] Yi Ding and Xue Li. "Time weight collaborative filtering." In: *CIKM*. Ed. by Otthein Herzog, Hans-Jörg Schek, Norbert Fuhr, Abdur Chowdhury, and Wilfried Teiken. ACM, Feb. 10, 2006, pp. 485–492. ISBN: 1-59593-140-6.

[DH00] P. Domingos and G. Hulten. "Mining High Speed Data Streams." In: *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 2000.

[DH01] Pedro Domingos and Geoff Hulten. "Catching up with the Data: Research Issues in Mining Data Streams." In: *DMKD*. 2001.

[DCP14] Karl B. Dyer, Robert Capo, and Robi Polikar. "COMPOSE: A Semisupervised Learning Framework for Initially Labeled Nonstationary Streaming Data." In: *IEEE Trans. Neural Netw. Learning Syst.* 25.1 (2014), pp. 12–26.

[EK95] R C Eberhart and J Kennedy. "A new optimizer using particle swarm theory." In: *International Symposium on Micro Machine and Human Science* (1995).

[ERR16] Mehdi Elahi, Francesco Ricci, and Neil Rubens. "A survey of active learning in collaborative filtering recommender systems." In: *Computer Science Review* 20 (2016), pp. 29–50.

[FA10] A. Frank and A. Asuncion. *UCI Machine Learning Repository*. 2010.

[FO16] Evgeny Frolov and Ivan Oseledets. "Tensor Methods and Recommender Systems." In: *CoRR* abs/1603.06038 (2016).

[Gam12] João Gama. "A Survey on Learning from Data Streams: Current and Future Trends." In: *Progress in Artificial Intelligence* 1.1 (2012), pp. 45–55. ISSN: 2192-6352. DOI: 10.1007/s13748-011-0002-6.

[GSR09] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. "Issues in evaluation of stream learning algorithms." In: *KDD*. 2009.

[Gam+14] João Gama, Indre Zliobaite, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. "A survey on concept drift adaptation." In: *ACM Comput. Surv.* 46.4 (2014), 44:1–44:37.

[Gem+15] Marco de Gemmis, Pasquale Lops, Cataldo Musto, Fedelucio Narducci, and Giovanni Semeraro. "Semantics-Aware Content-Based Recommender Systems." In: *Recommender Systems Handbook*. Ed. by Francesco Ricci, Lior Rokach, and Bracha Shapira. Boston, MA: Springer US, 2015, pp. 119–159. ISBN: 978-1-4899-7637-6. DOI: 10.1007/978-1-4899-7637-6_4.

[GP10] Mustansar Ali Ghazanfar and Adam Prügel-Bennett. "A Scalable, Accurate Hybrid Recommender System." In: *WKDD*. IEEE Computer Society, 2010, pp. 94–98. ISBN: 978-0-7695-3923-2.

[Gol+92] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. "Using Collaborative Filtering to Weave an Information Tapestry." In: *Commun. ACM* (1992).

[GH16] Carlos A. Gomez-Uribe and Neil Hunt. "The Netflix Recommender System: Algorithms, Business Value, and Innovation." In: *ACM Trans. Management Inf. Syst.* 6.4 (2016), p. 13.

[HH12] Yaroslav O. Halchenko and Michael Hanke. "Open is Not Enough. Let's Take the Next Step: An Integrated, Community-Driven Computing Platform for Neuroscience." In: *Front. Neuroinform.* (2012).

[HK16] F. Maxwell Harper and Joseph A. Konstan. "The MovieLens Datasets: History and Context." In: *TiiS* 5.4 (2016), p. 19.

[Has+09] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and Ebooks Corporation. *The Elements of Statistical Learning*. Dordrecht: Springer, 2009. Chap. 9.2.3, pp. 324–329. ISBN: 9780387848587 0387848584.

[Her+99] Jonathan L. Herlocker, Joseph A. Konstan, Al Borchers, and John Riedl. "An algorithmic framework for performing collaborative filtering." In: *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. New York, NY, USA: ACM, 1999, pp. 230–237.

[Her+13] Antonio Hernando, Jesús Bobadilla, Fernando Ortega, and Jorge Tejedor. "Incorporating reliability measurements into the predictions of a recommender system." In: *Information Sciences* 218 (2013), pp. 1–16. ISSN: 0020-0255.

[HT12] Balázs Hidasi and Domonkos Tikk. "Fast ALS-based tensor factorization for context-aware recommendation from implicit feedback." In: *CoRR* abs/1204.1259 (2012).

[Hoe63] Wassily Hoeffding. "Probability inequalities for sums of bounded random variables." In: *J. Amer. Statist. Assoc.* 58 (1963), pp. 13–30. ISSN: 0162-1459.

[Hol93] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1993.

[Hsu+06] William H. Hsu, Andrew L. King, Martin S. R. Paradesi, Tejaswi Pydimarri, and Tim Weninger. "Collaborative and Structural Recommendation of Friends using Weblog-based Social Network Analysis." In: *Computational Approaches to Analyzing Weblogs - Papers from the 2006 Spring Symposium* (2006). AAAI Press Technical Report SS-06-03. Stanford, USA, March 2006., pp. 24–31.

[HKV08] Yifan Hu, Yehuda Koren, and Chris Volinsky. "Collaborative Filtering for Implicit Feedback Datasets." In: *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*. 2008, pp. 263–272. DOI: 10.1109/ICDM.2008.22.

[HSD01a] G. Hulten, L. Spencer, and P. Domingos. "Mining Time-Changing Data Streams." In: *ACM SIGKDD* (2001).

[HSD01b] Geoff Hulten, Laurie Spencer, and Pedro Domingos. "Mining time-changing data streams." In: *KDD*. Ed. by Doheon Lee, Mario Schkolnick, Foster J. Provost, and Ramakrishnan Srikant. ACM, 2001, pp. 97–106. ISBN: 1-58113-391-X.

[HHL11] Frank Hutter, H. H. Hoos, and K. Leyton-Brown. "Sequential Model-Based Optimization for General Algorithm Configuration." In: *LION*. LNCS. 2011.

[Hut+14] Frank Hutter, Thomas Stützle, Kevin Leyton-Brown, and Holger H. Hoos. "ParamILS: An Automatic Algorithm Configuration Framework." In: *CoRR* (2014).

[JSW98] Donald R. Jones, Matthias Schonlau, and William J. Welch. "Efficient Global Optimization of Expensive Black-Box Functions." In: *J. Global Optimization* (1998).

[Kar+10] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. "Multiverse Recommendation: N-dimensional Tensor Factorization for Context-aware Collaborative Filtering." In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. Barcelona, Spain: ACM, 2010, pp. 79–86. ISBN: 978-1-60558-906-0.

[Kar+11] Rasoul Karimi, Christoph Freudenthaler, Alexandros Nanopoulos, and Lars Schmidt-Thieme. "Towards Optimal Active Learning for Matrix Factorization in Recommender Systems." In: *ICTAI*. IEEE, 2011, pp. 1069–1076. ISBN: 978-1-4577-2068-0.

[Kar+12] Rasoul Karimi, Christoph Freudenthaler, Alexandros Nanopoulos, and Lars Schmidt-Thieme. "Exploiting the characteristics of matrix factorization for active learning in recommender systems." In: *RecSys*. Ed. by Padraig Cunningham, Neil J. Hurley, Ido Guy, and Sarabjot Singh Anand. ACM, 2012, pp. 317–320. ISBN: 978-1-4503-1270-7.

[Kar+15] Rasoul Karimi, Christoph Freudenthaler, Alexandros Nanopoulos, and Lars Schmidt-Thieme. "Comparing Prediction Models for Active Learning in Recommender Systems." In: *LWA*. Ed. by Ralph Bergmann, Sebastian Görg, and Gilbert Müller. Vol. 1458. CEUR Workshop Proceedings. CEUR-WS.org, 2015, pp. 171–180.

[KJV83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. "Optimization by Simulated Annealing." In: *Science* (1983).

[KBV09] Y. Koren, R. Bell, and C. Volinsky. "Matrix Factorization Techniques for Recommender Systems." In: *Computer* 42.8 (Aug. 2009), pp. 30–37. ISSN: 0018-9162. DOI: 10.1109/MC.2009.263.

[Kor09] Yehuda Koren. "Collaborative filtering with temporal dynamics." In: *KDD*. Paris, France, 2009. ISBN: 978-1-60558-495-9. DOI: 10.1145/1557019.1557072.

[Koy00] I. Koychev. "Gradual Forgetting for Adaptation to Concept Drift." In: *ECAI 2000 Workshop on Current Issues in Spatio-Temporal Reasoning, Berlin, Germany*. 2000, pp. 101–106.

[LMV00] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. "A Multilinear Singular Value Decomposition." In: *SIAM J. Matrix Anal. Appl.* 21.4 (2000), pp. 1253–1278. ISSN: 0895-4798. DOI: 10.1137/S0895479896305696.

[LH14] Dokyun Lee and Kartik Hosanagar. "Impact of Recommender Systems on Sales Volume and Diversity." In: *ICIS*. Ed. by Michael D. Myers and Detmar W. Straub. Association for Information Systems, 2014. ISBN: 978-0-615-15788-7.

[LBD07] Xue Li, Jorge M. Barajas, and Yi Ding. "Collaborative filtering on streaming data with interest-drifting." In: *Intell. Data Anal.* 11.1 (2007), pp. 75–87.

[Lin14] Zhijie Lin. "An empirical investigation of user and system recommendations in e-commerce." In: *Decision Support Systems* 68 (2014), pp. 111–124.

[LLL11] Lindawati, Hoong Chuin Lau, and David Lo. "Instance-Based Parameter Tuning via Search Trajectory Similarity Clustering." In: *LION*. 2011.

[Liu+10] Nathan Nan Liu, Min Zhao, Evan Wei Xiang, and Qiang Yang. "Online evolutionary collaborative filtering." In: *Proc. of the ACM RecSys*. 2010.

[LW15] Andreas Lommatzsch and Sebastian Werner. "Optimizing and Evaluating Stream-Based News Recommendation Algorithms." In: *CLEF*. Ed. by Josiane Mothe, Jacques Savoy, Jaap Kamps, Karen Pinel-Sauvagnat, Gareth J. F. Jones, Eric SanJuan, Linda Cappellato, and Nicola Ferro. Vol. 9283. Lecture Notes in Computer Science. Springer, 2015, pp. 376–388. ISBN: 978-3-319-24026-8.

[LGS11] Pasquale Lops, Marco de Gemmis, and Giovanni Semeraro. "Content-based Recommender Systems: State of the Art and Trends." In: *Recommender Systems Handbook*. Ed. by Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. Springer, 2011, pp. 73–105. ISBN: 978-0-387-85819-7.

[Lü+12] Linyuan Lü, Matus Medo, Chi Ho Yeung, Yi-Cheng Zhang, Zi-Ke Zhang, and Tao Zhou. "Recommender Systems." In: *CoRR* abs/1202.1112 (2012).

[MKL07] Hao Ma, Irwin King, and Michael R. Lyu. "Effective missing data prediction for collaborative filtering." In: *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. SIGIR '07. 2007. DOI: 10.1145/1277741.1277751.

[Man+11] Nikos Manouselis, Hendrik Drachsler, Riina Vuorikari, Hans Hummel, and Rob Koper. "Recommender Systems in Technology Enhanced Learning." In: *Recommender Systems Handbook*. Ed. by Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. Boston, MA: Springer US, 2011, pp. 387–415. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_12.

[MA06] Paolo Massa and Paolo Avesani. "Trust-aware bootstrapping of recommender systems." In: *ECAI Workshop on Recommender Systems*. Citeseer. 2006, pp. 29–33.

[Mat+16] Pawel Matuszyk, Rene Tatua Castillo, Daniel Kottke, and Myra Spiliopoulou. "A Comparative Study on Hyperparameter Optimization for Recommender Systems." In: *Workshop on Recommender Systems and Big Data Analytics (RSBDA'16) @ iKNOW 2016*. Ed. by Elisabeth Lex, Roman Kern, Alexander Felfernig, Kris Jack, Dominik Kowald, and Emanuel Lacic. 2016.

[MKS13] Pawel Matuszyk, Georg Krempl, and Myra Spiliopoulou. "Correcting the Usage of the Hoeffding Inequality in Stream Mining." In: *IDA*. Ed. by Allan Tucker, Frank Höppner, Arno Siebes, and Stephen Swift. Vol. 8207. Lecture Notes in Computer Science. Springer, 2013, pp. 298–309. ISBN: 978-3-642-41397-1.

[MS14a] Pawel Matuszyk and Myra Spiliopoulou. "Hoeffding-CF: Neighbourhood-Based Recommendations on Reliably Similar Users." English. In: *User Modeling, Adaptation, and Personalization*. Ed. by Vania Dimitrova, Tsvi Kuflik, David Chin, Francesco Ricci, Peter Dolog, and Geert-Jan Houben. Vol. 8538. Lecture Notes in Computer Science. Springer

International Publishing, 2014, pp. 146–157. ISBN: 978-3-319-08785-6. DOI: 10.1007/978-3-319-08786-3_13.

[MS14b] Pawel Matuszyk and Myra Spiliopoulou. "Selective Forgetting for Incremental Matrix Factorization in Recommender Systems." In: *Discovery Science*. Vol. 8777. LNCS. Springer International Publishing, 2014, pp. 204–215.

[MS15] Pawel Matuszyk and Myra Spiliopoulou. "Semi-supervised Learning for Stream Recommender Systems." In: *Discovery Science*. Ed. by Nathalie Japkowicz and Stan Matwin. Vol. 9356. LNCS. Springer International Publishing, 2015, pp. 131–145. ISBN: 978-3-319-24281-1.

[MS17] Pawel Matuszyk and Myra Spiliopoulou. "Stream-based semi-supervised learning for recommender systems." In: *Machine Learning* (2017), pp. 1–28. ISSN: 1573-0565.

[Mat+15] Pawel Matuszyk, João Vinagre, Myra Spiliopoulou, Alípio Mário Jorge, and João Gama. "Forgetting Methods for Incremental Matrix Factorization in Recommender Systems." In: *Proceedings of the ACM SAC*. SAC '15. Salamanca, Spain: ACM, 2015, pp. 947–953.

[Mat+17] Pawel Matuszyk, João Vinagre, Myra Spiliopoulou, Alípio Mário Jorge, and João Gama. "Forgetting Methods for Incremental Matrix Factorization in Recommender Systems." In: *(under review)*. 2017.

[McN47] Quinn McNemar. "Note on the Sampling Error of the Difference between Correlated Proportions or Percentages." In: *Psychometrika* 12.2 (1947), pp. 153–157.

[Mei13] Christoph Meinel. *'Selbstplagiat' und gute wissenschaftliche Praxis*. 2013. URL: http://www.uni-regensburg.de/universitaet/ombudspersonen/medien/selbstplagiat-memo.pdf (visited on 12/22/2016).

[MJ08] C. Miranda and A.M. Jorge. "Incremental Collaborative Filtering for Binary Ratings." In: *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT '08. IEEE/WIC/ACM International Conference on*. Vol. 1. Dec. 2008, pp. 389–392. DOI: 10.1109/WIIAT.2008.263.

[MJ09] Catarina Miranda and AlípioMário Jorge. "Item-Based and User-Based Incremental Collaborative Filtering for Web Recommendations." In: *Progress in Artificial Intelligence*. Ed. by Luís Seabra Lopes, Nuno Lau, Pedro Mariano, and LuísM. Rocha. Vol. 5816. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 673–684. ISBN: 978-3-642-04685-8.

[NGL11] Jeffrey Naruchitparames, Mehmet Hadi Gunes, and Sushil J. Louis. "Friend recommendations in social networks using genetic algorithms and network topology." In: *IEEE Congress on Evolutionary Computation*. IEEE, 2011, pp. 2207–2214. ISBN: 978-1-4244-7834-7.

[Nas+07] Olfa Nasraoui, Jeff Cerwinske, Carlos Rojas, and Fabio A. González. "Performance of Recommendation Systems in Dynamic Streaming Environments." In: *SDM*. SIAM, Sept. 6, 2007.

[Nas+03] Olfa Nasraoui, Cesar Cardona Uribe, Carlos Rojas Coronel, and Fabio A. González. "TECNO-STREAMS: Tracking Evolving Clusters in Noisy Data Streams with a Scalable Immune System Learning Model." In: *Proceedings of the IEEE ICDM 2003*. 2003, pp. 235–242.

[NM65] J. A. Nelder and R. Mead. "A simplex method for function minimization." In: *Computer Journal* 7 (1965).

[Pap+05] Manos Papagelis, Ioannis Rousidis, Dimitris Plexousakis, and Elias Theoharopoulos. "Incremental Collaborative Filtering for Highly-Scalable Recommendation Algorithms." In: *Proceedings of the 15th International Symposium on Methodologies of Intelligent Systems (ISMIS'05*. 2005.

[PNH15] D. Paraschakis, B. J. Nilsson, and J. Holländer. "Comparative Evaluation of Top-N Recommenders in e-Commerce: An Industrial Perspective." In: *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. Dec. 2015, pp. 1024–1031. DOI: 10.1109/ICMLA.2015.183.

[Par+12] Deuk Hee Park, Hyea Kyeong Kim, Il Young Choi, and Jae Kyeong Kim. "A literature review and classification of recommender systems research." In: *Expert Syst. Appl.* 39.11 (2012), pp. 10059–10072.

[Pat07] Arkadiusz Paterek. "Improving regularized singular value decomposition for collaborative filtering." In: *Proc. KDD Cup Workshop at SIGKDD'07*. 2007.

[PB07] Michael J. Pazzani and Daniel Billsus. "Content-Based Recommendation Systems." In: *The Adaptive Web*. Ed. by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. Vol. 4321. Lecture Notes in Computer Science. Berlin / Heidelberg: Springer, 2007, pp. 325–341.

[PHK07] Bernhard Pfahringer, Geoffrey Holmes, and Richard Kirkby. "New Options for Hoeffding Trees." In: *Australian Conference on Artificial Intelligence*. 2007, pp. 90–99.

[Plu+11] Till Plumbaum, Andreas Lommatzsch, Ernesto William De Luca, and Sahin Albayrak. "SERUM: Collecting Semantic User Behavior for Improved News Recommendations." In: *UMAP Workshops*. Ed. by Liliana Ardissono and Tsvi Kuflik. Vol. 7138. Lecture Notes in Computer Science. Springer, 2011, pp. 402–405. ISBN: 978-3-642-28508-0.

[PMS10] Christine Preisach, Leandro Balby Marinho, and Lars Schmidt-Thieme. "Semi-supervised Tag Recommendation - Using Untagged Resources to Mitigate Cold-Start Problems." In: *Advances in Knowledge Discovery and Data Mining. 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part I*. Ed. by Mohammed Javeed Zaki, Jeffrey Xu Yu, B. Ravindran, and Vikram Pudi. Vol. 6118. Lecture Notes in Computer Science. Springer, 2010, pp. 348–357. ISBN: 978-3-642-13656-6. DOI: 10.1007/978-3-642-13657-3_38.

[RGS14] Giuseppe Ricci, Marco de Gemmis, and Giovanni Semeraro. "Mathematical Methods of Tensor Factorization Applied to Recommender Systems." In: *New Trends in Databases and Information Systems: 17th East European Conference on Advances in Databases and Information Systems*. Ed. by Barbara Catania, Tania Cerquitelli, Silvia Chiusano, Giovanna Guerrini, Mirko Kämpf, Alfons Kemper, Boris Novikov, Themis Palpanas, Jaroslav Pokorný, and Athena Vakali. Cham: Springer International Publishing, 2014, pp. 383–388. ISBN: 978-3-319-01863-8. DOI: 10.1007/978-3-319-01863-8_40.

[RGB08] Pedro Pereira Rodrigues, João Gama, and Zoran Bosnic. "Online Reliability Estimates for Individual Predictions in Data Streams." In: *ICDM Workshops*. IEEE Computer Society, 2008, pp. 36–45.

[RHS05] Chuck Rosenberg, Martial Hebert, and Henry Schneiderman. "Semi-Supervised Self-Training of Object Detection Models." In: *WACV/MOTION*. IEEE Computer Society, Feb. 9, 2005, pp. 29–36. ISBN: 0-7695-2271-8.

[Rut+12] Leszek Rutkowski, Lena Pietruczuk, Piotr Duda, and Maciej Jaworski. "Decision Trees for Mining Data Streams Based on the McDiarmid's Bound." In: *IEEE Trans. on Knowledge and Data Engineering* (2012). accepted in 2012.

[Sar+00] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John T. Riedl. "Application of Dimensionality Reduction in Recommender System - A Case Study." In: *In ACM WEBKDD Workshop*. 2000.

[Sar+01] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. "Item-based collaborative filtering recommendation algorithms." In: WWW '01. Hong Kong, Hong Kong, 2001. ISBN: 1-58113-348-0. DOI: 10.1145/371920.372071.

[SKR99] J. Schafer, J. Konstan, and J. Reidl. "Recommender Systems in E-Commerce." In: *Proceedings of ACM Conference on Electronic Commerce*. Denver, Colorado, USA, Nov. 1999.

[SVR09] Shilad Sen, Jesse Vig, and John Riedl. "Tagommenders: connecting users to items through tags." In: *WWW*. Ed. by Juan Quemada, Gonzalo León, Yoëlle S. Maarek, and Wolfgang Nejdl. ACM, 2009, pp. 671–680. ISBN: 978-1-60558-487-4.

[Sha95] J P Shaffer. "Multiple Hypothesis Testing." In: *Annual Review of Psychology* 46.1 (1995), pp. 561–584.

[SG11] Guy Shani and Asela Gunawardana. "Evaluating Recommendation Systems." In: *Recommender Systems Handbook*. Ed. by Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. 2011.

[SNB05] V. Sindhwani, P. Niyogi, and M. Belkin. "A Co–Regularized Approach to Semi–supervised Learning with Multiple Views." In: *Proceedings of the ICML Workshop on Learning with Multiple Views*. 2005.

[SLA12] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. "Practical Bayesian Optimization of Machine Learning Algorithms." In: *NIPS*. 2012.

[Sou+15] Vinícius M. A. de Souza, Diego Furtado Silva, João Gama, and Gustavo E. A. P. A. Batista. "Data Stream Classification Guided by Clustering on Nonstationary Environments and Extreme Verification Latency." In: *SDM*. Ed. by Suresh Venkatasubramanian and Jieping Ye. SIAM, 2015, pp. 873–881. ISBN: 978-1-61197-401-0.

[SPV14] John Z. Sun, Dhruv Parthasarathy, and Kush R. Varshney. "Collaborative Kalman Filtering for Dynamic Matrix Factorization." In: *IEEE Trans. Signal Processing* 62.14 (2014), pp. 3499–3509.

[SNM10] P. Symeonidis, A. Nanopoulos, and Y. Manolopoulos. "A Unified Framework for Providing Recommendations in Social Tagging Systems Based on Ternary Semantic Analysis." In: *IEEE Transactions on Knowledge and Data Engineering* 22.2 (Feb. 2010), pp. 179–192. ISSN: 1041-4347.

[SNM08] Panagiotis Symeonidis, Alexandros Nanopoulos, and Yannis Manolopoulos. "Tag recommendations based on tensor dimensionality reduction." In: *RecSys '08: Proceedings of the 2008 ACM conference on Recommender systems*. Lausanne, Switzerland: ACM, 2008, pp. 43–50. ISBN: 978-1-60558-093-7. DOI: 10.1145/1454008.1454017.

[JZ00] "Fourier, Laplace, and Mellin Transforms." In: *Table of Integrals, Series, and Products (Sixth Edition)*. Ed. by Alan Jeffrey and Daniel Zwillinger. Sixth Edition. San Diego: Academic Press, 2000, pp. 1099–1125. ISBN: 978-0-12-294757-5. DOI: 10.1016/B978-012294757-5/50021-0.

[Tak+09] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. "Scalable Collaborative Filtering Approaches for Large Recommender Systems." In: *J. Mach. Learn. Res.* 10 (2009). ISSN: 1532-4435.

[VJ12] João Vinagre and Alípio Mário Jorge. "Forgetting mechanisms for scalable collaborative filtering." In: *Journal of the Brazilian Computer Society* 18.4 (2012), pp. 271–282. ISSN: 0104-6500.

[VJG14] João Vinagre, Alípio Mário Jorge, and João Gama. "Fast Incremental Matrix Factorization for Recommendation with Positive-Only Feedback." In: *UMAP*. 2014, pp. 459–470.

[VJG15a] João Vinagre, Alípio Mário Jorge, and João Gama. "An overview on the exploitation of time in collaborative filtering." In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 5.5 (2015), pp. 195–215. ISSN: 1942-4795.

[VJG15b] João Vinagre, Alípio Mário Jorge, and João Gama. "Evaluation of recommender systems in streaming environments." In: *CoRR* abs/1504.08175 (2015).

[WP14] Wiesner Martin and Pfeifer Daniel. "Health Recommender Systems: Concepts, Requirements, Technical Basics and Challenges." In: *International Journal of Environmental Research and Public Health* 11.3 (Feb. 2014), pp. 2580–2607. ISSN: 1661-7827 1660-4601.

[WLZ15] Dianshuang Wu, Jie Lu, and Guangquan Zhang. "A fuzzy tree matching-based personalized e-learning recommender system." In: *IEEE Transactions on Fuzzy Systems* 23.6 (2015), pp. 2412–2426.

[Xio+10] Liang Xiong, Xi Chen, Tzu-Kuo Huang, Jeff Schneider, and Jaime G Carbonell. "Temporal collaborative filtering with bayesian probabilistic tensor factorization." In: *Pro-*

ceedings of the 2010 SIAM International Conference on Data Mining. SIAM. 2010, pp. 211–222.

[Xu+11] Wenhua Xu, Zheng Qin, Hao Hu, and Nan Zhao. "Mining Uncertain Data Streams Using Clustering Feature Decision Trees." In: *ADMA (2)*. Ed. by Jie Tang, Irwin King, Ling Chen, and Jianyong Wang. Vol. 7121. Lecture Notes in Computer Science. Springer, 2011, pp. 195–208. ISBN: 978-3-642-25855-8.

[Yin+12] Hongzhi Yin, Bin Cui, Jing Li, Junjie Yao, and Chen Chen. "Challenging the Long Tail Recommendation." In: *PVLDB* 5.9 (2012), pp. 896–907.

[Zha+14] Mi Zhang, Jie Tang, Xuchen Zhang, and Xiangyang Xue. "Addressing cold start in recommender systems: a semi-supervised co-training algorithm." In: *SIGIR*. ACM, 2014.

[Zhe+16] Xiaolin Zheng, Weifeng Ding, Zhen Lin, and Chaochao Chen. "Topic tensor factorization for recommender system." In: *Inf. Sci.* 372 (2016), pp. 276–293.

[ZL07] Zhi-Hua Zhou and Ming Li. "Semisupervised Regression with Cotraining-Style Algorithms." In: *IEEE Transactions on Knowledge and Data Engineering* 19.11 (2007). ISSN: 1041-4347.

[ZZQ07] Zhi-Hua Zhou, De-Chuan Zhan, and Yang Qiang. "Semi-Supervised Learning with Very Few Labeled Training Examples." In: *AAAI*. AAAI Press, 2007, pp. 675–680. ISBN: 978-1-57735-323-2.

[Zhu+10] T. Zhu, B. Hu, J. Yan, and X. Li. "Semi-Supervised Learning for Personalized Web Recommender System." In: *Computing and Informatics* 29.4 (2010), pp. 617–627.

[Zhu05] Xiaojin Zhu. *Semi-supervised learning literature survey*. Tech. rep. 1530. Computer Sciences, University of Wisconsin-Madison, 2005.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Final Version as of September 14, 2017 (`classicthesis` version 4.1).

EHRENERKLÄRUNG

Ich versichere hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; verwendete fremde und eigene Quellen sind als solche kenntlich gemacht. Insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.
Ich habe insbesondere nicht wissentlich:

- Ergebnisse erfunden oder widersprüchliche Ergebnisse verschwiegen,
- statistische Verfahren absichtlich missbraucht, um Daten in ungerechtfertigter Weise zu interpretieren,
- fremde Ergebnisse oder Veröffentlichungen plagiiert,
- fremde Forschungsergebnisse verzerrt wiedergegeben.

Mir ist bekannt, dass Verstöße gegen das Urheberrecht Unterlassungs- und Schadensersatzansprüche des Urhebers sowie eine strafrechtliche Ahndung durch die Strafverfolgungsbehörden begründen kann. Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Magdeburg, 13.09.2017

Pawel Matuszyk