Mach Learn (2017) 106:771–798
DOI 10.1007/s10994-016-5614-4

Stream-based semi-supervised learning for recommender systems

Pawel Matuszyk[1] · **Myra Spiliopoulou**[1]

Received: 28 April 2016 / Accepted: 18 November 2016 / Published online: 2 February 2017

Abstract To alleviate the problem of data sparsity inherent to recommender systems, we propose a semi-supervised framework for stream-based recommendations. Our framework uses abundant unlabelled information to improve the quality of recommendations. We extend a state-of-the-art matrix factorization algorithm by the ability to add new dimensions to the matrix at runtime and implement two approaches to semi-supervised learning: co-training and self-learning. We introduce a new evaluation protocol including statistical testing and parameter optimization. We then evaluate our framework on five real-world datasets in a stream setting. On all of the datasets our method achieves statistically significant improvements in the quality of recommendations.

Keywords Recommender systems · Semi-supervised learning · Matrix factorization · Collaborative filtering · Stream mining

1 Introduction

Recommender systems learn users' preferences and recommend to them a small selection of only relevant items. To train preference models recommender systems use users' feedback on relevance of items. A big class of recommender system algorithms, the collaborative filtering algorithms, use ratings as users' feedback (e.g. five stars for high relevance and one star for irrelevant items). However, only a small set of items has ratings. Ratings are the counter-part of labels in the context of machine learning. Consequently, a great majority of items is not labelled by users. This results in an extreme data sparsity. Typically, sparsity of a user-item-

Editors: Nathalie Japkowicz and Stan Matwin.

✉ Pawel Matuszyk
pawel.matuszyk@iti.cs.uni-magdeburg.de

Myra Spiliopoulou
myra@iti.cs.uni-magdeburg.de

[1] Otto-von-Guericke-University Magdeburg, Universitätsplatz 2, 39106 Magdeburg, Germany

rating matrix in recommender systems reaches 99% (only 1% is rated). Especially, new users and new items suffer from the feedback sparsity.

Therefore, it is challenging to provide accurate recommendations, given that only around 1% of possible label information is available. To tackle this problem we propose a semi-supervised recommender system that not only uses the labelled information to train preference models, but also employs the abundant unlabelled information. This framework is relevant to all existing stream-based collaborative filtering algorithms (including matrix factorization) as a method of tackling the sparsity problem.

In our method we follow the co-training-approach. According to this approach, there are multiple learners running in parallel. A reliable prediction of one of those learners is used as a label by the remaining learners (for details cf. Sect. 3). To prove the improvements achieved by semi-supervised learning (SSL) we compare our method to an analogous algorithm without SSL. For a further comparison we also implemented another approach to semi-supervised learning—the self-learning approach. In this approach a single learner provides reliable predictions to itself and uses them as labels. However, we show on five real-world datasets that the improvements achieved by self-learning are not as substantial as in the case of co-training.

Our method works incrementally on a stream of ratings and incorporates new user feedback into its preference models. This has an essential advantage of adaptivity to changes of preferences and to new information. Furthermore, because our framework works on stream of ratings, a running recommender system can take advantage of SSL immediately, as a streaming-based method learns continuously upon arrival of new information. However, the streaming-based approach poses additional challenges for our method. Computation in real time is one of them. Therefore, in our evaluation we also focus on measuring the computation time.

A further challenge is the incremental update of preference models as the stream goes on. To tackle this challenge we use an incremental version of the BRISMF algorithm (Takács et al. 2009). We extend it so that is able to extend the dimensions of the user-item-matrix incrementally as new users or items appear in the stream and call it hereafter extBRISMF. It is a state-of-the-art matrix factorization (MF) algorithm. While there are numerous different MF algorithms, most of them are based on the same core idea as the BRISMF algorithm. To make our findings as generalizable as possible, we chose to extend BRISMF as a representative to many other algorithms.

In an empirical study on five real-world datasets we show that our SSL framework significantly improves the quality of recommendations. To prove that the improvement is not due to chance, we introduce a novel evaluation protocol for stream-based recommender systems that includes statistical tests and a correction for multiple testing.

Contributions. To summarize, our contributions are as follows:

- a novel framework with semi-supervised learning (SSL) for stream-based recommender systems with two approaches
  - co-training
  - self-learning
- a new evaluation protocol with significance testing for stream-based recommender systems

This work is an extended issue of our conference paper (Matuszyk and Spiliopoulou 2015). Compared to the conference paper we added the self-learning approach, several components of the framework and a new evaluation protocol. All experiments were re-run using the

new evaluation that also includes statistical testing and analysis of impact of framework components onto the predictive performance of a recommender system.

Organization. This paper is structured as follows. In the next section we discuss related work and point out the differences from our approach to existing work. In Sect. 3 we present our SSL framework. Instances of the framework components are presented in Sect. 4. In Sect. 5 we describe our evaluation protocol including statistical testing and parameter optimization. Experimental results are presented in Sect. 6. Finally, we conclude our work in Sect. 7 and discuss remaining open issues.

2 Related work

Recommender systems are an active research topic that started with first approaches to collaborative filtering with work by Goldberg et al. (1992). Already then, the authors recognized the need for filtering the relevant information from a plethora of available information and applied their method onto text documents in a collaborative filtering method called Tapestry.

Sarwar et al. (2001) introduced an item-based variant of collaborative filtering (CF). Since then, the neighbourhood-based CF has been researched intensively in numerous publications (Desrosiers and Karypis 2011; Matuszyk and Spiliopoulou 2014a; Su and Khoshgoftaar 2009; Deshpande and Karypis 2004; Linden et al. 2003). All those approaches exploit the neighbourhood relations between items or users and are, therefore, called neighbourhood-based CF. They are based on the assumption that users, who rated items similarly in the past, will also rate future items similarly.

Nowadays, a different class of collaborative filtering algorithms is considered state-of-the-art. Those are matrix factorization algorithms (MF). The first application of MF in recommender systems goes back to year 2002 by Sarwar et al. (2002). However, only around year 2009 MF algorithms became popular, after they proved their superior performance and flexibility in the Netflix competition, for which, especially, work by Koren was relevant (Koren 2008, 2009; Koren et al. 2009).

In our work we use the BRISMF algorithm (Biased Regularized Incremental Simultaneous Matrix Factorization) by Takács et al. (2009). BRISMF and many other MF algorithms decompose the original user-item-rating matrix R into two other matrices P and Q. The decomposition is achieved using stochastic gradient descent (SGD). While R is typically extremely sparse, the latent matrices P and Q are complete. Predicting missing values in the matrix R is performed by applying the following formula $\widehat{R} = P \cdot Q$. Since the matrix $\widehat{R}$ is a product of two complete matrices, $\widehat{R}$ is complete as well. While this is the core idea of MF algorithms in recommender systems, the BRISMF algorithm uses a more sophisticated decomposition that also includes regularization and biases (cf. Takács et al. 2009 for details and more background information on matrix factorization).

The BRISMF algorithm has been proposed in two versions. One of them is batch-based i.e. it uses a batch of training data at once and creates a static model. Second version (cf. Algorithm 2 in Takács et al. 2009) is an incremental algorithm, i.e. it works on a stream of ratings and updates its preference model as new ratings appear in the stream. This incremental version of BRISMF, however, is not capable of handling new users and items i.e. new dimensions in the matrix. Since recommender systems are applied in volatile domains, handling new dimensions is an important feature. Thus, we extend BRISMF, so that it is capable to extend dimensions of the matrix (cf. Sect. 4.1).

Incremental setting for recommender systems poses new challenges for evaluation. Since MF algorithms usually start with an initial training phase, splitting of a dataset into training and test set is not trivial. For this purpose we use our splitting method from Matuszyk et al. (2015) and Matuszyk and Spiliopoulou (2014b) (cf. Sect. 5.2). According to this method, the main mode of an algorithm is the streaming mode. In our previous work we used the prequential evaluation, as proposed by Gama et al. (2009). However, prequential evaluation suffers from problems with statistical testing. Since, one data instance (rating) in a stream is used for both, training and testing, applying a statistical test onto a evaluation measure is not possible due to violation of independence of observations. Therefore, we introduce a novel evaluation protocol that doesn't use prequential evaluation and allows for hypothesis tests (cf. Sect. 5.4).

Our main contribution is a framework for stream-based semi-supervised learning (SSL) for recommender systems. SSL has been investigated thoroughly in conventional data mining and machine learning (Zhou et al. 2007), also in the stream setting (Dyer et al. 2014; de Souza et al. 2015). A comprehensive survey of those techniques can be found in Zhu (2005). Those techniques encompass both co-training (Sindhwani et al. 2005) and self-learning techniques (Rosenberg et al. 2005). Semi-supervised approaches for regression problems also have been proposed (Zhou and Li 2007). However, the problem in recommender systems is inherently different from the conventional classification or regression. In recommender systems an entire matrix of real or binary 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. (2005) proposed a model-based recommender system using the k-means algorithm with SSL. 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. (2013) 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. Rodrigues et al. (2008) and Bosnić et al. (2014) 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. Sect. 4.5).

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 (Zhang et al. 2014). 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. (2010) proposed a graph-based tag recommender system that employs untagged items. 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. (2010) proposed a recommender system for web pages that uses conventional classification with self-learning on natural language data. Also this method is not applicable to the general collaborative filtering scenario in recommender systems.

To summarize, there is no semi-supervised approach for stream based recommender systems. Therefore, we propose a novel and flexible framework that implements such an approach and is applicable to state-of-the-art matrix factorization methods. This paper extends our work from Matuszyk and Spiliopoulou (2015). In comparison to our previous work, here, we add several new instantiations for framework components, self-learning approach and a new evaluation protocol, according to which we re-evaluated all presented methods. Since each component in our framework has several possible implementations (e.g. different types of reliability measures), we additionally analyse the impact of framework components onto the quality of recommendations to find out the best implementation of each component (cf. Sect. 6.3).

3 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 Sect. 3.1 and then explain how it is applied in two alternative approaches: co-training (cf. Sect. 3.2) and self-learning (cf. Sect. 3.3). In Table 1 we present a summary of notation and abbreviations used in this work. Figure 1 gives a simplified overview over the framework components and their interaction. Sections 3.1 and 3.2 come from our conference paper (Matuszyk and Spiliopoulou 2015).

3.1 Incremental recommendation algorithm

The core of our framework is a recommendation system algorithm. Figure 2 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 (Biased Regularized Simultaneous Matrix Factorization) algorithm, etxBRISMF, as described in Sect. 4.1.

3.2 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. (2014) for batch recommender systems. 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.

Table 1 Summary of notation and abbreviations used in this paper

Notation	Meaning
SSL	Semi-supervised learning
SSL3	Semi-supervised learning with co-training using three parallel learners
USL	Unsupervised learning
noSSL	Algorithm without SSL (i.e. supervised learning only); it is used as a comparison baseline
SL	Self-learning
CF	Collaborative filtering
MF	Matrix factorization
BRISMF	Biased regularized incremental simultaneous matrix factorization (cf. Takács et al. 2009)
IR@10	Incremental recall at 10 (cf. Cremonesi et al. 2010)
$Co-Tr_n$	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
$\widehat{r}_{xCo-Tr_n}$	A prediction of value of rating x made by the Co-Trainer n
$\widehat{r}_{xAgg}$	An aggregate of all predictions of rating x made by all Co-Trainers from C
$\widehat{r}^u_x$	A prediction of rating x, where no ground truth exists ("u" for unlabelled)
$rel(\widehat{r}^u_{iCo-Tr_a})$	Reliability of prediction $\widehat{r}^u_i$ by $Co-Tr_a$
SGD	Stochastic gradient descent
k	Number of latent dimensions in matrix factorization
λ	Regularization parameter for matrix factorization
η	Learning rate for SGD
$\overrightarrow{p_u}$	Latent vector of user u
$\overrightarrow{q_i}$	Latent vector of item i

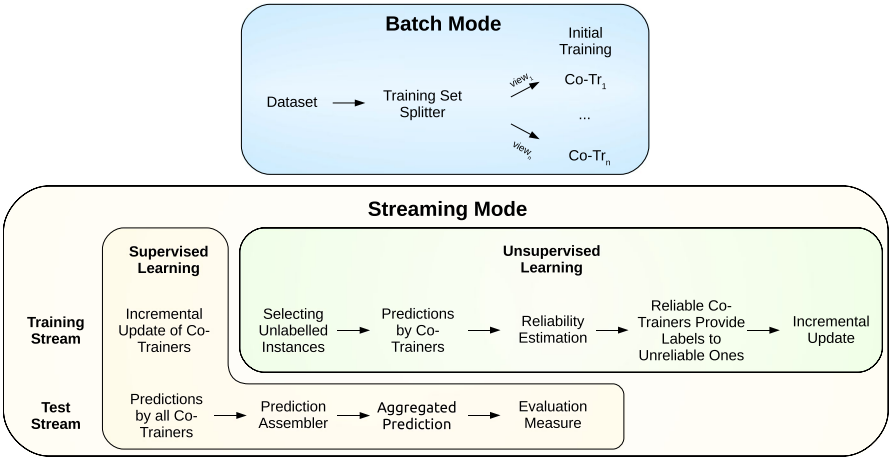

Fig. 1 A simplified overview of the framework components

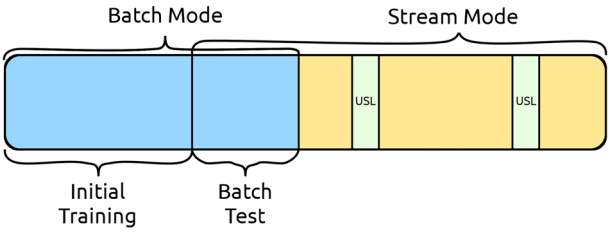

Fig. 2 Division of a dataset (entire *rectangle*) into batch (*blue* part) and stream mode (*yellow* part). The stream mode is the main part of an algorithm with incremental learning. Batch mode is used for initial training (Matuszyk and Spiliopoulou 2015) (Color figure online)

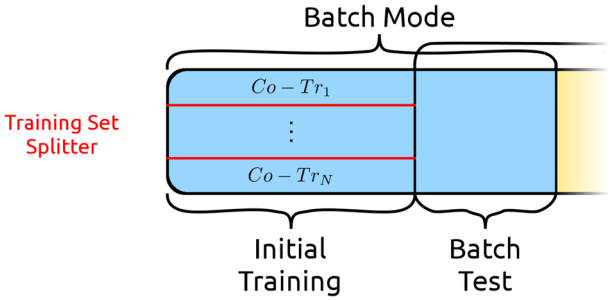

Fig. 3 Different co-trainers are trained on different parts of the initial training set. The component responsible for splitting the training set is **training set splitter** (Matuszyk and Spiliopoulou 2015)

3.2.1 Initial training

The specialization of the algorithms takes place already in the initial training. In Fig. 3 we present a close-up of the batch mode from Fig. 2. Here, the initial training set is divided between N co-trainers from the set $C = \{Co - Tr_1, \ldots, Co - Tr_N\}$, where $N \geq 2$.

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. 3; cf. Sect. 4.2 for instances of this component). Formally, a training set splitter is a function that relates all co-trainers to subsets of all ratings in the initial training set $R_{initialTrain}$:

$$f(C, R_{initialTrain}) : \forall n \left\{ (Co - Tr_n \in C) \rightarrow R_{initialTrain}^{Co-Tr_n} \right\} \tag{1}$$

with $n = 1, \ldots, N$ and $R_{initialTrain}^{Co-Tr_n} \subseteq R_{initialTrain}$. This function is not a partitioning function, since overlapping between different $R_{initialTrain}^{Co-Tr_n}$ is allowed and often beneficial. Implementations of this component are provided in Sect. 4.2.

3.2.2 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 4 is a close-up of the stream mode from Fig. 2. It represents a stream of ratings $r_1, r_2, \ldots$. The yellow part of the figure depicts the supervised learning, whereas the green part symbolizes the unsupervised learning (cf. next section).

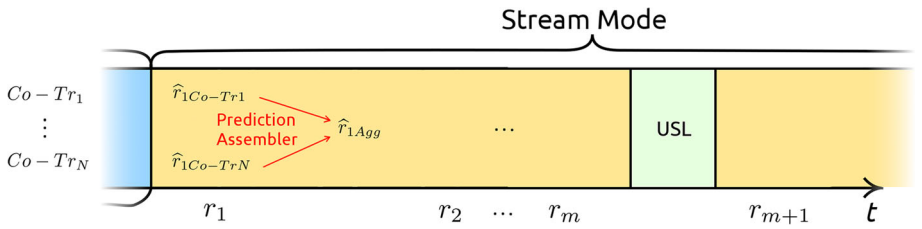

Fig. 4 A close-up of the stream mode from Fig. 2. The *yellow* part represents the supervised learning and the *green* one unsupervised learning. Predictions made by co-trainers are aggregated by a **prediction assembler** (Matuszyk and Spiliopoulou 2015) (Color figure online)

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:

$$\forall n : Co - Tr_n(r_x) = \widehat{r}_{xCo-Tr_n} \tag{2}$$

Please, note that co-trainers are instances of the extBRISMF algorithm (cf. Sect. 4.1). 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 1).

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 Sect. 4.3). The function of prediction assembler is as follows:

$$predictionAssembler(r_x, C) = \widehat{r}_{xAgg} \tag{3}$$

In Fig. 4 this process is visualized only for the rating r_1 due to space constraints, however in a real application, it is repeated for all ratings in the stream with known ground truth (supervised learning). For instances with no ground truth the procedure is different.

3.2.3 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. 5). Unlabelled instances in recommender systems are user-item-pairs that have no ratings. We indicate those instances with r_z^u ("u" for unsupervised). 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. Sect. 4.4).

Once the unlabelled instances $r_1^u, \ldots, r_z^u$ are selected, co-trainers are used again to make predictions:

$$\forall n, i : Co - Tr_n\left(r_i^u\right) = \widehat{r}_{iCo-Tr_n}^u \tag{4}$$

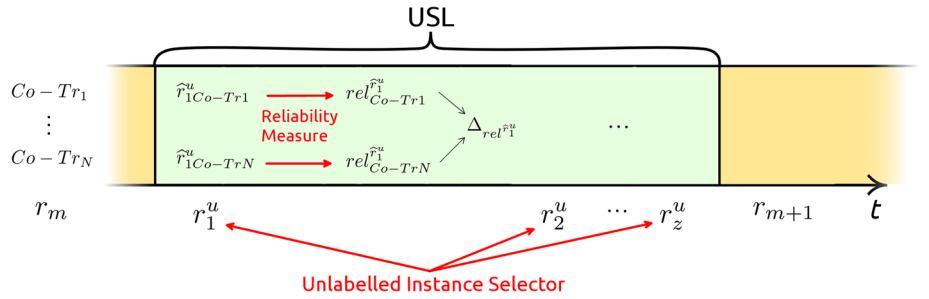

Fig. 5 The procedure of unsupervised learning. User-item-pair without ratings are selected using an **unlabelled instance selector**. Predictions and their **reliabilities** are estimated. The most reliable predictions are used as labels for the least reliable co-trainers (Matuszyk and Spiliopoulou 2015)

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

$$reliability : \left(Co - Tr_n, \widehat{r}^u_{iCo-Tr_n}\right) \rightarrow [0; 1] \tag{5}$$

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 reliabilities of the predictions for r^u_i :

$$\Delta = \left| rel\left(\widehat{r}^u_{iCo-Tr_a}\right) - rel\left(\widehat{r}^u_{iCo-Tr_b}\right)\right| \tag{6}$$

for all $a, b = 1, \ldots, N$ 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.

3.3 Stream-based self-learning

Our second approach to semi-supervised learning on streams is self-learning (SL). According to this approach a single learner provides labels to itself. Those labels are its own predictions, whose reliabilities were assessed the highest. 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. 5 changes as shown in Fig. 6. 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 $r^u_1, r^u_2, \ldots, r^u_z$. The reliability of this predictions is assessed using a reliability measure (cf. Sect. 4.5). Differently than in co-training, here the difference in

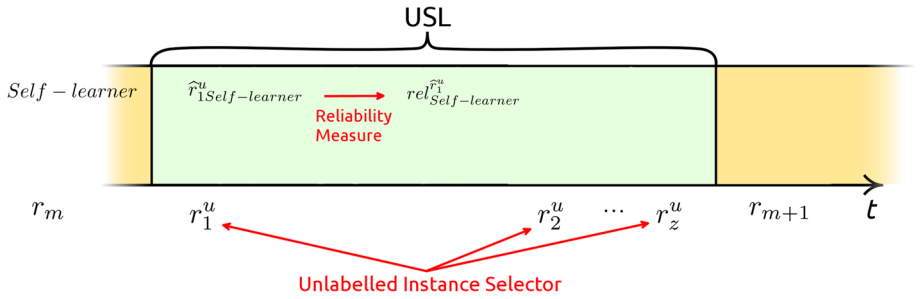

Fig. 6 Adjusted procedure of unsupervised learning from Fig. 5 for the self-learning approach. In this approach there is only one learner, whose predictions are assessed using a reliability measure (Matuszyk and Spiliopoulou 2015)

reliability of learners, Δ from Eq. 6, 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. Sect. 4.4 for details).

4 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 component. The reliability measure, for example, has many possible implementations (cf. Sect. 4.5). Except for Sects. 4.2.4–4.2.6, 4.3.4, 4.4.3, 4.5.4, the Sects. 4.1–4.5 come from our conference paper (Matuszyk and Spiliopoulou 2015).

4.1 Incremental recommendation algorithm: extBRISMF

The core of our framework is a matrix factorization algorithm. We extended the BRISMF algorithm by Takács et al. (2009) 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.

For decomposition of the rating matrix R into two latent matrices $R \approx PQ$ we use stochastic gradient descent (SGD). P is a matrix of latent user factors with elements p_{uk}, where u is a user and k is a latent dimension. Similarly, Q is a matrix of latent item factors with elements q_{ik}, where i is an item. That results in the following update formulas for SGD (formulas from Takács et al. 2009):

$$\begin{aligned} p_{u,k} &\leftarrow p_{u,k} + \eta \cdot (predictionError \cdot q_{i,k} - \lambda \cdot p_{u,k}) \\ q_{i,k} &\leftarrow q_{i,k} + \eta \cdot (predictionError \cdot p_{u,k} - \lambda \cdot q_{i,k}) \end{aligned} \tag{7}$$

where η is a learning rate and λ a regularization parameter that prevents overfitting. A rating prediction can be obtained by multiplying the corresponding item and user vector from latent matrices $\widehat{r}_{ui} \approx p_u \cdot q_i$.

Algorithm 1 extBRISMF - trainIncrementally($r_{u,i}$)

Input: $r_{u,i}$, P, Q, η, k, λ
1: $\overrightarrow{p_u} \leftarrow$ getLatentUserVector(P, u)
2: $\overrightarrow{q_i} \leftarrow$ getLatentItemVector(Q, i)
3: **if** $\overrightarrow{p_u} = null$ **then**
4: $\quad\overrightarrow{p_u} \leftarrow$ getAverageVector(P) + randomVector
5: $\quad\overrightarrow{p_{u_1}} \leftarrow 1$
6: $\quad P \leftarrow P.append(p_u)$
7: **end if**
8: **if** $\overrightarrow{q_i} = null$ **then**
9: $\quad\overrightarrow{q_i} \leftarrow$ getAverageVector(Q) + randomVector
10: $\quad\overrightarrow{q_{i_2}} \leftarrow 1$
11: $\quad Q \leftarrow Q.append(q_i)$
12: **end if**
13: $\widehat{r}_{u,i} = \overrightarrow{p_u} \cdot \overrightarrow{q_i}$ //predict a rating for $r_{u,i}$
14: evaluatePrequentially($\widehat{r}_{u,i}, r_{u,i}$) //update evaluation measures
15: $epoch = 0$
16: **for all** $epoch \in \{1, \ldots, optimalNumberOfEpochs\}$ **do**
17: $\quad\overrightarrow{p_u} \leftarrow$ getLatentUserVector(P, u)
18: $\quad\overrightarrow{q_i} \leftarrow$ getLatentItemVector(Q, i)
19: $\quad predictionError = r_{u,i} - \overrightarrow{p_u} \cdot \overrightarrow{q_i}$
20: $\quad$**for all** latent dimensions k **do**
21: $\quad\quad$if $k \neq 1$: $p_{u,k} \leftarrow p_{u,k} + \eta \cdot (predictionError \cdot q_{i,k} - \lambda \cdot p_{u,k})$
22: $\quad\quad$if $k \neq 2$: $q_{i,k} \leftarrow q_{i,k} + \eta \cdot (predictionError \cdot p_{u,k} - \lambda \cdot q_{i,k})$
23: $\quad$**end for**
24: **end for**

In Algorithm 1 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.

4.2 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. 2). All of them have one parameter p that controls the degree of overlapping between the co-trainers.

4.2.1 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**.

4.2.2 Random splitter

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

4.2.3 Dimensions 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 all users and items from the initial training set. This might be beneficial for methods not able to extend the dimensions of their matrices over time.

4.2.4 User variance splitter

As the name suggests, this splitter assigns users to different co-trainers based on their rating variance. For all users their 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 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.

4.2.5 Item variance splitter

Similarly to the user variance splitter, this splitter is also based on rating variance. However, here the variance is calculated for item ratings. 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.

4.2.6 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.

4.3 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 $w(\widehat{r}_{u,i}, Co - Tr_j)$:

$$\widehat{r}_{u,iAgg} = \frac{\sum_{j=0}^{N} w(\widehat{r}_{u,i}, Co - Tr_j) \cdot \widehat{r}_{u,iCo-Tr_j}}{\sum_{j=0}^{N} w(\widehat{r}_{u,i}, Co - Tr_j)} \tag{8}$$

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

4.3.1 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:

$$w(\widehat{r}_{u,i}, Co - Tr_j) = recall(Co - Tr_j) \tag{9}$$

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 $recall(Co - Tr_j)$ can be substituted with $recall(Co - Tr_j, u)$ or $recall(Co - Tr_j, i)$.

4.3.2 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.

$$w(\widehat{r}_{u,i}, Co - Tr_j) = RMSE(Co - Tr_j) \tag{10}$$

4.3.3 Reliability-weighted prediction assembler

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

$$w(\widehat{r}_{u,i}, Co - Tr_j) = rel^{\widehat{r}_{u,i}}_{Co-Tr_j} \tag{11}$$

4.3.4 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:

$$w(\widehat{r}_{u,i}, Co - Tr_j) = \begin{cases} 1, & \text{if } rel^{\widehat{r}_{u,i}}_{Co-Tr_j} = \max\limits_{k=1,\ldots,N} rel^{\widehat{r}_{u,i}}_{Co-Tr_k} \\ 0, & \text{otherwise} \end{cases} \tag{12}$$

4.4 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.

4.4.1 Latent disagreement selector

For all users each co-trainer stores a latent vector. We denote this vector as $p_u^{Co-Tr_n}$. In this method we search for users, where the disagreement of the latent user vectors among the

co-trainers is the highest. We define the disagreement among two co-trainers upon a user u as follows:

$$disagreement(Co-Tr_a, Co-Tr_b, u) = \left| p_u^{Co-Tr_a} - p_u^{Co-Tr_b} \right| \tag{13}$$

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.

4.4.2 Random selector

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

4.4.3 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**.

4.5 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.

4.5.1 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 that occurs in the stream is incorporated incrementally into the model. The changes of this vector can be captured using the following formula:

$$\Delta_{p_u} = \sum_{i=0}^{k} \left(p_{u,i}^{t+1} - p_{u,i}^{t} \right)^2 \tag{14}$$

where $p_{u,i}^{t+1}$ and $p_{u,i}^{t}$ are user vectors 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. last paragraph of this section).

4.5.2 Popularity-based reliability measure

Zhang et al. (2014) proposed 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. last paragraph of this section).

4.5.3 Random reliability measure

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

4.5.4 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.

4.5.5 Normalization of reliability measures

As defined in Sect. 3.2, 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. In our framework we use the following sigmoid function for normalization:

$$f(reliability) = \frac{1}{1 + e^{\alpha \cdot (reliability - \mu)}} \tag{15}$$

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).

5 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. (2010)
- significance testing

In the following subsections we describe each of the components.

5.1 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.

To create the holdout subsets we sample randomly a small percentage of users from the original dataset. Those percentages are listed in Table 3 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.

5.2 Dataset splitting

Splitting a dataset into a training and test set is not trivial in the streaming scenario. The state-of-the art method for doing it is the prequential evaluation proposed by Gama et al. (2009). According to this method, all instances (ratings) in a stream are first used for testing the performance of a learner and then they are used for updating the model incrementally. The separation between test and training set is temporal here (first testing, then training). This method is efficient in terms of data usage because all instances in the stream are used both for testing and training.

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 $Q_t, Q_{t+1}, \ldots, Q_{t+n}$ 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, usage of hypothesis tests is prohibited.

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 $Q_t, Q_{t+1}, \ldots, Q_{t+n}$ for a hypothesis test are independent for non-cumulative quality measures. In Sect. 5.4 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 components:

- 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. 7. 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 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.

5.3 Evaluation measure

As an evaluation measure we use the incremental recall by Cremonesi et al. (2010) (no to be confused with the conventional recall). Incremental recall measures how often a recommender system is able to find a relevant item among random items. It is computed as follows: for each relevant item in a stream (based on a relevance threshold) 1000 further items are drawn randomly. Those 1000 random items are assumed to be irrelevant and put into one set together with the one relevant item. The recommender systems is asked to rank the items according

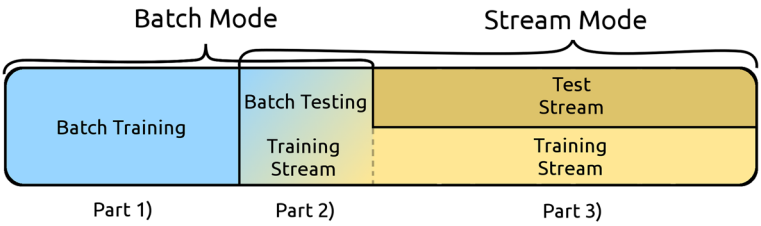

Fig. 7 Splitting of the dataset between the batch and streaming mode. Separation of training and test datasets in each of the modes (Matuszyk et al. 2015)

Table 2 Example input to McNemar's test

Algorithm	Timepoint (rating)				
	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})	…
SSL	1	0	1	1	…
noSSL	1	0	0	1	…

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

to their relevance. If the relevant item has a ranking position $\leq N$, then a hit is counted. The value of $incremental Recall@N$ is $\frac{\#hits}{|relevant\ Items|}$.

Compared to the conventional measures used in similar scenarios (e.g. accuracy, recall, RMSE, etc.), incremental recall brings several advantages. It considers the ranking of recommendations and counts only hits for relevant items. Error-based measures, such as RMSE or MAE, estimate the quality based on predictions of all ratings. Considering that ca. 99% of items is not relevant to a user, it is not desirable to incorporate the prediction error on items that do not matter into the quality measure.

A further advantage is the ability of incremental recall to deal with extremely skewed distributions. A negative example for this aspect is the accuracy measure. A classifier that predicts all items to be irrelevant would yield more than 99% accuracy. A distinction between a good and a bad classifier would be possible only with very high precision, where the numerical error plays a big role. Because incremental recall considers only the relevant items, it is free from this problem.

5.4 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 at 10 (cf. 5.3). An example for observations of this quality measure is represented in Table 2.

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 SSL algorithm 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 SSL algorithm 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 example table we see that the semi-supervised algorithm (SSL) scored more hits than the algorithm without semi-supervised learning (noSSL). However, the evidence in this example is not sufficient to consider any of them superior. To test if the improvement of the SSL algorithm is statistically significant, we use the McNemar's test (McNemar 1947). This test is used for paired, nominal, dichotomous data, same as presented here.

For this test, our example input from Table 2 is transformend 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 the tested algorithm is significantly better than a noSSL algorithm. All p values reported in Sect. 6 result from this test (lower p values are better).

Gama et al. (2009) 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 Hommel's method (Shaffer 1995). All p values in Sect. 6 have been corrected using this method.

6 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 (SSL) to a single learner without semi-supervised learning (noSSL). In both cases the algorithm used is the extBRISMF (cf. Sect. 4.1), 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)

In Sect. 6.2 we present the results of the approximately best parameter setting from the grid search. The grid search was performed on a cluster running the (Neuro)Debian operating system (Halchenko and Hanke 2012). In total we conducted more than 700 experiments. In Sect. 6.3 we analyse the impact of different instances of framework components (e.g. which reliability measure performs the best).

Table 3 Dataset statistics; "ratio of users for parameter optimization" indicates what percentage of users was used for parameter optimization using a grid search. Extreme sparisty values show the abundance of unlabelled information

Dataset	Ratings	Users	Items	Sparsity (%)	Ratio of users for parameter optimization
ML1M	1,000,209	6040	3706	95.53	0.05
ML100k	100,000	943	1682	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

6.1 Datasets

In Table 3 we present summary statistics of five real-world dataset that we used in our evaluation. Those datasets are: MovieLens 1M and 100k[1] (Harper and Konstan 2016), a sample of 5000 users from the extended Epinions (Massa and Avesani 2006) dataset, a sample of 10,000 users from the Netflix dataset[2] and a sample of the same size from the Flixster dataset.[3] From some of the 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. Extreme sparsity values in the fifth column show the amount of unlabelled information in a dataset. Our semi-supervised framework exploits this abundantly available information.

6.2 Performance of SSL

In this section we present results of the evaluation of our semi-supervised framework. In the evaluation we use the protocol described in Sect. 5. We compare four methods: co-training with two and three learners (SSL2 and SSL3), self-learning (SL) and a non SSL algorithm (noSSL). All compared methods use the same matrix factorization algorithm, so that changes in the performance are due to the application of SSL. 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. Sect. 5). Therefore, for each of the methods we have only one setting used in the final evaluation.

In Fig. 8 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 8 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.

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

[2] https://www.netflix.com

[3] https://www.flixster.com

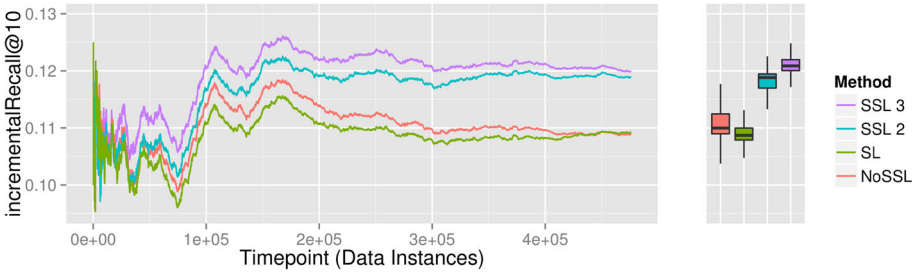

Fig. 8 Incremental Recall@10 over time on the **Movielens 1M** dataset achieved by different SSL methods as compared to noSSL (higher values are better). The *box plot* on the *right* visualizes a simplified distribution of incremental recall

Table 4 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 Sect. 5.4. In Table 5 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. 9. 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. Table 4). Statistical significance was achieved by both SSL3 and SSL2 (cf. Table 5).

On the Flixster dataset (random sample of 10,000 users) the self-learning method showed the best performance (cf. Fig. 10). 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 5000 users), which shows the highest sparsity of all tested datasets, SSL2 performed the best (cf. Fig. 11). 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. Self-learning preformed well initially, but did not achieve a significant improvement over noSSL (cf. Table 5). Both SSL2 and SSL3 yielded a significant improvement. However, the computation time for a data instance rose from 18.3 ms with noSSL to 311.7 ms with SSL3.

Table 4 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 $\widehat{t}$

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 5 p values from the McNemar's test, corrected for multiple testing according to the Hommel's method (cf. Sect. 5.4). p values lower than 0.05 are marked in bold face. They indicate a statistically significant improvement over the noSSL algorithm (lower values are better)

Dataset	p values SSL3 versus noSSL	SSL2 versus noSSL	SL versus 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**

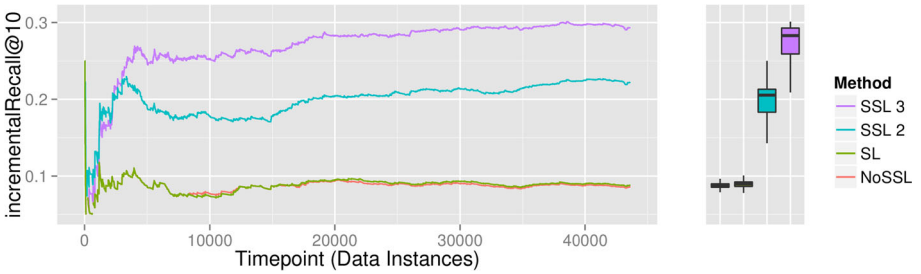

Fig. 9 Incremental Recall@10 over time on the **Movielens 100k** dataset achieved by different SSL methods as compared to noSSL (higher values are better)

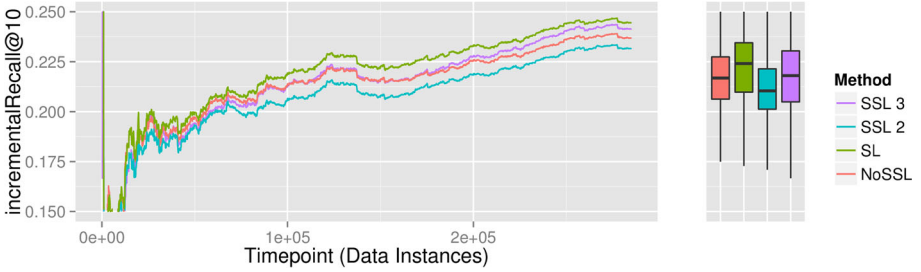

Fig. 10 Incremental Recall@10 over time on the **Flixster (10k users)** dataset achieved by different SSL methods as compared to noSSL (higher values are better)

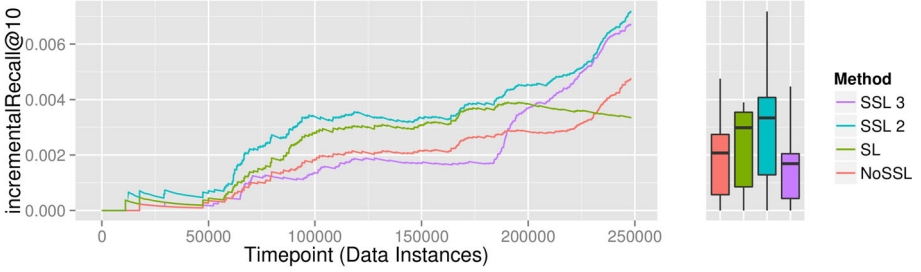

Fig. 11 Incremental Recall@10 over time on the **Epinions (5k users)** dataset achieved by different SSL methods as compared to noSSL (higher values are better)

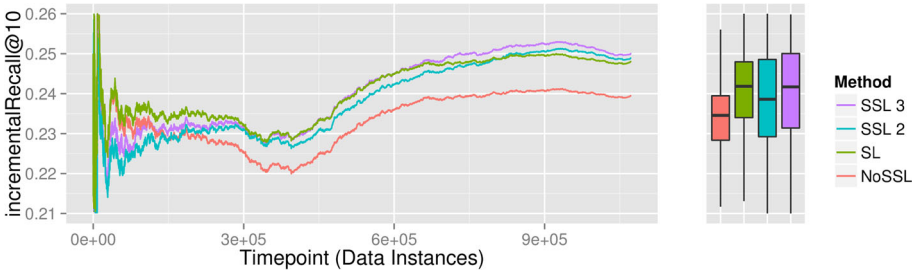

Fig. 12 Incremental Recall@10 over time on the **Netflix (10k users)** dataset achieved by different SSL methods as compared to noSSL (higher values are better)

On the Netflix dataset (random sample of 10,000 users) improvements of recommendation quality are clear for all SSL methods (cf. Fig. 12). This is also reflected by p values in Table 5.

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.

6.3 Impact analysis 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 (incremental recall).

In Fig. 13 we present the results of this analysis. Each sub-plot represents the impact analysis of one component. To analyse how much 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. 13 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 considered low), we normalized the bar hight 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 IR of 0.006 on the Epinions dataset is higher that 0.058 on the ML1M dataset.

In the first subplot of Fig. 13 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 user popularity reliability estimator with the stream-based normalization (ST for self-tuning). It is followed by item recall-based estimator, user popularity estimator and user recall-based estimator with similar results.

The second 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.

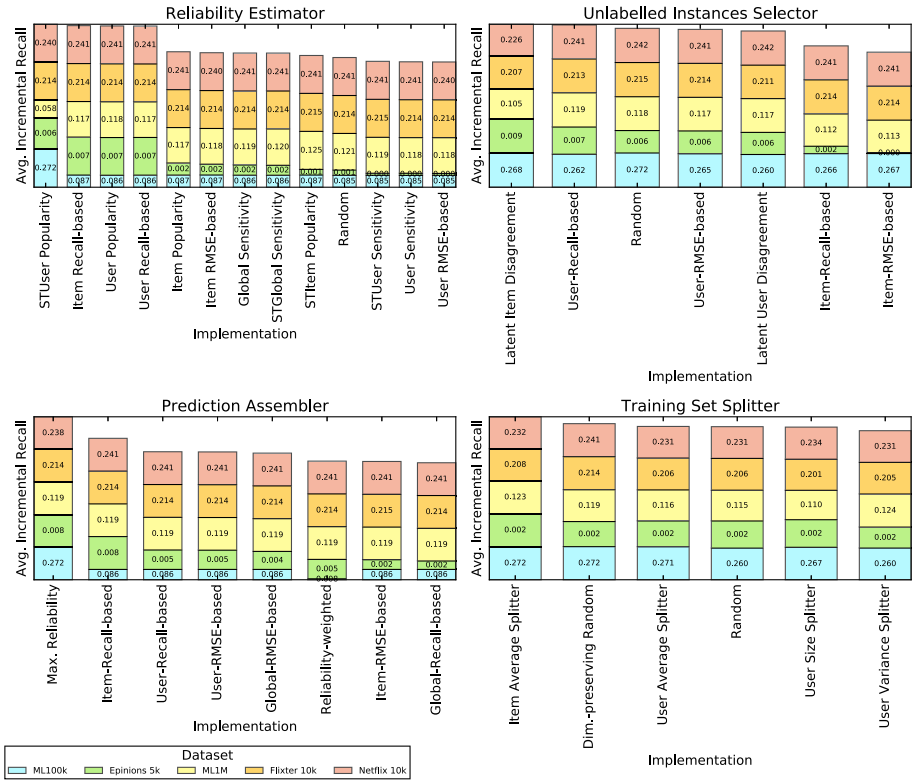

Fig. 13 Analysis of impact of component instances onto the quality of recommendations (avg. incremental recall). We conducted experiments with the optimal parameter setting, where only one component varied (e.g. reliability estimator in the first subplot). Component instances with the highest cumulative sum of performance on all dataset are the best (leftmost in all subplots)

The best implementation of the prediction assembler component is based on maximal reliability, i.e. the prediction with maximal reliability serves as the final prediction of all co-trainers. 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.

7 Conclusions

Recommender systems suffer from an extreme data sparsity. Only few items can be labelled by users. Therefore, the number of unlabelled items is unproportionally higher than the number of labelled ones. We propose a novel framework for stream-based semi-supervised learning for recommender systems that exploits this abundant unlabelled information and alleviates the sparsity problem.

This is the first such framework for stream-based recommender systems. We implemented two semi-supervised learning (SSL) approaches: self-learning and co-training and evaluated

them in a streaming setting on five real-world datasets. We showed 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.

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), which proves the applicability of our framework to real-time applications.

In our experiments we used the BRISMF algorithm by Takács et al. (2009), 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. The number of possible co-trainers is strongly limited. 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 onto 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. 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 retraining of the models with new data can restore this advantage. 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.

As our framework is modular and can be extended easily, in our future work we plan to implement further modules such as reliability measures and prediction assemblers. A further open challenge is the increased computation time in the co-training approach.

Acknowledgements The authors would like to thank Daniel Kottke and Dr. Georg Krempl for suggestions regarding self-tuning normalization for streams and evaluation. We also thank to the Institute of Psychology II at the University of Magdeburg for making their computational cluster available for our experiments and to our student, Florian Schweighöfer, who helped us with implementation of selected training set splitters.

References

Bosnić, Z., Demšar, J., Kešpret, G., Rodrigues, P. P., Gama, J., & Kononenko, I. (2014). Enhancing data stream predictions with reliability estimators and explanation. *Engineering Applications of Artificial Intelligence*, *34*, 178–192.

Christakou, C., Lefakis, L., Vrettos, S., & Stafylopatis, A. (2005). A movie recommender system based on semi-supervised clustering. *CIMCA/IAWTIC*, *2*, 897–903.

Cremonesi, P., Koren, Y., & Turrin, R. (2010). Performance of recommender algorithms on top-n recommendation tasks. In *RecSys '10*. ACM.

de Souza, V. M. A., Silva, D. F., Gama, J., & Batista, G. E. A. P. A. (2015). Data stream classification guided by clustering on nonstationary environments and extreme verification latency. In S. Venkatasubramanian & J. Ye (Eds.), *SDM* (pp. 873–881). SIAM.

Deshpande, M., & Karypis, G. (2004). Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems (TOIS)*, *22*(1), 143–177.

Desrosiers, C., & Karypis, G. (2011). A comprehensive survey of neighborhood-based recommendation methods. In F. Ricci, L. Rokach, B. Shapira, & P. B. Kantor (Eds.), *Recommender systems handbook* (pp. 107–144). Berlin: Springer.

Dyer, K. B., Capo, R., & Polikar, R. (2014). COMPOSE: A semisupervised learning framework for initially labeled nonstationary streaming data. *IEEE Transactions on Neural Networks and Learning Systems*, *25*(1), 12–26.

Gama, J., Sebastião, R., & Rodrigues, P. P. (2009). Issues in evaluation of stream learning algorithms. In *KDD*. ACM.

Goldberg, D., Nichols, D., Oki, B. M., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, *35*(12), 61–70.

Halchenko, Y. O., & Hanke, M. (2012). Open is not enough. let's take the next step: An integrated, community-driven computing platform for neuroscience. *Frontiers in Neuroinformatics, 6*, 22.

Harper, F. M., & Konstan, J. A. (2016). The MovieLens datasets: History and context. *TiiS*, *5*(4), 19.

Hernando, A., Bobadilla, J., Ortega, F., & Tejedor, J. (2013). Incorporating reliability measurements into the predictions of a recommender system. *Information Sciences*, *218*, 1–16.

Koren, Y. (2008). Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 426–434). ACM.

Koren, Y. (2009). Collaborative filtering with temporal dynamics. In *KDD '09*. ACM.

Koren, Y., Bell, R., & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, *42*(8), 30–37.

Linden, G., Smith, B., & York, J. (2003). Amazon.Com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing, 7*, 76–80.

Massa, P., & Avesani, P. (2006). Trust-aware bootstrapping of recommender systems. In *ECAI workshop on recommender systems* (pp. 29–33).

Matuszyk, P., & Spiliopoulou, M. (2014a). Hoeffding-CF: Neighbourhood-based recommendations on reliably similar users. In V. Dimitrova, T. Kuflik, D. Chin, F. Ricci, P. Dolog & G. J. Houben (Eds.), *User modeling, adaptation, and personalization, Lecture notes in computer science* (Vol. 8538, pp. 146–157). Springer.

Matuszyk, P., & Spiliopoulou, M. (2014b). Selective forgetting for incremental matrix factorization in recommender systems. In *Discovery science, LNCS*. Springer.

Matuszyk, P., & Spiliopoulou, M. (2015). Semi-supervised learning for stream recommender systems. In: N. Japkowicz & S. Matwin (Eds.), *Discovery science, Lecture notes in computer science* (Vol. 9356, pp. 131–145). Springer.

Matuszyk, P., Vinagre, J., Spiliopoulou, M., Jorge, A. M., & Gama, J. (2015). Forgetting methods for incremental matrix factorization in recommender systems. In *Proceedings of the SAC'15 conference*. ACM.

McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, *12*(2), 153–157.

Preisach, C., Marinho, L.B., & Schmidt-Thieme, L. (2010). Semi-supervised tag recommendation—Using untagged resources to mitigate cold-start problems. In M. J. Zaki, J. X. Yu, B. Ravindran & V. Pudi (Eds.), *Advances in knowledge discovery and data mining, Lecture notes in computer science* (Vol. 6118, pp. 348–357). Springer.

Rodrigues, P. P., Gama, J., & Bosnic, Z. (2008). Online reliability estimates for individual predictions in data streams. In *ICDM workshops* (pp. 36–45). IEEE Computer Society.

Rosenberg, C., Hebert, M., & Schneiderman, H. (2005). Semi-supervised self-training of object detection models. In WACV/MOTION (pp. 29–36). IEEE Computer Society.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web, WWW '01* (pp. 285–295). ACM, New York.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2002). Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth international conference on computer and information science* (pp. 27–28).

Shaffer, J. P. (1995). Multiple hypothesis testing. *Annual Review of Psychology*, *46*(1), 561–584.

Sindhwani, V., Niyogi, P., & Belkin, M. (2005). A co-regularized approach to semi-supervised learning with multiple views. In *Proceedings of the ICML workshop on learning with multiple views*.

Su, X., & Khoshgoftaar, T. (2009). A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, *2009*, 4.

Takács, G., Pilászy, I., Németh, B., & Tikk, D. (2009). Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research, 10*, 623–656.

Zhang, M., Tang, J., Zhang, X., & Xue, X. (2014). Addressing cold start in recommender systems: A semi-supervised co-training algorithm. In *SIGIR*. ACM.

Zhou, Z. H., & Li, M. (2007). Semisupervised regression with cotraining-style algorithms. *IEEE Transactions on Knowledge and Data Engineering, 19*(11), 1479–1493.

Zhou, Z. H., Zhan, D. C., & Qiang, Y. (2007). Semi-supervised learning with very few labeled training examples. In *AAAI* (pp. 675–680). AAAI Press.

Zhu, T., Hu, B., Yan, J., & Li, X. (2010). Semi-supervised learning for personalized web recommender system. *Computing and Informatics*, *29*(4), 617–627.

Zhu, X. (2005). Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison.