

Semi-supervised Learning for Stream Recommender Systems

Pawel Matuszyk^(✉) and Myra Spiliopoulou

Otto-von-Guericke-University Magdeburg, Universitätsplatz 2,
39106 Magdeburg, Germany
{pawel.matuszyk,myra}@iti.cs.uni-magdeburg.de

Abstract. Recommender systems suffer from an extreme data sparsity that results from a large number of items and only a limited capability of users to perceive them. Only a small fraction of items can be rated by a single user. Consequently, there is plenty of unlabelled information that can be leveraged by semi-supervised methods. We propose the first semi-supervised framework for stream recommender systems that can leverage this information incrementally on a stream of ratings. We design several novel components, such as a sensitivity-based reliability measure, and extend a state-of-the-art matrix factorization algorithm by the capability to extend the dimensions of a matrix incrementally as new users and items occur in a stream. We show that our framework improves the quality of recommendations at nearly all time points in a stream.

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

1 Introduction

Data sparsity is a known problem in recommenders. It is amplified by the introduction of new items and the appearance of new users, on which and whom little is known. In [11], Zhang et al. proposed to deal with this problem with semi-supervised learning. In this study, we demonstrate the potential of semi-supervised learning (SSL) as cure to data sparsity in **stream** recommenders. The streaming context poses several challenges on semi-supervised algorithms, which do not show in the static context: on which data of the stream should the learning be done, on which data should the learner be tested before being applied on the **ongoing** stream, how should an algorithms treat new users and items? To deal with these challenges, we propose a semi-supervised stream recommender that deals with data sparsity by deriving predictions from part of the stream (unlabelled information) and using them for learning. To deal with new users and items we extend a state-of-the-art matrix factorization algorithm BRISMF [10] by the ability to deal with growing dimensions of the matrix on the ongoing stream. Our framework encompasses novel reliability measures, selectors for unlabelled data and further components specified in Sect. 3. To our knowledge, this is the first such framework for **stream** recommender systems.

Sparsity and cold start problems are often tackled by using context or external sources of information (e.g. demographics of users, characteristics of items, etc.). These approaches, however, narrow down the palette of applicable algorithms to the few ones able to use them and it excludes many practitioners, who do not have the required data. Our framework does not rely on any additional source of information, but only on the user-item-rating matrix, which makes it general and applicable to any collaborative filtering algorithm.

In an empirical study on real-world datasets we show that our SSL framework improves the quality of recommendations at nearly all time points in the stream.

Contributions. To summarize, our contributions are as follows:

- we propose the **first SSL framework for stream recommenders** including novel reliability measures, selectors for unlabelled instances, etc.
- we extend the BRISMF algorithm by the ability to deal with growing dimensions of the matrix
- we show that SSL for stream recommenders improves the quality of recommendations.

Organization. This paper is structured as follows. In Sect. 2 we discuss related work. Section 3 gives an overview over our framework and explains the interplay of its components. The following section describes an instantiation of the components of the general framework. Evaluation protocol is described in Sect. 5. Our results are explained in Sect. 6. Finally, in Sect. 7, we conclude our work and discuss open issues.

2 Related Work

Recommender systems have been researched thoroughly in the recent years. State-of-the-art in the group of collaborative filtering approaches are nowadays matrix factorization methods. Their predictive performance has been shown in several publications [5, 6, 10]. We focus on their incremental version, since those methods are applicable to streams of ratings. In this work we extend the BRISMF algorithm (biased regularized incremental simultaneous matrix factorization) proposed by Takács et al. [10]. BRISMF exists in two versions. One of them was developed for batch processing. Takács et al., however, also developed an incremental version of it (cf. Algorithm 2 in [10]). In this version the latent item vectors are fixated and updated as new ratings occur in the stream. Latent user vectors are updated, however, no new users are added to the matrix. In our work we lift those limitations of the BRISMF algorithm.

While semi-supervised classification has been investigated thoroughly in the field of data mining, semi-supervised regression, a discipline that matrix factorization belongs to, is a less researched problem [12]. Recommender system domain is even more specific due to its idiosyncrasies, such as dealing with large matrices that are typically up to 99 % empty, cold start problem and many more. Due to those challenges only little work was done on semi-supervised learning

in recommender systems. The work by Zhang et al. [11] belongs to the few ones. They proposed a co-training method for stationary, batch-based recommender systems. Their approach, however, is not incremental and not appropriate for streams of ratings and, therefore, would require a frequent retraining of the models. The framework proposed by us lifts those limitations by incrementally incorporating new rating information into the models and adapting to changes.

One of the biggest challenges in recommender systems is an extreme sparsity of data. Many techniques have been developed in order to tackle this problem. One of the most straightforward techniques is filling of the missing values in the matrix with default values (e.g. averages). This method, however, is very time and memory consuming and it lacks personalization. Another approach involves active learning techniques, where an algorithm chooses what label (rating) to request from a user in order to maximize a predefined gain for the model [4]. Active Learning techniques base on the assumption that a user knows the requested label and is willing to share it. This is often not the case in real applications. Semi-supervised learning provides here an important advantage of not having to rely on users' input.

3 Semi-supervised Framework for Stream Recommenders

In this section we present our main contribution - a semi-supervised framework for stream recommenders with a description of the components and their function. Our new components are marked in red in the figures below. This section gives definitions and an overview of how the components are interrelated. An instantiation and implementation of the components is provided in Sect. 4.

3.1 Incremental Recommendation Algorithm

The core of our framework is a recommendation system algorithm. Figure 1 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 also takes place in this phase (green bars).

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 prediction take place simultaneously. Our extended version of the BRISMF algorithm, etxBRISMF, is described in Sect. 4.1.

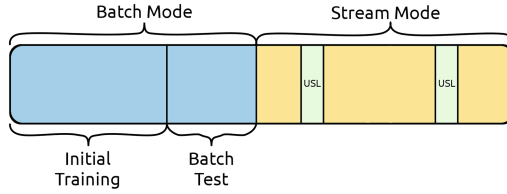


Fig. 1. Division of a dataset (entire rectangle) into batch (blue part) and stream mode (yellow part). The stream mode is the main part of an algorithm with incremental learning and predictions. Batch mode is used for initial training (Color figure online).

3.2 Stream Co-Training Approach

For semi-supervised learning we use the co-training 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 in the co-training approach can outperform a single model that uses all information.

Initial Training. The specialization of the algorithms takes place already in the initial training. In Fig. 2 we present a close-up of the batch mode from Fig. 1. Here, the initial training set is divided between N co-trainers from the set $C = \{Co - Tr_1, \dots, Co - Tr_N\}$, where $N \geq 2$.

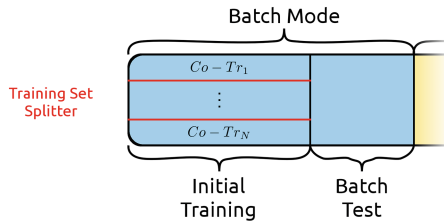


Fig. 2. Different co-trainers are trained on different parts of the initial training set. The component responsible for splitting the training set is **training set splitter** (Color figure online).

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

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

with $n = 1, \dots, N$ 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.

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. First, a prediction is made and evaluated, then the models are updated using the new information according to the prequential evaluation (cf. Sect. 5).

Figure 3 is a close-up of the stream mode from Fig. 1. It represents a stream of ratings r_1, r_2, \dots . The yellow part of the figure depicts the supervised learning, whereas the green part symbolizes the unsupervised learning (cf. next section). For each rating r_x in the stream all co-trainers calculate a prediction:

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

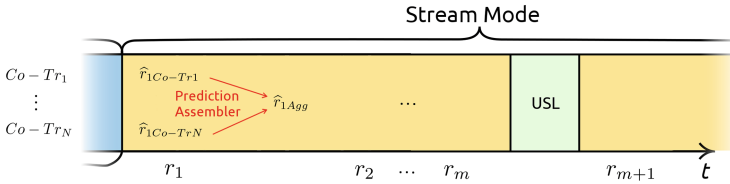


Fig. 3. Close-up of the stream mode from Fig. 1. The yellow part represents the supervised learning and the green one unsupervised learning. Predictions made by co-trainers are aggregated by a **prediction assembler** (Color figure online).

In order to aggregate all predictions made by co-trainers into one prediction \hat{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:

$$predictionAssembler(r_x, C) = \hat{r}_{xAgg} \quad (3)$$

In Fig. 3 this process is visualized only for the rating r_1 due to space constraints, however in 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.

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. 4). 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 than 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).

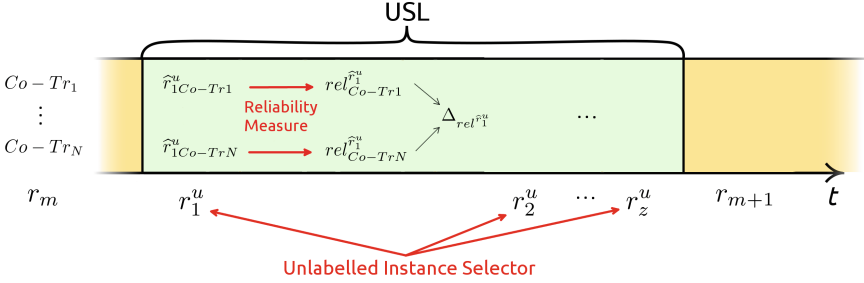


Fig. 4. The procedure of unsupervised learning. User-item-pair without ratings are selected using an **unlabelled instance selector**. Predictions and their **reliabilities** are estimated. The most reliable predictions are used as labels for the least reliable co-trainers.

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

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

where $i = 1, \dots, z$ and $n = 1, \dots, N$. 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:

$$reliability : (Co - Tr_n, \widehat{r}_{iCo-Tr_n}^u) \rightarrow [0; 1] \quad (5)$$

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_i^u :

$$\Delta = |rel(\widehat{r}_{iCo-Tr_a}^u) - rel(\widehat{r}_{iCo-Tr_b}^u)| \quad (6)$$

for all $a, b = 1, \dots, N$ 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 r_i^u and the unreliable co-trainer learns incrementally using the provided label.

4 Instantiation of Framework Components

While, in previous section, we provided definitions of our components, here we present several **instances** of each component and their implementations.

4.1 extBRISMF - Dimensionality Extending BRISMF

The core of our framework is a matrix factorization algorithm. We extended the BRISMF algorithm by Takács et al. [10] 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 $p_{u,k}$, where u is a user and k is a latent dimension. Similarly, Q is a matrix of latent item factors with elements $q_{i,k}$, where i is an item. That results in the following update formulas for SGD [10]:

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

where η is a learn 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 $\hat{r}_{ui} \approx p_u \cdot q_i$.

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. We initialize them with an average vector of the corresponding matrix plus a small random component instead of just a random vector.

4.2 Training Set Splitter

We propose three types of training set splitter (cf. Fig. 1). All of them have one parameter p that controls the degree of overlapping between the co-trainers.

User Size Splitter. This splitter discriminates between users of different sizes. Size of a user is defined as the number of rating he/she 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).

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

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.

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

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

- 1: $\vec{p}_u \leftarrow \text{getLatentUserVector}(P, u)$
- 2: $\vec{q}_i \leftarrow \text{getLatentItemVector}(Q, i)$
- 3: **if** $\vec{p}_u = \text{null}$ **then**
- 4: $\vec{p}_u \leftarrow \text{getAverageVector}(P) + \text{randomVector}$
- 5: $P \leftarrow P.append(p_u)$
- 6: **end if**
- 7: **if** $\vec{q}_i = \text{null}$ **then**
- 8: $\vec{q}_i \leftarrow \text{getAverageVector}(Q) + \text{randomVector}$
- 9: $Q \leftarrow Q.append(q_i)$
- 10: **end if**
- 11: $\hat{r}_{u,i} = \vec{p}_u \cdot \vec{q}_i$ //predict a rating for $r_{u,i}$
- 12: evaluatePrequentially($\hat{r}_{u,i}, r_{u,i}$) //update evaluation measures
- 13: $epoch = 0$
- 14: **while** $epoch < \text{optimalNumberOfEpochs}$ **do**
- 15: $epoch++$; //for all retained ratings
- 16: $\vec{p}_u \leftarrow \text{getLatentUserVector}(P, u)$
- 17: $\vec{q}_i \leftarrow \text{getLatentItemVector}(Q, i)$
- 18: $predictionError = r_{u,i} - \vec{p}_u \cdot \vec{q}_i$
- 19: **for all** latent dimensions k **do**
- 20: **if** $k \neq 1$: $p_{u,k} \leftarrow p_{u,k} + \eta \cdot (predictionError \cdot q_{i,k} - \lambda \cdot p_{u,k})$
- 21: **if** $k \neq 2$: $q_{i,k} \leftarrow q_{i,k} + \eta \cdot (predictionError \cdot p_{u,k} - \lambda \cdot q_{i,k})$
- 22: **end for**
- 23: **end while**

4.3 Prediction Assembler

Prediction assembler aggregates rating predictions from all co-trainers into a single value. We propose three ways of calculating this aggregation.

Recall-based Prediction Assembler assembles predictions of N co-trainers using a weighted average with weights depending on their past recall values.

$$\hat{r}_{u,iAgg} = \frac{\sum_{j=0}^N \text{recall}(Co - Tr_j) \cdot \hat{r}_{u,iCo-Tr_j}}{\sum_{j=0}^N \text{recall}(Co - Tr_j)} \quad (8)$$

In the above formula recall is measured globally for an entire co-trainer. Alternatively, recall can be measured also on user or item level. In this case $\text{recall}(Co - Tr_j)$ can be substituted with $\text{recall}(Co - Tr_j, u)$ or $\text{recall}(Co - Tr_j, i)$.

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.

$$\hat{r}_{u,iAgg} = \frac{\sum_{j=0}^N \text{RMSE}(Co - Tr_j) \cdot \hat{r}_{u,iCo-Tr_j}}{\sum_{j=0}^N \text{RMSE}(Co - Tr_j)} \quad (9)$$

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

$$\hat{r}_{u,iAgg} = \frac{\sum_{j=0}^N rel_{Co-Tr_j}^{\hat{r}_{u,i}} \cdot \hat{r}_{u,iCo-Tr_j}}{\sum_{j=0}^N rel_{Co-Tr_j}^{\hat{r}_{u,i}}}. \quad (10)$$

4.4 Selector of Unlabelled Instances

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

Latent Disagreement Selector. For all known users each co-trainer stores a latent vector that is specific for this co-trainer. 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:

$$disagreement(Co-Tr_a, Co-Tr_b, u) = |p_u^{Co-Tr_a} - p_u^{Co-Tr_b}| \quad (11)$$

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 analogously applied onto latent item vectors.

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

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.

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:

$$\Delta_{p_u} = \sum_{i=0}^k (p_{u,i}^{t+1} - p_{u,i}^t)^2 \quad (12)$$

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).

Popularity-based Reliability Measure. Zhang et al. proposed in [11] 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).

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

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:

$$f(\text{reliability}) = \frac{1}{1 + e^{\alpha \cdot (\text{reliability} - \mu)}} \quad (13)$$

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.

5 Evaluation Setting

Incremental matrix factorization algorithm works on a stream of ratings, however, it requires a short initialization phase that also needs to be taken into account, while evaluating our framework. Therefore, a simple split into a training and test datasets is not sufficient. Also peculiarities of a stream based evaluation should be considered. In [8] we developed an evaluation framework suitable for this type of algorithms. We adopt it also here to measure the performance of our SSL algorithms and explain it shortly in this section.

5.1 Evaluation Protocol

Our evaluation protocol consists of two modes that are presented in Fig. 5. The figure represents an entire dataset and splitting of it into the two aforementioned modes, as well as into training and test datasets within the modes. The first mode is a batch mode used for initial training and tuning (blue colour in Fig. 5). The batch mode itself consists of two further parts. Part (1) of a dataset is used as initial training set. Part (2) is used for testing of the initial training and for adjusting parameters of the model. This part is crucial especially for matrix factorization algorithms using stochastic gradient descent. Once the initial training is finished, the algorithm switches into the stream mode (yellow part). From this moment on, evaluation and training are performed simultaneously as proposed by Gama et al. in prequential evaluation [2].

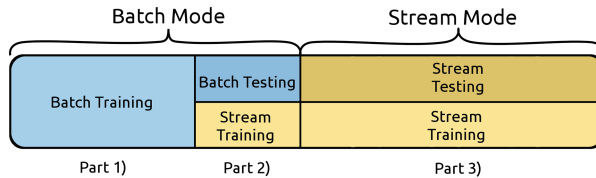


Fig. 5. Splitting of the dataset between the batch and streaming mode. Separation of training and test datasets in each of the modes [9] (Color figure online).

In Fig. 5 we can see that part (1) and (3) are user for learning and part (2) for testing in batch mode. Excluding part (2) from learning would create a temporal gap in the stream. This gap can be problematic for many incremental methods that rely on time aspects and are sensitive to ordering of data instances. In order to avoid this gap, we use part (2) for learning in the streaming mode as well, but not for testing, since it has been used for batch testing already. Testing in the stream mode starts in part (3).

In our framework the streaming mode is the main part of the preference learning, where SSL methods are used. Therefore, our results refer always to the streaming mode of an algorithm. To investigate the effect of SSL, we compare results of **exrBRISMf with the SSL setting and without it.**

5.2 Evaluation Measure - Incremental Recall

In our experiments we use an incremental recall measure proposed by Cremonesi et al. [1]. In the incremental setting precision can be derived from incremental recall (cf. [1]) and, therefore, we do not present it. In contrast to purely rating-based measures, such as RMSE or MAE, the incremental recall measure also considers ranking of a predicted item. Another problem of RMSE and MAE is that they weight all predictions uniformly, regardless of relevance of items, i.e. predictions on all irrelevant items (typically > 99%) influence the error measure

equally strong as predictions on relevant items. In incremental recall the ranking of relevant items only counts.

The procedure of measuring *incrementalRecall@N* is as follows. At each new rating in a stream the relevance of the corresponding item is determined using a rating threshold (e.g. $r_{ui} > 4$ is considered relevant). For each relevant item additional 1000 random irrelevant items, are selected. For all of those items rating predictions are made and sorted. Finally, the rank p of the relevant item among the irrelevant ones is determined. If the rank p is lower than N , a hit is counted. The value of *incrementalRecall@N* is set to $\frac{\#hits}{|Testset|}$.

6 Experiments

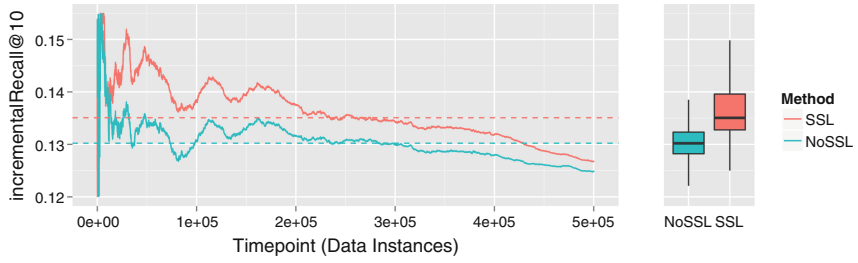
To show the effect of our SSL framework we compared the results of the extBRISMf algorithm alone (abbreviated hereafter as **NoSSL**) and extBRISMf with our SSL framework (abbrv. as **SSL**). Both, the algorithm and the framework require setting parameters, such as learn rate η in gradient descent, etc. Therefore, in order to find approximately best parameter setting we performed a grid search in the parameter space. The grid search was performed on a cluster running the (Neuro) Debian operating system [3]. In total we conducted more than 350 experiments. For brevity we present here only the best result achieved by the SSL and NoSSL method on each dataset.

Datasets. In our experiments we use four real-world datasets from the recommender systems domain. We stress out that our framework is applicable to all datasets in form of a user-item-rating matrix unlike similar SSL frameworks that rely on external sources of information. The datasets we used encompass MovieLens 100k, MovieLens 1M¹ datasets, as well as random samples of 1000 users of the Netflix² and Epinions (extended) [7] datasets. We used sampling on the large datasets due to a large numbers of experiments in the grid search. The percentage of labelled data out of all possible user-item-pairs in those datasets amounts to values between 0.03% and 6.3%. This shows how much of the unlabelled information is available in the process of preference learning.

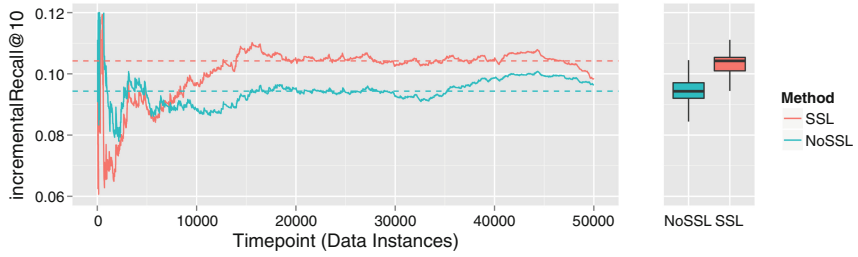
Results. Figure 6 shows the incremental recall@10 over time on the vertical axis (higher values are better) and the time dimension on the horizontal axis. The red curves represent the SSL method, whereas the blue ones stand for the NoSSL method. The dashed lines in colours of both curves represents the median value of the incremental recall. They correspond to the medians in the boxplots in the right part of the figures that visualize the distribution of incremental recall in a simplified way. In all parts of Fig. 6 we can see that the **SSL method dominates the NoSSL** one at nearly all time points. More precise results with the corresponding settings are presented in Table 1. The columns 2–5 of the table represent the components (e.g. a reliability estimator based on user popularity in the first row). Rows with no components, but with “NoSSL” stand for the

¹ www.movielens.org.

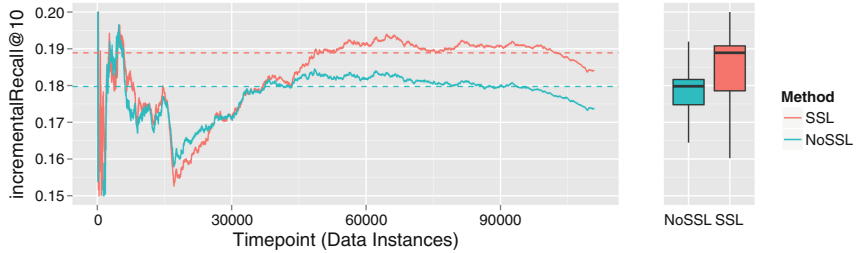
² www.netflix.com.



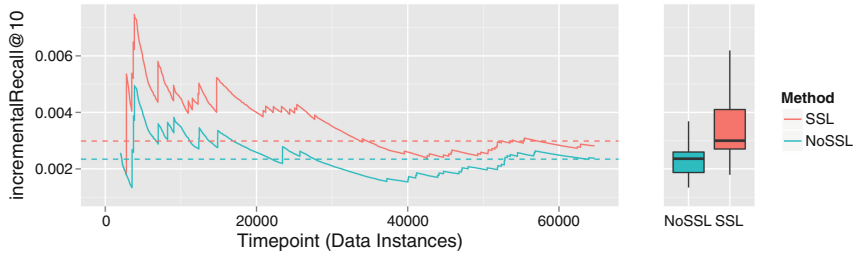
(a) MovieLens 1M



(b) MovieLens 100k



(c) Netflix (random sample of 1000 users)



(d) Epinions extended (random sample of 1000 users)

Fig. 6. *Incremental Recall* on four real datasets (higher values are better). Application of SSL techniques yields an improvement on all datasets at nearly all time points (Color figure online).

Table 1. Average incremental recall@10 and computation time for a single instance. Our SSL framework is marked in blue.

Dataset	Reliability Estimator	Prediction Assembler	Unlabelled Instance Selector	Training Set Splitter	Avg. Incremental Recall@10	Avg. Time for Instance (ms)
ML100k	User Popularity	User Recall	Latent User Disagreement	Random	0.101235	2.417228
	NoSSL				0.095099	0.138642
ML1M	Sensitivity Global	User Recall	Latent User Disagreement	Dimensions Preserving	0.136564	10.88977
	NoSSL				0.130721	0.250437
Netflix 1000 users	Sensitivity Global	RMSE Global	Latent Item Disagreement	Dimensions Preserving	0.184150	6.888166
	NoSSL				0.177380	0.400382
Epinions 1000 users	Sensitivity Global	RMSE Global	Latent User Disagreement	User Size	0.003312	131.7290
	NoSSL				0.002289	0.979214

extBRISMF alone with no SSL used. The sixth column contains the average incremental recall@10 for each of the setting. Best results are marked in bold. Also here we can recognize that the SSL setting dominated the NoSSL one on all datasets. From the components there are no clear winners, except for latent disagreement instance selectors, which performed the best on all datasets. From reliability measures the sensitivity-based ones were mostly successful.

The last column in Table 1 contains the average runtime for a single instance in milliseconds. We observed that the computation time increased considerably, when using SSL. Nevertheless, the runtime still stayed in the range of a few milliseconds, except for the Epinions dataset with 131 ms, which is still feasible in real-world applications. Remaining settings used in the experiments are the regularization parameter $\lambda = 0.01$, number of latent dimensions $k = 40$ and learn rate $\eta = 0.003$. The framework used USL every $m = 50$ ratings, where $z = 100$ unlabelled instances were selected. Although our framework was developed for an arbitrary number of co-trainers, in this work we used two of them. Experiments with a larger number of co-trainers are part of our future work.

7 Conclusions

In this work we proposed a semi-supervised framework for stream recommender systems based on the co-training approach. To our knowledge, it is the first such framework that can deal with a stream of ratings and incremental algorithms. Within the framework we proposed several generic components including training set splitter, reliability measures, prediction assemblers and selectors for unlabelled instances. For each of those components we developed several instantiations e.g. sensitivity-based reliability measure, latent disagreement-based instance selector and many more. Furthermore, we extended the BRISMF algorithm [10] by the ability to extend the dimensions of the matrix incrementally.

In experiments on four real datasets we showed that **our SSL framework outperforms the non-SSL method at nearly all time points**. This is,

because our framework is able to leverage the unlabelled information, which in recommender systems is abundant. Using this information allows us alleviate the problem of sparsity even without using any context information. The improvement is, however, at the cost of longer computation time. Nevertheless, the computation time of a single instance still remains in a range of a few milliseconds (normally around 6 ms, except for Epinions dataset - ca. 131 ms). Therefore, this framework applicable in real-world recommenders.

Our immediate next steps are to investigate, how to make this framework faster by e.g. sharing parts of the matrix among co-trainers and by using efficient data structures. Furthermore, we plan to experiment with more than two co-trainers and to implement further instances of the aforementioned components.

Acknowledgements. The authors would like to thank Daniel Kottke and Dr. Georg Krempl for suggestions regarding self-tuning normalization for streams and also the Institute of Psychology II at the University of Magdeburg for making their computational cluster available for our experiments.

References

1. Cremonesi, P., Koren, Y., Turrin, R.: Performance of recommender algorithms on top-N recommendation tasks. In: *RecSys 2010*. ACM (2010)
2. Gama, J., Sebastião, R., Rodrigues, P.P.: Issues in evaluation of stream learning algorithms. In: *KDD*. ACM (2009)
3. Halchenko, Y.O., Hanke, M.: Open is not enough. Let's take the next step: an integrated, community-driven computing platform for neuroscience. *Front. Neuroinform.* **6**, 1–4 (2012)
4. Karimi, R., Freudenthaler, C., Nanopoulos, A., Schmidt-Thieme, L.: Towards optimal active learning for matrix factorization in recommender systems. In: *ICTAI*, pp. 1069–1076. IEEE (2011)
5. Koren, Y.: Collaborative filtering with temporal dynamics. In: *KDD 2009*. ACM (2009)
6. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. *Computer* **42**(8), 30–37 (2009)
7. Massa, P., Avesani, P.: Trust-aware bootstrapping of recommender systems. In: *ECAI Workshop on Recommender Systems*, pp. 29–33. Citeseer (2006)
8. Matuszyk, P., Spiliopoulou, M.: Selective forgetting for incremental matrix factorization in recommender systems. In: Džeroski, S., Panov, P., Kocev, D., Todorovski, L. (eds.) *DS 2014*. LNCS, vol. 8777, pp. 204–215. Springer, Heidelberg (2014)
9. Matuszyk, P., Vinagre, J., Spiliopoulou, M., Jorge, A.M., Gama, J.: Forgetting methods for incremental matrix factorization in recommender systems. In: *Proceedings of the SAC 2015 Conference*. ACM (2015)
10. Takács, G., Pilászy, I., Németh, B., Tikk, D.: Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res.* **10**, 623–656 (2009)
11. Zhang, M., Tang, J., Zhang, X., Xue, X.: Addressing cold start in recommender systems: a semi-supervised co-training algorithm. In: *SIGIR*. ACM (2014)
12. Zhou, Z.-H., Li, M.: Semisupervised regression with cotraining-style algorithms. *IEEE Trans. Knowl. Data Eng.* **19**(11), 1479–1493 (2007)