Addressing Cold Start in Recommender Systems: A Semi-supervised Co-training Algorithm

Mi Zhang[1,2] Jie Tang[3] Xuchen Zhang[1,2] Xiangyang Xue[1,2]
[1]School of Computer Science, Fudan University
[2]Shanghai Key Laboratory of Intelligent Information Processing
[3]Department of Computer Science and Technology, Tsinghua University
{mi_zhang,13210240075,xyxue}@fudan.edu.cn, jietang@tsinghua.edu.cn

ABSTRACT

Cold start is one of the most challenging problems in recommender systems. In this paper we tackle the cold-start problem by proposing a context-aware semi-supervised co-training method named CSEL. Specifically, we use a factorization model to capture fine-grained user-item context. Then, in order to build a model that is able to boost the recommendation performance by leveraging the context, we propose a semi-supervised ensemble learning algorithm. The algorithm constructs different (weak) prediction models using examples with different contexts and then employs the co-training strategy to allow each (weak) prediction model to learn from the other prediction models. The method has several distinguished advantages over the standard recommendation methods for addressing the cold-start problem. First, it defines a fine-grained context that is more accurate for modeling the user-item preference. Second, the method can naturally support supervised learning and semi-supervised learning, which provides a flexible way to incorporate the unlabeled data.

The proposed algorithms are evaluated on two real-world datasets. The experimental results show that with our method the recommendation accuracy is significantly improved compared to the standard algorithms and the cold-start problem is largely alleviated.

Categories and Subject Descriptors

H.4.m [**Information Systems Applications**]: Miscellaneous

Keywords

Cold-start; Recommendation; Semi-supervised Learning

1. INTRODUCTION

Recommendation plays an important role in many fields and has attracted a lot of research interest. For example, Netflix has released an interesting fact that about 75% of its subscribers watch are from recommendations. In a recommender system such as Netflix and Amazon, users can browse items and choose those items they are interested in, while the system would also recommend to them the items that the system thought best match their preferences. Afterward, the user may provide feedback (such as rating, usually represented as a score between, for example, 1 and 5) on how the user thinks about an item after she/he has experienced the item. One important task for the recommendation engine is to understand users' personalized preferences from their historic rating behaviors.

SIGIR'14, July 6–11, 2014, Gold Coast, Queensland, Australia.
Copyright 2014 ACM 978-1-4503-2257-7/14/07 ...$15.00.
http://dx.doi.org/10.1145/2600428.2609599.

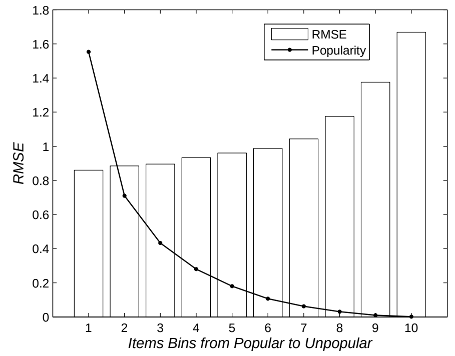

Figure 1: Average popularity and RMSE on items with different popularity. The dataset is from MovieLens (D_1', Cf. Section 6.1 for details). We estimate the popularity of each item based on the number of ratings and partition all the items into 10 bins according to their popularity with equal number. The average RMSE scores are obtained using a standard collaborative filtering approach (Cf. Eq. 1).

Another important, and actually more challenging task is how to improve the recommendation accuracy for the new (or rarely rated) items and the new (or inactive) users. Comparing to the popular items, for the newly released ones and the old items that are rarely rated by users, it is difficult for the standard recommendation approaches such as collaborative filtering approach to provide high-quality recommendations. Figure 1 shows some preliminary results in our experiments. The recommendation error (by mean-square error, i.e., RMSE) increases quickly with the decrease of popularity of the item. The average error of the most unpopular items (Bin_{10}) almost doubles that of the popular items (Bin_1, Cf. Table 2 for details). The problem also exists for the newly entered users or the inactive users who have not contributed enough ratings. Technically, this problem is referred to as *cold start*. It is prevalent in almost all recommender systems, and most existing approaches suffer from it [22].

Despite that much research has been conducted in this field, the cold-start problem is far from solved. Schein [22] proposed a method by combining content and collaborative data under a single probabilistic framework. However, their method is based on Bayes classifier, which cannot accurately model the user's fine-grained preference. Lin et al. [17] addressed the cold-start problem for App recommendation. They use the social information from Twitter to help App recommendation. The method is effective for dealing with the problem as it introduces external social information for building the recommendation model. However, if we deepen the analysis, we can easily find that for building a practical and accurate recommendation model, the available (useful) information is usually from different sources. A single model would be ineffective in this sense. Thus the question is, how to build models using the information of different sources and, more importantly, how the different models can help each other? Besides, there are also some other challenges in the cold-start problem. For example, since the available labeled data from different sources is often limited, it is important to develop a semi-supervised learning method so as to leverage the unlabeled data.

Solution and Contributions. In this paper, we aim to conduct a systematical investigation to answer the above questions. In order to capture users' preferences, a fine-grained context-aware approach is proposed which incorporates additional sources of information about the users and items rather than users' rating information only. Furthermore, we propose a semi-supervised co-training method to build the recommendation model. The model not only is able to make use of unlabeled data to help learn the recommendation model, but also has the capacity to build different sub models based on different views (contexts). The built sub models are then combined by an ensemble method. This significantly alleviates the cold-start problem.

We evaluate the proposed model on a publicly available dataset, MovieLens. The results clearly demonstrate that our proposed method significantly outperforms alternative methods in solving the cold-start problem. The overall RSME is reduced by 3-4%, and for the cold-start users and items the prediction accuracy is improved up to 10% ($p < 0.05$ with t-test). Beyond accurate recommendation performance, our method is also insensitive to parameter tuning as confirmed in the sensitivity analysis. Finally, we use several case studies as the anecdotal evidence to further demonstrate the effectiveness of our method. To summarize, contributions of this work include:

- By providing an in-depth analysis about the existing algorithms, we propose a fine-grained modeling method to capture the user-item contexts;
- We propose a semi-supervised co-training method named CSEL to leverage the unlabeled examples. With the context-aware model given beforehand, CSEL is able to construct different regressors by incorporating different contexts, which ideally helps capture the data's characteristics from different views.
- Our empirical study on real-world datasets validates the effectiveness of the proposed semi-supervised co-training algorithm.

Organization. Section 2 formulates the problem; Section 3 presents a fine-grained user-item context model; Section 4 describes the proposed semi-supervised co-training algorithm (CSEL); Section 5 discusses related work; finally, Section 6 presents the experimental results and Section 7 concludes the work.

2. OVERVIEW

Here we present required definitions and formulate the cold-start problem in recommender systems.

Let $\mathcal{I}$ be the set of items and $\mathcal{U}$ be the set of users in the system. We use r_{ui} to denote a rating that user $u \in \mathcal{U}$ gives to the item $i \in \mathcal{I}$, and use $L = \{r_{ui}\} \subset \mathcal{I} \times \mathcal{U}$ to denote the set of all ratings. Further let $U = \{x_{ui}\}$ denote all the unknown ratings, i.e., $U = \mathcal{I} \times \mathcal{U} - L$. We use the notation $|.|$ to denote the cardinality of a set, for example $|L|$ indicates the number of ratings in the set L. In addition, each user/item may have some attributes. For example, a user may have gender, age and other attributes and an item may be associated with multiple genres. In this work gender, age and occupation are referred to as the *context of users*, whereas genres are referred to as the *context of items*. The goal of the cold-start recommendation is to leverage all the available information to learn a recommendation function f so that we can predict the rating of user u for item i, i.e.,

$$f((u,i)|L,U) \rightarrow r_{ui}$$

Here in the definition, if the rating r_{ui} is already available, i.e., $r_{ui} \in L$, the learning objective is to minimize the difference of the estimated rating and the real rating $|\hat{r}_{ui} - r_{ui}|$, otherwise the goal is to predict the most possible rating for $x_{ui} \in U$ with high accuracy.

In the following, for an item $i \in \mathcal{I}$ let d_i be i's popularity, which is obtained by $d_i = \frac{m_i}{m}$, where m_i is the number of users who have rated i and m is the number of users in the recommender system. For a user $u \in \mathcal{U}$, let d_u be u's popularity, which is obtained by $d_u = \frac{n_u}{n}$, where n_u is the number of items rated by u and n is the total number of items.

2.1 Approach Overview

To deal with a general recommendation problem, we can consider the standard factorization algorithm [15] as a baseline model. It is a *model-based collaborative filtering* (CF) algorithm proposed by the winning team in the Netflix competition [1], which is designed for improving the prediction accuracy of the Netflix movie recommender system. This algorithm can be regarded as a *state-of-the-art* algorithm. The model is defined as

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \quad (1)$$

Here, the observed rating is factorized into four components: global average μ, item bias b_i, user bias b_u, and user-item interaction $q_i^T p_u$ that captures the user u's personalize preference on item i. By minimizing the difference of the real ratings and the factorized ratings in the labeled training data, we can estimate the unknown parameters $(\{b_u\}, \{b_i\}, \{q_i\}, \{p_u\})$ in Eq. 1. This model is referred to as **FactCF** in the rest of the paper.

However, the standard CF approach cannot deal with the cold-start problem. As shown in Figure 1, the recommendation error increases quickly when directly applying the standard approach to unpopular items.

In this paper, we propose a novel semi-supervised co-training approach, i.e., **CSEL**, to address the cold-start problem. Specifically, at the high-level, the approach consists of two stages:

- **Context-aware factorization.** We propose a fine-grained context model by adding in the contexts of users and items, specifically, the interactions between them into the model. The model is the basis to deal with the cold-start problem;
- **Semi-supervised Co-training.** Based on the context modeling results, a semi-supervised learning framework is built for addressing the cold-start problem in recommender systems, where multiple models are constructed using different

[1]http://www.netflixprize.com/leaderboard

information and then co-training is applied so that the learned models can help (teach) each other. This is the major technical contribution of this work. Besides the specific models used in this work, the proposed framework is very flexible and can accommodate any kind of models without any extra information.

3. CONTEXT-AWARE FACTORIZATION

Often in a recommender system, many users supply very few ratings, making it difficult to reach general conclusions on their taste. One way to alleviate this problem is to incorporate additional sources of information, for example, recommender systems can use the attributes of the users and items to build a *context* of the user preference. Following this thread, we propose a context-aware factorization model by leveraging "more general" sources such as age and gender of a user, or the genres and tags of an item that are independent of ratings. Saying that they are "more general" than the ratings is in a sense that a rating is given by a specific user to a specific item, whereas these sources can be learned from all the users or items that share the same category. In this section, based on the FactCF algorithm given in Eq. 1 we propose a context-aware model that incorporates the context information into the model.

Here we start enhancing the model by letting item biases share components for items linked by the genres. For example, items in a certain genre may be rated somewhat higher than the average. We therefore add shared bias parameters to different items with a common information. The expanded model is as follows

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u + \frac{1}{|genres(i)|} \sum_{g \in genres(i)} b_g, \quad (2)$$

then the total bias associated with an item i sums both its own specific bias modifier b_i, together with the mean bias associated with its genres $\frac{1}{|genres(i)|} \sum_{g \in genres(i)} b_g$. One could view these extensions as a gradual accumulation of the biases. For example, when modeling the bias of i, the start point is $\frac{1}{|genres(i)|} \sum_{g \in genres(i)} b_g$, and then b_i adds a residual correction on the top of this start point. A similar method has been also used for music recommendations [7].

The second type of "general" information we incorporate into the model is the context (attributes) of users such as age and gender. For example, we can enhance Eq. 2 as follows:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u + \frac{\sum_{g \in genres(i)} b_g}{|genres(i)|} + b_a + b_o + b_s \quad (3)$$

where b_a, b_o and b_s are the biases associated with the user's age, occupation and gender, respectively. Moreover, we propose mixing the user and item's contexts as below to obtain a further optimization:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u + \frac{\sum_{g \in genres(i)} b_{ug}}{|genres(i)|} + b_{ia} + b_{io} + b_{is} \quad (4)$$

where b_{ug}, b_{ia}, b_{io} and b_{is} are the parts mixing the contexts. Specifically, b_{ug} is user u's bias in genre g, i.e., b_{ug} catches u's preference in g. We can use a stochastic gradient descent (SGD) algorithm to learn the parameters. Compared to b_g that is independent of specific users, here b_{ug} is only updated when the current rating (in L) used for training is given by u. Likewise, b_{ia}, b_{io} and b_{is} are item i's preferences by the users in the categories of a, o and s, respectively, and they'll be updated in the learning process when the rating is associated with i and the users are in the corresponding categories.

Model Learning. By combining all the context information, we can define an objective function to learn the context model by minimizing the prediction errors over all the examples in L.

$$\min_{p_*, q_*, b_*} \sum_{(u,i) \in K} (r_{ui} - \hat{r}_{ui})^2 + \lambda(\|p_u\|^2 + \|q_i\|^2 + \sum b_*^2) \quad (5)$$

where $\hat{r}_{ui}$ can be any model defined in Eqs. 2-4; $||\cdot||$ indicate the 2-norm of the model parameters and λ is the regularization rate. The objective function can be solved by the stochastic gradient descent (SGD). SGD processes the training examples one-by-one, and update the model parameters corresponding to each example. More specifically, for training example r_{ui}, SGD lowers the squared prediction error $e_{ui}^2 = (r_{ui} - \hat{r}_{ui})^2$ by updating each individual parameter θ by

$$\Delta\theta = -\gamma \frac{\partial e_{ui}^2}{\partial \theta} - \lambda\theta = 2\gamma e_{ui} \frac{\partial \hat{r}_{ui}}{\partial \theta} - \lambda\theta$$

where γ is the learning rate. Thus, the parameters are updated by moving in the opposite direction of the gradient, yielding:

$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$$

$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$$

$$b_* \leftarrow b_* + \gamma \cdot (e_{ui} - \lambda \cdot b_*)$$

For each type of learned parameter we set a distinct learning rate and regularization rate. This grants us the flexibility to tune learning rates such that, e.g., parameters that appear more often in a model are learned more slowly (and thus more accurately). Similarly, the various regularization coefficients allow assuming different scales for different types of parameters.

The model given by Eq. 4 turns out to provide a better performance than the other ones in terms of prediction accuracy therefore it is adopted in this work for the further optimization with semi-supervised co-training.

4. SEMI-SUPERVISED CO-TRAINING

Based on the learned contexts, we propose a semi-supervised co-training (CSEL) framework to deal with the cold-start problem in recommender systems. CSEL aims to build a semi-supervised learning process by assembling two models generated with the above context-aware model, in order to provide more accurate predictions. Specifically, CSEL consists of three major steps.

- **Constructing multiple regressors.** Since we are addressing a regression problem, the first step is to construct the two regressors, i.e. h_1 and h_2 from L, each of which is then refined with the unlabeled examples that are labeled by the latest version of their peer regressors.
- **Co-Training.** In the co-training, each regressor can learn from each other. More accurately, those examples with high confidences are selected for the regressor and being labeled (predicted) by it, and later to be used to "teach" the other regressors.
- **Assembling the results.** In the end the results obtained by the individual regressors are assembled to form the final solution.

```
Input: the training set $L$ with labeled (rated) examples, and the
       unlabeled (not rated) example set $U$;
Output: regressor $h(u,i)$ for the rating prediction;
Step 1: Construct multiple regressors;
Generate the two regressors $h_1$ and $h_2$ by manipulating the training
set (M$_s$) or manipulating the attributes (M$_v$ in Section 4.1);
Create a pool $U'$ by randomly picking examples from $U$;
Initialize the teaching sets $T_1 = \phi; T_2 = \phi$;
Step 2: Co-Training;
repeat
    for $j = 1$ to 2 do
        foreach $x_{ui} \in U'$ do
            Obtain the confidence $\mathcal{C}_j(x_{ui})$ by Eq. 10:
            $\mathcal{C}_j(x_{ui}) = \frac{d_u^{(j)} \times d_i^{(j)} \times \prod_{c \in G,O,A,S} d_c^{(j)}}{\mathcal{N}}$;
        end
        Select $N$ examples to form $T_j$ by the Roulette algorithm
        with the probability $Pr(x_{ui}, j)$ given by Eq. 11:
        $T_j \leftarrow Roulette(Pr(x_{ui}, j))$;
        $U' = U' - T_j$;
    end
    Teach the peer regressors: $L_1 = L_1 \cup T_2; L_2 = L_2 \cup T_1$;
    Update the two regressors: $h_1 \leftarrow L_1; h_2 \leftarrow L_2$;
until $t$ rounds;
Step 3: Assembling the results (with the methods in Section 4.3);
$h(u,i) = Assemble(h_1(u,i), h_2(u,i))$.
```
Algorithm 1: Semi-supervised Co-Training (CSEL).

Algorithm 1 gives the algorithm framework. In the following more details will be given about the methods of constructing multiple regressors, constructing the teaching sets, and assembling. Note that in principle, the algorithm can be easily extended to accommodate multiple regressors (more than two). In this work, for the sake of simplicity we focus on two regressors.

4.1 Constructing Multiple Regressors

Generally speaking, in order to generate different learners for ensemble, one way is to train the models with different examples by manipulating the training set, while the other way is to build up different views by manipulating the attributes [6], which are both adopted in this paper to generate multiple regressors for co-training.

Constructing Regressors by Manipulating the Training Set. A straightforward way of manipulating the training set is Bagging. In each run, Bagging presents the learning algorithm with a training set that consists of a sample of k training examples drawn randomly with replacement from the original training set. Such a training set is called a *bootstrap* replicate of the original training set.

In this work two subsets are generated with the Bagging method from the original training set, and two regressors are trained on the two different subsets, respectively, for the purpose of working collaboratively in the co-training process. The basic learner for generating the regressors could be the standard factorization model (Eq. 4), i.e.,

$$h_1(u,i) = h_2(u,i) = \mu + b_u + b_i + q_i^T p_u + \frac{\sum_{g \in genres(i)} b_{ug}}{|genres(i)|} + b_{ia} + b_{io} + b_{is} \quad (6)$$

Some related work tries to use *diverse* regressors to reduce the negative influence of the newly labeled noisy data [30]. Our experiments also showed that a good diversity between the regressors indeed helps much in improving the performance of co-training. Thus it is an important condition for generating good combinations of regressors. Intuitively the diversity between the two regressors generated in this way are coming from the different examples they are trained on, and can be evaluated by their difference on predictions. Moreover, another important condition to make a good ensemble is, the two regressors should be *sufficient* and *redundant*. Then being translated to this case, it means each training set should be sufficient for learning, respectively and the predictions made by the two individual regressors should be as accurate as possible.

However, the above two conditions, diversity and sufficiency, are contradictive to each other. That is, let the size of the two subsets generated with the Bagging method both be k, then with the original training set size fixed, when k is big, the overlap between the two subsets will be big as well, which will result in a small diversity between regressors. In the mean time the accuracy performance of individual regressors will be good with a big set of training examples. And vice versa when k is small. Thus in the experiments we need to find an appropriate value of k for the trade-off between these two criteria. This method is referred to as $\mathbf{M}_s$ hereafter.

Constructing Regressors by Manipulating the Attributes. Another way to generate multiple regressors is to divide attributes into multiple *views*, such as

$$h_1(u,i) = \mu + b_u + b_i + q_i^T p_u + \frac{1}{|genres(i)|} \sum_{g \in genres(i)} b_{ug} \quad (7)$$

$$h_2(u,i) = \mu + b_u + b_i + q_i^T p_u + b_{ia} + b_{io} + b_{is} \quad (8)$$

The two regressors are both a part of the model given in Eq. 4. Unlike M$_s$, here the regressors are both trained on the entire training set. To guarantee enough diversity between the constructed regressors, the contexts are separated, specifically, user-related contexts and item-related contexts are given to the different regressors. Further, in order to keep a good performance in terms of prediction accuracy (with respect to the sufficient condition) for the individual regressors, the common part, $\mu + b_u + b_i + q_i^T p_u$, is kept for both regressors. This method is referred to as $\mathbf{M}_v$ hereafter.

Constructing Regressors by a Hybrid Method. As described above, M$_s$ is to train a same model on different subsets whereas M$_v$ is to construct different models and let them be trained on a single set. The hybrid method is to construct different regressors as well as training them on the different subsets of the attributes. Obviously, it will bring in more diversity between the regressors. The hybrid method is referred to as $\mathbf{M}_{sv}$.

4.2 Semi-supervised Co-training

Now with the multiple regressors at hand, the task becomes how to co-train the different regressors. Specifically, we need to construct a "*teaching set*" T_j for each regressor j and use it to teach its peer (the other regressors). This is a key step for launching the semi-supervised learning (SSL). One challenge here is how to determine the criteria for selecting unlabeled examples from U' so as to build the teaching set. The criteria we used in this work is the regressor's confidence. Every candidate example has a specific confidence value to reflect the probability of its prediction to be accurate. As given in the algorithm, before SSL processes a smaller set $U' \subset U$ is drawn randomly from U because U could be huge.

Confidence for the FactCF Model. In many cases the user can benefit from observing the confidence scores [11], e.g. when the system reports a low confidence in a recommended item, the user may tend to further research the item before making a decision. However, not every regression algorithm can generate a confidence by itself, like the factorization regressor this work is based on. Thus it is needed to design a confidence for it.

In [30] the authors proposed the predictive confidence estimation for the kNN regressors. The idea is that the most confidently labeled example of a regressor should decrease most the error of the regressor on the labeled example set, if it is utilized. However, it requires the example to be merged with the labeled data and re-train the model, and this process needs to be repeated for all candidate examples. Similarly, in [10] another confidence measure was proposed, again, for the kNN regressor. The idea is to calculate the conflicts level between the neighbors of the current item for the current user. However, the above two methods are both based on the k nearest neighbors of the current user and cannot be directly applied to our model. In the following we define a new confidence measure that suits our model.

Confidence in the recommendation can be defined as the system's trust in its recommendations or predictions [11]. As we have noted above, collaborative filtering recommenders tend to improve their accuracy as the amount of data over items/users grows. In fact, how well the factorization regressor works directly depends on the number of labeled examples (ratings) that is associated with each factor in the model. And the factors in the model pile up together to have an aggregate impact on the final results. Based on this idea, we propose a simple but effective definition for the confidence on a prediction made by regressor j for example x_{ui}:

$$\mathcal{C}_j(x_{ui}) = \frac{d_u^{(j)} \times d_i^{(j)}}{\mathcal{N}} \tag{9}$$

where $\mathcal{N}$ is the normalization term; $d_u^{(j)}$ and $d_i^{(j)}$ are the user u and item i's popularities associated with regressor j.

The idea is, the more chances (times) a factor gets to be updated during the training process, the more confident it is with the predictions being made with it. The rating number that is associated with the user-related factors b_u and p_u in the model is expressed by d_u, while the rating number that is associated with the item-related factors b_i and q_i is expressed by d_i. Here the assumption is that the effects of the parameters are accumulative.

Confidence for the Context-aware Models. With the similar idea the confidence for the context-aware models is defined as below:

$$\mathcal{C}_j(x_{ui}) = \frac{d_u^{(j)} \times d_i^{(j)} \times \prod_{c \in G,O,A,S} d_c^{(j)}}{\mathcal{N}} \tag{10}$$

where c stands for the general information including genre, age, gender, occupation, etc.; d_c represents the fraction of items that fall in a certain category c. For example, d_{g_1} is the fraction of items that fall in the genre "Fictions", d_{g_2} is the fraction of items that fall in the genre "Actions", etc., with $G = \{g_1, ..., g_{18}\}$ (there are 18 genres in the MovieLens dataset). Likewise, d_a, d_s, d_o, are associated with the categories of $A = \{a_1, ..., a_7\}$ (the entire range of age is divided into 7 intervals), $S = \{male, female\}$ and $O = \{o_1, ..., o_{20}\}$ (20 kinds of occupations), respectively, can be obtained in the same way.

Specifically, for the model with the item-view, i.e., h_1 in Eq. 7, the confidence for x_{ui} is $\mathcal{C}_1(x_{ui}) = d_u^{(1)} \times d_i^{(1)} \times \bar{d}_{gk}$, where $\bar{d}_{g_k} = \frac{1}{|genres(i)|} \sum_{g_k \in genres(i)} d_{g_k}$ and $|genres(i)|$ is the number of genres i belongs to. On the other hand, for the model with the user-view, i.e., h_2 in Eq. 8, the confidence is $\mathcal{C}_2(x_{ui}) = d_u^{(2)} \times d_i^{(2)} \times d_a \times d_s \times d_o$. This way the confidence depends on the view applied by the model.

With this definition, the two regressors (given by Eq. 6) being trained on different sample sets (M$_s$) will naturally have different d_u, d_i and d_c. E.g., for a user u, its $d_u^{(1)}$ for h_1 is the rating numbers of u being divided into the first sub-set, while its $d_u^{(2)}$ for h_2 is the rating numbers of u being divided into the second sub-set, and d_i likewise. Therefore, it will result in two different confidence values for these two models.

Note that in this work the confidence value of a prediction, $\mathcal{C}_j(x_{ui})$, is for selecting examples within a single model j, in order to steer the process of semi-supervised learning. Hence they are not comparable between different models. For the same reason the normalization part $\mathcal{N}$ is a constant within a model and thus can be omitted.

Constructing and Co-training with the Teaching Set. For each regressor j, when constructing the "teaching set" T_j with $\mathcal{C}_j(x_{ui})$, to avoid focusing on the examples that are associated with the most popular users and items that have relatively high $d_u^{(j)}$ and $d_i^{(j)}$, rather than directly selecting the examples with the highest confidence, we obtain a probability for each candidate example based on $\mathcal{C}_j(x_{ui})$ and select with the Roulette algorithm [1]. Specifically, the probability given to an example x_{ui} is calculated by:

$$Pr(x_{ui}, j) = \frac{\mathcal{C}_j(x_{ui})}{\sum_{x_k \in U'} \mathcal{C}_j(x_k)} \tag{11}$$

What's noteworthy is, in the experiments we found that simply selecting examples with $Pr(x_{ui}, j)$ sometimes cannot guarantee the performance. Moreover, we observed that a big diversity between the outputs of the two regressors is important for the optimization. The experiments show that for the selected examples, if the difference between the predictions made for them is not big enough the teaching effect can be trivial. This is consistent with the conclusion in [20] where the authors emphasize the diversity between the two learners are important. To guarantee a progress in SSL we propose to apply an extra condition in the teaching set selection. That is, only those examples with a big difference between the predictions made by the two regressors are selected. Specifically, when an example x_{ui} is drawn from U' with $Pr(x_{ui}, j)$, its predictions made by the two regressors are compared to a threshold τ. If the difference between them is bigger than τ then x_{ui} is put into T_j, otherwise it is discarded.

Next, the examples in the teaching set are labeled with the predictions made by the current regressor, likewise for its peer regressor. Note again that for the sake of diversity the teaching sets for the two regressors should be exclusive.

At last the regressors are updated (re-trained) with these new labeled data. In this way the unlabeled data are incorporated into the learning process. According to the results given in the experiments, by incorporating the unlabeled data the CSEL strategy provides a significant boost on the system performance. This process of constructing the teaching sets and co-training will be repeated for t iterations in the algorithm.

4.3 Assembling the Results

The final step of the algorithm is to assemble the results of the regressors after the optimization by SSL. A straightforward method is to take the average, i.e.,

$$h(u,i) = \frac{1}{l} \sum_{j=1..l} h_j(u,i) \tag{12}$$

where l is the number of regressors being assembled. In our case, to assemble results from two regressors it is $h(u,i) = \frac{1}{2}[h_1(u,i)+$

$h_2(u, i)]$. However, this method does not consider the confidence of the different regressors. To deal with this problem, we assemble the results by a weighted vote of the individual regressors. Here the confidence can be used as the weights of assembling, i.e.,

$$h(u, i) = \sum_{j=1..l} \frac{\mathcal{C}_j(x_{ui})}{\sum_{k=1..l} \mathcal{C}_k(x_{ui})} h_j(u, i) \tag{13}$$

Another option is to weight the regressors by ω_j that corresponds to regressor j's accuracy in the training set: $h(u, i) = \sum_{j=1...l} \omega_j h_j(u, i)$. It is trained with the loss function $\arg\min_{\omega_j} \sum_j \sum_{r_k \in L} (\omega_j h_j(u_k, i_k) - r_k)^2$ and addressed by linear regression. However, this method requires the model to be trained on all the labeled examples again and in the end it did not yield a better performance in our experiments. Some other options can also be adopted such as selecting the regressor h_j with the minimum errors in the training set, i.e., $h(u, i) \leftarrow \arg\min_j \sum_{r_k \in L} (h_j(u_k, i_k) - r_k)^2$, where r_k is a labeled example in L that is associated with u_k and i_k. This method also turned out to fail at providing the optimal results in our experiments.

Note that these ensemble methods can either be applied right after the training of individual models without semi-supervised learning, or after the semi-supervised learning. The former way makes use of the labeled data only, while the latter one makes use of both labeled and unlabeled data.

Moreover, the ensemble results can be used in different ways. That is, in each iteration the ensemble of the two newest regressors' predictions is obtained and used to label the examples in the teaching sets, rather than using their own predictions as described before. The reason is, firstly, according to our experiments the ensemble predictions always performs better than the individual results, which promises a bigger improvement for the algorithm; Secondly, since selecting the examples with high confidences only means the current regressor is more confident about them than the other examples, it does not say anything about its peer, which means if the original prediction is actually better noises will be brought into the peer regressors.

5. RELATED WORK

Cold-start is a problem common to most recommender systems due to the fact that users typically rate only a small proportion of the available items [21], and it is more extreme to those users or items newly added to the system since they may have no ratings at all [22]. A common solution for these problems is to fill the missing ratings with default values [5], such as the middle value of the rating range, and the average user or item rating. A more reliable approach is to use content information to fill out the missing ratings [4, 18]. For instance, the missing ratings can be provided by autonomous agents called filterbots [9], that act as ordinary users of the system and rate items based on some specific characteristics of their content. The content similarity can also be used "instead of" or "in addition to" rating correlation similarity to find the nearest-neighbors employed in the predictions [16, 23]. From a broader viewpoint, our problem is also related to the recommendation problem, which has been intensively studied in various situations such as collaborative recommendation [13] and cross-domain recommendation [26]. However, the cold-start problem is still largely unsolved.

Dimensionality reduction methods [2, 8, 14, 25] address the problems of limited coverage and sparsity by projecting users and items into a reduced latent space that captures their most salient features. There are also some more recent work that proposed some new methods to tackle the cold-start problem in different applications such as [17, 19, 24, 29]. Work [19] tries to address the cold-start problem as a ranking task by proposing a pairwise preference regression model and thus minimizing the distance between the real rank of the items and the estimated one for each user. Work [29] follows the idea of progressively querying user responses through an initial interview process. [24] is based on an interview process, too, by proposing an algorithm that learns to conduct the interview process guided by a decision tree with multiple questions at each split. [17] aims to recommend apps to the Twitter users by applying latent Dirichlet allocation to generate latent groups based on the users' followers, in order to overcome the difficulty of cold-start app recommendation. Unlike the above work that addresses the cold-start problem in some specific applications with specific extra sources, our work possesses more generosity. The framework for the CSEL algorithm proposed in this work can basically accommodate any kind of regressors without any extra information required.

The semi-supervised learning algorithm adopted in this paper falls in the category of disagreement-based SSL [31]. This line of research started from Blum and Mitchell's work [3]. Disagreement-based semi-supervised learning is an interesting paradigm, where multiple learners are trained for the task and the disagreements among the learners are exploited during the semi-supervised learning process. However, little work has been done making use of the semi-supervised techniques in the literature of recommender systems although it is a natural way to solve these problems. Following this line of research we propose a semi-supervised learning algorithm specifically designed for the factorization regressors that outperforms the standard algorithm in both the overall system performance and providing high quality recommendations for the users and items that suffer from the cold-start problem.

6. EVALUATION

6.1 Experimental Setup

Datasets. We evaluate our methods on a public available dataset, MovieLens, consisting of $100,000$ ratings (1-5) from 943 users on $1,682$ movies. In the experiments, the algorithms are performed at two stages of the system. The first stage is the early stage, i.e., when the system has run for 3 month. $50,000$ ratings are collected given by 489 users to $1,466$ items. The second stage is the late stage with the entire dataset collected during 7 months. The datasets at these two stages are referred to as D_1 and D_2, respectively.

The proposed methods are also evaluated on a different version of MovieLens dataset consisting of $1,000,000$ ratings provided by $6,040$ users for $3,900$ movies. The experiments are performed on two stages of it as well. The first stage is when the system has run for 3 months, by this time $200,000$ ratings are collected for $3,266$ items given by 1429 users. The second stage is again the full dataset collected in 34 months. They are referred to as D'_1 and D'_2 for this larger version MovieLens, respectively. The reason we did not do the evaluations on the Netflix dataset is, it only provides the ratings without any attributes or context information about users or items, which makes it impossible to perform the context-aware algorithm and the CSEL method by manipulating the attributes (M_v) proposed in this work.

Besides the historical ratings MovieLens datasets also provides some context information, including 19/18 genres associated with items (movies) and age, occupation and gender associated with users. Gender is denoted by "M" for male and "F" for female, age is given from 7 ranges and there are 20 types of occupations.

In the evaluation, the available ratings were split into train, validation and test sets such that 10% ratings of each user were placed in the test set, another 10% were used in the validation set, and the rest were placed in the training set. The validation dataset is used for early termination and for setting meta-parameters, then we fixed the meta-parameters and re-built our final model using both the train and validation sets. The results on the test set are reported.

Comparison Algorithms. We compare the following algorithms.

- FactCF: the factorization-based collaborative filtering algorithm given by Eq. 1 [15];
- UB k-NN and IB k-NN: the user-based and item-based k-nearest neighbor collaborative filtering algorithms;
- Context: the context-aware model we propose in Section 3, given by Eq. 4;
- Ensemble$_s$ and Ensemble$_v$: directly assembling the results of h_1 and h_2 generated by manipulating the training set (M$_s$) and manipulating the attributes (M$_v$ in Section 4.1), respectively. Among the options of ensemble method in Section 4.3, the weighted method in Eq. 13 gives a slightly better result than the other ones and thus being adopted.
- CSEL$_s$, CSEL$_v$ and CSEL$_{sv}$: the semi-supervised co-training algorithms using the h_1 and h_2 generated by M$_s$, M$_v$ and M$_{sv}$ (Cf. Section 4.1), respectively.

Besides the state-of-the-art FactCF algorithm we also use another standard algorithm, k-NN, to serve as a baseline. Two kinds of k-NN algorithms are implemented and compared to our methods. With user-based k-NN, to provide a prediction for x_{ui}, the k nearest neighbors of u who have also rated i are selected according to their similarities to u, and the prediction is made by taking the weighted average of their ratings. With item-based k-NN, the k nearest neighbors of i that have also been rated u are selected to form the prediction. Note that as another standard recommendation approach, the *content-based* algorithm has a different task from ours and thus not considered in this work. It aims to provide top-N recommendations with the best content similarities to the user, rather than providing predictions.

When training the models we mainly tuned the parameters manually and sometimes resorted to an automatic parameter tuner (APT) to find the best constants (learning rates, regularization, and log basis). Specifically, we were using APT2, which is described in [27]. All the algorithms are performed with 10-fold cross-validations.

Evaluation Measures. The quality of the results is measured by the root mean squared error of the predictions:

$$RMSE = \sqrt{\sum_{(u,i)\in TestSet} \frac{(r_{ui} - \hat{r}_{ui})^2}{|TestSet|}}$$

where r_{ui} is the real rating and $\hat{r}_{ui}$ is the rating estimated by a recommendation model.

6.2 Results

The overall performance on the four datasets are given in Table 1. The CSEL methods outperform both k-NN and FactCF algorithms in terms of the overall RMSE. Specifically, compared to FactCF the prediction accuracy averaged on all the test examples is improved by 3.4% on D_1, 3.6% on D_2, 3.2% on D'_1 and 3% on D'_2 respectively, with CSEL$_v$ performing the best among the methods. An unexpected result is that the hybrid method CSEL$_{sv}$ does not outperform CSEL$_v$. It is because the double splitting, i.e., on both the training set and views, leads to a bigger diversity (see Table 4) as well as worse accuracy for the regressors. As expected, FactCF outperforms k-NN on the overall performance by exploiting the latent factors. We tried different values of k from 10 to 100, $k = 20$ gives the best results. CSEL presents a much bigger advantage over the k-NN algorithms, i.e., 12% over UB and 16% over IB. The optimal solution is obtained by the CSEL$_v$ for all datasets.

Table 1: The overall RMSE Performance on the Four Datasets.

Models	D_1	D_2	D'_1	D'_2
UB k-NN	1.0250	1.0267	1.0224	0.9522
IB k-NN	1.0733	1.0756	1.0709	1.0143
FactCF	0.9310	0.9300	0.9260	0.8590
Context	0.9164	0.9180	0.9140	0.8500
Ensemble$_s$	0.9174	0.9170	0.9156	0.8490
Ensemble$_v$	0.9131	0.9153	0.9130	0.8452
CSEL$_s$	0.9013	0.9061	0.9022	0.8375
CSEL$_v$	**0.8987**	**0.8966**	**0.8963**	**0.8334**
CSEL$_{sv}$	0.9012	0.9020	0.9000	0.8355

Cold-start Performance. Except for the overall RMSE performance, we also present the recommendation performance according to the popularity of items. Specifically, we estimate the popularity of each item based on the number of ratings and partition all the items into 10 bins according to their popularity with equal number, the average RMSE scores are obtained for each bin. The results on these 10 bins over dataset D_2 are depicted in Table 2 corresponding to $Bin_i^{(1)}$ through $Bin_i^{(10)}$. Likewise, all the users are partitioned into 10 bins, the average RMSE scores are reported for each bin, corresponding to $Bin_u^{(1)}$ through $Bin_u^{(10)}$ in the table. The average rating number in each bin is given, too. With the users and items being grouped by their popularity we can observe how well the cold-start problem is solved. For the presented results, the differences between the algorithms are statistically significant ($p < 0.05$ with t-test). Due to space limitation, we do not list results on all the datasets, as according to the experimental results the behaviors of the algorithms on the four datasets (D_1, D_2, D'_1, and D'_2) turned out to be similar with each other.

Table 2 demonstrates how well the cold-start problem can be addressed by the proposed CSEL algorithm. The performance of the k-NN algorithms shows again that the cold-start problem is common to all kinds of recommendation algorithms, i.e., the items and users with less ratings receive less accurate predictions. The reason is that the effectiveness of the k-NN collaborative filtering recommendation algorithms depends on the availability of sets of users/items with similar preferences [12]. Compared to the baseline algorithms, CSEL successfully tackles the cold-start problem by largely improving the performance of the unpopular bins. Again, CSEL$_v$ provides the best results, compared to FactCF, RMSE drops up to 8.3% for the most unpopular user bins and 10.1% for the most unpopular item bins. Compared to UB k-NN, it drops up to 13.5% for the most unpopular user bins and 22.3% for the most unpopular item bins, respectively.

6.3 Analysis and Discussions

We further evaluate how the different components (context-aware factorization, ensemble methods, etc.) contribute in the proposed algorithm framework. In the following the performance of the proposed context-aware model and ensemble methods are given, and the results of CSEL are further analyzed.

Table 2: RMSE Performance of Different Algorithms on D_2. $Bin_u^{(1)}$ – the set of most active users; $Bin_u^{(10)}$ – the set of most inactive users. $Bin_i^{(1)}$ – the set of most popular items; $Bin_i^{(10)}$ – the set of most unpopular items.

D_2	Total RMSE	$Bin_u^{(1)}$	$Bin_u^{(2)}$	$Bin_u^{(3)}$	$Bin_u^{(4)}$	$Bin_u^{(5)}$	$Bin_u^{(6)}$	$Bin_u^{(7)}$	$Bin_u^{(8)}$	$Bin_u^{(9)}$	$Bin_u^{(10)}$
Rating #	106 (AVG#)	340	207	148	110	77	57	44	33	26	21
UB k-NN	1.0267	0.9816	0.9939	1.0191	1.0292	1.0696	1.0739	1.1043	1.1145	1.1381	1.1694
IB k-NN	1.0756	1.0156	1.0470	1.0536	1.0729	1.1041	1.1245	1.1396	1.1446	1.1852	1.2768
FactCF	0.9300	0.8929	0.9019	0.9100	0.9432	0.9789	0.9933	1.0291	1.0435	1.0786	1.0923
Context	0.9180	0.8797	0.8887	0.8920	0.9253	0.9503	0.9630	0.9861	1.0370	1.0573	1.0656
Ensemble$_s$	0.9170	0.8810	0.8867	0.8995	0.9280	0.9478	0.9522	0.9734	1.0100	1.0338	1.0448
Ensemble$_v$	0.9153	0.8836	0.8877	0.8983	0.9179	0.9451	0.9585	0.9760	0.9938	1.0268	1.0422
CSEL$_s$	0.9061	0.8779	0.8815	0.8928	0.9170	0.9282	0.9335	0.9714	0.9927	1.0054	1.0316
CSEL$_v$	0.8966	**0.8694**	0.8773	**0.8832**	**0.9078**	**0.9215**	**0.9223**	**0.9687**	0.9849	**0.9918**	**1.0023**
CSEL$_{sv}$	0.9020	0.8700	**0.8752**	0.8849	0.9100	0.9254	0.9367	0.9700	**0.9835**	0.9985	1.0153
D_2	Total RMSE	$Bin_i^{(1)}$	$Bin_i^{(2)}$	$Bin_i^{(3)}$	$Bin_i^{(4)}$	$Bin_i^{(5)}$	$Bin_i^{(6)}$	$Bin_i^{(7)}$	$Bin_i^{(8)}$	$Bin_i^{(9)}$	$Bin_i^{(10)}$
Rating #	60 (AVG#)	254	130	80	53	35	21	12	7	3	1
UB k-NN	1.0267	0.9677	0.9888	1.0089	1.0281	1.0592	1.0836	1.1003	1.1632	1.2977	1.3860
IB k-NN	1.0756	0.9865	1.0035	1.0360	1.0556	1.0930	1.1360	1.2608	1.4398	1.4742	1.5553
FactCF	0.9300	0.9071	0.9099	0.9300	0.9824	1.0344	1.0709	1.1066	1.1375	1.1531	1.1983
Context	0.9180	0.8871	0.9023	0.9177	0.9607	0.9980	1.0316	1.0649	1.1017	1.1171	1.1485
Ensemble$_s$	0.9170	0.8934	0.9050	0.9215	0.9656	0.9938	1.0384	1.0522	1.0957	1.1174	1.1242
Ensemble$_v$	0.9153	0.8819	0.8973	0.9163	0.9454	0.9740	1.0307	1.0576	1.1067	1.1098	1.1317
CSEL$_s$	0.9061	0.8821	0.8990	0.9184	0.9383	0.9572	1.0066	1.0166	1.0357	1.0661	1.0868
CSEL$_v$	0.8966	**0.8741**	**0.8907**	0.9168	**0.9235**	0.9561	**1.0008**	**1.0237**	**1.0454**	**1.0592**	**1.0768**
CSEL$_{sv}$	0.9020	0.8781	0.8920	**0.9151**	0.9243	**0.9537**	1.0076	1.0370	1.0470	1.0654	1.0800

Table 3: RMSE Performance of the evolving model. RMSE reduces while adding model components.

#	Models	D_1	D_2	D_1'	D_2'
1)	$\mu + b_u + b_i + q_i^T p_u$	0.931	0.93	0.926	0.859
2)	$1) + b_{io} + b_{ia} + b_{is}$ (h_{v1})	0.922	0.925	0.920	0.853
3)	$1) + \frac{\sum_{g \in genres(i)} b_{ug}}{\lvert genres(i) \rvert}$ (h_{v2})	0.919	0.920	0.917	0.850
4)	$2) + \frac{\sum_{g \in genres(i)} b_{ug}}{\lvert genres(i) \rvert}$	0.915	0.918	0.914	0.850

Context-aware Model. Table 3 depicts the results of the evolving model by adding on the biases of b_{i*} and b_{u*} separately. This way the effect of each part can be observed. Obviously, b_{ug} contributes more than b_{i*} for the final improvement in RMSE. The reason can be traced back again to the rating numbers falling into into each category. Specifically, the average rating number of items is 60 and the average rating number of users is 106. The b_{i*} and b_{u*} are related to the specific i and u now, which means these ratings are then divided into these categories, i.e., S, O, A and G. Roughly speaking, the ratings for each item are divided by 2 for S, 20 for O and 7 for A, respectively, whereas the ratings for each user are shared between 18 genres but not exclusively, which leaves relatively richer information for b_{ug} than b_{io}, b_{ia} and b_{is}.

With the context-aware model, the overall RMSE is improved by 1.7% on D_1, 1.3% on D_2, 1.5% on D_1' and 1% on D_2', respectively, compared to FactCF. The performance on the cold-start problem is depicted in Table 2. There is a certain level improvement on all the bins comparing to the baseline algorithms.

Ensemble Results. In Table 1 we give the ensemble results obtained by two approaches M$_s$ and M$_v$, respectively. Let h_{s1} and h_{s2} be the regressors generated by M$_s$, h_{v1} and h_{v2} be the ones generated by M$_v$. Their performance serves as a start point for the SSL process. The corresponding ensemble results are depicted as Ensemble$_s$ and Ensemble$_v$, respectively. The ensemble methods present a much better results in overall RMSE than the FactCF algorithm. Given the individual regressors h_{v1} and h_{v2}'s performance in Table 3, by simply assembling them together before any optimization with SSL, the accuracy has been largely improved.

Table 2 shows the performance on bins. Comparing to Context, with a slightly better performance on the overall RMSE, their performance on the unpopular bins are better. The biggest improvement is observed in the last bin, where the RMSE is reduced by 4.6% for users and 6% for items. The empirical results demonstrate that the ensemble methods can be used to address the cold-start problem. For a more theoretical analysis for the ensemble methods, please refer to [6].

Semi-supervised Co-training Algorithm (CSEL). In order to generate the bootstrap duplicates with the M$_s$ method, we set a parameter $0 \leq \zeta \leq 1$ as the proportion to take from the original training set. In our evaluation, by trying different values ζ is set to 0.8, i.e., $0.8 \times |T|$ examples are randomly selected for each duplicate, which provides the best performance in most cases.

As mentioned in Section 4.2, selecting the examples with a big difference between h_1 and h_2 to form the teaching set is essential to the success of the SSL process. In order to do so, a threshold β is set to select the examples with the top 10% biggest difference. So, according to the distribution of the difference depicted in Figure 2 we have β =0.675 for M$_s$, 0.85 for M$_v$ and 1 for M$_{sv}$, respectively. As expected, M$_{sv}$ generates the most diverse regressors whereas M$_s$ generates the least. During the iterations the difference changes with the updated regressors. That is, if there is not enough unlabeled examples meet the condition the threshold will automatically shrink by $\beta = 0.8 \times \beta$. The performance on solving the cold-start problem on D_2' is depicted by the comparison of the left and right figures in Figure 3, which presents the results on user bins of the FactCF method and the CSEL$_v$ method, respectively. The

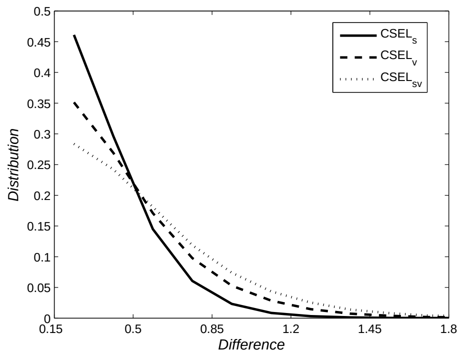

Figure 2: Difference Distribution between the Multiple Regressors Generated for the CSEL Methods (on D_2). We estimate the difference between the predictions made by the multiple regressors generated with different methods, i.e., $\mathbf{M}_s$, $\mathbf{M}_v$ and $\mathbf{M}_{sv}$. The threshold of difference is set according to the distributions for selecting the examples to form the teaching set.

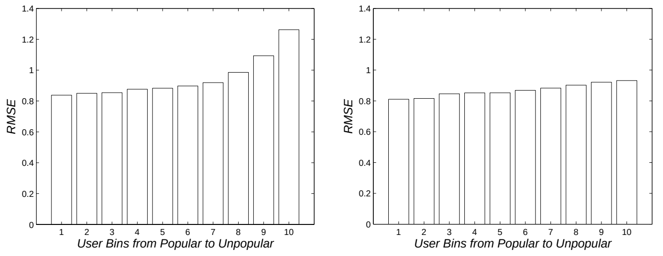

Figure 3: Comparison between the FactCF algorithm (left) and CSEL$_v$ (right) on the cold-start problem for D_2'. The average RMSE scores on user bins with different popularity are given.

RMSE on the cold-start bins drops dramatically. The performance on those bins are much more closer to the performance on the popular bins now. In other words, the distribution of RMSE becomes much more balanced than FactCF. Meanwhile, the performance on the popular bins has also been improved, which means the overall RMSE performance is generally improved. The improvement on the unpopular bins means the cold-start problem is well addressed in the sense that the predictions for the users and items that possess limited ratings become more accurate, which makes it possible to provide accurate recommendations for them.

Efficiency Performance. Roughly speaking, the cost of the algorithm is t times of the standard factorization algorithm since the individual models are re-trained for t times. In the experiments we found that teaching set size N has a big influence on convergence. We tried many possible values of N from $1,000$ up to $20,000$ with respect to the optimization effect and time efficiency, and found that 20% of the training set size is a good choice, e.g., $N = 8,000$ for D_1 and $N = 16,000$ for D_2, which leads to a fast convergence and optimal final results. Besides, we found the quality of the teaching set, i.e., difference and accuracy of the predictions made for the selected examples, has a even bigger impact on the performance and convergence of CSEL. With the methods described in Section 4.2 we managed to guarantee the best examples that satisfy the conditions are selected. The convergence point is set to when the RMSE in the validation set does drop any more for 2 iterations in a row. The entire training process is performed off-line.

Table 4: Diversity of the Individual Regressors h_1 and h_2 Pre- and Post- CSEL Methods.

Methods	D_1	D_2	D_1'	D_2'
Before CSEL$_s$	0.43	0.45	0.4	0.45
After CSEL$_s$	0.13	0.12	0.11	0.12
Before CSEL$_v$	0.49	0.5	0.5	0.52
After CSEL$_v$	0.15	0.11	0.12	0.1
Before CSEL$_{sv}$	0.51	0.57	0.55	0.56
After CSEL$_{sv}$	0.13	0.15	0.12	0.11

Table 5: Predictions made for the users and items with different popularity.

Methods	x_{11}	x_{12}	x_{21}	x_{22}
Real Rating	3	4	1	5
FactCF	3.3124	3.2171	2.0155	3.7781
Context	3.3084	3.4605	1.9560	4.2217
Ensemble$_v$	3.2934	3.5071	1.8032	4.3433
CSEL$_s$	3.2313	3.4913	1.8351	4.241
CSEL$_v$	3.1971	3.7655	1.5456	4.671

Diversity Analysis. The diversity of the regressors pre- and post-performing the CSEL Methods is depicted in Table 4 for all the datasets under estimation. Here diversity is defined as the average difference between the predictions made by the regressors. In the experiments we observed that the diversity drops rapidly when the accuracy goes up. At the point of convergence the regressors become very similar to each other and the ensemble of the results does not bring any more improvements at this point. As mentioned before diversity is an important condition to achieve a good performance. Apparently, simply distinguishing the teaching sets of the two regressors cannot help enough with maintaining the diversity and some strategies can be designed in this respect.

Moreover, when generating the multiple regressors with M$_v$ or M$_s$ there is a setback on their performance, i.e., h_1 and h_2 do not perform as good as the full model (Eq. 4) being trained on the full training set. For example, as shown in Table 3, h_{v1} and h_{v2} have a setback in accuracy compared to the full model. Although by simply assembling the two regressors together the accuracy can be brought back (see Table 1), some improvement could still be done in this respect to obtain an even better initial point at the beginning of the SSL process.

6.4 Qualitative Case Study

To have a more intuitive sense about how the algorithms work in different cases, we pick two users and two items from the dataset and look at their predictions. Let u_1 be an active user and u_2 be an inactive user, i_1 be a popular item and i_2 be an unpopular item. Then four examples $x_{11}, x_{12}, x_{21}, x_{22}$ are drawn from the dataset with the given users and items, where x_{kj} is given by u_k to i_j. This way x_{11} is rated by an active user to a popular item whereas x_{22} is rated by an inactive user to an unpopular item, etc., and the performance of the algorithms on them can be observed. In this case study we pick the four examples that have the real ratings, $r_{11}, r_{12}, r_{21}, r_{22}$.

The results are given in Table 5, showing that providing accurate predictions also helps with personalization. That is, to recommend items that the end-user likes, but that are not generally popular, which can be regarded as the users' *niche* tastes. By providing more accurate predictions for these items, the examples such as

x_{12} or x_{22} can be identified and recommended to the users who like them. Thus the users' personalized tastes can be retrieved.

7. CONCLUSION

This paper resorts to the semi-supervised learning methods to solve the cold-start problem in recommender systems. Firstly, to compensate the lack of ratings, we propose to combine the contexts of users and items into the model. Secondly, a semi-supervised co-training framework is proposed to incorporate the unlabeled examples. In order to perform the co-training process and let the models teach each other, a method of constructing the teaching sets is introduced. The empirical results show that our strategy improves the overall system performance and makes a significant progress in solving the cold-start problem.

As future work, there are many things worth to try. For example, our framework provides a possibility for accommodating any kinds of regressors. One interesting problem is how to choose the right regressors for a specific recommendation task. Meanwhile, this semi-supervised strategy can be easily expanded to include more than two regressors. For example, three regressors can be constructed and a semi-supervised tri-training can then be performed on them. Finally, it is also intriguing to further consider rich social context [28] in the recommendation task.

Acknowledgements. This work is partially supported by National Natural Science Foundation of China (61103078, 61170094), National Basic Research Program of China (2012CB316006, 2014CB340500), NSFC (61222212), and Fundamental Research Funds for the Central Universities in China and Shanghai Leading Academic Discipline Project (B114). We also thank Prof. Zhi-Hua Zhou for his valuable suggestions.

8. REFERENCES

[1] T. Back. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.

[2] R. Bell, Y. Koren, and C. Volinsky. Modeling relationships at multiple scales to improve accuracy of large recommender systems. In *KDD'07*, pages 95–104, 2007.

[3] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *COLT'98*, 1998.

[4] M. Degemmis, P. Lops, and G. Semeraro. A content-collaborative recommender that exploits wordnet-based user profiles for neighborhood formation. *User Modeling and User-Adapted Interaction*, 17(3):217–255, 2007.

[5] M. Deshpande and G. Karypis. Item-based top-n recommendation algorithms. *ACM Transaction on Information Systems*, 22(1):143–177, 2004.

[6] T. G. Dietterich. Ensemble methods in machine learning. In *Workshop on Multiple Classifier Systems*, pages 1–15, 2000.

[7] G. Dror, N. Koenigstein, and Y. Koren. Yahoo! music recommendations: Modeling music ratings with temporal dynamics and item taxonomy. In *RecSys'11*, pages 165–172, 2011.

[8] K. Goldberg, T. Roeder, D.Gupta, and C. Perkins. Eigentaste: A constant time collaborative filtering algorithm. *Information Retrieval*, 4(2):133–151, 2001.

[9] N. Good, J. Schafer, and J. etc. Combining collaborative filtering with personal agents for better recommendations. In *AAAI'99*, pages 439–446, 1999.

[10] G. Guo. Improving the performance of recommender systems by alleviating the data sparsity and cold start problems. In *IJCAI'13*, pages 3217–3218, 2013.

[11] J. Herlocker, J. Konstan, and J. Riedl. Explaining collaborative filtering recommendations. In *CSCW'00*, pages 241–250, 2000.

[12] P. B. Kantor, F. Ricci, L. Rokach, and B. Shapira. *Recommender Systems Handbook*. Springer, 2010.

[13] I. Konstas, V. Stathopoulos, and J. M. Jose. On social networks and collaborative recommendation. In *SIGIR'09*, pages 195–202, 2009.

[14] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD'08*, pages 426–434, 2008.

[15] Y. Koren. The bellkor solution to the netflix grand prize. *Technical report*, 2009.

[16] J. Li and O. Zaiane. Combining usage, content, and structure data to improve web site recommendation. In *EC-Web'04*, 2004.

[17] J. Lin, K. Sugiyama, M.-Y. Kan, and T.-S. Chua. Addressing cold-start in app recommendation: Latent user models constructed from twitter followers. In *SIGIR'13*, pages 283–293, 2013.

[18] P. Melville, R. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *AAAI'02*, pages 187–192, 2002.

[19] S.-T. Park and W. Chu. Pairwise preference regression for cold-start recommendation. In *RecSys'09*, pages 21–28, 2009.

[20] L. Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1):1–39, 2010.

[21] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *CSCW'98*, pages 345–354, 1998.

[22] A. Schein, A. Popescul, L. Ungar, and D. Pennock. Methods and metrics for cold-start recommendations. In *SIGIR'02*, pages 253–260, 2002.

[23] I. Soboroff and C. Nicholas. Combining content and collaboration in text filtering. In *IJCAI Workshop on Machine Learning for Information Filtering*, pages 86–91, 1999.

[24] M. Sun, F. Li, J. Lee, K. Zhou, G. Lebanon, and H. Zha. Learning multiple-question decision trees for cold-start recommendation. In *WSDM'13*, pages 445–454, 2013.

[25] G. Takacs, I. Pilaszy, B. Nemeth, and D. Tikk. Scalable collaborative filtering approaches for large recommender systems. *Journal of Machine Learning Research*, 10:623–656, 2009.

[26] J. Tang, S. Wu, J. Sun, and H. Su. Cross-domain collaboration recommendation. In *KDD'12*, pages 1285–1294, 2012.

[27] A. Toscher and M. Jahrer. The bigchaos solution to the netflix prize 2008. *Technical Report*, 2008.

[28] Z. Yang, K. Cai, J. Tang, L. Zhang, Z. Su, and J. Li. Social context summarization. In *SIGIR'11*, pages 255–264, 2011.

[29] K. Zhou, S.-H. Yang, and H. Zha. Functional matrix factorizations for cold-start recommendation. In *SIGIR'11*, pages 315–324, 2011.

[30] Z.-H. Zhou and M. Li. Semi-supervised regression with co-training. In *IJCAI'05*, pages 908–913, 2005.

[31] Z.-H. Zhou and M. Li. Semi-supervised learning by disagreement. *Knowledge and Information Systems*, 24(3):415–439, 2010.