Novel Boosting Frameworks to Improve the Performance of Collaborative Filtering

Xiaotian Jiang 0532yangya@163.com
Zhendong Niu zniu@bit.edu.cn
Jiamin Guo minminte@hotmail.com
Ghulam Mustafa gmustafa@bit.edu.cn
Zihan Lin linzihan@bit.edu.cn
Baomi Chen 2120111120@bit.edu.cn
Qian Zhou 2120111231@bit.edu.cn
School of Computer Science, Beijing Institute of Technology
Zhongguancun South Street.5, 100081 Beijing, China

Editor: Cheng Soon Ong and Tu Bao Ho

Abstract

Recommender systems are often based on collaborative filtering. Previous researches on collaborative filtering mainly focus on one single recommender or formulating hybrid with different approaches. In consideration of the problems of sparsity, recommender error rate, sample weight update, and potential, we adapt AdaBoost and propose two novel boosting frameworks for collaborative filtering. Each of the frameworks combines multiple homogeneous recommenders, which are based on the same collaborative filtering algorithm with different sample weights. We use seven popular collaborative filtering algorithms to evaluate the two frameworks with two MovieLens datasets of different scale. Experimental result shows the proposed frameworks improve the performance of collaborative filtering.

Keywords: Boosting, Collaborative Filtering, Recommender Systems, AdaBoost.

1. Introduction

Recommender Systems (RS) predict a user's preference on his unrated items based on the existing rating data and knowledge. Recommendation algorithms can be divided into multiple categories like content-based approaches, collaborative filtering, knowledge-based approaches, etc. As one of the most representative categories, collaborative filtering is prevalent for its high performance and simple requirements (Koren and Bell (2011)).

There are two main categories of collaborative filtering: memory-based and model-based. In memory-based approaches, a user' existing rating data is directly exploited for predicting their preference on other items. While in model-based approaches, the prediction is achieved by building a model that describes the interaction between users and items. Of all the model-based approaches, latent factor model (such as SVD-based models) earns popularity for its high accuracy as well as reduced online response delay.

Most previous recommendation algorithms dedicate to improve the performance of collaborative filtering on a single recommender (like Zhao et al. (2013)), or they combine different types of recommendation strategies with simple ensemble approaches like weighting or cascade (Burke (2007), Chen et al. (2012)). Yet, to our best knowledge, only a few researchers showed their interest in combining multiple similar recommenders that are based on the same single common-used collaborative filtering algorithm with different sample weights. A known work is Bar et al. (2013), in which paper a boosting framework is proposed to improve the performance of common-used collaborative filtering. However, this framework suffers from the complexity brought by iteratively calculating and storing the similarity matrix.

Boosting is first employed to enhance the performance of classification by integrating multiple weak classifiers into a better classifier with higher accuracy. It is generally acknowledged that classification and (numeric) prediction are two major components of prediction problems (Han et al. (2006)). In recommender systems, rating prediction is a common practice to get a user's preference on his unrated items. Based on these observations, there is an instinctive idea that when boosting is applied as a general framework to improve the performance of recommendation algorithms, satisfying results can be achieved.

Since AdaBoost is both popular and typical among all the boosting approaches, in this paper we adapt it for collaborative filtering and propose two boosting frameworks: one for memory-based collaborative filtering, and the other for SVD-based collaborative filtering. Each of these frameworks combines multiple similar recommenders, which are based on the same collaborative filtering algorithm with different sample weights controlling the recommenders in their modeling or prediction phase.

The major considerations for the adaptation process include sparsity, recommender error rate, sample weight update, and potential. Specifically, we consider how to generate recommenders without aggravating the sparsity of recommendation data, how to use sample weights to accurately evaluate the error rate of the generated recommender, how to appropriately update sample weights, and how to transfer the system's attention to the samples with potential to further reduce its error.

Details of the adaptation process is elaborated in section 3. In our experiment, we carefully selected seven common-used collaborative filtering approaches. We put four of them under the memory-based framework and the other three under the SVD-based framework. Experimental result showed that almost for each of these approaches, the proposed frameworks generated a strong recommender that offers better performance than Bar et al. (2013). Not only this, the proposed frameworks are also more advantageous in time and space complexity with the avoidance of calculating similarity matrix iteratively during each boosting loop.

The rest of this paper is organized as follows. The preliminaries are introduced in section 2. In the next section, we elaborate the proposed two frameworks for collaborative filtering. In section 4, we conduct experiments on seven widely used collaborative approaches and carry on analysis on the results. In the last section, we draw conclusions and list future works.

2. Preliminaries

For the convenience of making clear expressions, we reserve special indexing letters to distinguish the following concepts. We denote the user set by U and the item set by I. A

Rating dataset includes two subsets : T_{train} for the training set and T_{test} for the test set. u,v indicate the users in U; i,j indicate the items in I. We denote the rating of user u to item i by r_{ui}. Each record (u, i, r_{ui}) in T is called a sample, and sometimes we also use (u, i) for short. A larger r_{ui} indicates user u's higher interest in item i. $N(i)$ represents the users who rated item i and $N(u)$ the items rated by user u. Given a similarity function, we denote by $S(u, k)$ the k-nearest users around u and $S(i, k)$ the k-nearest items around i. $B(i; u, k)$ denotes item i's k-nearest items among those rated by user u, and $B(u; i, k)$ denotes user u's k-nearest users among those who rated item i.

2.1. Baseline Predictor

Collaborative filtering focuses on modeling the interaction between users and items. However, many effects that contribute to the ratings, such as user bias and item bias, are not associated with this interaction (Koren and Bell (2011)). User bias indicates the observed individual difference among users, while item bias indicates the observed bias for each item compared to the overall average. For example, a lower user bias corresponds to a critical user who is prone to rate low, and a high item bias corresponds to a good item that earns higher ratings. A baseline predictor (Koren (2010)) tries to capture these biases and separate them from the whole effects so as to enable other models better concentrating on user-item interaction. It is composed of three components: the overall average μ, user bias b_u, and item bias b_i. That is,

$$b_{ui} = \mu + b_u + b_i \tag{1}$$

We obtain the unknown parameters b_u and b_i by minimizing the following loss function:

$$\min_{b_*} \sum_{(u,i)\in T_{train}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda(b_u^2 + b_i^2) \tag{2}$$

where $\lambda(b_u^2 + b_i^2)$ is a regularizing term that prevents this model from over-fitting and λ is set to adjust the extent of regularization. A variant of Stochastic Gradient Descent (SGD) (Funk) is often employed to rapidly optimize functions like (2). For each sample (u, i, r_{ui}) in T_{train}, SGD denotes $e_{ui} \overset{\text{def}}{=} r_{ui} - \hat{b}_{ui}$ and executes the following steps to move these parameters towards the opposite direction of the gradient:

- $b_u \leftarrow b_u + \gamma \cdot (e_{ui} - \lambda \cdot b_u)$
- $b_i \leftarrow b_i + \gamma \cdot (e_{ui} - \lambda \cdot b_i)$

where γ controls the learning rate of this process. After several iterations on the training dataset, b_u and b_i will gradually converge to stable values.

2.2. Memory-based Collaborative Filtering

User-based collaborative filtering (UserCF, Resnick et al. (1994)) and Item-based collaborative filtering (ItemCF, Sarwar et al. (2001)) are two most famous algorithms in memory-based collaborative filtering.

UserCF assumes that a user will like the items enjoyed by the users whose rating perference are like him (also called his neighbors). The prediction formula is

$$\hat{r}_{ui} = \frac{\sum_{v \in B(u;i,k)} s_{uv} \cdot r_{vi}}{\sum_{v \in B(u;i,k)} s_{uv}} \tag{3}$$

where s_{uv} is the similarity value between u and v. In this paper, we employ the cosine similarity. Likewise, ItemCF assumes that a user will like the items similar to those he has rated high. Its formula is achieved by switching the role of item and user in formula (3).

Integrating memory-based approaches with baseline predictor can obtain better performance. For itemCF, we can get the following formula (Koren (2010)):

$$\hat{r}_{ui} = \mu + b_u + b_i + \frac{\sum_{j \in B(i;u,k)} s_{ij} \cdot (r_{uj} - b_{uj})}{\sum_{j \in B(i;u,k)} s_{ij}} \tag{4}$$

Likewise, formula for UserCF can be easily deduced too. Experiments in section 4 confirm the performance improvements of these two integrated algorithms, called BUserCF and BItemCF respectively.

2.3. SVD-based Collaborative Filtering

In SVD-based models, both users and items are projected into the same f-dimensional latent feature space. The interaction between users and items is modeled as inner products in this space.

Formally, each user u corresponds to vector $p_u \in \mathbb{R}^f$, and each item i corresponds to vector $q_i \in \mathbb{R}^f$. For a given item i, each component of q_i indicates the extent that item i contains this feature. Likewise, each component of p_u denotes to what extent user u feels this feature interesting. So, the interaction between u and i can be modeled as $q_i^T p_u$. Integrating it with the baseline predictor, we retrieve the final rating prediction formula, that is:

$$r_{ui} = \mu + b_u + b_i + q_i^T p_u \tag{5}$$

We denote the model above as BiasSVD. The form of its lost function are alike to that of the baseline predictor. Optimal values of these parameters can also be approximated by SGD, which has been depicted in section 2.1.

BiasSVD can not react for the newly created ratings. Koren (2010) proposed a factorized neighborhood model (FNM) to deal with this problem. Not only this, this model lowers the complexity and offers better performance. Like BiasSVD and baseline predictor, FNM also employs SGD to minimize a loss function to get its unknown parameters. Details of the optimization progress are demonstrated in Koren and Bell (2011), which is quite different from that of BiasSVD.

2.4. AdaBoost

AdaBoost (Han et al. (2006)) combines multiple weak classifiers (better than random guess) to obtain a stronger classifier with better performance. In this algorithm, first the same sample weight is assigned to each sample in T_{train}, which is used to generate a sample collection of $|T_{train}|$ by sampling with replacement. This collection is used to train the first weak classifier M_1 and evaluate its error rate $err(M_1)$. Then the weights of the misclassified samples are increased and the weights of the correct classified ones are decreased. This adjustment changes the probability distribution of sample weights. So when current sample weights are used to generate M_2, the new classifier will pay more attention on samples that are misclassified by M_1. Then the error rate of M_2 can be used to obtain M_3. This iteration is repeated until K classifiers and their error rates are obtained. In classification process, the K classifiers are combined and an integral result is obtained based on majority voting.

3. Two Boosting Frameworks for Collaborative Filtering

3.1. The Boosting Framework for Memory-based Collaborative Filtering

Since item-based approaches and user-based approaches have similar formulae, in this subsection, if not explicitly declared, the demonstrations will be based on the item-based context. The same method is also applicable to user-based approaches, by switching the roles of users and items.

In the boosting scheme in classification, there are some requirements for the weak classifier (e.g., for binary classification the accuracy rate should larger than 50%). However, since the employed base recommenders are all significantly better than random guess, in this paper we do not specify this requirement.

To adapt AdaBoost and propose the boosting framework for memory-based collaborative filtering, it is necessary to consider the following four major factors: sparsity, recommender error rate, sample weights update, and potential.

3.1.1. Sparsity

As mentioned above, AdaBoost employs sampling with replacement to generate its actual training dataset (D_i) for each classifier. In recommender systems, whose dataset is often very sparse, that does not work. Sampling with replacement brings about massive repetitive samples, making D_i even much sparser than T_{train}. Recommender systems cannot guarantee acceptable results or even any legitimate results with such a training set. The information loss that sampling brings about is intolerable. For example, in memory-based collaborative filtering a sparser dataset may mean that items cannot even find there neighbors, not to mention the recommendation.

To deal with this problem, we employ a more direct approach called reweighting (Seiffert et al. (2008)). Instead of AdaBoost's generating D_i by sampling, the reweighting strategy directly uses sample weights to control the new classifier in its shaping or prediction phase. For ItemCF, we turn its prediction formula into the form below:

$$\hat{r}_{ui}^{m} = \frac{\sum_{j \in B(i;u,k)} s_{ij} \cdot w_{uj}^{m} \cdot r_{uj}}{\sum_{j \in B(i;u,k)} s_{ij} \cdot w_{uj}^{m}} \tag{6}$$

where w_{uj}^{m} denotes the weight of sample (u, j) in the m-th recommender. In this formula, the larger sample (u, j) is weighted, the closer $\hat{r}_{ui}^{m}$ is to r_{uj}. On the other hand, the smaller sample (u, j) is weighted, the less relevant $\hat{r}_{ui}^{m}$ is to r_{uj}.

3.1.2. Recommender Error Rate

When AdaBoost is employed in classification, the weight of a sample directly indicates that to what extent the sample is expected to be correctly classified. However, in collaborative filtering especially the memory-based approaches, the effect of sample weights may not be that explicit.

For memory-based collaborative filtering, like ItemCF and UserCF, a sample's weight does not function directly on its own prediction. Instead, it affects the rating predictions of all the samples nearby. Correspondingly, the prediction value of this sample is also affected by the weights of all the samples nearby. That is to say, a larger sample weight does not indicate the greater eagerness for the system to decrease its error any more. So we come up with another approach to indicate this eagerness and the following formula for $err(R_m)$:

$$err(R_m) = \sum_{(u,i)\in T_{train}} \left(\frac{|r_{ui} - \hat{r}^m_{ui}| \times \sum_{j\in B(i;u,k)} w^m_{uj}}{\sum_{(u,i)\in T_{train}} \sum_{j\in B(i;u,k)} (w^m_{uj} \times |r_{ui} - \hat{r}^m_{ui}|)} \times \frac{|r_{ui} - \hat{r}^m_{ui}|}{D} \right) \tag{7}$$

where D is a constant used to denote the drop between the upper bond and the lower bond of the rating interval. w^m_{uj} indicates the weight of sample (u, j) for the m-th recommender. Comparing with the classifier error rate formula in AdaBoost, we can clearly observe that here we use $|r_{ui} - \hat{r}^m_{ui}|/D$ to represent sample error $err(X_j)$. Moreover, for each sample we use the sample weights of all its k-nearest neighbor samples rated by u to represent the eagerness for the recommender system to decrease its error. Also, we weight the sum value by $|r_{ui} - \hat{r}^m_{ui}|$ to further emphasize samples that have got extra concern but still have large sample error.

3.1.3. Sample Weight Update

In AdaBoost, the objective of updating sample weights is to pay more attention on the misclassified samples. To make this applicable to memory-based collaborative filtering, here we employ an alternative method: we minimize sample errors by updating all the sample weights of its k-nearest neighbors.

Specifically, suppose that we observed $r_{ui} > \hat{r}_{ui}$ in one loop. Naturally we expect $\hat{r}_{ui}$ to be higher and closer to r_{ui}. To achieve this, for each of its k-nearest neighbors (u, j), we adjust its sample weight by the following process: if $r_{uj} > \hat{r}_{ui}$, we increase its sample weight; if $r_{uj} < \hat{r}_{ui}$, we decrease the sample weight. The circumstance that $r_{ui} < \hat{r}_{ui}$ can be solved by a reverse process. Formally speaking, for sample (u, i) we define the signal of its neighbor (u, j) as

$$SGN^m_{ui}(j) = sgn(r_{ui} - \hat{r}^m_{ui}) \times sgn(r_{uj} - \hat{r}^m_{ui}) \tag{8}$$

where $sgn(x)$ is a signal function.

Not only that, we also take the individual difference of each sample into consideration. For samples with higher average error, which indicates the system's long-term insufficiency of attention for their correctness, a larger update extent may be more reasonable. The following formula describes such consideration.

$$UE^m_{ui} = 1 + \eta \times \frac{1}{m} \sum_{i=1}^{m} \frac{r_{ui} - \hat{r}^i_{ui}}{D} \tag{9}$$

where η denotes how much the average sample error influences the update process. In this paper, we set $\eta = 0.5$ by experience. It is obvious that UE_{ui}^m is in proportion to the average error of sample (u, i).

The sample weight update fashion in Adaboost (Han et al. (2006)) is acute. We tried to apply it directly, but the results are unsatisfactory. So we keep the update process more steady by using the update fashion depicted below. For each sample (u, i) among the largest $T(\leq 1)$ of sample errors $|r_{ui} - \hat{r}_{ui}^m|$ in T_{train}, we update the sample weights of its k-nearest neighbors (u, j) as

$$w_{uj}^{m+1} = w_{uj}^1 \times \left(1 + SGN_{ui}^m(j) \times \frac{err(R_m)}{1 - err(R_m)} \times UE_{uj}^m \times \rho\right) \tag{10}$$

where we denote by ρ the update rate that controls the impact of $err(R_m)$ and UE_{ui}^m in the update process. In this formula we choose $\frac{err(R_m)}{1-err(R_m)}$ to indicate the effect of R_m as AdaBoost does.

3.1.4. Potential

In memory-based collaborative filtering, $|r_{ui} - \hat{r}_{ui}^m|$ may not reduce to an arbitrarily small value. This is because $\hat{r}_{ui}$ —the predicted rating of sample(u, i) —always lays in the rating interval of this sample's neighbors. Unfortunately, if the real rating of (u, i) is out of this interval, then what we can do to minimize the sample error only gets it closer to one bound of this interval. For instance, supposing that $\hat{r}_{ui}$ is 2.5, the rating interval of its neighbors is [1,3], and the real rating r_{ui} is 5. Then no matter how we adjust the sample weights of (u, i)'s neighbors to get $\hat{r}_{ui}$ closer to r_{ui}, $\hat{r}_{ui}$ can not exceed 3 — upper bound of the interval [1,3]. When samples like (u, i) get close enough to their bounds, compared with keeping on adjusting the weights to minimize the errors of these samples, which often receive little effect as the cost of increasing errors of the other samples, it may be wiser to pay more attention on samples that may have lower error but larger room to minimize their errors.

So we use $err_{threshold}$ as an indicator. Suppose that $[\hat{r}_{min}, \hat{r}_{max}]$ is the rating interval of the neighbors of sample (u, i). For the m-th recommender, if $\hat{r}_{ui}^m > r_{ui}$ and $\hat{r}_{ui}^m - \hat{r}_{min} < err_{threshold}$, or if $\hat{r}_{ui}^m < r_{ui}$ and $\hat{r}_{max} - \hat{r}_{ui}^m < err_{threshold}$, then we believe sample (u, i) has reached its intrinsic error $\hat{r}_{min} - r_{ui}$ or $r_{ui} - \hat{r}_{max}$. At this time, proceeding with weight update cannot bring about apparent error reduction, conversely it leads to precision loss on other samples due to the neighbor relationship.

Based on all the analysis above, we propose the boosting framework for memory-based collaborative filtering and describe it with item-based collaborative filtering in Algorithm 1. We name this framework MemCFBoost.

3.2. The Boosting Framework for SVD-based Collaborative Filtering

With the similar idea described in subsection 3.1, we propose a boosting framework for SVD-based collaborative filtering denoted by SVDCFBoost (Algorithm 2). Most of this framework is the same as Algorithm 1, so in this subsection we just depict the differences.

For SVD-based collaborative filtering approaches, the adaptations are more intuitive. Instead of using neighbors to predict unknown ratings, SVD-based approaches obtain prediction model by employing SGD to gradually optimize a loss function. If sample weights

Algorithm 1 MemCFBoost: The Boosting Framework for Memory-based Collaborative Filtering

```
Train Model
Initialize the weight of each sample in $T_{train}$ with 1;
for each $m \in [1, M]$ do
    (1) Use formula (6) to train the $m$-th recommender $R_m$.
    (2) Predict ratings with the training set, calculate the error rate of recommender $R_m$ with formula (7).
    (3) For each sample $(u, i)$ among the largest T of sample errors $|r_{ui} - \hat{r}^m_{ui}|$, define the rating interval of its neighbors as $[\hat{r}_{min}, \hat{r}_{max}]$. If $(u, i)$ does not match the following conditions:
        1. $\hat{r}^m_{ui} > r_{ui}, and \quad \hat{r}^m_{ui} - \hat{r}_{min} < err_{threshold}$
        2. $\hat{r}^m_{ui} < r_{ui}, and \quad \hat{r}_{max} - \hat{r}^m_{ui} < err_{threshold}$
    For each of its k-nearest neighbors $(u, j)$, adjust the sample weight $w^{m+1}_{uj}$ with formula (10).
    (4) Normalize sample weights;
end for

Prediction
(1) For each recommender, set its weight as:
    $$w_m = log\frac{1 - err(R_m)}{err(R_m)}$$
(2) For each sample $X(u, i)$, its predicted rating is: $R(X) = \sum_{m=1}^{M} R_m(X) \times w_m$;
```

are used to adjust the value of parameter λ so as to control the extent of regularization for each sample during the optimization process, the weights can directly affect the prediction model and properly represent the attention paid for the samples' correctness. Specifically, in model training process of the m-th recommender, for each sample (u, i) we change its original learning rate λ to $\lambda \cdot w^m_{ui}$, where w^m_{ui} is acquired by the last boosting process.

When calculating the error rates of recommenders, we use sample weights directly. That is,

$$err(R_m) = \sum_{(u,i)\in T_{train}} \frac{w^m_{ui}}{\sum_{(u,i)\in T_{train}} w^m_{ui}} \times \frac{|r_{ui} - \hat{r}^m_{ui}|}{D} \tag{11}$$

Not only this, for each sample (u, i) among the largest $T(\leq 1)$ of the sample errors we directly decrease the sample weight of its own with the formula below.

$$w^{m+1}_{ui} = w^1_{ui} \times \left(1 - \frac{err(R_m)}{1 - err(R_m)} \times UE^m_{uj} \times \rho\right) \tag{12}$$

By weakening the extent of regularization to get the predicted rating values fitter to the existing rating data, the generated model focuses more on samples that was poorly predicted by the last recommender.

3.3. Complexity Analysis

Complexity analysis of the two proposed frameworks are elaborated as follows.

The time complexity of ItemCF's training process includes that of calculating item-item similarity matrix ($O(|I|^2|U|)$) and sorting all items for each item according in similarity descending order ($O(|I|^2 log|I|)$). For each loop in the training process of Algorithm 1, we first calculate the estimated value of each sample in the training dataset($O(n|I|)$), then use these results to calculate $err(R_m)$ ($O(n|I|)$). In update phase, the time complexity is $O(n|I|)$. The last phase of the training process is normalization, and its time complexity is $O(n)$. So the time complexity of the training process of the proposed model is $O(|I|^2 \times (log|I|+|U|)+Mn|I|)$, and that of the prediction part is $O(M|I|)$. For most recommendation datasets, whose sparsity is under 8%, Mn is less than $|I||U|$. So time complexity can be $O(|I|^2(log|I| + |U|))$, which is the same as that of ItemCF. As for the space complexity, the main space cost is on the storage of item-item similarity matrix ($O(|I|^2)$), the sorted item list for each item ($O(|I|^2)$), sample weights for each recommender ($O(Mn)$) and $err(R_m)$ ($O(M)$). So the space complexity of the algorithm is $O(|I|^2 + Mn)$. The complexity is higher than the original form, but in consideration of that M is usually small (3 to 8) and the recommendation is very sparse, the time and space cost is acceptable.

Complexity of Algorithm 2 is less than that in Algorithm 1. This is because this framework directly updates the sample itself instead of updating the weights of its neighbors. Here we directly present the time and space complexity in Table 1, where S denotes the iteration times of SVD-based approaches.

Table 1: Time and Space Complexity Comparisons

	Training Time	Online Time	Space
ItemCF	$O(\|I\|^2(log\|I\| + \|U\|))$	$O(\|I\|)$	$O(\|I\|^2)$
Algorithm 1	$O(\|I\|^2(log\|I\| + \|U\|) + Mn\|I\|)$	$O(M(\|I\|))$	$O(\|I\|^2 + Mn)$
BiasSVD	$O(SnF)$	$O(F)$	$O(F(\|U\| + \|I\|))$
Algorithm 2	$O(MnSF)$	$O(MF)$	$O(MF(\|U\| + \|I\|))$

4. Experiments

To evaluate the performance of the two proposed frameworks, we carried out experiments on seven popular collaborative filtering algorithms and present the experimental process in this section. The detailed experimental settings and result analysis is depicted below.

4.1. Datasets

We use two datasets of difference scale for test: MovieLens-1M and MovieLens-100K. Their statistics are listed in Table 2, and the rating scale is $[1, 5]$.

4.2. Evaluation Metrics

In the above-mentioned two datasets, each user had rated over 20 items. Based on this feature, we divided each of the two datasets into five disjoint splits. Each split contains

Table 2: Statistics of MovieLens-100K and MovieLens-1M

	ML-100K	ML-1M
Users	943	6,040
Items	1,682	3,952
Ratings	100,000	1,000,209
Avg Ratings per user	106.04	165.60
Sparsity	0.063	0.0422

20% of the items that each user had rated. One split was used for testing and the rest four for training. We repeated our experiments on different split and found that the change of splits do not lower the improvement of the proposed frameworks, though the performance on the original algorithms may vary.

In this paper, we employ Root Mean Square Error (RMSE) to evaluate the performance of our frameworks. The definition is as follows.

$$RMSE = \sqrt{\frac{\sum_{(u,i)\in T_{test}} (r_{ui} - \hat{r}_{ui})^2}{|T_{test}|}} \tag{13}$$

It is worth noting that Mean Absolute Error (MAE) is also a famous measure. In this paper we employ RMSE because this measure is stricter than MAE. It further increases the penalty to bad predictions. And this is why RMSE was also employed by Netflix Prize as a better indictor to the performance of recommender systems.

4.3. Evaluated Algorithms

In experimental section, we employ UserCF, ItemCF, BUserCF, BItemCF to evaluate the performance of the boosting framework for memory-based collaborative filtering, and Baseline(baseline predictor), BiasSVD, FNM to evaluate the performance of the boosting framework for SVD-based collaborative filtering. We employ a parallel set of parameters in both datasets to make the experimental result more comparable. Detailed parameter settings are based on those presented in the referenced papers, such as in Koren and Bell (2011) Resnick et al. (1994) and Sarwar et al. (2001). Parameters of the proposed methods, ρ the update extent, m the number of recommenders in use and T the update proportion, are chosen by a grid search process. The relationship between ρ and m is detailedly illustrated in section 4.4.

4.4. Experimental Results and Analysis

The experimental results are listed in Table 3, where methods that have clearly improvements are marked in bold.

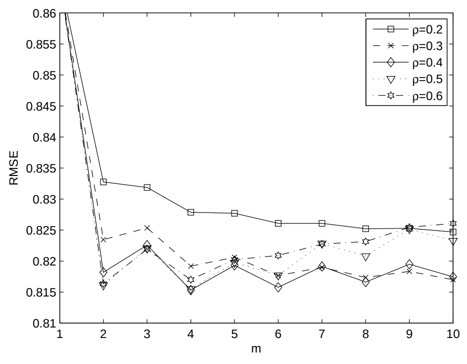
From this table, we can clearly perceive that most of the methods have better performances after applying the proposed framework. It is worth noting that methods that earned higher improvement when using the proposed frameworks on Movielens-1M are also the methods that performed better on Movelens-100K. On both datasets, ItemCF obtains the best improvement. Specifically, the RMSE drops from 0.8655 to 0.8140 on Movielens-1M and from 0.9410 to 0.8954 on Movielens-100K. This improvement is quite impressive

Table 3: Performance Comparisons on Movielens-1M and Movielens-100K.

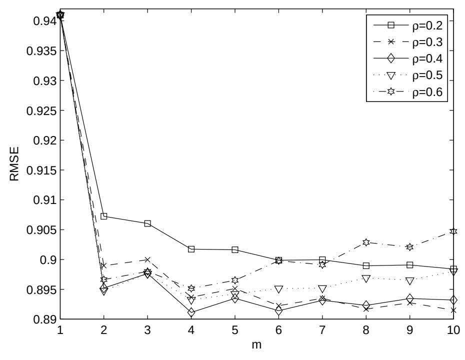
Methods	Movielens-1M		Movielens-100K	
	unboosted	boosted	unboosted	boosted
Baseline	0.8581	**0.8239**	0.9188	**0.8918**
UserCF	0.9327	**0.9272**	1.0335	**1.0299**
BUserCF	0.8091	**0.8018**	0.8844	0.8807
ItemCF	0.8655	**0.8140**	0.9410	**0.8954**
BItemCF	0.7502	**0.7368**	0.8315	0.8200
BiasSVD	0.7392	**0.7055**	0.8449	**0.8262**
FNM	0.7306	**0.7225**	0.8374	0.8353

given that the gap between ItemCF and BiasSVD, which is famous for its high accuracy, is only about 0.13. Methods like Baseline, BItemCF and BiasSVD also received obvious performance improvements. Equally noteworthy is that the performance of the boosted version of BItemCF is in exceed of that of BiasSVD.

During the experimental process, it is found that the relationship between ρ and the numbers of recommenders is interesting, so we employed ItemCF as a base recommender and drew Fig.1(a) and Fig.1(b), where m means the number of recommenders.



(a) Movielens-1M



(b) Movielens-100K

Figure 1: The relationship between ρ and m on Movielens-1M and Movielens-100K

From these two tables, we can clearly see that ρ actually affects the number of recommenders needed to achieve the largest improvement that the framework can offer. As ρ gets smaller, the curve gets flatter. So we need more recommenders to achieve the best performance, but it is more possible to choose an appropriate value of m. As ρ get larger, the curve gets steeper. The number of recommenders we need is decreasing, but a steep curve may lead to the incapability to approach the optimal performance. Supposing that the largest improvement is achieved when $m_{opt} = 7.5$. Since the m in use must be an integer, we have to suffer loss when choosing 6 or 8 as the actually adopted value. If a steer curve is given, there may be a heavy loss. So, a larger ρ means less improvement with quicker calculation, while a smaller ρ means greater improvement with slower calculation.

5. Conclusion

In this paper, we propose two boosting frameworks to improve the performance of memory-based collaborative filtering and SVD-based collaborative filtering. The experimental results on seven popular collaborative filtering algorithms show their effectiveness.

The future work is concentrated on efficiency and Top-N recommendation. Efficiency of the framework can be improved by two means. On one hand, we can employ and modify the parallel version of AdaBoost (Chen et al. (2008)) for recommendation. On the other hand, we can also adjust the parameter ρ to generate fewer recommenders, though it may cause performance degradation.

The real challenge comes from how to adapt this framework to the Top-N Recommendation. We attempted to use the Top-N metrics proposed in Cremonesi et al. (2010) to test its performance. However, the improvements are not striking. That may be due to that Boosting, as a classification method, does not take ranking information into consideration. Instead of ensuring the correctness of few Top-N samples, the aim of boosting is to improve the performance on the overall samples. This aim coincides with rating prediction, so we obtain notable improvements on RMSE. In the future, we hope better results are achievable by adapting boosting to fit the Top-N recommendation.

6. Acknowledgement

This work is supported by the National Natural Science Foundation of China (no. 61250010 and no. 61370137), the Program for Beijing Municipal Commission of Education (grant no.1320037010601) and the 111 Project of Beijing Institute of Technology.

References

Ariel Bar, Lior Rokach, Guy Shani, Bracha Shapira, and Alon Schclar. Improving simple collaborative filtering models using ensemble methods. In Zhi-Hua Zhou, Fabio Roli, and Josef Kittler, editors, *Multiple Classifier Systems*, volume 7872 of *Lecture Notes in Computer Science*, pages 1–12. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-38066-2. doi: 10.1007/978-3-642-38067-9_1. URL http://dx.doi.org/10.1007/978-3-642-38067-9_1.

Robin Burke. Hybrid web recommender systems. In *The adaptive web*, pages 377–408. Springer, 2007.

Wei Chen, Zhendong Niu, Xiangyu Zhao, and Yi Li. A hybrid recommendation algorithm adapted in e-learning environments. *World Wide Web*, pages 1–14, 2012.

Yen-Kuang Chen, Wenlong Li, and Xiaofeng Tong. Parallelization of adaboost algorithm on multi-core processors. In *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*, pages 275–280. IEEE, 2008.

Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM, 2010.

Simon Funk. Netflix update: Try this at home (december 2006). *URL http://sifter. org/˜ simon/journal/20061211. html.*

Jiawei Han, Micheline Kamber, and Jian Pei. *Data mining: concepts and techniques*. Morgan kaufmann, 2006.

Yehuda Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4(1):1, 2010.

Yehuda Koren and Robert Bell. Advances in collaborative filtering. In *Recommender Systems Handbook*, pages 145–186. Springer, 2011.

Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

Chris Seiffert, Taghi M Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. Resampling or reweighting: A comparison of boosting implementations. In *Tools with Artificial Intelligence, 2008. ICTAI'08. 20th IEEE International Conference on*, volume 1, pages 445–451. IEEE, 2008.

Xiangyu Zhao, Zhendong Niu, and Wei Chen. Interest before liking: Two-step recommendation approaches. *Knowl.-Based Syst.*, pages 46–56, 2013.