Hybrid Recommendation Models for Binary User Preference Prediction Problem

Siwei Lai[*] SWLAI@NLPR.IA.AC.CN
Yang Liu LIUYANG09@NLPR.IA.AC.CN
Huxiang Gu HXGU@NLPR.IA.AC.CN
Liheng Xu LHXU@NLPR.IA.AC.CN
Kang Liu KLIU@NLPR.IA.AC.CN
Shiming Xiang SMXIANG@NLPR.IA.AC.CN
Jun Zhao[†] JZHAO@NLPR.IA.AC.CN
National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing 100190 China

Rui Diao DIAORUI@LSEC.CC.AC.CN
Institute of Computational Mathematics and Scientific/Engineering Computing, Chinese Academy of Sciences, Beijing 100190 China

Liang Xiang LIANGXIANG@HULU.COM
Hang Li HANGLI@HULU.COM
Dong Wang WANGDONG@HULU.COM
Hulu R&D Beijing, Beijing 100080 China

Editor: G. Dror, Y. Koren and M. Weimer

Abstract

This paper presents detailed information of our solutions to the task 2 of KDD Cup 2011. The task 2 is called binary user preference prediction problem in the paper because it aims at separating tracks rated highly by specific users from tracks not rated by them, and the solutions of this task can be easily applied to binary user behavior data. In the contest, we firstly implemented many different models, including neighborhood-based models, latent factor models, content-based models, etc. Then, linear combination is used to combine different models together. Finally, we used robust post-processing to further refine the special user-item pairs. The final error rate is 2.4808% which placed number 2 in the Leaderboard.

Keywords: recommender system, collaborative filtering, item-based, latent factor model, model ensemble

1. Introduction

The tasks of KDD Cup 2011 (Dror et al., 2012) arise from recommender system. Recommender system is an important tool to help users find information of their interests by analyzing their historical behaviors. KDD Cup 2011 releases a large dataset of users' rating behaviors in Yahoo! Music and asks participants to design models which can predict user

[†] The team "The art of lemon" is lead by Siwei Lai, it is combined by two teams: "Lemon" (lead by Siwei Lai, Rui Diao) and "The art of war" (lead by Liang Xiang)
[†] Corresponding Author

preferences very well. It has two tasks. The first task is rating prediction problem. It asks participants to predict users' rating for items by analyzing users' previous ratings for items. This is very similar to the Netflix Prize (Bell and Koren, 2007). The main difference is that KDD Cup 2011 also releases some taxonomy data to describe relationships between tracks, albums, artists and genres. Competitors may use these content information to make more accurate prediction.

The task 2 is a new problem in recommender systems. This task asks participants to separate 3 tracks rated highly by each specific user from 3 tracks not rated by her. In this task, the most important thing is to predict which tracks the active users will rate. This is very similar to top-N recommendation task. We can easily regard it as a top-3 recommendation task. However, there are still some differences between task 2 and traditional top-N recommendation task. In many top-N recommendation tasks, recommending popular items may gain high accuracy, but it suffers from the problem of lacking of novelty and diversity (Hurley and Zhang, 2011). Furthermore, in the Yahoo! Music datasets, users may not like some popular tracks. The task 2 introduces negative samples for every user. These negative samples contain the items which are very popular but have not been rated by the user. Thus, in task 2, recommending popular items will not achieve high accuracy because negative samples include many popular items. Therefore, we regard it as binary user preference prediction problem rather than top-N recommendation problem.

To solve binary user preference prediction problem, we try to find some data insights. After analyzing the dataset carefully, we make two useful observations.

1. **Taxonomy data is useful in binary user preference prediction problem.** If a user has rated an artist/album, she will have large probability to rate the tracks of the artist/album.

2. **Time ordering of user's rating data is useful in binary user preference prediction problem.** User's rating data in task 2 may be ordered by time. Time ordering in user's rating data is useful as user's taste and mood may change. So we guess if two items are close in the rating sequence, they may have large similarity. This is the key discovery to gain high accuracy in task 2.

We mainly implement 3 types of different recommendation algorithms and improve them by using above two useful observations. These algorithms include content filtering, neighborhood-based collaborative filtering, latent factor model on both rating and binary data, etc. After that, we combine our models together by a linear model to gain high accuracy. Finally, we use robust post-processing to further refine the special user-item pairs.

The rest of the paper is organized as follows. Section 2 will introduce main notations in the paper and the methods of the validation dataset generation. In Section 3, we will propose different models used in the final solution. Model ensemble and post-processing methods are presented in Section 4. Experimental results are given in Section 5 and conclusions are made in Section 6.

2. Preliminary

2.1. Notations

We use many different models to solve the task 2. So, there are many different notations used in this paper. Table 1 lists some main notations all through the paper. In the paper, users are denoted by u, v. There are four types of items: track, album, artist, genre. Tracks are denoted by i, j, albums are denoted by b, artists are denoted by a, genres are denoted by g. The dataset also includes some taxonomy data. Given a track i, its album is denoted by $b(i)$ and its artist is denoted by $a(i)$.

Symbols	**Definition**
$\mathcal{U}$	users set
$\mathcal{I}$	items set
$\mathcal{I}_T$	tracks set
$\mathcal{I}_A$	artists set
$\mathcal{I}_B$	albums set
$\mathcal{I}_G$	genres set
r_{ui}	rating of user u assigned to item i
$a(i)$	artist of track i
$b(i)$	album of track i
n_i	popularity of track i (the number of users rated for this track)
ρ_i	popularity rank of track i in descending order
n_a	popularity of artist a
n_b	popularity of album b
$n_{i,s}$	popularity of item i which rated no less than s
$\mathcal{A}(a)$	set of tracks belongs to artist a
$\mathcal{B}(b)$	set of tracks belongs to album b
$\mathcal{I}(u)$	a set of items user u rated
$\mathcal{U}(i)$	a set of users who rated item i
$\mathcal{U}(i, T_r)$	a set of users who rated item i higher than T_r

Table 1: Main notations used in the paper

2.2. Validation Set Generation

The task 2 releases a training dataset and a test dataset. The training dataset contains 61,944,406 rating records from 249,012 users and 296,111 items, and the test dataset contains 607,032 user-item pairs from 101,172 users and 118,239 items. Competitors need to train models in training data and predict the test data.

The test dataset is divided into two sets, called “Test1” and “Test2”. Test1 is used to rank contestants on the Leaderboard and Test2 is used for the final ranking. We use Test1 to evaluate our models. In this paper, Test1 is denoted by $\mathcal{T}_S$.

In order to test models without submitting prediction results, we firstly generate a validation dataset $\mathcal{T}_V$ from the training data which has identical distribution with the test data. For user u in the test data, three tracks which are rated higher than 80 in the training data by u are extracted as positive samples and three tracks which have not been rated by the user u are selected as negative samples. However, there are usually more than three tracks with ratings higher than 80 or not rated by user u. Hence, similar to the official's selection, we choose the three positive samples randomly from the user's training tracks scored higher than 80 and select the negative samples according to the probability proportional to the number of high (80 or higher) ratings that a track received in the overall population. Additionally, neither the positive nor the negative samples should be included in the given 6 tracks in the test dataset. Fortunately, we get a validation dataset $\mathcal{T}_V$ which also includes 607,032 user-item pairs from 101,172 users and 118,239 items. In every submission, the error rate in the validation data is very similar to the error rate in the test data, the error rate in $\mathcal{T}_V$ is about 0.03% to 0.05% higher than the error rate in $\mathcal{T}_S$.

2.3. Evaluation Metric

The task 2 uses error rate as evaluation. Given a test dataset $\mathcal{T}$, $\hat{r}_{ui}$ or $\hat{r}(u,i)$ is denoted as user u's preference on item i. For user u, top 3 items with highest $\hat{r}_{ui}$ will be predicted as positive samples, and the other 3 items will be predicted as negative samples. The error rate of predictions in the test data $\mathcal{T}$ is denoted by $Error(\mathcal{T})$ throughout the paper.

3. Recommendation Models

3.1. Content-based Model

In task 2, we observe one important property of the taxonomy data.

- If a user has rated an artist/album, she will have large probability to rate the tracks of the artist/album.

Specifically, for all the people in the training set, a user will have a probability of 45% to rate a track if she has rated the track's artist. If she has rated an album, she will have a probability of 75% to rate the tracks of the album. For the users in the test set, the number will become 58% and 75%.

We use the above rule to design a simple content-based model and it can achieve error rate of 10.8006% in $\mathcal{T}_S$.

Following is the formulation of content-based model:

$$\hat{r}_{ui} = \widetilde{r}_{u,a(i)} + \widetilde{r}_{u,b(i)} + \frac{\sum_{j\in\mathcal{I}(u):a(j)=a(i)} r_{uj}}{\sum_{j\in\mathcal{I}(u):a(j)=a(i)} 1} + \frac{\sum_{j\in\mathcal{I}(u):b(j)=b(i)} r_{uj}}{\sum_{j\in\mathcal{I}(u):b(j)=b(i)} 1} + \beta(a(i),b(i)) \quad (1)$$

where $\widetilde{r}_{u,a(i)} = r_{u,a(i)}$ if user u has rated artist of track i, or $\widetilde{r}_{u,a(i)} = 0$ otherwise. $\beta(a(i),b(i)) =$ 35 if the both artist and album of track i are NONE or $\beta(a(i),b(i)) = 0$ otherwise.

It means that the track i is barely rated highly by the user u when $\hat{r}_{ui}$ equals zero. However, the missing values of artist and album cannot indicate anything (user u may like or dislike track i). So $\hat{r}_{ui}$ should be much larger than 0 when both artist and album of some track are NONE. In this case, the first four terms in Equation 1 are zeros, then we let $\beta(a(i),b(i)) = 35$ to achieve the best precision.

3.2. Item-based Collaborative Filtering Model

Item-based collaborative filtering (ItemCF) (Linden et al., 2003) is the most famous recommendation algorithm, it can be easily applied to task 2. In ItemCF, the preference of user u on item i is measured by:

$$\hat{r}_{ui} = \sum_{j \in S(\mathcal{I}(u),k)} w_{ij} \cdot (1 + r_{uj})^{\gamma_1} \tag{2}$$

where $S(\mathcal{I}(u), k)$ includes top k items in $\mathcal{I}(u)$ that are similar to item i, w_{ij} is the similarity between item i and item j, r_{ui} is the rating user u assigned to item i, and γ_1 is a parameter to control the impact of the ratings on final results. We use $\gamma_1 = 0.65$ in final results.

The simplest way to measure item similarity is using the Jaccard Index:

$$w_{ij} = \frac{|\mathcal{U}(i) \cap \mathcal{U}(j)|}{|\mathcal{U}(i) \cup \mathcal{U}(j)|} \tag{3}$$

where $\mathcal{U}(i)$ is a set of users who rate item i. Using this similarity metric can gain error rate of 9% in the test set.

The most important discovery is that we find the user rating data in the training data may be ordered by time. Such temporal information is valuable, as verified in past works Xiang et al. (2010). We make this assumption for two reasons. Firstly, in task 2, most of the users will firstly rate an artist and then rate the tracks of the artist. Secondly, the training data have anchoring effect. i.e. if a user gives a high rating to one item, she is more likely to give high rating to the next item she wants to rate. Furthermore, experimental results show that using this assumption can decline error rate significantly (about 1% improvement on one single model and 0.1% improvement on combined models, which will be illustrated in details in Section 5.2). Under this assumption, two items will have large similarity if they are close in the rating sequence. Let d_{ui} be position of item i in user u's rating sequence, then the similarity between items can be measured by:

$$w_{ij} = \frac{\sum_{u \in \mathcal{U}(i) \cap \mathcal{U}(j)} \frac{1}{|d_{ui} - d_{uj}|^{\gamma_3}}}{|\mathcal{U}(i) \cup \mathcal{U}(j)|} \tag{4}$$

We use $\gamma_3 = 1.1$ in final results.

If many users give high ratings to item i and low ratings to item j, these two items may have low similarity. So, rating information is also useful to measure item similarity. We add rating information into Equation 4 by the following formulation:

$$w_{ij} = \frac{\sum_{u \in \mathcal{U}(i) \cap \mathcal{U}(j)} \frac{1}{|d_{ui} - d_{uj}|^{\gamma_3} (1 + |r_{ui} - r_{uj}|)^{\gamma_4}}}{|\mathcal{U}(i) \cup \mathcal{U}(j)|} \tag{5}$$

We use $\gamma_4 = 0.2$ in final results.

To leverage on the taxonomy data, we add taxonomy information into Equation 2 in the following ways:

$$\hat{r}_{ui} = \sum_{j \in S(\mathcal{I}(u),k)} w_{ij} \cdot (1 + r_{uj})^{\gamma_1} \cdot (1 + \gamma_2 \cdot I(a(i) = j \vee b(i) = j)) \tag{6}$$

Here, $I(a(i) = j \vee b(i) = j) = 1$ if item j is the artist/album of item i, or $I(a(i) = j \vee b(i) = j) = 0$ otherwise. γ_2 is a parameter, and we use $\gamma_2 = 1.5$ in final result.

At last, we find that popular tracks have higher prediction scores than unpopular tracks in ItemCF's results. Due to the reason that general ItemCF algorithm usually result in Matthew effect (the popular items become more popular), we add a penalty function to popular items. Therefore our prediction function (Equation 6) can be converted to

$$\hat{r}_{ui} = \sum\nolimits_{j \in S(\mathcal{I}(u),k)} w_{ij} \cdot (1 + r_{uj})^{\gamma_1} \cdot (1 + \gamma_2 \cdot I(a(i) = j \vee b(i) = j)) \cdot \frac{1}{\sqrt{n_i}} \tag{7}$$

Here n_i is popularity of item i.

The ItemCF model consider many different factors. Table 2 shows the test set error rate of ItemCF after adding different factors. Experimental results show that temporal information and popular bias play the most important role in reducing error rate of predictions. The parameters $\gamma_1, \cdots, \gamma_4$ are set manually.

Factors	**Error Rate%**
initial model (Eqn. 2 & 3)	8.8950
+ removing popular bias	5.2467
+ using temporal information	3.9125
+ using rating information	3.8341
+ using taxonomy information (Eqn. 7 & 5)	3.6407

Table 2: Test set error rate of ItemCF after using different information

3.3. Latent Factor Model on Rating Data

In task 2, the positive samples are tracks that users will rate higher than 80, and the negative samples are tracks that users will not rate. In this way, rating data are also very useful. We also apply two famous latent factor model, RSVD (Paterek, 2007) (Equation 8) and SVD++ (Koren, 2008) (Equation 9) to task 2 to predict users' rating on tracks in the test data. Table 3 list error rate of these two models.

$$\hat{r}_{ui} = \mu + b_u + b_i + \boldsymbol{p}_u^T \boldsymbol{q}_i \tag{8}$$

$$\hat{r}_{ui} = \mu + b_u + b_i + \boldsymbol{q}_i^T (\boldsymbol{p}_u + \frac{1}{\sqrt{|\mathcal{I}(u)|}} \sum_{j \in \mathcal{I}(u)} \boldsymbol{y}_j) \tag{9}$$

Model	**Latent Factor Number**	**Error Rate %**
RSVD	200	17.1486
SVD++	200	26.4302

Table 3: Test set error rate of two latent factor model on rating data

3.4. Binary Latent Factor Model

Latent Factor Model (LFM) have been used in Netflix prize competition and achieved great success. There are many different types of LFM, such as RSVD, NSVD, SVD++, etc. However, LFM are mostly used in rating prediction task, such as task 1 in KDD Cup, few researchers have used it in top-N recommendation problem with binary user behavior data because binary user behavior data only contain positive samples. Pan et al. (2008) proposed that LFM can be applied to binary user behavior data after generating negative samples from missing values randomly. Pan et al. (2008) proposed many different methods to generate negative samples, such as using all missing values as negative samples, sampling negative samples by uniform distribution, user-oriented sampling, item-oriented sampling.

In order to apply LFM to binary user preference prediction problem, we need to firstly generate some negative samples for every user. For user u, let $\mathcal{I}(u)$ be items that user u rated, and $\mathcal{I}(u, T_r)$ be items that user u rated higher than score r. Then, given a rating threshold T_r, we will randomly choose $|\mathcal{I}(u, T_r)|$ items that the user have not rated with a probability proportional to number of their high ($\geq T_r$) ratings. Then, we use Equation 10 as our predictor:

$$\hat{r}_{ui} = \boldsymbol{p}_u^T \boldsymbol{q}_i \tag{10}$$

where $\hat{r}_{ui}$ is the prediction of the probability user u rate item i highly, $\boldsymbol{p}_u$ is latent factor vector of user u and $\boldsymbol{q}_i$ is latent factor vector of item i. This model is denoted by BinarySVD in the paper.

Let $\mathcal{K}^+$ be all positive samples and $\mathcal{K}^-$ be all negative samples sampled from missing values. Then, model parameters $\boldsymbol{p}_u$, $\boldsymbol{q}_i$ are trained by minimizing the following cost function:

$$\sum_{(u,i)\in\mathcal{K}^+\cup\mathcal{K}^-} (r_{ui} - \boldsymbol{p}_u^T \boldsymbol{q}_i)^2 + \lambda(||\boldsymbol{p}_u||^2 + ||\boldsymbol{q}_i||^2) \tag{11}$$

Here, $r_{ui} = 1$ if user-item pair (u, i) belongs to positive samples set $\mathcal{K}^+$ and $r_{ui} = 0$ if user-item pair (u, i) belongs to negative samples set $\mathcal{K}^-$. Stochastic gradient-descent is used to minimize the cost function. We use bagging technique (Breiman, 1996) which means resampling is made in each iteration to further improve model's performance. Detailed algorithms are shown in Algorithm 1. We set the max iteration number to 30 steps in the experiment.

BinarySVD is very simple and do not use any taxonomy data provided in the task. It can get 6% error rate in the test set. Since taxonomy data is very important in KDD Cup task 2, we also design a complex BinarySVD model by using more taxonomy data. The model is defined in Equation 12 and it is denoted by BinarySVD+ in the paper.

$$\hat{r}_{ui} = b_u + b_i + b_{a(i)} + b_{b(i)} + b_{u,I(a(i)\in\mathcal{I}(u))} + b_{u,I(b(i)\in\mathcal{I}(u))} + \boldsymbol{p}_u^T(\boldsymbol{q}_i + \boldsymbol{x}_{a(i)} + \boldsymbol{y}_{b(i)}) \tag{12}$$

$a(i)$ is artist of item i if i is track or album, $a(i) = i$ if i is artist and $a(i) = -1$ if i is genre. $b(i)$ is album of item i if i is track, $b(i) = i$ if i is album and $b(i) = -1$ if i is artist or genre. $I(\cdot)$ is indicator function, $I(a(i) \in \mathcal{I}(u)) = 1$ if $a(i) \in \mathcal{I}(u)$ and $I(a(i) \in \mathcal{I}(u)) = 0$ otherwise. $I(b(i) \in \mathcal{I}(u)) = 1$ if $b(i) \in \mathcal{I}(u)$ and $I(b(i) \in \mathcal{I}(u)) = 0$ otherwise. $\boldsymbol{p}_u$ is the latent factor vector for user u, $\boldsymbol{q}_i$ is the latent factor vector for item i, $\boldsymbol{x}_{a(i)}$ is latent factor vector for artist $a(i)$, and $\boldsymbol{y}_{b(i)}$ is latent factor vector for artist $b(i)$. We also use stochastic gradient-descent to learn model parameters in BinarySVD+ and this model gain 3.49% error rate in the test set when latent factor number equals 500.

Algorithm 1: Pseudo code of binary latent factor model.

```
Data: user rating data $R$, rating threshold $T_r$, learning rate $\alpha = 0.01$, regularization parameter $\lambda = 0.01$
Result: parameters of binary latent factor model
for $step \leftarrow 0$ to 30 do
    foreach $u \in \mathcal{U}$ do
        $\mathcal{I}^+(u,\gamma) \leftarrow$ positive samples  $\mathcal{I}^-(u,\gamma) \leftarrow$ negative samples  foreach $i \in \mathcal{I}^+(u,\gamma) \cup \mathcal{I}^-(u,\gamma)$ do
            if $i \in \mathcal{I}^+(u,r)$ then
                $r_{ui} \leftarrow 1$
            else
                $r_{ui} \leftarrow -1$
            end
            $\hat{r}_{ui} \leftarrow \boldsymbol{p}_u^T \boldsymbol{q}_i$  $e_{ui} \leftarrow r_{ui} - \hat{r}_{ui}$  foreach latent factor $f$ do
                $p_{uf} \leftarrow p_{uf} + \alpha \cdot (e_{ui} q_{if} - \lambda p_{uf})$  $q_{if} \leftarrow q_{if} + \alpha \cdot (e_{ui} p_{uf} - \lambda q_{if})$
            end
        end
    end
end
```

3.5. Neighborhood-based Binary SVD Model

Similar to SVD++ model (Koren, 2008) in rating prediction task, we also design a neighborhood-based BinarySVD model in task 2:

$$r_{ui} = b_u + b_i + b_a + b_b + b_{u,I(a(i)\in\mathcal{I}(u))} + b_{u,I(b(i)\in\mathcal{I}(u))} + \frac{1}{\sqrt{|\mathcal{I}(u)|}} \boldsymbol{q}_i^T \Big(\sum_{j\in\mathcal{I}(u)} \boldsymbol{y}_j \Big) \tag{13}$$

Here, $\mathcal{I}(u)$ includes all items user u rated in the training data. In this model, users are modeled by items they rated ($\sum_{j\in\mathcal{I}(u)} \boldsymbol{y}_j$). Stochastic gradient-descent is used to train the model.

4. Model Ensemble and Post-processing

4.1. Model Ensemble

Many researches prove that combining numerous models together can gain higher accuracy than using one single model. For example, in the Netflix Prize, the wining team has used more than 100 models in their final solution. We use linear combination to do model ensemble and use simulated annealing (SA) (Laarhoven and Aarts, 1987) to train the combination parameters.

Several steps are adopted as follows in order to do model ensemble. Firstly, we split the training dataset ($\mathcal{K}$) into the local training dataset ($\mathcal{K}_L$) and the local test dataset (the validation dataset) ($\mathcal{T}_V$). The generation method of the validation dataset has been discussed in Section 2. Secondly, we train each single model in the local training dataset $\mathcal{K}_L$ and make predictions in the validation dataset $\mathcal{T}_V$. Thirdly, we train each model in the whole training dataset $\mathcal{K}$ and make predictions in the test set $\mathcal{T}_S$. Finally, linear combine model is trained in the validation dataset $\mathcal{T}_V$ and predictions are made in the test set $\mathcal{T}_S$.

Before model ensemble, we firstly normalize prediction results of each model in the validation dataset/the test dataset. Let $\hat{r}_L(u,i)$ be the normalized prediction results on the validation data. Normalization will make sure that mean value of all predictions equals 0 and the standard deviation of all predictions equals 1.

The final predictions on both the validation data and the test data are the linear combination of all models:

$$\hat{r}_L(u,i) = \sum_k \beta_k \hat{r}_L^k(u,i) \tag{14}$$

$$\hat{r}_S(u,i) = \sum_k \beta_k \hat{r}_S^k(u,i) \tag{15}$$

where $\hat{r}_L^k(u,i)$ is the normalized prediction results of user u's preference on item i by kth model on the validation data, $\hat{r}_S^k(u,i)$ is the normalized prediction results of user u's preference on item i by kth model in the test set, and β_k is the linear combination parameter for the kth model.

Ensemble parameters $\boldsymbol{\beta}$ are learned by minimize error rate of predictions in the validation dataset:

$$\boldsymbol{\beta} = \arg\min_{\boldsymbol{\beta}} Error(\sum_k \beta_k \hat{r}_L^k) \tag{16}$$

Since the error rate function $Error(\cdot)$ is a combinatorial function, SA is used to minimize it. Algorithm 2 is pseudo code of model ensemble. In the algorithm, $RandomModify(\cdot)$ is a function to change parameters randomly in every step.

Algorithm 2: Pseudo code of Simulated Annealing.

```
Data: $\hat{r}_L^k(u,i)$ from different models on the validation data
Result: linear combination parameters $\boldsymbol{\beta}$
$\boldsymbol{\beta} \leftarrow$ random number vector  $error \leftarrow Error(\boldsymbol{\beta})$  $T \leftarrow 10$  while $T > \varepsilon$ do
    $\boldsymbol{\beta}' \leftarrow RandomModify(\boldsymbol{\beta})$  $error' \leftarrow Error(\boldsymbol{\beta}')$  if  $\min(\exp(\frac{error-error'}{T}),1) > rand(0,1)$ then
        $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}'$  $error \leftarrow error'$
    end
    $T \leftarrow T \times 0.99$
end
Function: $RandomModify(\boldsymbol{\beta})$
begin
    $\boldsymbol{\beta}' \leftarrow \boldsymbol{\beta}$  if $rand(0,1) > 0.5$ then
        Choose two different positions $i$, $j$ randomly   $x \leftarrow \min(\beta_i,\beta_j) \times rand(0,0.2)$  $\beta'_i \leftarrow \beta_i + x$  $\beta'_j \leftarrow \beta_j - x$
    else
        Choose one position $i$ randomly   $\beta'_i \leftarrow \beta_i \times rand(0.8,1.2)$
    end
    return $\boldsymbol{\beta}'$
end
```

In order to overcome the problem of over-fitting in model ensemble, we also apply 8-fold cross validation to the validation dataset to get linear combination parameters. Firstly, the the validation dataset is divided into 8 parts. Then, 7 parts are used to train combination parameters with 1 part left. After training for 8 times, we will get 8 different combination parameters, and the final combination parameters is the average of the 8 different combination parameters. Experimental results show that cross validation can improve prediction accuracy significantly.

4.2. Post-processing

After combining all models, we get the predictions at the validation data $\mathcal{T}_V$ and the test data $\mathcal{T}_S$. Now, we can use post-processing methods to model some special data features. Firstly, we extract a subset of user-item pairs by some rules, then the prediction of these pairs in both $\mathcal{T}_V$ and $\mathcal{T}_S$ will be multiple by a parameter α to minimize the prediction error in $\mathcal{T}_V$. After applying one rule to extract user-item pairs and do post-processing, we use another rule on the new prediction results in $\mathcal{T}_V$ and $\mathcal{T}_S$. Pesudo code of post-processing are shown in Algorithm 3.

We want to find subsets of user-item pairs from the test data set, in which their results predicted by our models are worse than the performance of their own complementary sets in the test data set. In the contest, we used 9 different rules to select user-item pairs and do post-processing. Table 4 shows all symbols used in post-processing which are not listed in Table 1. The rules in the table are ordered by the time we discover them. However, we also try different orders, but they generate similar results.

Symbols	Definition
$\phi_{u,a}$	$\min_{i\in\mathcal{I}(u)} n_{a(i)}$
$\Phi_{u,a}$	$\max_{i\in\mathcal{I}(u)} n_{a(i)}$
$\phi_{u,b}$	$\min_{i\in\mathcal{I}(u)} n_{b(i)}$
$\Phi_{u,b}$	$\max_{i\in\mathcal{I}(u)} n_{b(i)}$
$\psi_{u,a}$	$\min_{i\in\mathcal{I}(u),a(i)=a} r_{ui}$
$\Psi_{u,a}$	$\max_{i\in\mathcal{I}(u),a(i)=a} r_{ui}$
$\psi_{u,b}$	$\min_{i\in\mathcal{I}(u),b(i)=b} r_{ui}$
$\Psi_{u,b}$	$\max_{i\in\mathcal{I}(u),b(i)=b} r_{ui}$

Table 4: Symbols used in post-processing

In the third line of Table 4, $n_{b(i)}$ is the popularity of the album of item i, and $\phi_{u,b}$ is denoted as the minimum popularity of $b(i)$ in the rated items of user u. In the first line of Table 5, the rule matches the ratings satisfying that the popularity of album $b(i)$ divided by $\phi_{u,b(i)}$, which was defined in the third line of Table 4, is less than 8, and $b(i)$ must be also rated by user u. In other words, Rule 1 finds out the user-item pairs that the user both rated item i and its album $b(i)$, and the popularity of $b(i)$ is not too larger than some other item with the same album $b(i)$.

We have listed all the post-processings in Table 5 in the form of equation. Here we explain some of them. Rule 7 is used to select the top 5 popular tracks. Our models do not

Id	Rules
1	$n_{b(i)}/\phi_{u,b(i)} < 8, b(i) \in \mathcal{I}(u)$
2	$n_{b(i),80}/n_{a(i),80} > 2$
3	$\lvert\mathcal{A}(a(i))\rvert < 8, n_i > 3000$
4	$\Psi_{u,b(i)} < 60$
5	$b(i) \in \mathcal{I}(u), r_{u,b(i)} \leq 30$
6	$b(i) \in \mathcal{I}(u), r_{u,b(i)} = 100$
7	$\rho_i < 6$
8	$r_{u,b(i)} = 90, r_{u,a(i)} = 10$
9	$n_i = \min_{j,b(j)=b(i)} n_j$

Table 5: Rules used in post-processing

perform very well in these tracks, so we should refine their prediction results. While rule 8 is adopted to find out some "strange" ratings. The users rated very high for some track's album, but very low for the same track's artist. We don't know why the users behave like that, but we know our models do not handle these user-item pairs very well. The other rules almost act in the same way. We choose these rules so that we can improve the performance of our model in both the validation dataset and the test dataset.

Algorithm 3: Pseudo code of post-processing.

```
Data: user rating data $R$, prediction results in the validation dataset $\mathcal{T}_V$ and the test dataset $\mathcal{T}_S$.
Result: new prediction results in the validation dataset $\mathcal{T}_V$ and the test dataset $\mathcal{T}_S$
foreach Rule $\kappa$ in rules set do
    Extract subset $\mathcal{K}_\kappa$ of user-item pairs by the rule  $e \leftarrow 1$  $\alpha \leftarrow 1$  for $\alpha' \leftarrow 1.2; \alpha' \geq 0.8;$
    $\alpha' * = 0.99$ do
        foreach $(u,i) \in \mathcal{K}_\kappa \cap \mathcal{T}_L$ do
            $\hat{r}_{ui} \leftarrow \hat{r}_{ui} \cdot \alpha'$
        end
        $e' \leftarrow$ error rate in $\mathcal{T}_V$  if $e' < e$ then
            $e \leftarrow e'$  $\alpha \leftarrow \alpha'$
        end
    end
    foreach $(u,i) \in \mathcal{K}_\kappa \cap (\mathcal{T}_L \cup \mathcal{T}_S)$ do
        $\hat{r}_{ui} \leftarrow \hat{r}_{ui} \cdot \alpha$
    end
end
```

5. Experiments

5.1. Results of Different Models

In the contest, 19 different models are used in the final model ensemble. Table 6 lists detailed information of these models, including model type, and error rate in the test set. In the table, many models have the same type, but they differ in model parameters. Furthermore, results show that BinarySVD+ has the best performance as a single model. It can gain 3.5362% error rate in the validation set, and 3.4941% error rate in the test set.

Id	Model Type	Error Rate (%)
1	Content-based	10.8006
2	ItemCF	3.6407
3	ItemCF	3.8044
4	ItemCF	3.8341
5	ItemCF	4.2830
6	ItemCF	4.7682
7	ItemCF	4.7709
8	ItemCF	5.6330
9	BinarySVD	6.0515
10	BinarySVD	6.4047
11	BinarySVD	7.1098
12	BinarySVD+	**3.4941**
13	BinarySVD+	3.5278
14	NBinarySVD	6.7163
15	NBinarySVD+	3.7965
16	NBinarySVD+	3.8064
17	UserCF	15.4837
18	RSVD	17.1486
19	SVD++	26.4302

Table 6: Test set error rate of different models

5.2. Model Ensemble Results

We apply 8-fold cross validation (CV) to model ensemble. Table 8 shows final ensemble parameters of every model. If temporal information is removed from our ItemCF model (Id = 2, 3 and 4 in Table 6), the error rate will increase by almost 0.1%, as shown in Table 7.

Table 8 shows that the best single model does not have the largest combination parameter. The ItemCF model(Id = 4) has the largest combination parameter. One possible reason of this is that the model has used the temporal information in the training data, and other models have not. Each of the 19 models are necessary, removing any of them will increase the error rate by at least 0.02%.

	Validation set	Test set
All models	2.5593%	2.5033%
Models without temporal information	2.6400%	2.5985%

Table 7: Validation set rate and Test set error rate of predictions by model ensemble

Id	Model Type	β_{Id}
1	Content-based	0.002
2	ItemCF	0.079
3	ItemCF	0.231
4	ItemCF	**0.438**
5	ItemCF	0.030
6	ItemCF	0.001
7	ItemCF	0.001
8	ItemCF	0.013
9	BinarySVD	0.003
10	BinarySVD	0.011
11	BinarySVD	0.020
12	BinarySVD+	0.006
13	BinarySVD+	0.017
14	NBinarySVD	0.003
15	NBinarySVD+	0.033
16	NBinarySVD+	0.025
17	UserCF	0.049
18	RSVD	0.027
19	SVD++	0.010

Table 8: Linear combination parameters in model ensemble

5.3. Post-processing Results

After model ensemble, we use post-processing to gain lower error rate. 9 rules are used in post-processing to extract user-item pairs and Table 9 shows error rate in the test set after applying every rule. Overall, post-processing can reduce validation set error rate from 2.5593 to 2.5102, and reduce test set error rate from 2.5033 to 2.4808.

Id	Rules	Error Rate%
1	$n_{b(i)}/\phi_{u,b(i)} < 8, b(i) \in \mathcal{I}(u)$	2.4924
2	$n_{b(i),80}/n_{a(i),80} > 2$	2.4919
3	$\lvert\mathcal{A}(a(i))\rvert < 8, n_i > 3000$	2.4913
4	$\Psi_{u,b(i)} < 60$	2.4894
5	$b(i) \in \mathcal{I}(u), r_{u,b(i)} \leq 30$	2.4888
6	$b(i) \in \mathcal{I}(u), r_{u,b(i)} = 100$	2.4873
7	$\rho_i < 6$	2.4864
8	$r_{u,b(i)} = 90, r_{u,a(i)} = 10$	2.4841
9	$n_i = \min_{j,b(j)=b(i)} n_j$	2.4808

Table 9: Experimental results of post-processing

6. Conclusion and Future Work

Most of the researchers focus on solving rating prediction problems since Grouplens publish their first article (Resnick et al., 1994) in recommender systems. However, predicting which items users will rate is more important than predicting which rating the active user will assign to the given items, because the first problem is a kind of top-N recommendation problem which is the real problem of many on-line recommender systems. Furthermore, the solution of first problem can be also easily applied to implicit feedback data, such as watching history, browsing history, etc. The main contribution of the task 2 problem is that it proposed a way to avoid popularity bias in top-N recommendations by using popular items as negative samples. Our solution to the binary user preference prediction problem is implementing 3 types of different recommendation algorithms and improving them by using our two useful observations. In future, we will do more online test to see if this method can really help us increase recommendation quality in real system.

Taking genre data into consideration is another direction for improvement in future work. The experimental results show that genre data have little impact on final results mainly because of the following two reasons. First, each track has many different genre labels. We tried to find the most popular genre labels, and re-label all the tracks, unfortunately it does not work very well. Second, there are some human factors in labeling these genre data, in other words, the genre data itself is inaccurate to some extent. In future, we will try to consider these two main problems in order to further improve the predicting accuracy in real recommender systems.

Acknowledgments

The work is supported by the National Natural Science Foundation of China (Grant No. 61070106), the National Basic Research Program of China (Grant No. 2012CB316300), and Tsinghua National Laboratory for Information Science and Technology (TNList) Cross-discipline Foundation.

References

Robert M. Bell and Yehuda Koren. Lessons from the Netflix prize challenge. *SIGKDD Explor. Newsl.*, 9:75–79, December 2007.

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, August 1996.

Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. The Yahoo! music dataset and KDD-Cup'11. *Journal Of Machine Learning Research W&CP*, 18:1–12, 2012.

Neil Hurley and Mi Zhang. Novelty and diversity in top-n recommendation - analysis and evaluation. *ACM Trans. Internet Techn.*, 10(4):14, 2011.

Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '08, pages 426–434, 2008.

P. J. M. Laarhoven and E. H. L. Aarts, editors. *Simulated annealing: theory and applications*. 1987.

G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, January 2003.

Rong Pan, Yunhong Zhou, Bin Cao, Nathan N. Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, pages 502–511, 2008.

Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. In *Proc. KDD Cup Workshop at SIGKDD'07, 13th ACM Int. Conf. on Knowledge Discovery and Data Mining*, pages 39–42, 2007.

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*, CSCW '94, pages 175–186, 1994.

Liang Xiang, Quan Yuan, Shiwan Zhao, Li Chen, Xiatian Zhang, Qing Yang, and Jimeng Sun. Temporal recommendation on graphs via long- and short-term preference fusion. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 723–732, 2010.