Novel Models and Ensemble Techniques to Discriminate Favorite Items from Unrated Ones for Personalized Music Recommendation

Todd G. McKenzie, Chun-Sung Ferng, Yao-Nan Chen, Chun-Liang Li, Cheng-Hao Tsai, Kuan-Wei Wu, Ya-Hsuan Chang, Chung-Yi Li, Wei-Shih Lin, Shu-Hao Yu, Chieh-Yen Lin, Po-Wei Wang, Chia-Mau Ni, Wei-Lun Su, Tsung-Ting Kuo, Chen-Tse Tsai, Po-Lung Chen, Rong-Bing Chiu, Ku-Chun Chou, Yu-Cheng Chou, Chien-Chih Wang, Chen-Hung Wu, Hsuan-Tien Lin, Chih-Jen Lin, Shou-De Lin

Department of Computer Science and Information Engineering, National Taiwan University

{d97041, r99922054, r99922008, b97018, b97705004, b96018, b96025, b96069, b96113, b95076, b97042, b97058, b96092, b96110, d97944007, r98922028, r99922038, b97114, r99922095, b96115, d98922007, b96055, htlin, cjlin, sdlin}@csie.ntu.edu.tw

ABSTRACT

The track 2 problem in KDD Cup 2011 (music recommendation) is to discriminate between music tracks highly rated by a given user from those which are overall highly rated, but not rated by the given user. The training dataset consists of not only user rating history but also the taxonomic information of track, artist, album, and genre. This paper describes the solution of the National Taiwan University team which ranked first place in the competition. We exploited a diverse of models (neighborhood models, latent models, BPR-based models, and random-walk models) with local blending and global ensemble to achieve 97.45% in accuracy on the testing dataset.

1 Introduction

The current work is based on a competition for recommendation, an area of research popularized by the recent Netflix competition[7, 16, 13]. A departure from the standard mode of item rating prediction, the task of KDD-Cup 2011 Track 2 requires discrimination of tracks rated highly by a given user from tracks which are highly rated in general. In other words, in the context of tracks which are highly rated, what distinguishes the given user from his or her peers? This paper describes the solution proposed by the National Taiwan University team through a course "Machine Learning and Data Mining: Theory and Practice" that consists of three instructors, three TAs, and 19 students.

The Yahoo! Labs KDD-Cup 2011 Track 2 Dataset[3] finds its origin in Yahoo! Music, where users may rate tracks, albums, artists, and genres. Information about this taxonomy is provided for over 90% of rated tracks.

Spanning $249,012$ users and $296,111$ items, the provided training and testing datasets consist of $61,944,406$ and $607,032$ records, respectively. For each user in the test set, six songs are given: three tracks rated highly by the user (positives) and three not rated by the given user, but rated highly by other users (negatives). The task is to classify these six tracks in binary fashion to minimize the percentage error.

This paper is organized as follows: Section 2 describes our general framework, Section 3∼6 describes different types of models we exploit, Section 7 and 8 describing our blending and ensemble methods, and we provide our take home points in Section 9. We provide the notation description and other supplemental materials in the appendix.

2 Framework and Global Settings/Strategies

2.1 The Architecture of Our System

Our solution can be divided into three core stages: models, blends, and ensembling, as shown in Figure 1.

In the first stage, several models are designed and applied independently for this problem. In the blending stages, some blending models that consist of a linear or non-linear combinations of the results from a small portion of single models are generated. We then utilize the individual model results along with the blend results in the ensemble process, which is made up of two stages. Such general framework is similar to what the Netflix winner has used and our solution to track 1 in KDD Cup 2011.

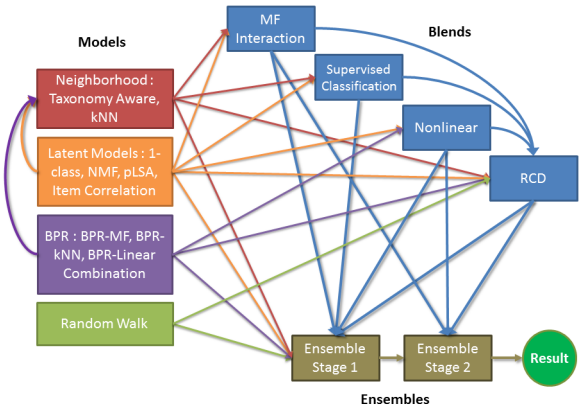


Figure 1: Discrimination Framework

The strategies and methods described in this section are used by multiple models in our framework. In Appendix B, we create a table showing which techniques were used by which individual models.

2.2 Sampling

The goal is to divide the six items into two groups: one group which contains tracks highly rated by the user and one which contains highly rated tracks (i.e. rated ≥ 80 by many other users) not yet rated by the given user. Since in the provided dataset negative examples are less transparent (i.e. only given ratings are given), a reasonable first-step is to sample negative items for each user as training data for our models.

We therefore follow the method that was used to produce the testing dataset to sample a set of negative examples. The probability that a track is sampled as negative for a particular user is proportional to the number of high ratings (≥ 80) it received in the training data. In a portion of our models, a variation named taxonomic sampling is used to reject a negative sample if its corresponding album or artist has been rated by the given user, as in this case the track is more likely to have been rated previously by the user. The number of negative examples per user is set to the number of items the user has rated. We re-sample negative data in every iteration during optimization for most of the models. For some models that require exact item rating values r_{ui} of negative examples, they are designed as a parameter for optimization, and generally we found values between -1 and -100 to be suitable.

2.3 Validation Set

A validation set is usually needed for parameter adjusting to avoid overfitting. For efficiency purpose, we did not do cross-validation but simply choose an individual validation set to serve such purpose. Unlike Track 1, a validation set was not provided for Track 2 by the Cup organizers. An internal validation set was thus created from the training data utilizing the same method as that for creating the test dataset as described within the KDD-Cup website[1]. The items in the validation dataset are exclusively tracks. Three were randomly selected from the user's highly rated items (positives), and three were sampled with probability proportional to the number of times it receives high ratings (negatives). This internal validation dataset is used for evaluating individual models results and for optimizing blending and ensemble methods. We find that the trend of performance of our models for the validation data is highly consistent with that of the leaderboard which gives us high confidence to use such a validation set to evaluate the quality of our models internally.

2.4 Residual-Based Models

A residual-based model generally works along with a basic prediction model. The basic model is first trained for prediction, and then its residuals are used as the inputs to train a residual-based model. During prediction, a similar procedure is applied and the outputs from both models are summed up as the final output.

2.5 Adaptive Learning Rate

An adaptive learning rate was employed in a number of the methods outlined in our framework. The idea behind this method is to test whether the decrease in validation error rate between iterations has dropped below a given threshold. If it has, we decrease the learning rate. This strategy worked 0.1∼0.6% better than fixed learning rate in most of the models.

[1] `http://kddcup.yahoo.com`

2.6 Quasi-Album/Artist Data

We propose a method to model the taxonomic structure through enriching our training dataset by adding some additional sample data includes:

- Quasi-Album data : each track with album information in both training and testing datasets is replaced by its album
- Quasi-Artist data: each track or album with artist information specified in both training and testing datasets is replaced by its artist

In effect, if a user rates more than one track per album, then there will be more than one album rating by the user. The experiments show that adding such data provide 0.9∼1.8% improvement for certain models, which is considered as very significant in this competition.

2.7 Pseudo-Taxonomy

Roughly 9% of the tracks in the official training dataset lack album information. For each of them, we first represent it using a vector of binary elements where each element represent whether a specific user rated it, then we apply k-means clustering algorithm on these vectors to group those tracks and finally assign a pseudo-album to each group. Similarly, we assigned pseudo-artists to the tracks without artist information. Furthermore, we preform similar clustering on every item (including those with official taxonomic information), providing an additional track feature. Several of our models reach 0.4∼0.9% improvement in accuracy using this information.

3 Neighborhood Models

3.1 Taxonomy-Aware Model

It is natural to assume that a user may rate a track high given having rated the associated album high. Based on this conjecture, we consider each album as a group that contains the tracks which belong to it. Similarly, each artist and genre can also be treated as a group. A taxonomy-aware neighborhood model is described as below:

Let $\mathfrak{G}$ be the set of all groups, and $\mathfrak{G}_i$ be the set of groups containing item i. Then, the score of user u for item i is defined as

$$s_{ui} = \frac{1}{|\mathfrak{G}_i|} \sum_{j \in R(u)} (r_{uj} + 1) \cdot \left(\sum_{G \in \mathfrak{G}_i \cap \mathfrak{G}_j} \frac{1}{|G|} \right) \qquad (1)$$

The term $\sum_{G \in \mathfrak{G}_i \cap \mathfrak{G}_j} \frac{1}{|G|}$ represents the similarity between item i and j, which is the sum of weights of the common groups containing items i and j. The weight of each group is set to the inverse of its size as we believe that two items which coexist in a smaller group are likely to be more similar. The score is normalized by the number of groups the item i belongs to $\frac{1}{|\mathfrak{G}_i|}$.

The items with top-3 s_{ui} scores are labeled `1`, while the others are labeled `0`. The validation error of this model is 7.5123% and the Test1 error is 7.2959%.

3.1.1 Regularization Terms

We find that the scores of items with little or no group information are under-estimated in our model, so two regularization terms are added into Eq. 1 to mitigate this issue.

$$s_{ui} = \frac{1}{\mid \mathfrak{G}_i \mid} \sum_{j \in R(u)} (r_{uj} + 1) \left(\lambda + \sum_{G \in \mathfrak{G}_i \cap \mathfrak{G}_j} \frac{1}{\mid G \mid + \lambda_g} \right) \qquad (2)$$

Table 1: Regularization Augmentation of Neighborhood Models Results

Correlations	λ_g	λ	%Test1
P	0	0	5.8515
BPR100, CUS	0	0	5.5003
BPR1000, CUS, P	0	0	5.1676
BPR1000, CUS, P, C	0	0.001	3.9978
BPR1000, CUS, P, C, K	100	0.005	3.8862
BPR2000, CUS, P, C, K	100	0.005	3.8797

CUS = common-user-support, P=Pearson, C=Cosine, K=Kulczynski, $BPRn$ represents the correlation trained by BPR-kNN with n features.

where λ_g is used to counter the impact of small group-size and λ is used to compensate for neighbor items which have no common group with i.

3.1.2 Augmenting by Correlation Measures

We refine the previous formulation by considering the correlation between two items.

$$s_{ui} = \frac{1}{|\mathfrak{G}_i|} \sum_{j \in R(u)} (r_{uj}+1) \cdot \left(\lambda + \sum_{G \in \mathfrak{G}_i \cap \mathfrak{G}_j} \frac{1}{|G| + \lambda_g} \right) \cdot Corr(i,j) \tag{3}$$

The correlation between two items is based on their ratings by users. For each item, we first create a vector $\mathbf{v}_i$ whose length is equivalent to the number of users. An element in $\mathbf{v}_i$ is 0 if the corresponding user has not rated it and $\ln(N_i/|R(u)|)$ otherwise. We consider several correlation measures among $\mathbf{v}_i$ and $\mathbf{v}_j$ as $Corr(i,j)$:

- Cosine: $\frac{\mathbf{v}_i^T \mathbf{v}_j}{\|\mathbf{v}_i\|_2 \cdot \|\mathbf{v}_j\|_2}$
- Common user support: $\frac{\mathbf{v}_i^T \mathbf{v}_j}{\|\mathbf{v}_i\|_1 \cdot \|\mathbf{v}_j\|_1}$
- Kulczynski's coefficient: $\frac{1}{2}\left(\frac{\mathbf{v}_i^T \mathbf{v}_j}{\|\mathbf{v}_i\|_2^2} + \frac{\mathbf{v}_i^T \mathbf{v}_j}{\|\mathbf{v}_j\|_2^2}\right)$
- Pearson's correlation coefficient: $\frac{N_u \cdot \mathbf{v}_i^T \mathbf{v}_j - \|v_i\|_1 \|v_j\|_1}{\sqrt{(N_u \|v_i\|_2^2 - \|v_i\|_1^2) \cdot (N_u \|v_j\|_2^2 - \|v_j\|_1^2)}}$

Besides these common correlation measures, the Bayesian Personalized Ranking (BPR) correlation generated by the BPR-kNN model (see Section 5.2) may also be used as $Corr(i,j)$ in the neighborhood model. Also, multiple correlation measures can be applied concurrently. Here we consider only non-linear combinations as

$$Corr(i,j) = \prod_k Corr_k(i,j) \tag{4}$$

Table 1 shows the results of different variations. It should be noted that with the large number of items, the calculation of item-item similarities/correlation measures presents a challenge. While the number of items is large, the number of items within the validation/test datasets is quite limited. In addition, only the items in $R(u)$ are paired with i for validation and testing (u,i) tuples. The computation can be done efficiently via the use of sparse data representation and parallelization. We pre-calculate correlation measures for the nearly 250M effective pairs and store to disk, requiring roughly 3G of memory and disk space.

3.2 User-based k-Nearest Neighbors

As the previous neighborhood model considers only item-based neighbors, we also implement user-based neighborhood approaches. The main idea is to predict how a user rates an item by checking how similar users rated the item. Adopting the k-Nearest Neighbor (kNN) approach, the score of a user u to an item i is defined as

$$s_{ui} = \left(\frac{1}{ln(|R(i)| + 10)} \right)^m \sum_{\substack{\text{top-}k \text{ sim.} \\ v \in R(i)}} sim(v,u) \tag{5}$$

where m is a parameter set to 2 and k is a parameter set to 20 in our experiment.

Unlike items, there is no taxonomy for users. For this, we define an asymmetric similarity measure between users:

$$sim(v,u) = \frac{1}{|R(v)|} \sum_{i \in R(u) \cap R(v)} \frac{1}{|R(i)|} \tag{6}$$

The validation error of this model is 8.7112%. Note that this kNN approach is also applied on the item side. Although it produces inferior results as compared to the user-based model, it is still useful in the final ensemble.

3.3 Predicting on Neighbors

The kNN approach can also be used to improve the results of other models, such as pLSA. For example, to predict the rating of a user-item pair, we first use pLSA to predict the ratings of the top-k neighbors of this item by pLSA, and then weigh the neighbor scores by their cosine similarity to this item. Finally, the weighted scores of the top-k neighbors are summed as the prediction for this user-item rating. This method improves the validation error of pLSA with 200 features from 8.8447% to 8.1462%, and from 4.8742% to 4.6548% for BPR-MF with 200 features.

4 Models that Exploit Latent Information

Models based on collaborative filtering approaches often exploit latent information or hidden structures of users and items from the given ratings. We modify these models to solve the Track 2 problem, which can be considered a one-class collaborative filtering(OCCF) problem [12] as the user-item pairs given in training data are all rated. The negative examples are sampled as described in Section 2.2, and the latent information is learned from the existing and sampled data using MF, pLSA and pPCA models.

4.1 Matrix Factorization

MF methods have been widely used in CF to predict ratings [7, 16, 13]. At the core, MF methods approximate a rating r_{ui} by the inner product of an f-dimensional user feature vector $\mathbf{p}_u$ and an item feature vector $\mathbf{q}_i$. A standard method to learn $\mathbf{p}_u, \mathbf{q}_i$ is to minimize the sum of square errors between the predicted and actual ratings in the training data and often L2 regularization is applied to prevent overfitting.

However, directly applying MF to predict ratings for discrimination yields poor performance for the Track 2 task. This is a reasonable finding as the ratings in the training do not provide sufficient information about negative instances. To provide for negative samples, we exploit the negative sampling technique introduced in Section 2. The MF technique is then applied to a more condensed matrix, now containing negative ratings. This model is called OCCF-MF. For prediction, the six tracks associated with each user are ranked by the score $s_{ui} = \mathbf{p}_u^T \mathbf{q}_i$. The top three items are assigned label `1`, while the remaining items are assigned label `0`.

We propose several variations of OCCF-MF to further improve its performance.

4.1.1 Optimization Methods

A typical optimization method for MF is Stochastic Gradient Descent (SGD) which we combine with Adaptive Learning Rate (ALR), as described in Section 2 to form our naïve MF solver. In the SGD algorithm, we update one positive and one negative instance in each turn. For acceleration, we utilize Coordinate Descent (CD) for solving the OCCF-MF optimization problem for its fast convergence properties. By solving each $\mathbf{p}_u$ and $\mathbf{q}_i$ separately, the time complexity of updating becomes linear. In practice, the initial value of all $\mathbf{p}_u$ and $\mathbf{q}_i$ are set to $\sqrt{\bar{r}/f}$, where $\bar{r}$ is the average rating in the training data [11]. Also, the rating values of the sampled negative data should be set to a value which makes $\bar{r}$ zero otherwise the CD approach tends to overfit easily.

4.1.2 Quantizing Ratings

The task of Track 2 can be regarded as a classification task. If we focus solely on whether a user rates an item, we may discard the actual rating values and set $r_{ui} = 1$ for rated instances, and $r_{ui} = -1$ for sampled negative instances. Applying OCCF-MF on this kind of data leads to our two-class variant.

Furthermore, the items with high and low ratings may be separated into two groups as in Track 2 only highly rated items are considered positive. Thus we quantize $r_{ui} = 1$ for high ratings and $r_{ui} = 0$ for low ratings. The sampled negative data, we still set $r_{ui} = -1$. Again, OCCF-MF procedure is applied to solve this three-class variant.

4.1.3 Ranking Objective

The task of Track 2 problem may alternatively be viewed as a ranking problem, where the six items of a specific user are ranked by their estimated ratings. To reflect this variation, we change the optimization objective in OCCF-MF.

- pairwise ranking variant:

$$\min_{P,Q} \sum_u \sum_{i,j} \max(0, -(r_{ui} - r_{uj})(\mathbf{p}_u^T \mathbf{q}_i - \mathbf{p}_u^T \mathbf{q}_j))$$

- one-sided regression (OSR) [17] variant:

$$\min_{P,Q} \sum_u \sum_{i,j} \Big(\max(0, r_{ui} - \mathbf{p}_u^T \mathbf{q}_i)\Big)^2 + \Big(\max(0, \mathbf{p}_u^T \mathbf{q}_j - r_{uj})\Big)^2$$

In both objectives, i stands for positive examples, while j stands for the sampled negative example.

4.1.4 Optimization with Constraint

Following non-negative MF [9], we may perform OCCF-NMF by adding the constraint that all entries of P and Q are non-negative. The constraints are satisfied using projected gradient method during optimization [**?**]. Note that the negative ratings are shifted to non-negative before training. SGD with adaptive learning rate is used to optimize OCCF-NMF, using the same details as that of OCCF-MF. In fact, the only difference between the OCCF-MF and OCCF-NMF methods is the non-negative constraint.

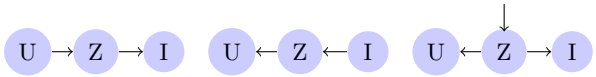
Based on the idea of neighborhood models, another variant is formed by adding the constraint

$$\mathbf{p}_u = \frac{1}{|R(u)|} \sum_{i \in R(u)} \frac{r_{ui} + 1}{101} \mathbf{q}_i$$

The prediction function $\mathbf{p}_u^T \mathbf{q}_i$ can then be viewed as a item-based neighborhood model with low rank approximation of the correlation matrix. We call this variant Item Correlation MF (ICMF).

4.2 Probabilistic Latent Semantic Analysis

Probabilistic Latent Semantic Analysis (pLSA) generally models the hidden latent structure in the data [6]. It introduces f unobserved variables Z with each occurrence of an user-item pair in our context. The conditional probabilities are learned using the Expectation-Maximization (EM) algorithm. There are several ways to model the dependency, as shown in the following figure.



Regular $P(Z|U)P(I|Z)$ | Inverse $P(Z|I)P(U|Z)$ | Symmetric $P(Z)P(U|Z)P(I|Z)$

We find the inverse model has superior performance and therefore it is this version which is included in our final ensemble. Note that the rating is not modeled here, instead we use the rating information to control the updated weight in the M-step of EM. That is, the distribution of higher-rated items are updated in a faster manner than their lower-rated counterparts. Moreover, the tempered EM (TEM) [6] algorithm is applied as an alternative optimization method.

4.3 Probabilistic Principle Component Analysis

Probabilistic Principle Component Analysis (pPCA) uses the EM algorithm to determine the principle axes of observed data [15]. This method is equivalent to Probabilistic Matrix Factorization (PMF) [8]. We included this approach into our solution with the modification of sampled negative data. At the beginning of pPCA, some unrated user-item pairs are sampled as negative data. The principle axes are learned from both the original training data and sampled negative data. The parameter f decides the number of principle axes to learn, or equivalently the number of features in PMF.

4.4 Other Exploited Strategies

In addition to the basic models and variants, some ideas are applied to further improve the performance.

- The parameters of models, including learning rate, regularization weight, and rating value of negative samples are fine tuned by an automatic parameter tuner[16].
- Besides the ordinal training set, the quasi-album/artist data described in Section 2 is also used to generate models for further blending.
- With the same training data, several models can be generated with different random initial values. Combining these models helps to improve the final result.

The models with their parameters and results are shown in Table 2.

5 Bayesian Personalized Ranking

The goal of Bayesian Personalized Ranking (BPR)[14] is to optimize the pair-wise ranking of items for each user. We believe this idea is especially suitable for the Track 2 task as BPR allows the explicit distinction of favorite items from unrated ones. Specifically, pairwise ranking of items is optimized for each individual user. We choose three basic learning models for BPR: Matrix Factorization, K-Nearest Neighbors and Linear Combination of User/Item features to generate ratings for comparison. We use Stochastic Gradient Descent as the solver. In each step SGD updates a pair of items. One item in the pair represents the positive item (i.e. high rating), while the other represents the negative item.

Table 2: Results of latent information models

Model	Method	f	%Valid
OCCF-MF	SGD	3200	4.2514
OCCF-MF	ALR	3200	4.1197
OCCF-MF	CD	800	7.3571
OCCF-MF	SGD[a]	800	6.5644
OCCF-MF w/ quasi-album	SGD[a]	800	6.8052
OCCF-MF w/ quasi-artist	SGD[a]	800	9.9814
OCCF-MF w/ taxonomy sample	SGD	3200	4.3840
OCCF-NMF	ALR	800	4.7230
OCCF-NMF	ALR	3200	4.0400
OSR	SGD	800	16.3692
2-class	SGD	800	6.5279
3-class	SGD	800	5.0735
pairwise ranking	SGD	800	6.9874
ICMF	ALR	200	7.8737
ICMF & pairwise ranking	ALR	800	7.9485
pLSA	EM	100	9.8374
pLSA	EM	200	8.7818
pLSA	EM	2000	7.2695
pLSA on album data	EM	500	8.1712
pLSA on artist data	EM	500	8.4780
pLSA	TEM	2000	5.4643
pLSA	TEM	2000	5.3997
pPCA	EM	20	9.2460

[a] The sampled negative data is fixed in these models. Also, it updates all positive instances of one user before updating the negative instances of that user.

5.1 BPR: Matrix Factorization

We use the BPR-MF[14] to optimize the pairwise ranking of items that are rated using the MF method. Our approach is slightly different from the original BPR-MF approach in the following respects:

1. We use explicit ratings instead of implicit feedback information.
2. The learning rates are chosen based on the rating difference of a pair of items.

$$\eta(u,i,j) = \frac{r_{ui} - r_{uj} + \alpha}{100}$$

where α is a small number between 0 and 10. Furthermore, we use the adaptive learning rates, specifically halving the rate if the improvement of the validation error is within 0.01% over 10 iterations.

3. We set the regularization terms to 0 or very small numbers such as 0.001.
4. We use non-negative MF because OCCF-NMF outperforms OCCF-MF.
5. We use the quasi-album/artist data in training.

5.2 BPR: kNN

We follow the standard training approach[14], but modify the kNN prediction formula. We add a rating term and a regularization term $(r_{ui} + \lambda)$ to be multiplied with the item similarity term c as shown in equation 7.

$$s_{ui} = \sum_{j \in R(u), j \neq i} (r_{ui} + \lambda) c_{ij} \tag{7}$$

We hereby take the rating information into account as we believe the highly rated items and the lower rated ones should convey different preferences in kNN. The term λ is very important for Track 2 as it distinguishes the unrated items from zero-rated items. The training complexity is proportional to the number of ratings per user, which can become a problem as the number of rating increases. For users with a large number of ratings, we randomly sample a subset of their ratings from the training set.

The memory requirement for this method is 3-4GB for the $62,551,438$ ratings using standard dynamically allocated 2-dim arrays. While ratings are used in a piecewise fashion, this is done to mitigate time not memory complexity. Indeed, all ratings are stored to memory when performing sampling, for example.

Table 3: BPR-MF Results

Variants	#features	α	#iter	%Valid	%Test1
-	200	-	94	6.2364	6.0484
QAL	200	-	174	5.9002	-
QAL	200	-	192	5.8712	5.8885
QAR	200	-	190	8.4407	-
RES	50	10	356	5.9272	5.8852
WLR	50	10	242	6.3506	-
WLR	200	10	228	5.2679	5.2798
WLR	400	10	154	5.3177	-
ALR	400	5	610	4.6614	4.6817
ALR & NMF	400	5	354	4.8699	4.8448
ALR & NVT	400	5	434	4.5530	4.5867

QAL/QAR: quasi-album/artist data
RES: residual data of OCCF-MF with $f = 400$
WLR: weight learning rate by rating difference
ALR: adaptive learning rate, $\eta = 0.01$ initially
NMF: non-negative MF
NVT: reject sampled ratings in validation/testing data
Regularization: $\lambda_u = \lambda_i = 10^{-3}$, $\lambda_j = 10^{-4}$ for ALR, and $\lambda_u = \lambda_i = \lambda_j = 0$ for others.

Table 4: BPR K-Nearest Neighbor Results

#ratings[a]	#features	#iter	η	%Valid	%Test1
100	200	142	0.008	8.4398	8.4307
100	2000	224	0.006	7.4853	-

[a] The maximum number of ratings per user used in training. For users with more ratings, only a subset of this size is sampled for training.

5.3 BPR: Linear Combination

Here we fix either the user or item matrix in MF, while updating the other. The fixed matrix is obtained from other model types such as OCCF-MF. We call it BPR-Linear Combination as the columns to be learned are in the form of a linear combination.

6 Random Walk

Random Walk (RW) [2] based methods are performed on graphs that generally include the items to be rated, and eventually use the stationary probability of the walkers on a node as the prediction of the rating for that node. Such ratings can then be utilized to discriminate items in Track 2. Random Walk algorithms themselves are unlikely to outperform the mainstream algorithms such as an MF-based one, but are nevertheless useful since they can provide diversity into our ensemble as it is the only algorithm we exploit which explicitly considers the higher-order relationships among items and users. Variations of RW come from different kinds of graphs used, different surfing strategies, and different initialization and restarting methods.

Table 5: BPR Linear Combination Results

Methods	Features from	%Valid	%Test1
Linear MFuser	OCCF-MF	6.1789	-
F400	$\lambda_i = \lambda_j = 0$, itr=212		
Linear MFitem	OCCF-MF	5.7101	5.7373
F400	$\lambda_u = 0.0005$, itr=40		
Linear BPRKNN	BPR-kNN	6.8850	6.7645
F2000	$\lambda_u = 0.0005$, itr=184		
Square MFitem	OCCF-MF	5.3559	5.3551
F400	$\lambda_u = 0.0005$, itr=110		
Square BPRKNN	BPR-kNN	6.6859	6.2687
F1000	$\lambda_u = 0.0005$, itr=86		

Linear: use features generated by other models directly
Square: use original values and their squares as new features
Fn: This means the feature dimension.

Table 6: Best Single Model Comparison

Type	Model	%Valid
Neighborhood	Taxonomy Aware	4.2864
Neighborhood	User-based kNN	8.7112
Neighborhood	Predict on Neighbors	4.6548
Latent Information	MF	4.0400
Latent Information	pLSA	5.3997
Latent Information	pPCA	9.2460
BPR	BPR-MF	4.5530
BPR	BPR-kNN	7.4853
BPR	BPR-Linear Combination	5.3559
Random Walk	Random Walk	6.0929

6.1 Query Centered Random Walk on Taxonomy Graph

Our experimental results indicate that moving random surfers on a item-user graphs yields an inferior performance comparing to moving them on the item-relationship graph (i.e. the taxonomy). The formula

$$r(t) = (1-\alpha) \cdot A \cdot r(t-1) + \alpha \cdot q \tag{8}$$

where $r(t)$ is a stationary distribution of the random surfer after t iterations, A is a symmetric adjacency matrix describing the connection of item taxonomy, q can be treated as the restarting prior preference distribution of the specified user, and α is a real number in [0,1].

6.1.1 Enhancing Similar Items

We wish to rank the six items associated with each user. A naïve idea is to assign q as the ratings of the user to items. However, performing this operation on the hierarchy graph indeed abandons the opportunity to exploit item similarity. Here we propose a new idea to multiply the item similarity (i.e. user-based Pearson correlation values) with the ratings to generate each element of q (note that we adjust the negative correlation values to 0, and then add 1 for each rated item to distinguish them from the unrated ones). Then for each item to be rated, we generate its own q and execute RW to produce its rating.

6.1.2 Enhancing RW-model by Neighborhood Info

The predicted ratings generated previously are adjusted using other tracks in the pseudo-taxonomy as described in Section 2.

$$s_{ui} = r_{u,i} + \frac{1}{|G_i|} \sum_{t \in G_i} \frac{r_{u,t}}{|G_t|} + c_{u,i} \tag{9}$$

where the reinforcing term is

$$c_{u,i} = \begin{cases} \sum\limits_{t \in R(u)} \frac{r_{u,t} p_{i,t}}{K} & : i \text{ has no album or artist} \\ 0 & : \text{otherwise} \end{cases}$$

where $r_{u,i}$ is the RW score, G_i contains the items directly related to i in the actual and pseudo-taxonomy and $p_{i,t}$ is the Pearson correlation of i and t. We further adjust the predictions for the tracks with no album/artist information by adding an additional value which captures item similarity. Note that the number K is usually very large in order to avoid the dominance of such a term.

6.2 Experiment

In pre-processing, we assign tracks without albums or artist to one of 50 pseudo-albums or artists. We also create 20 additional pseudo-groups, every item being assigned to one. For Random Walk, we use $\alpha = 0.3$, and K=10^6 in the reinforcing term. We find $t = 2$ obtains the best performance. The results reach 6.1% in validation.

7 Blending

The aim of blending is to combine a subset of single models to boost performance, producing diverse hybrid models for the final ensemble. We use both linear and non-linear blending methods in our methodology. Table 6 summarizes the best single model performance of models in our framework.

7.1 Score Transformation

Prior to blending, a transformation of the outputs of each model into a consistent format is used as described below.

- raw score: no transformation
- normalized score: normalized to $[0, 1]$
- global ranking: replace the score by its rank among all validation and testing instances, normalizing to $[0, 1]$
- user local ranking: replace the score by its rank among the six items associated with the same user, normalizing to $[0, 1]$

Two (or more) transforms can be used concurrently in blending methods. For example, the scores from M models are transformed into $2M$ inputs for blending as if there were $2M$ models. Furthermore, we may apply 2-degree polynomial expansion on the transformed score to produce $\frac{2M(2M+1)}{2}$ additional inputs.

7.2 MF Interaction Blending

The Track 2 problem requires the system to address two issues: whether an item is rated by the user, and if so, whether the rating is high. Our neighborhood and pLSA models focus more closely on the first issue, while the MF model addresses the second.

We combine the MF model with either neighborhood or pLSA models by multiplication, $(MF)^x \cdot (NM \text{ or } pLSA)$. Here we use standard MF and not OCCF-MF as we wish to restrict focus to "whether the rating is high." The optimal exponential term x can be learned through validation data as shown in Table 7.

7.3 Supervised Classification

The task of Track 2 can be regarded as a classification task, where highly rated items are in class `1` and unrated items are in class `0`. At the onset, this problem is very

Table 7: MF Interaction Blending Results

Model multiplied with MF	x	%Valid	%Test1
Taxonomy-aware Neighborhood	6	5.4510	4.8159
User-based kNN	1	6.9423	-
pLSA	1.2	5.2712	5.2139

Individual Test1 error rate: MF 15.3534%, taxonomy-aware neighborhood 5.8168%, pLSA 5.4643%. Validation error rate: user-based kNN 8.7112%

unlike one of classification due to the lack of explicit features. In our data, a rating is only associated with (u, i) and the taxonomy of i. To apply classification algorithms to our task, we first extract meaningful features from inputs and individual models, and then exploit a classifier to learn combination strategies. The classification algorithms investigated are support vector machines (SVM) and neural networks (NN). For prediction, the top-3 ranked testing items (according to the decision value of the classifier) are considered to be in class `1`, while the remaining three are considered to be in class `0`.

7.3.1 Feature Engineering

Two kinds of features are used: factorization-based features and taxonomy-based features. We take the user and item feature vectors $(\mathbf{p}_u, \mathbf{q}_i)$ learned by MF or BPR-MF methods as the factorization-based features of users and items. To utilize the full power of MF and BPR-MF, we also include the prediction score (i.e. the inner product of user and item feature vectors) as one feature.

The taxonomy-based features consist of one prediction score from a variant of a taxonomy-aware neighborhood model, and 8 features describing the taxonomic information of the testing user-item pair. The prediction score is obtained from Eq (1), except that the normalization term $\frac{1}{\mathfrak{G}_i}$ vanishes.

The taxonomic features are listed below.

- binary indicator of whether the corresponding album/artist of this track was rated by this user
- rating of this user of the corresponding album/artist
- the proportion of items rated by this user in the corresponding album/artist/genre groups
- the proportion of items rated by this user in the union of all groups which contain this track

In the event that the taxonomic information for a given track is missing, we use the pseudo-taxonomy.

Finally, 2-degree polynomial expansion is performed to further expand the feature space.

7.3.2 Results

LIBLINEAR [4] is used as the SVM solver, with parameters `-g 0.125 -c 1`. We also implemented a neural network version containing 1 hidden layer, 50 neurons, and a learning rate of 10^{-5}. The results are shown in Table 8. Note that for B200 the number of features is prohibitively large, thus we include only the prediction score as model-based features for classification.

7.4 Nonlinear Blending

We utilize non-linear methods to capture more information through blending. AdaBoost, LogitBoost and Random Forest methods provided by Weka [5] are exploited to blend the taxonomy-based features (with pseudo-taxonomy) described in Section 7.3, together with the scores predicted by BPR-MF, pLSA and NMF models. The results are shown in Table 9.

Table 8: Supervised Classification Result

Model	Features[a]	%Valid
SVM	TX	8.2391
	TXP	7.4566
	B50	5.9928
	B50, TX	3.5237
	B50, TXP	3.5003
	M50, TX	5.5872
	N50, TX	4.2340
	N50, TXP	4.1853
	B200, TX	3.6571
	B200, TXP	3.6341
NN	N50, TX	4.9595
	M50, TX	4.2311
	B50, TXP	3.9085

[a] TX: taxonomy-based features without pseudo-taxonomy
TXP: taxonomy-based features with pseudo-taxonomy
B50: factorization-based features: BPR-MF with $f = 50$
B200: prediction score of BPR-MF with $f = 200$
M50: factorization-based features: OCCF-MF with $f = 50$
N50: factorization-based features: OCCF-NMF with $f = 50$

Table 9: Nonlinear Blending Performance

Method	Features	%Valid
Adaboost	taxonomy	7.5581
Random Forest	taxonomy	9.0275
Adaboost	taxonomy+models	5.3496
Random Forest	taxonomy+models	5.5243
Logitboost	taxonomy+models	3.8555

7.5 Linear Blending by Random Coordinate Descent

Random Coordinate Descent (RCD) is an algorithm for non-smooth optimization [10]. The main idea is to iteratively update the weight vector by choosing a random direction and an optimal descent step. To apply RCD for linear blending, we begin with a base model, and iteratively update it with a random linear combination of models.

We first discuss the base model and random combinations, then introduce the update procedure. Finally, we present the usage of bootstrap aggregation to further improve the performance.

7.5.1 Base Model and Random Linear Combinations

The base model is a linear combination of individual models, where the weight of each model is learned by linear regression for pairwise ranking. Let $\mathbf{x}_{u1}, \cdots, \mathbf{x}_{u6}$ be the vectors of scores predicted on the six validation items of user u, and $b_{u1}, \cdots, b_{u6}$ be the binary label (`1` or `0`). The base model is formed by the solution of

$$\min_{\mathbf{w}} \sum_{u} \sum_{i,j \in \{1,\cdots,6\}, b_{ui} \neq b_{uj}} ((b_{ui} - b_{uj}) - \mathbf{w}^T(\mathbf{x}_{ui} - \mathbf{x}_{uj}))^2$$

We borrow an idea from sparse coding to choose candidates for binary blending. In iteration t, we blend the best existing blended model with another model M_t. M_t itself is a random linear combination of k_t arbitrary models where

$$k_t = \begin{cases} 1 & : \text{ if in iteration } t-1 \text{ blending yields better results} \\ k_{t-1} + 1 & : \text{ otherwise} \end{cases}$$

7.5.2 Optimal Linear Blending of Two Models

The updating procedure aims to solve the optimal linear blending of two models. We use an efficient algorithm to

Table 10: RCD Blending Results

M	transform[a]	K or[b] $B \times K$	%Valid	%Test1
101	G	20000	2.7175	2.6333
18	G,N,P	10000	2.6691	2.5825
27	G,N,P	10000	2.5995	2.5706
27	G,N,P	90×1000	2.6084	2.5515
24	G,N,P	132×1000	2.6463	-

[a] G: global ranking, N: normalized score, P: 2-degree polynomial expansion (see Section 7.1)

[b] Bootstrap aggregation is applied if × appears.

solve this problem. Let x_{ui} denotes the score of the i-th validation item of user u predicted by one model, and y_{ui} be the score predicted by another model. We would like to determine w_x, w_y to minimize the error rate when ranking the items by $s_{ui} = w_x x_{ui} + w_y y_{ui}$. When fixing $w_x = 1$, the error rate is piecewise constant respect to $r = \frac{w_y}{w_x}$ and the changes only happen when two items with different labels swap their ranks. Every r must satisfy $x_{ui} + r y_{ui} = x_{uj} + r y_{uj}$ for some $i, j \in \{1, \cdots, 6\}$ and $b_{ui} \neq b_{uj}$.

For each user, there are only nine such rs, and we can readily calculate the error for each instance to determine the one which minimizes the error. The detailed algorithm is listed in Algorithm 1. It takes only $O(N_u \log N_u)$ time, which can be completed for Track 2 data in a couple of seconds on standard hardware.

7.5.3 Bootstrap Aggregation

To avoid overfitting, we apply bootstrap aggregation [1]. We generate B new validation sets by sampling instances with replacement from our validation set. Blending by Random Coordinate Descent is done on each of the B sets independently, resulting B $\mathbf{w}$s. Binary predictions are made by each $\mathbf{w}$, and the final score s_{ui} is the number of $\mathbf{w}$s which predict `1` on this u, i pair.

The results of blending by RCD are shown in Table 10. The models to be blended are subsets of our models. K is the number of iterations in RCD.

8 Ensemble

We believe that ensemble methods in this late stage should be simple aggregate functions (e.g. linear) to avoid overfitting. We investigate three types of simple ensembles: voting, average, and weighted average.

We use two distinct versions of the weighted average ensemble: single-phase and three-phase. In the single-phase solution, we consider all of the track data per user while performing the weight optimization phase. However, in the three-phase solution, we divide this task into three phases. In the first phase, weights are learned to assign the most plausible positive instance and the most plausible negative instance. In the second phase, a different set of weights are learned to assign among the remaining 4 unassigned instances the most likely positive and negative instances. In the third phase, we perform the same operation on the two remaining instances. Thus, there is a phase weight vector $\mathbf{w}$ for each phase which minimizes the validation error, selecting the pair of previously unassigned tracks which also minimizes the error at each phase.

As mentioned in our framework in Section 2, our ensemble process was divided into two stages. The former utilizes the global ranking score transformation (as discussed in the blending section), while the latter combines a set of ensemble results from the first stage together with the blending models using raw score to produce the final outputs.

9 Discussion

Track 2 of the KDD-Cup 2011 competition presented a new challenge in recommendation. Instead of the standard prediction of rating or similar metric on items in the dataset, this task instead requires the discrimination of two types of highly rated items: items rated high by the given user and the items rated highly by others, but not rated by the given user.

During the course of the competition, we made several observations that are worth sharing.

- We find diversity to be very important and in combining many diverse models great performance improvement can be made. Both exploiting different optimization techniques for blending and blending the same models using different parameters proved useful.
- As we aim to classify tracks rather than predict their ratings (the latter akin to Track 1 of the KDD-Cup 2011), we find utilizing models which optimize the track ranking and rated/unrated status is prudent.
- We observe that taxonomic information is very useful, so developing multiple efficient methods which make effective use of this information is important.
- As a validation set was not provided and the submission chances to the leaderboard are limited, creating a good validation set for internal use is crucial. The error rate of our validation set was quite consistent with the leaderboard score.
- We find that interaction between different models is very useful. For example, using the output of one model as the input of another.
- It is useful to perform negative sampling to capture implicit feedback.
- Raw score transformation, such as ranking, is useful for ensembling the predictions of multiple models.

10 Acknowledgments

We thank the organizers for holding this interesting and meaningful competition. We also thank National Taiwan University for providing a stimulating research environment. Moreover, we thank Prof. Jane Hsu for the valuable comments and thank En-Hsu Yen for fruitful discussions. This work was supported by the National Science Council, National Taiwan University and Intel Corporation under Grants NSC99-2911-I-002-201, 99R70600, and 10R70500.

11 References

[1] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, August 1996.

[2] H. Cheng, P. N. Tan, J. Sticklen, and W. F. Punch. Recommendation via query centered random walk on k-partite graph. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 457–462. IEEE, 2007.

[3] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer. he Yahoo! Music Dataset and KDD-Cup'11. *KDD-Cup Workshop*, 2011.

[4] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[5] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. *The WEKA data*

mining software: an update, 2009.

[6] T. Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the Twenty-Second Annual International SIGIR Conference on Research and Development in Information Retrieval (SIGIR-99)*, 1999.

[7] Y. Koren. The bellkor solution to the netflix grand prize. Tech. report, 2009.

[8] N. D. Lawrence and R. Urtasun. Non-linear matrix factorization with Gaussian processes. In *ICML*, 2009.

[9] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, 2001.

[10] L. Li and H.-T. Lin. Optimizing 0/1 loss for perceptrons by random coordinate descent. In *IJCNN*, pages 749–754. IEEE, 2007.

[11] C.-C. Ma. Large-scale collaborative filtering algorithms. Master's thesis, National Taiwan University, 2008.

[12] R. Pan, Y. Zhou, B. Cao, N. Liu, R. Lukose, M. Scholz, and Q. Yang. One-class collaborative filtering. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 502–511. IEEE, 2008.

[13] M. Piotte and M. Chabbert. The Pragmatic Theory solution to the Netflix Grand prize. Tech. report, 2009.

[14] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 452–461. AUAI Press, 2009.

[15] M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *JRSS-B*, 61:611–622, 1999.

[16] A. Töscher and M. Jahrer. The BigChaos solution to the Netflix grand prize. Tech. report, 2009.

[17] H.-H. Tu and H.-T. Lin. One-sided support vector regression for multiclass cost-sensitive classification. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 1095–1102, Haifa, Israel, June 2010. Omnipress.

APPENDIX

A Notation

The mathematical notation listed below are used in this paper.

- u, v - user ID
- i, j - item ID
- r_{ui} - the real rating of item i given by user u
- s_{ui} - the predicted score of user u for item i
- $R(u)$ - the set of all items rated by user u in the Train and Validation datasets
- $R(i)$ - the set of all users who gave rating to item i in Train and Validation datasets
- N_u - the number of users
- N_i - the number of items
- N_{val} - the number of u, i pairs in validation set
- $\alpha, \beta, \gamma, \delta$ - parameters in each model, which can be tuned
- η - the learning rate in SGD algorithm
- λ - regularization weight in SGD algorithm
- %Valid - Error rate on validation set
- %Test1 - Error rate on Test1 (leaderboard)

B Model Properties

Table 11 outlines the properties of the models used in our framework.

Table 11: Model Properties

Type	Model	Sampled	Taxonomy	Residual	ALR	Quasi	Pseudo
Neighborhood	Taxonomy Aware		✓				✓
Neighborhood	User-based kNN						
Neighborhood	Predict on Neighbors			✓			
Latent Information	MF	✓			✓	✓	
Latent Information	pLSA					✓	
Latent Information	pPCA	✓					
BPR	BPR-MF	✓		✓	✓	✓	
BPR	BPR-kNN	✓			✓		
BPR	BPR-Linear Combination	✓			✓		
Random Walk	Random Walk		✓				✓

ALR: Adaptive Learning Rate
Quasi: Quasi-album/artist data
Pseudo: Pseudo-taxonomy

C Algorithms

Algorithm 1 Optimal Linear Blending of Two Models

Input: $x_{u1} \cdots x_{u6}$ and $y_{u1} \cdots y_{u6}$ for each user u
Output: the optimal w_x and w_y

```
S ← ∅
for all user u appears in validation set do
  for all i, j ∈ {1, ··· , 6} do
    if the i-th and j-th item have different labels then
      r ← (x_uj − x_ui) / (y_ui − y_uj)
      Δerror ← E_u(r + ε) − E_u(r − ε) {E_u(r) is the number of errors on the 6 items of user u when taking score s_ui = x_ui + r y_ui.}
      S ← S ∪ {(r, Δerror)}
    end if
  end for
end for
sort S according to r in increasing order
r_old ← −∞, err ← Σ_u error(r_old, u)
T ← ∅
for all (r, Δerror) ∈ S do
  if r ≠ r_old then
    T ← T ∪ {((r_old + r)/2, err)}
    r_old ← r, err ← err + Δerror
  end if
end for
T ← T ∪ {(∞, err)}
(r_min, err_min) ← min_err(T)
(r_max, err_max) ← max_err(T)
if N_val − err_max < err_min then
  w_x ← −1, w_y ← −r_max
else
  w_x ← 1, w_y ← r_min
end if
```

D Predictor List

In this section we list all 127 models which participated in our final ensemble. The listed error rate is measured on our internal validation set.

D.1 Neighborhood Predictors

- NBH-1: error: 7.6378%
  Taxonomy-aware neighborhood model, $\lambda = \lambda_g = 0$, no normalized term $\frac{1}{\mathfrak{S}_i}$.
- NBH-2: error: 6.7591%
  Taxonomy-aware neighborhood model, $\lambda = \lambda_g = 0$.
- NBH-3: error: 6.3937%
  Taxonomy-aware neighborhood model augmented by Pearson's correlation, $\lambda = \lambda_g = 0$.
- NBH-4: error: 6.0227%
  Taxonomy-aware neighborhood model augmented by Pearson's correlation, $\lambda = \lambda_g = 0$, pseudo-taxonomy.
- NBH-5: error: 6.4678%
  Taxonomy-aware neighborhood model augmented by Pearson's correlation, $\lambda = \lambda_g = 0$, square $(r_{uj} + 1)$.
- NBH-6: error: 6.0303%
  Taxonomy-aware neighborhood model augmented by Pearson's correlation, $\lambda = \lambda_g = 0$, square $(r_{uj} + 1)$, pseudo-taxonomy.
- NBH-7: error: 5.8086%
  Taxonomy-aware neighborhood model augmented by cosine similarity, common user support and square of BPR-kNN correlation with $f = 1000$, $\lambda = \lambda_g = 0$.
- NBH-8: error: 5.8168%
  Taxonomy-aware neighborhood model augmented by cosine similarity, Pearson's correlation, common user support and square of BPR-kNN correlation with $f = 1000$, $\lambda = \lambda_g = 0$.
- NBH-9: error: 4.2858%
  Taxonomy-aware neighborhood model augmented by cosine similarity, Pearson's correlation, common user support, Kulczynski's coefficient and BPR-kNN correlation with $f = 1000$, $\lambda = 0.005$, $\lambda_g = 100$.
- NBH-10: error: 4.6670%
  Taxonomy-aware neighborhood model augmented by cosine similarity, Pearson's correlation, common user support, Kulczynski's coefficient and square of BPR-kNN correlation with $f = 1000$, $\lambda = 0.00005$, $\lambda_g = 0$.

- NBH-11: error: 4.4960%
  Taxonomy-aware neighborhood model augmented by cosine similarity, Pearson's correlation, common user support, Kulczynski's coefficient and square of BPR-kNN correlation with $f = 1000$, $\lambda = 0.005$, $\lambda_g = 0$.
- NBH-12: error: 8.7112%
  User-based kNN, $k = 20$, $m = 2$
- NBH-13: error: 8.7752%
  User-based kNN, $k = 20$, $m = 2$, $\ln(|R(i)|)$ instead of $\ln(|R(i)| + 10)$
- NBH-14: error: 8.6065%
  User-based kNN, $k = 20$, $m = 2$, $\ln(|R(i)|)$ instead of $\ln(|R(i)| + 10)$, $sim'(v, u) = sim(v, u) \cdot \ln(r_{vi} + 20)$
- NBH-15: error: 10.5500%
  User-based kNN, $k = 20$, $m = 1$, $\ln(|R(i)|)$ instead of $\ln(|R(i)| + 10)$, $sim'(v, u) = sim(v, u) \cdot r_{vi}$
- NBH-16: error 19.0046%
  Item-based kNN, only consider items with the same artist as neighbors, $k = 50$.
- NBH-17: error 10.0835%
  Item-based kNN, only consider items with the same artist as neighbors, increasing similarity by $\frac{1}{\#genre}$ if two items belong to the same genre, $k = 50$.
- NBH-18: error: 8.1462%
  Predicting on neighbors using inverse pLSA, $f = 200$.
- NBH-19: error: 4.6548%
  Predicting on neighbors using BPR-MF, $f = 200$.

D.2 Latent Information Predictors

η denotes the learning rate. λ_u and λ_i are regularization weights of user and item features respectively. r_{neg} stands for the rating value of negative samples.

- LI-1: error 7.7858%
  OCCF-MF, optimized by coordinate descent method, $f = 100$, $\lambda_u = \lambda_i = 500$.
- LI-2: error 7.4046%
  OCCF-MF, optimized by coordinate descent method, $f = 400$, $\lambda_u = \lambda_i = 714$.
- LI-3: error 7.3571%
  OCCF-MF, optimized by coordinate descent method, $f = 800$, $\lambda_u = \lambda_i = 500$.
- LI-4: error 7.7182%
  OCCF-MF, optimized by coordinate descent method, $f = 1200$, $\lambda_u = \lambda_i = 500$.
- LI-5: error 6.0909%
  OCCF-MF, optimized by SGD, fixed negative samples, $f = 800$, $\eta = 0.0001$, $\lambda_u = \lambda_i = 1$, $r_{neg} = -10$.
- LI-6: error 6.0567%
  OCCF-MF, optimized by SGD, fixed negative samples, $f = 800$, $\eta = 0.0001$, $\lambda_u = \lambda_i = 1$, $r_{neg} = -10$.
- LI-7: error 5.3885%
  OCCF-MF, optimized by SGD, $f = 800$, $\lambda_u = \lambda_i = 1$, $r_{neg} = -10$.
- LI-8: error 4.7230%
  OCCF-NMF, optimized by SGD, $f = 800$, $\eta = 0.0001$, $\lambda_u = 0.3164$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-9: error 4.4146%
  OCCF-MF, optimized by SGD, $f = 800$, $\lambda_u = 0.3164$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-10: error 4.2514%
  OCCF-NMF, optimized by SGD, $f = 3200$, $\eta = 0.0001$, $\lambda_u = 0.3164$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-11: error 4.3177%
  OCCF-MF, optimized by SGD, $f = 3200$, $\lambda_u = 0.3245$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-12: error 4.3398%
  OCCF-MF, optimized by SGD, $f = 3200$, $\lambda_u = 0.2654$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-13: error 4.3391%
  OCCF-MF, optimized by SGD, $f = 3200$, $\lambda_u = 0.28440$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-14: error 4.2854%
  OCCF-MF, optimized by SGD, $f = 3200$, $\lambda_u = 0.31638$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-15: error 4.2848%
  OCCF-MF, optimized by SGD, $f = 3200$, $\lambda_u = 0.32021$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-16: error 4.3177%
  OCCF-MF, optimized by SGD, $f = 3200$, $\lambda_u = 0.32445$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-17: error 4.1197%
  OCCF-MF, optimized by SGD with adaptive learning rate, $f = 3200$, $\eta = 0.0001$, $\lambda_u = 0.3164$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-18: error 4.3840%
  OCCF-MF, optimized by SGD, taxonomic sampling, $f = 3200$, $\eta = 0.0001$, $\lambda_u = 0.3164$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-19: error 6.8052%
  OCCF-MF on quasi-album data, optimized by SGD, fixed negative samples, $f = 800$, $\eta = 0.0001$, $\lambda_u = \lambda_i = 1$, $r_{neg} = -10$.
- LI-20: error 9.9814%

OCCF-MF on quasi-artist data, optimized by SGD, fixed negative samples, $f = 800$, $\eta = 0.0001$, $\lambda_u = \lambda_i = 1$, $r_{neg} = -10$.

- LI-21: error 6.9874%
  OCCF-MF, pairwise ranking objective, optimized by SGD, $f = 800$, $\lambda_u = 1.5625$, $\lambda_i = 1.2193$.
- LI-22: error 6.5249%
  OCCF-MF two class, optimized by SGD, $f = 800$, $\lambda_u = \lambda_i = 0.0001$.
- LI-23: error 5.0735%
  OCCF-MF three class, optimized by SGD, $f = 800$, $\lambda_u = 0.00560$, $\lambda_i = 0.00864$.
- LI-24: error 6.9878%
  OCCF-MF three class, optimized by SGD, $f = 800$, $\lambda_u = 0.00448$, $\lambda_i = 0.00297$.
- LI-25: error 4.3602%
  Combining 2 OCCF-MF with different initial, each optimized by SGD, $f = 3200$, $\lambda_u = 0.3202$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-26: error 4.3447%
  Combining 5 OCCF-MF with different initial, each optimized by SGD, $f = 800$, $\lambda_u = 0.2654$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-27: error 4.0400%
  OCCF-NMF, optimized by SGD, $f = 800$, $\eta = 0.0001$, $\lambda_u = 0.3164$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-28: error 7.8737%
  Item correlation MF, optimized by SGD with adaptive learning rate, $f = 200$, $\lambda = 0.064$, $r_{neg} = -1$.
- LI-29: error 10.3161%
  Item correlation MF, pairwise ranking objective, optimized by SGD with adaptive learning rate, $f = 20$, $\lambda = 0.00064$.
- LI-30: error 9.7385%
  Item correlation MF, pairwise ranking objective, optimized by SGD with adaptive learning rate, only treat high ratings are positive, $f = 20$, $\lambda = 0.00064$.
- LI-31: error 7.9485%
  Item correlation MF, pairwise ranking objective, optimized by SGD with adaptive learning rate, only treat high ratings are positive, $f = 800$, $\lambda = 0.000512$.
- LI-32: error 5.4643%
  Inverse pLSA, optimized by TEM, $f = 2000$, $\beta = 0.95$.
- LI-33: error 5.3997%
  Inverse pLSA, optimized by TEM, $f = 2000$, $\beta = 0.95$.
- LI-34: error 8.1712%
  Inverse pLSA on quasi-album data, optimized by EM, $f = 500$.
- LI-35: error 8.4780%
  Inverse pLSA on quasi-artist data, optimized by EM, $f = 500$.
- LI-36: error 5.5391%
  Weighted average of 9 inverse pLSA predictors.
- LI-37: error 9.2460%
  pPCA, $f = 20$, $\#iter = 50$.

D.3 Bayesian Personalized Ranking Predictors

- BPR-1: error 6.2364%
  BPR-MF, $f = 200$, $\#iter = 94$.
- BPR-2: error 5.9002%
  BPR-MF on quasi-album data, $f = 200$, $\#iter = 174$.
- BPR-3: error 5.8712%
  BPR-MF on quasi-album data, $f = 200$, $\#iter = 192$.
- BPR-4: error 5.9272%
  BPR-MF based on residual of OCCF-MF, $f = 50$, $\alpha = 10$, $\#iter = 356$.
- BPR-5: error 5.2679%
  Weighted BPR-MF, $f = 200$, $\#iter = 228$.
- BPR-6: error 5.2699%
  Weighted BPR-MF, $f = 200$.
- BPR-7: error 4.6614%
  Weighted BPR-MF, optimized by SGD with adaptive learning rate, $f = 400$, $\alpha = 5$, $\#iter = 610$.
- BPR-8: error 4.8699%
  Weighted BPR-NMF, optimized by SGD with adaptive learning rate, $f = 400$, $\alpha = 5$, $\#iter = 354$.
- BPR-9: error 4.553%
  Weighted BPR-MF, optimized by SGD with adaptive learning rate, $f = 400$, $\alpha = 5$, $\#iter = 434$, reject sampled items if present in validation or testing data.
- BPR-10: error 8.4398%
  BPR-kNN, $f = 200$, $\eta = 0.008$, $\#iter = 142$.
- BPR-11: error 8.7261%
  BPR-kNN, $f = 200$, $\eta = 0.008$.

- BPR-12: error 7.4853%
  BPR-kNN, $f = 2000$, $\#iter = 224$.
- BPR-13: error 6.8850%
  BPR-Linear based on item features of BPR-kNN, $f = 2000$, $\lambda_u = 0.0005$.
- BPR-14: error 6.6859%
  BPR-Linear based on item features of BPR-kNN and square of the item features, $\lambda_u = 0.0005$, $\#iter = 86$.
- BPR-15: error 6.1789%
  BPR-Linear based on user features of OCCF-MF, $f = 400$, $\lambda_i = \lambda_j = 0$, $\#iter = 212$.
- BPR-16: error 5.7101%
  BPR-Linear based on item features of OCCF-MF, $f = 400$, $\lambda_u = 0.0005$, $\#iter = 40$.
- BPR-17: error 5.3559%
  BPR-Linear based on item features of OCCF-MF and square of the item features, $f = 800$, $\lambda_u = 0.0005$, $\#iter = 110$.

D.4 Random Walk Predictors

- RW-1: error 7.2713%
  Adjacency matrix trained via SGD, $\eta = 0.001$, $\#iter = 5$, $p_{restart} = 0.3$.
- RW-2: error 7.2561%
  Adjacency matrix trained via SGD, $\eta = 0.001$, $\#iter = 6$, $p_{restart} = 0.3$.
- RW-3: error 6.8151%
  Adjacency matrix trained via SGD, $\eta = 0.001$, $\#iter = 25$, $p_{restart} = 0.3$, using non-negative Pearson's correlation, adding 1 pseudo-album and pseudo-artist.
- RW-4: error 8.6910%
  Adjacency matrix set according to number of neighbors, $p_{restart} = 0.3$.
- RW-5: error 6.8105%
  Adjacency matrix set according to taxonomy, $p_{restart} = 0.3$, using 20 pseudo-albums and pseudo-artists.
- RW-6: error 6.2036%
  Adjacency matrix set according to taxonomy, $p_{restart} = 0.3$, using 50 pseudo-albums and pseudo-artists, and 20 pseudo groups.
- RW-7: error 6.8131%
  Adjacency matrix set according to taxonomy, $p_{restart} = 0.3$, using 20 pseudo-albums and pseudo-artists, $K = 10^{-6}$.
- RW-8: error 6.2488%
  Adjacency matrix set according to taxonomy, $p_{restart} = 0.3$, using 50 pseudo-albums and pseudo-artists, $K = 10^{-6}$.
- RW-9: error 6.0929%
  Adjacency matrix set according to taxonomy, $p_{restart} = 0.3$, using 50 pseudo-albums and pseudo-artists, and 20 pseudo groups, $K = 10^{-6}$.

D.5 Blending Predictors

These blending predictors are trained on our internal validation data. POLY2 stands for 2-degree polynomial expansion.

- BL-1: error 6.9423%
  MF interaction blending with user-based kNN, $x = 1$.
- BL-2: error 5.2712%
  MF interaction blending with pLSA, $x = 1.2$.
- BL-3: error 5.4510%
  MF interaction blending with taxonomy-aware neighborhood model, $x = 6$.
- BL-4: error 5.4824%
  MF interaction blending with taxonomy-aware neighborhood model and total common support, $x = 4$.
- BL-5: error 5.8013%
  MF interaction blending with taxonomy-aware neighborhood model and total Pearson's correlation, $x = 3$.
- BL-6: error 5.2814%
  MF interaction blending with taxonomy-aware neighborhood model and total common user support, $x = 5$.
- BL-7: error 5.6040%
  MF interaction blending with taxonomy-aware neighborhood model and total set correlation, $x = 3$.
- BL-8: error 6.4830%
  MF interaction blending with taxonomy-aware neighborhood model and total Spearman correlation, $x = 4$.
- BL-9: error 6.0656%
  MF interaction blending with taxonomy-aware neighborhood model and total common, $x = 7$.
- BL-10: error 6.6053%
  Linear SVM on 6 random walk predictions.
- BL-11: error 3.6192%
  Linear SVM on 2 OCCF-MF and 1 random walk predictions.
- BL-12: error 8.2391%
  Linear SVM on taxonomy-based features, $C = 1$, $\gamma = 0.125$.
- BL-13: error 7.4566%
  Linear SVM on taxonomy-based features with pseudo-taxonomy, $C = 1$, $\gamma = 0.125$.
- BL-14: error 5.5872%

Linear SVM on taxonomy-based features and direct MF ($f = 50$) features, $C = 1$, $\gamma = 0.125$.

- BL-15: error 5.5262%
  Linear SVM on taxonomy-based features with pseudo-taxonomy and direct MF ($f = 50$) features, $C = 1$, $\gamma = 0.125$.
- BL-16: error 3.5237%
  Linear SVM on taxonomy-based features and weighted BPR-MF ($f = 50$, $\alpha = 5$) features, $C = 1$, $\gamma = 0.125$.
- BL-17: error 3.5003%
  Linear SVM on taxonomy-based features with pseudo-taxonomy and weighted BPR-MF ($f = 50$, $\alpha = 5$) features, $C = 1$, $\gamma = 0.125$.
- BL-18: error 3.6571%
  Linear SVM on taxonomy-based features and weighted BPR-MF ($f = 200$, $\alpha = 5$) features, $C = 1$, $\gamma = 0.125$.
- BL-19: error 3.6341%
  Linear SVM on taxonomy-based features with pseudo-taxonomy and weighted BPR-MF ($f = 200$, $\alpha = 5$) features, $C = 1$, $\gamma = 0.125$.
- BL-20: error 3.9359%
  Linear SVM on taxonomy-based features and OCCF-MF ($f = 50$) features, $C = 1$, $\gamma = 0.125$.
- BL-21: error 4.2340%
  Linear SVM on taxonomy-based features and OCCF-NMF ($f = 50$) features, $C = 1$, $\gamma = 0.125$.
- BL-22: error 4.1853%
  Linear SVM on taxonomy-based features with pseudo-taxonomy and OCCF-NMF ($f = 50$) features, $C = 1$, $\gamma = 0.125$.
- BL-23: error 5.0623%
  Neural network on OCCF-NMF features, 1 hidden layer, 50 neurons, $\eta = 0.0001$.
- BL-24: error 4.9595%
  Neural network on OCCF-NMF features with POLY2, 1 hidden layer, 50 neurons, $\eta = 0.0001$.
- BL-25: error 4.2311%
  Neural network on OCCF-MF features with POLY2, 1 hidden layer, 50 neurons, $\eta = 0.0001$.
- BL-26: error 3.9085%
  Neural network on BPR-MF features with POLY2, 1 hidden layer, 50 neurons, $\eta = 0.0001$.
- BL-27: error 7.5581%
  Adaboost on taxonomy-based features, $\#iter = 500$.
- BL-28: error 5.3496%
  Adaboost on taxonomy-based features and predictions, $\#iter = 500$.
- BL-29: error 3.8555%
  Logitboost on taxonomy-based features and predictions, $\#iter = 200$.
- BL-30: error 9.0275%
  Random forests on taxonomy-based features, $\#tree = 40$, $maxdepth = 80$.
- BL-31: error 5.5243%
  Random forests on taxonomy-based features and predictions, $\#tree = 40$, $maxdepth = 80$.
- BL-32: error 6.2995%
  RCD on 6 taxonomy-aware neighborhood predictors, raw score with POLY2.
- BL-33: error 5.4122%
  RCD on 2 BPR-MF predictors with quasi-album data and 2 BPR-MF predictors with quasi-artist data, raw score.
- BL-34: error 2.7698%
  RCD on 18 predictors, global rank transform.
- BL-35: error 2.7539%
  RCD on 21 predictors, global rank transform.
- BL-36: error 2.7135%
  RCD on 21 predictors with global rank transform and 59 predictors with raw score.
- BL-37: error 2.7560%
  RCD on 59 predictors, raw score.
- BL-38: error 2.7362%
  RCD on 64 predictors, global rank transform.
- BL-39: error 2.7692%
  RCD on 73 predictors, user local rank transform.
- BL-40: error 2.6691%
  RCD on 18 predictors, POLY2 over normalized score and global rank transform, $\#iter = 10000$.
- BL-41: error 2.5995%
  RCD on 27 predictors, POLY2 over normalized score and global rank transform, $\#iter = 10000$.
- BL-42: error 2.6760%
  Bagging-RCD on 18 predictors, POLY2 over normalized score and global rank transform, 75 bags, $\#iter = 6000$.
- BL-43: error 2.6463%
  Bagging-RCD on 24 predictors, POLY2 over normalized score and global rank transform, 132 bags, $\#iter = 1000$.
- BL-44: error 2.6084%
  Bagging-RCD on 24 predictors, POLY2 over normalized score and global rank transform, 90 bags, $\#iter = 1000$.
- BL-45: error 2.5817%

Bagging-RCD on 27 predictors, POLY2 over normalized score and global rank transform, 10 bags, $\#iter = 1000$.