JMLR: Workshop and Conference Proceedings 18:101–135, 2012 Proceedings of KDD-Cup 2011 competition

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

{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

Department of Computer Science and Information Engineering, National Taiwan University

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

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, Bayesian Personalized Ranking 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 (Koren, 2009; Töscher and Jahrer, 2009; Piotte and Chabbert, 2009). Indeed, recommendation systems and specifically collaborative filtering techniques have received considerable attention in recent years (Adomavicius and Tuzhilin, 2005; Su and Khoshgoftaar, 2009). 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. This problem is important in recommendation systems as we wish not simply to recommend generally popular items to a user, but indeed to provide personalized recommendation based on his or her unique tastes. Discrimination between these two modes of recommendation lies at the core of this track of the competition. This paper describes the solution proposed by the National Taiwan University team through the 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 (Dror et al., 2011) finds its origin in the Yahoo! Music online music site, 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 tracks 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). Within the dataset, as stated by Yahoo!, these negative examples were drawn in proportion to their dataset popularity. This measure solidifies the objective of discriminating popular from user-preferred items. The task is to classify these six tracks in binary fashion in an effort to minimize the overall percentage error.

This paper is organized as follows: Section 2 describes our general framework; Sections 3∼6 describe the different types of models we employ; Sections 7 and 8 describe our blending and ensemble methods. Furthermore, we provide take home points in Section 9 and supplemental materials in the appendix.

2. Notation, Framework, and Global Settings/Strategies

2.1. Notation

For easy reference and to simplify the discussion in the following sections, we define global mathematical notation used in this paper as listed below.

- 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 the 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 the 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
- Test1(%) - Error rate on Test1 (leaderboard)

2.2. The Architecture of Our System

Our solution can be divided into three core stages: models, blends, and ensembles, 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 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 a general framework is similar to the solution of the Netflix prize winners (Koren, 2009; Töscher and Jahrer, 2009) and our solution to Track 1 of KDD-Cup 2011.

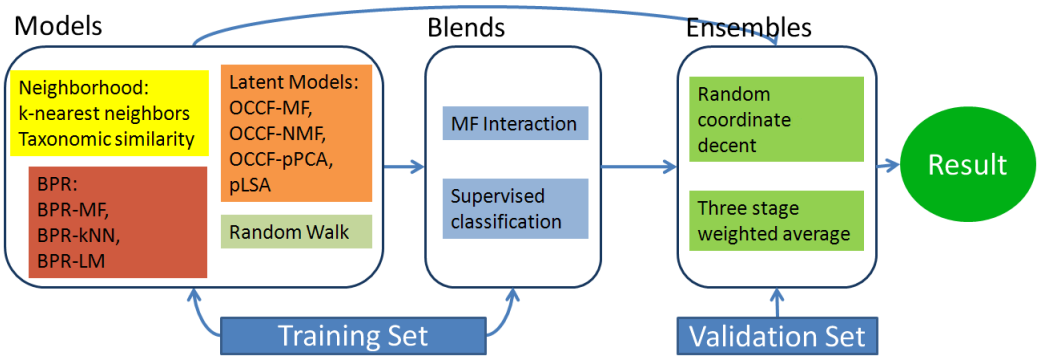


Figure 1: Three-stage System Architecture for our Discrimination Solution

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

2.3. Sampling

The goal is to divide the six tracks 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 latent information models (Section 4) and Bayesian Personalized Ranking models (Section 5) require negative examples which are unavailable directly from the dataset (i.e. only given ratings are provided), a reasonable first-step is to sample negative items for each user as training data for these models. Note that Neighborhood (Section 3) and Random Walk (Section 6) models do not utilize negative sample information.

To produce a sample set of negative examples, we follow the method that was used to produce the testing dataset. 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. In other words, as we herein seek highly-rated tracks not yet rated by the given user, tracks with a higher probability of having been rated by the given user are undesirable. 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 so that r_{ui} is a parameter for optimization, and generally we found values between -1 and -100 to be suitable for r_{ui} in these cases.

2.4. Validation Set

A validation set is usually needed for parameter adjustment to avoid overfitting. For the sake of efficiency, rather than perform full cross-validation, we simply chose an individual validation set to serve this purpose. Unlike Track 1, a validation set was not provided for Track 2 by the KDD-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 by 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 model 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.

2.5. Two-stage Prediction Models

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

2.6. Adaptive Learning Rate

An Adaptive Learning Rate (ALR) was employed when applying Stochastic Gradient Descent (SGD) in a number of the methods outlined in our framework. The idea behind this method is to decrease the learning rate (i.e. reduce the learning step size by a factor of γ) if the decrease in validation error rate between iterations has dropped below a given threshold θ for m consecutive iterations. One regular setting is $\gamma = 1.1$, $\theta = 0$, $m = 3$, which means the learning rate is divided by 1.1 whenever the validation error rate goes up for 3 consecutive iterations. These parameters can be tuned along with other parameters in individual models. This strategy worked 0.1∼0.6% better than fixed learning rate for most models.

2.7. Quasi-Album/Artist Data

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

1. http://kddcup.yahoo.com

- Quasi-Album data: each track rating with album information is removed and replaced by a rating on the corresponding album of this track. (eliminating actual "tracks" from the dataset)
- Quasi-Artist data: each track or album rating with artist information is removed and replaced by a rating on the corresponding artist of this track or album. (eliminating both actual "tracks" and "albums" from the dataset)

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 provides 0.9∼1.8% improvement for certain models, which is considered as very significant in this competition.

2.8. 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 represents 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 perform 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

In this approach, we assume that when a user rates an album highly, tracks within that same album are more likely to be rated highly by the same user. 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

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

The term $\sum_{G \in \mathfrak{G}_i \cap \mathfrak{G}_j} \frac{1}{|G|}$ represents the similarity between items 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 $|\mathfrak{G}_i|$, the number of groups to which item i belongs.

In prediction, 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 (1) to mitigate this issue.

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

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 Similarity Measures

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

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

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

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

Besides these common similarity measures, the Bayesian Personalized Ranking (BPR) similarity generated by the BPR-kNN model (see Section 5.2) may also be used as $\mathrm{Sim}(i, j)$ in the neighborhood model. Furthermore, several similarity measures can be applied concurrently. Here we consider only non-linear combinations as

$$\mathrm{Sim}(i, j) = \prod_k \mathrm{Sim}_k(i, j) \tag{4}$$

Table 1 shows the results of different variations. It should be noted that with the large number of items, the calculation of item-item similarity 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, we only have to compute $\mathrm{Sim}(i, j)$ for those (i, j), which has (u, i) in training set and (u, j) in validation or test set for some u. The computation can be done efficiently via the use of both sparse data representation and parallelization. We pre-calculate similarity measures for the nearly 250M effective pairs and store to disk, requiring roughly 3G of disk space.

Table 1: Results of Taxonomy-aware Neighborhood Models

Predictor	Similarity	λ_g	λ	Test1(%)
NBH-3	Pearson	0	0	5.8561
NBH-20	BPR100, CUS	0	0	5.5003
NBH-21	BPR1000, CUS, Pearson	0	0	5.1676
NBH-11	BPR1000, CUS, Pearson, Cosine	0	0.001	3.9978
NBH-9	BPR2000, CUS, Pearson, Cosine, Kulczynski	100	0.0005	3.8797

. CUS = common-user-support, BPRn represents the similarity measure learned by BPR-kNN with n features (c_{ij} in (7)).

3.2. User-based k-Nearest Neighbors

As previous neighborhood models consider only item-based neighbors, we also implement user-based neighborhood approaches. The main idea is to predict how a user might rate an item by checking how the item was rated by similar users.

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

$$\text{sim}(v,u) = \frac{1}{|R(v)|} \sum_{i \in R(u) \cap R(v)} \frac{1}{|R(i)|} \tag{5}$$

Adopting the k-Nearest Neighbor (kNN) approach, the score of a user u to an item i is defined as

$$s_{ui} = \left(\frac{1}{\ln(|R(i)| + 10)} \right)^m \sum_{\substack{\text{top-}k \text{ sim} \\ v \in R(i)}} \text{sim}(v,u) \tag{6}$$

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

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 models detailed in subsequent sections, such as probabilistic Latent Semantic Analysis (pLSA) described in Section 4.2. For example, to predict the score of an (u,i) pair, we first use pLSA to predict the score of (u,j), where $j \in$ top-k nearest neighbors of i. Next, the weight of each (u,j) score is determined by the cosine similarity between i and j. Lastly, the final prediction score of (u,i) is simply the sum of these weighted scores. 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 (see Section 5.1) 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 (Pan et al., 2008) as the ratings in training data are all positive examples. The negative examples are sampled as described in Section 2.3, and the latent information is learned from the existing and sampled data using Matrix Factorization (MF), pLSA and Probabilistic Principal Component Analysis (pPCA) models.

4.1. Matrix Factorization

MF methods have been widely used in CF to predict ratings (Koren, 2009; Töscher and Jahrer, 2009; Piotte and Chabbert, 2009). 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 on $\mathbf{p}_u$ and $\mathbf{q}_i$ 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 set do not provide sufficient information about negative instances. To provide 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, as discussed below.

4.1.1. Optimization Methods

A typical optimization method for MF is Stochastic Gradient Descent (SGD) which we combine with ALR, as described in Section 2.6 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 (Ma, 2008). 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 one of classification. If we focus solely on whether an user rates an item, we may discard raw rating values and use $\hat{r}_{ui} = 1$ for rated instances, and $\hat{r}_{ui} = -1$ for sampled negative instances. Although our OCCF-MF method above works on positive and negative rating values, it may also be applied on this kind of data, which leads to our two-class variant.

Furthermore, the items with high and low positive ratings may be separated into two groups as in Track 2 only highly rated items are considered positive. Thus we quantize $\hat{r}_{ui} = 1$ for high ratings ($r_{ui} \geq 80$) and $\hat{r}_{ui} = 0$ for low ratings ($0 \leq r_{ui} < 80$). We set

$\hat{r}_{ui} = -1$ for the sampled negative data. 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) (Tu and Lin, 2010) variant:

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

In both objectives, i stands for positive examples, while j stands for the sampled negative examples. The first objective penalizes wrong ranking order, i.e. ranking a negative instance before a positive one. The second objective is like regression, but it only penalizes the predictions falling on the wrong side: predicting a lower value than the true rating for positive instances, and predicting a higher value than the true ratings for negative instances. Since the ratings of positive instances are always higher than that of negative instances, minimizing such a one-sided penalty proves to be a good ranking function.

4.1.4. Optimization with Constraint

To address different perspectives of the problem and to introduce diversity into our models for further blending and ensemble, we add certain constraints during the optimization process.

Following non-negative MF (Lee and Seung, 2001), we may perform OCCF-NMF by adding the constraint that all entries of P and Q are non-negative. 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. Satisfaction of this constraint is forced while performing the optimization using the projected gradient method (Lin, 2007), with the features p_{uf} and q_{if} set to 0 if they get negative values during training.

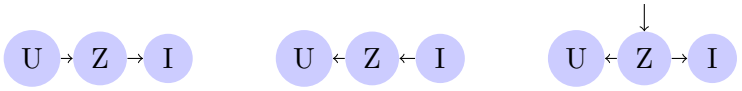
Based on the idea of neighborhood models, we may build user features $\mathbf{p}_u$ from the items rated by that user. This yields an asymmetric variant of OCCF-MF with the constraint

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

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 (Hofmann, 1999). 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 of superior performance and therefore it is this version which is included in our final ensemble. Note that the rating is not modeled in the original PLSA. Therefore our idea is to incorporate the rating as the learning rate 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. This method performs better than the original PLSA which discards rating information. Moreover, the tempered EM (TEM) (Hofmann, 1999) 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 (Tipping and Bishop, 1999). This method is equivalent to Probabilistic Matrix Factorization (PMF) (Lawrence and Urtasun, 2009). 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 proposed by Töscher and Jahrer (2009). The tuner applies a cyclic coordinate descent method on the parameter space to minimize the loss.
- 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.

Table 2: Results of Latent Information Models

Predictor	Model	Method	f	Test1(%)
LI-10	OCCF-MF	SGD	3,200	4.0969
LI-17	OCCF-MF	SGD+ALR	3,200	4.0368
LI-3	OCCF-MF	CD	800	7.0543
LI-5	OCCF-MF	SGD[a]	800	6.0265
LI-19	OCCF-MF w/ quasi-album	SGD[a]	800	6.8714
LI-20	OCCF-MF w/ quasi-artist	SGD[a]	800	9.9933
LI-18	OCCF-MF w/ taxonomy sample	SGD	3,200	4.2401
LI-8	OCCF-NMF	SGD+ALR	800	3.9081
LI-27	OCCF-NMF	SGD+ALR	3,200	3.9081
LI-22	2-class	SGD	800	6.4714
LI-23	3-class	SGD	800	5.1056
LI-21	pairwise ranking	SGD	800	6.9005
LI-28	ICMF	SGD+ALR	200	8.0340
LI-31	ICMF & pairwise ranking	SGD+ALR	800	7.9409
LI-38	pLSA	EM	2,000	6.9124
LI-34	pLSA on album data	EM	500	7.1678
LI-35	pLSA on artist data	EM	500	8.1495
LI-36	pLSA	TEM	2,000	5.4607
LI-33	pLSA	TEM	2,000	5.1313
LI-37	pPCA	EM	20	8.8519

a. The sampled negative data is fixed in these models. Also, all positive instances of one user are updated before the negative instances of that user.

5. Bayesian Personalized Ranking Models

The goal of Bayesian Personalized Ranking (BPR; Rendle et al., 2009) 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 SGD as the solver, updating a pair of items at each step. One item in the pair represents the positive item (i.e. high rating), while the other represents the negative item.

5.1. BPR: Matrix Factorization

We use the BPR-MF (Rendle et al., 2009) 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:

- We use explicit ratings instead of implicit feedback information.
- 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.

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

5.2. BPR: kNN

We follow the standard training approach (Rendle et al., 2009), 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 tuning parameter term c_{ij} as shown in (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 highly rated items and 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 ratings increases. For users with a large number of ratings, we randomly sample a subset of their ratings from the training set.

Table 3: BPR-MF Results

Predictor	Variants	#features	α	#iter	Test1(%)
BPR-1	-	200	-	94	6.2364
BPR-2	QAL	200	-	174	5.9499
BPR-3	QAL	200	-	192	5.8865
BPR-18	QAR	200	-	190	8.5872
BPR-4	PV	50	10	356	5.8852
BPR-19	WLR	50	10	242	6.3585
BPR-5	WLR	200	10	228	5.2798
BPR-20	WLR	400	10	154	5.3458
BPR-7	ALR	400	5	610	4.6817
BPR-8	ALR & NMF	400	5	354	4.8448
BPR-9	ALR & NVT	400	5	434	4.5867

. QAL/QAR: quasi-album/artist data
PV: prediction values of OCCF-MF with $f = 400$
WLR: weighted 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

Predictor	#ratings[a]	#features	#iter	η	Test1(%)
BPR-10	100	200	142	0.008	8.4307
BPR-12	100	2000	224	0.006	7.0860

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.

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.

5.3. BPR: Linear Combination

In this approach, we fix either the user or item matrix in MF, while updating its counterpart. 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 Models

Random Walk (RW; Cheng et al., 2007) 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

Table 5: BPR Linear Combination Results

Predictor	Methods	Features from	Test1(%)
BPR-15	Linear MFuser	OCCF-MF	6.1176
	F400	$\lambda_i = \lambda_j = 0$, itr=212	
BPR-16	Linear MFitem	OCCF-MF	5.7373
	F400	$\lambda_u = 0.0005$, itr=40	
BPR-13	Linear BPRKNN	BPR-kNN	6.7645
	F2000	$\lambda_u = 0.0005$, itr=184	
BPR-17	Square MFitem	OCCF-MF	5.3551
	F400	$\lambda_u = 0.0005$, itr=110	
BPR-14	Square BPRKNN	BPR-kNN	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.

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.

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

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

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, $\alpha \in \mathbb{R}$ and $0 \leq \alpha \leq 1$.

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's correlation values) with the ratings to generate each element of q (note that we adjust the negative similarity 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 Information

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

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

and where $r_{u,i}$ is the RW score, G_i contains the items directly related to i in the actual and pseudo-taxonomy and $pc_{i,t}$ is the Pearson's 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 borrow the staged blending/ensemble terminology used during the Netflix competition(Töscher and Jahrer, 2009). For the blending stage, we utilize both linear and non-linear methodologies. 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 $[0, 1]$

Table 6: Best Single Model Comparison

Predictor	Type	Model	Test1(%)
NBH-9	Neighborhood	Taxonomy Aware	3.8797
NBH-12	Neighborhood	User-based kNN	8.7324
NBH-19	Neighborhood	Predict on Neighbors	4.7035
LI-27	Latent Information	MF	3.9081
LI-33	Latent Information	pLSA	5.1313
LI-37	Latent Information	pPCA	8.8519
BPR-9	BPR	BPR-MF	4.5867
BPR-12	BPR	BPR-kNN	7.0860
BPR-17	BPR	BPR-Linear Combination	5.3551
RW-9	Random Walk	Random Walk	5.7096

Table 7: MF Interaction Blending Results

Predictor	Model multiplied with MF	β	Test1(%)
BL-3	Taxonomy-aware Neighborhood	6	4.8105
BL-1	User-based kNN	1	6.9094
BL-2	pLSA	1.2	5.2224

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

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, $(\text{MF})^{\beta} \cdot (\text{NM or pLSA})$. 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 β can be learned through validation data as shown in Table 7.

7.3. Supervised Classification

Track 2 can be thought of as a classification problem, where highly rated items (class `1`) and unrated items (class `0`) are to be discriminated. At the onset, this problem is very 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 (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 (Fan et al., 2008) 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 (Hall et al., 2009) 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

Predictor	Model	Features	Test1(%)
BL-12	SVM	TX	7.8867
BL-13		TXP	7.0319
BL-16		B50, TX	3.3403
BL-17		B50, TXP	3.3133
BL-14		M50, TX	4.8158
BL-21		N50, TX	4.0553
BL-22		N50, TXP	3.9972
BL-18		B200, TX	3.4743
BL-19		B200, TXP	3.4691
BL-24	NN	N50, TX	4.7544
BL-25		M50, TX	3.9978
BL-26		B50, TXP	3.8427

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

Table 9: Nonlinear Blending Performance

Predictor	Method	Features	Test1(%)
BL-27	Adaboost	taxonomy	7.0094
BL-30	Random Forest	taxonomy	8.4743
BL-28	Adaboost	taxonomy+models	4.2243
BL-31	Random Forest	taxonomy+models	4.3055
BL-29	Logitboost	taxonomy+models	3.5153

7.5. Linear Blending by Random Coordinate Descent

Random Coordinate Descent (RCD) is an algorithm for non-smooth optimization (Li and Lin, 2007). 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 solve this problem. Let x_{ui} denote 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 $s_{ui} = w_x x_{ui} + w_y y_{ui}$. Fixing $w_x = 1$, the error rate becomes a piecewise constant with respect to $\alpha = \frac{w_y}{w_x}$ and the changes only happen when two items with different labels swap their ranks. Every α must satisfy $x_{ui} + \alpha y_{ui} = x_{uj} + \alpha y_{uj}$ for some $i, j \in \{1, \cdots, 6\}$ and $b_{ui} \neq b_{uj}$.

For each user, there are only nine such values for α, 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 (Breiman, 1996). 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 $\mathbf{w}$ which predict `1` on this u, i pair.

Table 10: RCD Blending Results

Predictor	M	transform[a]	K or[b] $B \times K$	Test1(%)
BL-46	101	G	20000	2.6333
BL-40	18	G,N,P	10000	2.5825
BL-41	27	G,N,P	10000	2.5706
BL-44	27	G,N,P	90×1000	2.5495
BL-43	24	G,N,P	132×1000	2.5429

a. G: global ranking, N: normalized score, P: 2-degree polynomial expansion (see Section 7.1)
b. Bootstrap aggregation is applied if × appears

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 four 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. The final leaderboard results for our implementation were 2.4749% and 2.4291% for Test1 and Test2, respectively.

9. Discussion

9.1. Observations

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 ensemble formation using the predictions of multiple models.

9.2. Top Performing Models

All of the models described herein contribute to the reduction of the final prediction error rate. However, we found that MF and Taxonomy-Aware Neighborhood models were the top two contributors by some margin to the final solution. It was perhaps unsurprising that the MF modeling methodology, the classical collaborative-filtering technique for recommendation systems, proved to be quite effective in this competition. We believe the Taxonomy-Aware Neighborhood models performed especially well for the ability of these models to utilize the album, artist, and genre information included in the competition dataset.

9.3. Novelty

During the course of developing our solution to the competition, many new innovative ideas were introduced and put into practice. While these ideas have already previously been described in this paper, we highlight a few of these novel items as follows.

1. Regularization of Taxonomy-Aware Models (Section 3.1.1)
2. User-based kNN method (Section 3.2)
3. Item Correlation MF method (Section 4.1.4)
4. incorporation of the rating as the learning rate in the M-step of EM for PLSA (Section 4.2)

5. numerous customizations of BPR-MF (Section 5.1)
6. modification of prediction formula for BPR-kNN by including rating and regularization terms (Section 5.2)
7. BPR Linear Combination method (Section 5.3)
8. enhancements to the Random Walk model (Section 6.1)

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.

References

Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Transactions on Knowledge and Data Engineering*, 17(6):734–749, 2005.

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

Haibin Cheng, Pang-Ning Tan, Jon Sticklen, and William F. Punch. Recommendation via query centered random walk on k-partite graph. In *Proceedings of the Seventh IEEE International Conference on Data Mining*, pages 457–462, 2007.

Gideon Dror, Noam Koenigstein, Yehuda Koren, and Markus Weimer. The Yahoo! Music Dataset and KDD-Cup'11. *KDD-Cup Workshop*, 2011.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 50–57. ACM, 1999.

Yehuda Koren. The BellKor solution to the Netflix grand prize. Technical report, 2009.

Neil D. Lawrence and Raquel Urtasun. Non-linear matrix factorization with Gaussian processes. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 601–608. ACM, 2009.

Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. *Advances in Neural Information Processing Systems*, 13:556–562, 2001.

Ling Li and Hsuan-Tien Lin. Optimizing 0/1 loss for perceptrons by random coordinate descent. In *International Joint Conference on Neural Networks*, pages 749–754. IEEE, 2007.

Chih-Jen Lin. Projected gradient methods for non-negative matrix factorization. *Neural computation*, 19(10):2756–2779, 2007.

Chih-Chao Ma. Large-scale collaborative filtering algorithms. Master's thesis, National Taiwan University, 2008.

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

Martin Piotte and Martin Chabbert. The Pragmatic Theory solution to the Netflix Grand prize. Technical report, 2009.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars 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.

Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009:4, 2009.

Michael E. Tipping and Christopher M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622, 1999.

Andreas Töscher and Michael Jahrer. The BigChaos solution to the Netflix grand prize. Technical report, 2009.

Han-Hsing Tu and Hsuan-Tien Lin. One-sided support vector regression for multiclass cost-sensitive classification. In *Proceedings of the Twenty-Seventh International Conference on Machine learning*, pages 1095–1102, 2010.

Appendix A. Model Properties

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

Table 11: Model Properties

Type	Model	Sampled	Taxonomy	Two-Stage	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

Appendix B. 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
$S \leftarrow \emptyset$
forall the *users u appearing in the validation set* **do**
  forall the $i, j \in \{1, \cdots, 6\}$ **do**
    if *the i-th and j-th item have different labels* **then**
      $r \leftarrow \frac{x_{uj} - x_{ui}}{y_{ui} - y_{uj}}$
      $\Delta error \leftarrow E_u(r + \epsilon) - E_u(r - \epsilon)$; /* $E_u(r)$ is the number of errors on the 6 items */
                                                /* of user u when taking score $s_{ui} = x_{ui} + r y_{ui}$ */
      $S \leftarrow S \cup \{(r, \Delta error)\}$
    end
  end
end
Sort S according to r in increasing order
$r_{old} \leftarrow -\infty$, $err \leftarrow \sum_u error(r_{old}, u)$
$T \leftarrow \emptyset$
forall the $(r, \Delta error) \in S$ **do**
  if $r \neq r_{old}$ **then**
    $T \leftarrow T \cup \{(\frac{r_{old} + r}{2}, err)\}$
    $r_{old} \leftarrow r$, $err \leftarrow err + \Delta error$
  end
end
$T \leftarrow T \cup \{(\infty, err)\}$
$(r_{min}, err_{min}) \leftarrow \min_{err}(T)$
$(r_{max}, err_{max}) \leftarrow \max_{err}(T)$
if $N_{val} - err_{max} < err_{min}$ **then**
  $w_x \leftarrow -1$, $w_y \leftarrow -r_{max}$
else
  $w_x \leftarrow 1$, $w_y \leftarrow r_{min}$
end

Appendix C. 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.

C.1. Neighborhood Predictors

- NBH-1, Test1 Error=7.2081%
  Taxonomy-aware neighborhood model, $\lambda = \lambda_g = 0$, no normalized term $\frac{1}{\mathfrak{G}_i}$.

- NBH-2, Test1 Error=6.5599%
  Taxonomy-aware neighborhood model, $\lambda = \lambda_g = 0$.

- NBH-3, Test1 Error=5.8561%
  Taxonomy-aware neighborhood model augmented by Pearson's correlation, $\lambda = \lambda_g = 0$.

- NBH-4, Test1 Error=5.4429%
  Taxonomy-aware neighborhood model augmented by Pearson's correlation, $\lambda = \lambda_g = 0$, pseudo-taxonomy.

- NBH-5, Test1 Error=5.8627%
  Taxonomy-aware neighborhood model augmented by Pearson's correlation, $\lambda = \lambda_g = 0$, square $(r_{uj} + 1)$.

- NBH-6, Test1 Error=5.4541%
  Taxonomy-aware neighborhood model augmented by Pearson's correlation, $\lambda = \lambda_g = 0$, square $(r_{uj} + 1)$, pseudo-taxonomy.

- NBH-7, Test1 Error=5.1135%
  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, Test1 Error=5.1128%
  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, Test1 Error=3.8797%
  Taxonomy-aware neighborhood model augmented by cosine similarity, Pearson's correlation, common user support, Kulczynski's coefficient and BPR-kNN correlation with $f = 1000$, $\lambda = 0.0005$, $\lambda_g = 100$.

- NBH-10, Test1 Error=4.2751%
  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, Test1 Error=3.9978%
  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, Test1 Error=8.7324%
  User-based kNN, $k = 20$, $m = 2$

- NBH-13, Test1 Error=8.7324%
  User-based kNN, $k = 20$, $m = 2$, $\ln(|R(i)|)$ instead of $\ln(|R(i)| + 10)$

- NBH-14, Test1 Error=8.6809%
  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, Test1 Error=10.3359%
  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, Test1 Error=19.8849%
  Item-based kNN, only consider items with the same artist as neighbors, $k = 50$.
- NBH-17, Test1 Error=10.8019%
  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, Test1 Error=7.6385%
  Predicting on neighbors using inverse pLSA, $f = 200$.
- NBH-19, Test1 Error=4.7035%
  Predicting on neighbors using BPR-MF, $f = 200$.
- NBH-20, Test1 Error=5.5003%
  Taxonomy-aware neighborhood model augmented by common user support and square of BPR-kNN correlation with $f = 100$, $\lambda = 0$, $\lambda_g = 0$.
- NBH-21, Test1 Error=5.1676%
  Taxonomy-aware neighborhood model augmented by common user support, Pearson's correlation, and square of BPR-kNN correlation with $f = 1000$, $\lambda = 0$, $\lambda_g = 0$.

C.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, Test1 Error=6.9903%
  OCCF-MF, optimized by coordinate descent method, $f = 100$, $\lambda_u = \lambda_i = 500$.
- LI-2, Test1 Error=6.8147%
  OCCF-MF, optimized by coordinate descent method, $f = 400$, $\lambda_u = \lambda_i = 714$.
- LI-3, Test1 Error=7.0543%
  OCCF-MF, optimized by coordinate descent method, $f = 800$, $\lambda_u = \lambda_i = 500$.
- LI-4, Test1 Error=6.9909%
  OCCF-MF, optimized by coordinate descent method, $f = 1200$, $\lambda_u = \lambda_i = 500$.
- LI-5, Test1 Error=6.0265%
  OCCF-MF, optimized by SGD, fixed negative samples, $f = 800$, $\eta = 0.0001$, $\lambda_u = \lambda_i = 1$, $r_{neg} = -10$.

- LI-6, Test1 Error=5.9670%
  OCCF-MF, optimized by SGD, fixed negative samples, $f = 800$, $\eta = 0.0001$, $\lambda_u = \lambda_i = 1$, $r_{neg} = -10$.
- LI-7, Test1 Error=5.4482%
  OCCF-MF, optimized by SGD, $f = 800$, $\lambda_u = \lambda_i = 1$, $r_{neg} = -10$.
- LI-8, Test1 Error=4.6085%
  OCCF-NMF, optimized by SGD, $f = 800$, $\eta = 0.0001$, $\lambda_u = 0.3164$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-9, Test1 Error=4.3939%
  OCCF-MF, optimized by SGD, $f = 800$, $\lambda_u = 0.3164$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-10, Test1 Error=4.0969%
  OCCF-NMF, optimized by SGD, $f = 3200$, $\eta = 0.0001$, $\lambda_u = 0.3164$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-11, Test1 Error=4.2810%
  OCCF-MF, optimized by SGD, $f = 3200$, $\lambda_u = 0.3245$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-12, Test1 Error=4.2936%
  OCCF-MF, optimized by SGD, $f = 3200$, $\lambda_u = 0.2654$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-13, Test1 Error=4.3088%
  OCCF-MF, optimized by SGD, $f = 3200$, $\lambda_u = 0.28440$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-14, Test1 Error=4.2579%
  OCCF-MF, optimized by SGD, $f = 3200$, $\lambda_u = 0.31638$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-15, Test1 Error=4.2493%
  OCCF-MF, optimized by SGD, $f = 3200$, $\lambda_u = 0.32021$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-16, Test1 Error=4.2810%
  OCCF-MF, optimized by SGD, $f = 3200$, $\lambda_u = 0.32445$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.
- LI-17, Test1 Error=4.0368%
  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, Test1 Error=4.2401%
  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, Test1 Error=6.8714%
  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, Test1 Error=9.9933%
  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, Test1 Error=6.9005%
  OCCF-MF, pairwise ranking objective, optimized by SGD, $f = 800$, $\lambda_u = 1.5625$, $\lambda_i = 1.2193$.

- LI-22, Test1 Error=6.4714%
  OCCF-MF two class, optimized by SGD, $f = 800$, $\lambda_u = \lambda_i = 0.0001$.

- LI-23, Test1 Error=5.1056%
  OCCF-MF three class, optimized by SGD, $f = 800$, $\lambda_u = 0.00560$, $\lambda_i = 0.00864$.

- LI-24, Test1 Error=6.9599%
  OCCF-MF three class, optimized by SGD, $f = 800$, $\lambda_u = 0.00448$, $\lambda_i = 0.00297$.

- LI-25, Test1 Error=4.1510%
  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, Test1 Error=4.3220%
  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, Test1 Error=3.9081%
  OCCF-NMF, optimized by SGD, $f = 800$, $\eta = 0.0001$, $\lambda_u = 0.3164$, $\lambda_i = 0.32768$, $r_{neg} = -47.68$.

- LI-28, Test1 Error=8.0340%
  Item correlation MF, optimized by SGD with adaptive learning rate, $f = 200$, $\lambda = 0.064$, $r_{neg} = -1$.

- LI-29, Test1 Error=9.4896%
  Item correlation MF, pairwise ranking objective, optimized by SGD with adaptive learning rate, $f = 20$, $\lambda = 0.00064$.

- LI-30, Test1 Error=9.7715%
  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, Test1 Error=7.9409%
  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, Test1 Error=5.3808%
  Inverse pLSA, optimized by TEM, $f = 2000$, $\beta = 0.95$.

- LI-33, Test1 Error=5.1313%
  Inverse pLSA, optimized by TEM, $f = 2000$, $\beta = 0.95$.

- LI-34, Test1 Error=7.1678%
  Inverse pLSA on quasi-album data, optimized by EM, $f = 500$.

- LI-35, Test1 Error=8.1495%
  Inverse pLSA on quasi-artist data, optimized by EM, $f = 500$.

- LI-36, Test1 Error=5.4607%
  Weighted average of 9 inverse pLSA predictors.

- LI-37, Test1 Error=8.8519%
  pPCA, $f = 20$, $\#iter = 50$.

- LI-38, Test1 Error=6.9124%
  Inverse pLSA optimized by EM, $f = 2000$.

C.3. Bayesian Personalized Ranking Predictors

- BPR-1, Test1 Error=6.2364%
  BPR-MF, $f = 200$, $\#iter = 94$.

- BPR-2, Test1 Error=5.9499%
  BPR-MF on quasi-album data, $f = 200$, $\#iter = 174$.

- BPR-3, Test1 Error=5.8865%
  BPR-MF on quasi-album data, $f = 200$, $\#iter = 192$.

- BPR-4, Test1 Error=5.8852%
  BPR-MF based on the predicted values of OCCF-MF, $f = 50$, $\alpha = 10$, $\#iter = 356$.

- BPR-5, Test1 Error=5.2798%
  Weighted BPR-MF, $f = 200$, $\#iter = 228$.

- BPR-6, Test1 Error=5.5267%
  Weighted BPR-MF, $f = 200$.

- BPR-7, Test1 Error=4.6817%
  Weighted BPR-MF, optimized by SGD with adaptive learning rate, $f = 400$, $\alpha = 5$, $\#iter = 610$.

- BPR-8, Test1 Error=4.8448%
  Weighted BPR-NMF, optimized by SGD with adaptive learning rate, $f = 400$, $\alpha = 5$, $\#iter = 354$.

- BPR-9, Test1 Error=4.5867%
  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, Test1 Error=8.4307%
  BPR-kNN, $f = 200$, $\eta = 0.008$, $\#iter = 142$.

- BPR-11, Test1 Error=8.7654%
  BPR-kNN, $f = 200$, $\eta = 0.008$.

- BPR-12, Test1 Error=7.0860%
  BPR-kNN, $f = 2000$, $\eta = 0.006$, $\#iter = 224$.
- BPR-13, Test1 Error=6.7645%
  BPR-Linear based on item features of BPR-kNN, $f = 2000$, $\lambda_u = 0.0005$.
- BPR-14, Test1 Error=6.2687%
  BPR-Linear based on item features of BPR-kNN and square of the item features, $\lambda_u = 0.0005$, $\#iter = 86$.
- BPR-15, Test1 Error=6.1176%
  BPR-Linear based on user features of OCCF-MF, $f = 400$, $\lambda_i = \lambda_j = 0$, $\#iter = 212$.
- BPR-16, Test1 Error=5.7373%
  BPR-Linear based on item features of OCCF-MF, $f = 400$, $\lambda_u = 0.0005$, $\#iter = 40$.
- BPR-17, Test1 Error=5.3551%
  BPR-Linear based on item features of OCCF-MF and square of the item features, $f = 800$, $\lambda_u = 0.0005$, $\#iter = 110$.
- BPR-18, Test1 Error=8.5872%
  BPR-MF on quasi-artist data, $f = 200$, $\#iter = 190$.
- BPR-19, Test1 Error=6.3585%
  Weighted BPR-MF, $f = 50$, $\alpha = 10$, $\#iter = 242$.
- BPR-20, Test1 Error=5.3458%
  Weighted BPR-MF, $f = 400$, $\alpha = 10$, $\#iter = 154$.

C.4. Random Walk Predictors

- RW-1, Test1 Error=6.9414%
  Adjacency matrix trained via SGD, $\eta = 0.001$, $\#iter = 5$, $p_{restart} = 0.3$.
- RW-2, Test1 Error=6.9348%
  Adjacency matrix trained via SGD, $\eta = 0.001$, $\#iter = 6$, $p_{restart} = 0.3$.
- RW-3, Test1 Error=6.3862%
  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, Test1 Error=8.3937%
  Adjacency matrix set according to number of neighbors, $p_{restart} = 0.3$.
- RW-5, Test1 Error=6.4523%
  Adjacency matrix set according to taxonomy, $p_{restart} = 0.3$, using 20 pseudo-albums and pseudo-artists.
- RW-6, Test1 Error=5.8278%
  Adjacency matrix set according to taxonomy, $p_{restart} = 0.3$, using 50 pseudo-albums and pseudo-artists, and 20 pseudo groups.

- RW-7, Test1 Error=6.4542%
  Adjacency matrix set according to taxonomy, $p_{restart} = 0.3$, using 20 pseudo-albums and pseudo-artists, $K = 10^{-6}$.

- RW-8, Test1 Error=5.8192%
  Adjacency matrix set according to taxonomy, $p_{restart} = 0.3$, using 50 pseudo-albums and pseudo-artists, $K = 10^{-6}$.

- RW-9, Test1 Error=5.7096%
  Adjacency matrix set according to taxonomy, $p_{restart} = 0.3$, using 50 pseudo-albums and pseudo-artists, and 20 pseudo groups, $K = 10^{-6}$.

C.5. Blending Predictors

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

- BL-1, Test1 Error=6.9064%
  MF interaction blending with user-based kNN, $x = 1$.

- BL-2, Test1 Error=5.2224%
  MF interaction blending with pLSA, $x = 1.2$.

- BL-3, Test1 Error=4.8105%
  MF interaction blending with taxonomy-aware neighborhood model, $x = 6$.

- BL-4, Test1 Error=5.0844%
  MF interaction blending with taxonomy-aware neighborhood model and total common support, $x = 4$.

- BL-5, Test1 Error=5.3795%
  MF interaction blending with taxonomy-aware neighborhood model and total Pearson's correlation, $x = 3$.

- BL-6, Test1 Error=6.8741%
  MF interaction blending with taxonomy-aware neighborhood model and total common user support, $x = 5$.

- BL-7, Test1 Error=5.1887%
  MF interaction blending with taxonomy-aware neighborhood model and total set correlation, $x = 3$.

- BL-8, Test1 Error=6.0647%
  MF interaction blending with taxonomy-aware neighborhood model and total Spearman correlation, $x = 4$.

- BL-9, Test1 Error=5.7017%
  MF interaction blending with taxonomy-aware neighborhood model and total common, $x = 7$.

- BL-10, Test1 Error=5.8740%
  Linear SVM on 6 random walk predictions.
- BL-11, Test1 Error=3.4981%
  Linear SVM on 2 OCCF-MF and 1 random walk predictions.
- BL-12, Test1 Error=7.8867%
  Linear SVM on taxonomy-based features, $C = 1$, $\gamma = 0.125$.
- BL-13, Test1 Error=7.0319%
  Linear SVM on taxonomy-based features with pseudo-taxonomy, $C = 1$, $\gamma = 0.125$.
- BL-14, Test1 Error=4.8158%
  Linear SVM on taxonomy-based features and direct MF ($f = 50$) features, $C = 1$, $\gamma = 0.125$.
- BL-15, Test1 Error=7.0319%
  Linear SVM on taxonomy-based features with pseudo-taxonomy and direct MF ($f = 50$) features, $C = 1$, $\gamma = 0.125$.
- BL-16, Test1 Error=3.3403%
  Linear SVM on taxonomy-based features and weighted BPR-MF ($f = 50$, $\alpha = 5$) features, $C = 1$, $\gamma = 0.125$.
- BL-17, Test1 Error=3.3133%
  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, Test1 Error=3.4743%
  Linear SVM on taxonomy-based features and weighted BPR-MF ($f = 200$, $\alpha = 5$) features, $C = 1$, $\gamma = 0.125$.
- BL-19, Test1 Error=3.4691%
  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, Test1 Error=3.4658%
  Linear SVM on taxonomy-based features and OCCF-MF ($f = 50$) features, $C = 1$, $\gamma = 0.125$.
- BL-21, Test1 Error=4.0553%
  Linear SVM on taxonomy-based features and OCCF-NMF ($f = 50$) features, $C = 1$, $\gamma = 0.125$.
- BL-22, Test1 Error=3.9972%
  Linear SVM on taxonomy-based features with pseudo-taxonomy and OCCF-NMF ($f = 50$) features, $C = 1$, $\gamma = 0.125$.
- BL-23, Test1 Error=4.8666%
  Neural network on OCCF-NMF features, 1 hidden layer, 50 neurons, $\eta = 0.0001$.

- BL-24, Test1 Error=4.7544%
  Neural network on OCCF-NMF features with POLY2, 1 hidden layer, 50 neurons, $\eta = 0.0001$.

- BL-25, Test1 Error=3.9978%
  Neural network on OCCF-MF features with POLY2, 1 hidden layer, 50 neurons, $\eta = 0.0001$.

- BL-26, Test1 Error=3.8427%
  Neural network on BPR-MF features with POLY2, 1 hidden layer, 50 neurons, $\eta = 0.0001$.

- BL-27, Test1 Error=7.0094%
  Adaboost on taxonomy-based features, $\#iter = 500$.

- BL-28, Test1 Error=4.2243%
  Adaboost on taxonomy-based features and predictions, $\#iter = 500$.

- BL-29, Test1 Error=3.5153%
  Logitboost on taxonomy-based features and predictions, $\#iter = 200$.

- BL-30, Test1 Error=8.4743%
  Random forests on taxonomy-based features, $\#tree = 40$, $maxdepth = 80$.

- BL-31, Test1 Error=4.3055%
  Random forests on taxonomy-based features and predictions, $\#tree = 40$, $maxdepth = 80$.

- BL-32, Test1 Error=5.7393%
  RCD on 6 taxonomy-aware neighborhood predictors, raw score with POLY2.

- BL-33, Test1 Error=5.4508%
  RCD on 2 BPR-MF predictors with quasi-album data and 2 BPR-MF predictors with quasi-artist data, raw score.

- BL-34, Test1 Error=2.6439%
  RCD on 18 predictors, global rank transform.

- BL-35, Test1 Error=2.6287%
  RCD on 21 predictors, global rank transform.

- BL-36, Test1 Error=2.6109%
  RCD on 21 predictors with global rank transform and 59 predictors with raw score.

- BL-37, Test1 Error=2.6676%
  RCD on 59 predictors, raw score.

- BL-38, Test1 Error=2.6056%
  RCD on 64 predictors, global rank transform.

- BL-39, Test1 Error=2.6696%
  RCD on 73 predictors, user local rank transform.
- BL-40, Test1 Error=2.5825%
  RCD on 18 predictors, POLY2 over normalized score and global rank transform, $\#iter = 10000$.
- BL-41, Test1 Error=2.5706%
  RCD on 27 predictors, POLY2 over normalized score and global rank transform, $\#iter = 10000$.
- BL-42, Test1 Error=2.5944%
  Bagging-RCD on 18 predictors, POLY2 over normalized score and global rank transform, 75 bags, $\#iter = 6000$.
- BL-43, Test1 Error=2.5429%
  Bagging-RCD on 24 predictors, POLY2 over normalized score and global rank transform, 132 bags, $\#iter = 1000$.
- BL-44, Test1 Error=2.5495%
  Bagging-RCD on 24 predictors, POLY2 over normalized score and global rank transform, 90 bags, $\#iter = 1000$.
- BL-45, Test1 Error=2.5785%
  Bagging-RCD on 27 predictors, POLY2 over normalized score and global rank transform, 10 bags, $\#iter = 1000$.
- BL-46, Test1 Error=2.6333%
  RCD on 101 predictors, global rank transform, $\#iter = 20000$.