JMLR: Workshop and Conference Proceedings 75–97

Rating Prediction with Informative Ensemble of Multi-Resolution Dynamic Models

Zhao Zheng ZHENGZHAOZZ@CSE.UST.HK
Hong Kong University of Science and Technology, Hong Kong

Tianqi Chen TQCHEN@APEX.SJTU.EDU.CN
Shanghai Jiao Tong University, Shanghai, China

Nathan Liu NLIU@CSE.UST.HK
Hong Kong University of Science and Technology, Hong Kong

Qiang Yang QYANG@CSE.UST.HK
Hong Kong University of Science and Technology, Hong Kong

Yong Yu YYU@APEX.SJTU.EDU.CN
Shanghai Jiao Tong University, Shanghai, China

Abstract

The Yahoo! music rating data set in KDD Cup 2011 raises several interesting challenges: (1) The data covers a lengthy time period of more than eight years. (2) Not only are training ratings associated date and time information, so are the test ratings. (3) The items form a hierarchy consisting of four types of items: genres, artists, albums and tracks. To capture the rich temporal dynamics within the data set, we design a class of time-aware matrix/tensor factorization models, which adopts time series based parameterizations and models user/item drifting behaviors at multiple temporal resolutions. We also incorporate the taxonomical structure into the item parameters by introducing sharing parameters between ancestors and descendants in the taxonomy. Finally, we have identified some conditions that systematically affect the effectiveness of different types of models and parameter settings. Based on these findings, we designed an *informative ensemble* framework, which considers additional *meta features* when making predictions for a particular pair of user and item. Using these techniques, we built the best single model reported officially, and our final ensemble model got third place in KDD Cup 2011.

1. Introduction

Recommender systems have become an indispensable tool for helping users tackle information overload as new content (e.g., news, products) are growing at an explosive rate. Collaborative filtering (CF) is one of the most promising technologies for recommender systems. It works by discovering the correlation between users and items based on observed user preferences (i.e., ratings, clicks, etc.) so that unobserved user preferences can be inferred from the observed ones. For example, the well-known user based algorithm first finds the similarities between users based on their past ratings, then a target user's rating on a new item can be predicted from the ratings on that item from other similar users, also known as the user's neighborhood. Thanks to the widely publicized Netflix prize competition, there has been a surging interests in CF algorithm design in the research community in recent years.

In contrast to the Netflix prize competition which deals with the movie domain, this year's KDD Cup 2011 (Dror et al., 2011) provides access to one of the largest music rating data sets, which contains nearly thirty times more ratings than the Netflix movie ratings data set. In addition to the differences in data size, we also noted several other interesting new challenges raised by the Yahoo! music data set:

- The data set spans a very long time period of more than 6,000 days and there is a significant number of users that have been active in the system for multiple years. People's tastes in music are arguably much more diverse and unpredictable than in movies due to the much larger number of choices available. In the mean time, the popularity of genres and artists is also changing fast. The effect of temporal dynamics has to be carefully taken into account to cope with various drifting and transient characteristics of users/items over such a long period of time.

- Each rating in both the training and test data sets is associated with both date and time information. While previous work (Koren, 2009) has demonstrated the value of considering date information, there has been no previous work that jointly considers both date and time information in a single model. Intuitively, people's musical preference can naturally change over the day and people often prefer different types of music depending on his current context/status (e.g., at home vs. at work). Therefore, considering time in addition to date as an additional context dimension can lead to more accurate modeling of user behaviors as we will demonstrate. In addition, the task of rating prediction given a specific temporal context described by date plus time is also a novel task that has not been addressed in the Netflix prize competition or existing literature.

- Unlike traditional data sets which contain a single set of homogeneous items (e.g., movies), the Yahoo! music data set consists of 4 types of items: genres, artists, albums and tracks, which are naturally linked together as a directed acyclic graph (DAG) based on predefined taxonomical relations. Intuitively, a user's preference for a particular track can be highly influenced by whether he likes the singer or the genre of the song. Similarly, if a user hates a particular artist, he can hardly rate any of this artist's songs highly. To capture such correlated user preference over linked items, we also design a downward parameter sharing scheme such that the parameters of an item lower in the taxonomy will depend on the parameters of its ancestors, but not vice versa.

- The data set contains a huge number of items, which is nearly 30 times more than that of the Netflix data set. As a result, although the number of ratings is much larger, the data sparsity is actually much more severe than the Netflix data set. This also leads a user/item population with highly diverse characteristics, which can hardly be served using a single model. As we will show, different models or parameter settings perform differently on particular user/item segments. To more flexibly fuse multiple models so as to serve different user/item segments differently, we design an *informative ensemble learning* strategy, which augments the model predictions with an additional set of *meta features* describing various user/item characteristics, and then train a nonlinear model to make predictions based on this representation.

To solve these problems, we have designed various kinds of extensions of the well known matrix factorization model to handle the many unique aspects of the Yahoo! music data set. Based on extensive experiments, we find that incorporating date, time and taxonomy informationcan lead to significant improvement over basic matrix factorization and the informative ensemble of a collection of models can further significantly improve upon the best single model.

2. Preliminaries

In this section, we will give a concise overview of the matrix factorization model, which forms the foundation of our KDD Cup 2011 solution. We will also describe some of its recent extensions specifically designed for collaborative filtering problems. Before formally describing the models, we first define our notational conventions. Suppose we have m users and n items. Generally, we use u, v to denote users and i, j to denote items. Day and time indices are denoted by d and t respectively. The ratings are arranged in a $m \times n$ matrix $R = r_{ui}$. In the matrix, some cells have values, the others are empty and we let S denote the set of (u, i) indices for which ratings are observed.

Using the model-based approach to recommendation, one needs to develop a parametric model of the user ratings. Let the parameters in such a model be denoted by Θ. For any pair of user u and item i, the model enables us to make a prediction $\widehat{r}_{ui}$ based on the parameters Θ. As in any machine learning algorithm, building recommendation models involves three critical steps. First of all, one needs to specify the *parametrization*, which governs how the prediction $\widehat{r}_{ui}$ is produced by the model parameters Θ. Secondly, a *learning criterion* needs to be chosen that allows us to evaluate how well a particular model fits the observed user ratings. Finally, a *learning algorithm* provides the procedures for optimizing the learning criteria in order to find the best model parameters Θ^*. In this section, we describe how each of these steps are instantiated for the basic matrix factorization model, which forms the foundation for more complicated extensions such as time and taxonomy awareness.

2.1. Matrix Factorization Model for Collaborative Filtering

Matrix factorization is one of the state of the art models for large scale collaborative filtering as demonstrated by its success in the Netflix prize competition (Koren et al., 2009). In its basic form, every user u and item i is associated with a vector $p_u, q_i \in \mathbb{R}^k$ and a scalar a_u and b_i. The vectors p_u and q_i are generally referred to as the user and item factors where as a_u and b_i are referred to as the user and item biases. Under this model, the predictor $\widehat{r}_{ui}$ is parameterized as follows:

$$\widehat{r}_{ui} = \sigma(p_u^T q_i + a_u + b_i), \tag{1}$$

where the function $\sigma(\cdot)$ is the sigmoid function that maps any real value to a value between 0 and 1:

$$\sigma(x) = \frac{1}{1 + \exp(-x)}, \tag{2}$$

which can then be easily mapped to the scale of the Yahoo! music ratings by multiplication with 100 so the predicted ratings would be between 0 and 100.

The model's parameter collection Θ thus consists of the following $\{\mathbf{a} \in \mathbb{R}^m, \mathbf{b} \in \mathbb{R}^n, \mathbf{P} \in \mathbb{R}^{m\times k}, \mathbf{Q} \in \mathbb{R}^{n\times k}\}$, where elements of $\mathbf{a}$ and $\mathbf{b}$ correspond to the user and item biases and the rows of $\mathbf{P}$ and $\mathbf{Q}$ correspond to user and item factors.

2.2. Learning Criterion

The matrix factorization model is highly flexible as the number of free parameters can be huge given a high dimensionality and this can easily cause overfitting. So a good learning criterion should balance between the goodness of fit on the training data and the model complexity. One particularly popular learning criterion is the following optimization problem:

$$\min_{\Theta} \sum_{(u,i)\in S} (r_{ui} - \widehat{r}_{ui}(\Theta))^2 + \lambda ||\Theta||^2, \tag{3}$$

which is known as the regularized least squares problem. The first quantity of the objective function is the sum of the squared differences between the predicted ratings and the true ratings, which measures the model's accuracy. The second quantity is known as the $\ell 2$ norm, which is the sum of squares of every parameter of the model and quantifies the complexity of the model. The parameter $\lambda \in \mathbb{R}$ allows us to trade off between model accuracy and complexity.

2.3. Learning Algorithm

As the learning objective derived in the previous section is differentiable, an algorithm based on gradient descent would be a natural choice for optimizing this criterion. A general form of the gradient of the objective, denoted as $\mathcal{L}$, with respect to the model parameters is as follows:

$$\frac{\partial \mathcal{L}}{\partial \Theta} = \sum_{(u,i)\in S} e_{ui} \cdot \frac{\partial \widehat{r}_{ui}}{\partial \Theta} + \lambda \Theta, \tag{4}$$

where $e_{ui} = (r_{ui} - \widehat{r}_{ui})$ denotes the prediction error on the pair (u, i).

One commonly used algorithm for gradient descent is the full gradient descent, where in each step the full gradient over all training data is computed and used to update the data. However, this was found to have slow convergence for training matrix factorization models over large data sets. In this work, we use the other type of algorithm, which does stochastic gradient descent (SGD) over one randomly chosen rating each time, and update the model parameters with a learning rate α, as shown in Algorithm 1.

3. Incorporating Temporal Information

This section discusses a variety of techniques for utilizing the available date and time information associated with each rating. The key to incorporating temporal dynamics into the matrix factorization framework is to let the user/item factors and biases become time dependent. In the following two subsections, we discuss several schemes we attempted to design date-time dependent versions of user/item factors $p_u(d,t), q_i(d,t)$ and biases $a_u(d,t), b_i(d,t)$.

Algorithm 1: Stochastic Gradient Descent

```
initialize $\Theta$ ;
repeat
    randomly draw $(u, i)$ from $S$ ;
    $\Theta \leftarrow \Theta - \alpha\Big(e_{ui} \cdot \frac{\partial \widehat{r}_{ui}}{\partial \Theta} + \lambda\Theta\Big)$;
until convergence;
```

3.1. Date-Time Dependent User/Item Biases

The user bias a_u in basic matrix factorization is used to capture the general tendency of a user to assign high or low ratings, which is a rather person-dependent effect. For example, there are very picky users who rarely give high ratings and there are casual users who give high ratings to most items they find acceptable. On the other hand, the item bias b_i is used to capture the overall popularity of an item.

The date can have the following effects on the user/item biases as first suggested in (Koren, 2009): Firstly, a user may change their rating scale over time as they become more adept at use rating to express personal preferences or become more picky in the music listened to. Secondly, item popularity may change over time, which is especially true for the highly dynamic music domain where new artists and genres quickly rise and fall whereas a small number of classics may remain popular over time.

In addition to date, the time may also affect music ratings in the several ways. Firstly, a user may rate differently during different hours. For example, in the day time, as one is often busy with work at hand and less willing to spend time to rate items, a user may mostly assign low ratings for songs that he finds really annoying as a way to filter out bad songs. Secondly, different songs may be more (or less) popular during different hours. For example, dance music may be more preferred at night whereas light music may be generally more preferred in the morning.

3.1.1. Binning based Date/Time Dependent Bias Model

One simple approach to design date-time dependent biases is to assume date and time have independent effects and then simply assign a separate bias value to each date and time point. An important decision in this scheme is the granularity at which to treat data and time. Our design is based on dividing the 6,000 days and 24 hours of a day into equal sized bins such as every week and every 30 minutes and then designating a user/item bias for each date bin and time bin, which leads to the following formulation:

$$a_u(d, t) = a_u + a_{u,Bin(d)} + a_{u,Bin(t)}, \tag{5}$$

where $Bin(d)$ and $Bin(t)$ denote the bin index associated with the particular date and time.

3.1.2. Tensor based Bias Model

There are two major drawbacks of the previous two models for modeling date-time dependent user/item biases. Firstly, they are not capable of extrapolating into unseen date values or unseen hours. More specifically, the bias at any date later than the date of the latest rating or earlier than the first rating in the training data set can not be predicted at all. A second drawback is that they do not consider the interaction effects between date and time, such as songs with a happy mood are more popular at night during Christmas seasons. In this subsection, we describe a tensor factorization based framework for modeling date-time dependent user/item biases. The key idea is to treat the collections of date-time dependent biases $a_u(d,t)$ as being represented as a 3- dimensional tensor with dimensions corresponding to user, date and time.

$$a_u(d,t) = w_u^T \cdot p_{Bin(d)} + w_u^T \cdot q_{Bin(t)} + p_{Bin(d)}^T \cdot q_{Bin(t)}, \tag{6}$$

where we learn a new factor w_u for each user as well as a factor $p_{Bin(d)}$ and $q_{Bin(t)}$ for each date bin and time bin. The user's bias on at a specific time of a specific day is then produced the pairwise interactions between these three factors.

3.2. Date-Time Dependent User/Item Factors

3.2.1. Binning based Dynamic Factor Model

Making the user and item biases date/time dependent enables a model to capture the changing trend of a user or item's ratings. However, it is not able to capture more complicated temporal dynamics such as a user's dynamic preferences over the item space. For example, a user may be into Dance/Rock music when he is young but may gradually start to like Jazz/Classical music as he grow older. Also, a user may prefer different types of music during working hours versus during the night time. To handle such dynamic interactions between users and items, which we refer to as *second-order effects*, we need to also have a date-time dependent user factor $p_u(d,t)$, which leads to the following predictor:

$$\widehat{r}_{ui}(d,t) = a_u + b_i + p_u(d,t)^T q_i. \tag{7}$$

Suppose each user factor is of k dimensions, our idea is to partition the k dimensions into three subsets, $p_u(d,t) = [p_u^0, p_u(d), p_u(t)]$, where $p_u^0 \in \mathbb{R}^{k_0}$ is a static portion that is independent of date and time whereas $p_u(d) \in \mathbb{R}^{k_1}$ and $p_u(t) \in \mathbb{R}^{k_2}$ are two portions that depend on date and time respectively and are implemented by dividing the time line into bins and having a set of factors for each bin (i.e., $p_u(d) = p_{Bin(d)}$). By adjusting the values of k_0, k_1 and k_2, one can easily control the number of features devoted to modeling different types of dynamic properties of the user.

While the above formulation only emphasizes capturing time dependent user properties, we can apply the same idea to design a dynamic item factor $q_i(d,t)$. However, based on our experiments on the data set, making the item factor time-dependent resulted in much less improvement over a static model compared with making the user factor time-dependent. This seems to indicate that in the music domain it is the user properties which are more dynamic over time.

3.2.2. Time Centered Factor

We also tried a simpler way of making the user factor time-dependent by augmenting p_u with a center decay-style factor as follows:

$$p_u(t) = p_u + e^{-\beta_u |t-c(u)|} p_u^{(c)}. \tag{8}$$

Here $c(u)$ is the center point of user u's time line and $p_u^{(c)}$ is the time-centered user factor. The factors at other time points are interpolated from the centered user factor via an exponential decay function with a decay rate parameter β_u. β_u and $p_u^{(c)}$ is trained by the data set. Unlike the bin based model where the parameters of individual bins are independently learnt from each other, the time centered factor is learnt based on ratings from the user's entire history but with a stronger emphasis on the middle part of the history.

3.3. Session Locality

A quite common phenomenon that we have found in the KDD Cup 2011 data set is the presence of *rating sessions*, in which users rate multiple items consecutively. In this work, we define user's rating sessions as a sequence of ratings such that any pair of consecutive ratings are less than five minutes apart. One interesting effect we observed about rating sessions is that while a user can have both high and low ratings, the values of the ratings within a rating session are nevertheless highly correlated. As one way to demonstrate such correlation, we computed the mean of the standard deviations of a every user's full set of ratings, which has a value of 19.38. Similarly, we also computed the mean of the standard deviations of every identified sessions' set of ratings, which has a much lower value of 13.85. We refer to such temporal correlation in user's rating behavior as *session locality.*

To exploit session locality in order to improve prediction accuracy on the test data, we did some analysis and found that 72% of the ratings in test set can be found to be within some rating sessions in the combined training and validation data, which implies that we can use the observed ratings in the same session as a feature when predicting many of the test ratings. To achieve this goal, we first extract all the user rating sessions and compute a mean rating in each user session r_s. Then for each rating that were included in session s, we extend the predictor as follows:

$$\widehat{r}_{ui} = \sigma\Big(b_i + a_u + p_u^T q_i + w_s \cdot (r_s - \bar{b}_u)\Big), \tag{9}$$

where a constant bias with value $(r_s - \bar{b}_u)$ is added to the prediction with weight w_s as a learnable parameter.

3.4. Multi Resolution Dynamic Models

A critical design choice in both our date-time dependent factor and bias modeling is the number of bins, which controls the granularity at which the model deal with time varying user/item characteristics. Models with large number of bins can capture transient behaviors of users/items more effectively but may be subject to overfitting whereas a small number of bins can capture slowly drifting behaviors. So there seems to be a natural trade off between granularity and generalization. Even trickier is the fact that the optimal granularity for different users and items can be quite different and a particular setting with the best overall accuracy can be suboptimal for some users and items. To tackle this difficulty, our solution is to train a collection of dynamic factorizations with varying number of bins, each of which can capture temporal dynamics with different granularity. The final prediction will then be based on the combination of this collection of models rather than any single one. We refer to this framework as multi-resolution dynamic factorization model. In the experiment

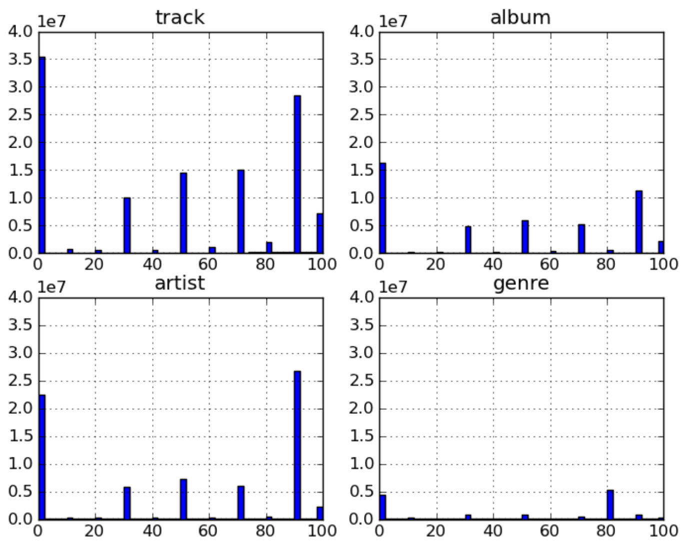


Figure 1: The Distribution of Ratings Scores on Each Category of Items

section, we systematically evaluate ways to deliberately generate and combine models with different temporal resolutions.

4. Incorporating Taxonomical Information

One interesting property of the Yahoo! music data set is that items belong to multiple categories, namely tracks, albums, artists, and genres. The relations among these items naturally form a directed acyclic graph (DAG) structure, where artists are above albums, which then contains tracks, whereas genres can annotate all the other three items. How to utilize this taxonomical structure thus raises another interesting challenge.

We hypothesize that the hierarchical relationships between items can have the following effect. Firstly, the user's rating behavior on different types of items may be intrinsically different. Secondly, a user's preferences over two hierarchically connected items are expected to be correlated. For example, a user who favors an artist is more likely to assign high ratings to his songs. Similarly, a user who hates a genre rarely give good ratings on songs in that genre. In the following subsections, we describe several further enhancements to the proposed matrix factorization model in order to exploit these two effects caused by taxonomy.

To verify the first hypothesis, we tried to examine the user's rating behaviors on different categories of items. In particular, we examine the distribution of rating values on each of the four types of items, which are plotted in Figure 1. It can be clearly seen that users do exhibit highly distinct rating behaviors across item types. In particular, we can note ratings on genres are much more polarized than ratings on other types of items.

In order to see the correlations of user ratings on hierarchically connected item pairs, we produced the following scatter plots showing user ratings on artists against the same user's ratings on the artist's tracks. We randomly choose two artists with more than 1000 ratings. Then for those users who have rated both the artist and some of his tracks, we plot a point with x and y coordinate corresponding to the track and artist rating minus the user's mean rating respectively. From these plots, we can see a clear positive correlation between users' ratings on the artist and the tracks.

Another observation we made about the KDD Cup 2011 data set is that the rating sparsity within different categories of items are highly different. From the Table 1, we can see that on average genres tend to receive the most ratings, followed by artists and then albums with tracks tend to have the fewest ratings on average. Such skewed data sparsity over different items will imply that user preference over more frequently rated items such as genres and artists can be estimated more robustly.

Table 1: Rating Statistics on Each Type of Items

Category	Num of Items	Total Num of Ratings	Avg Num of Ratings
Genre	992	13,829,235	13,940
Artist	27,888	74,985,515	2,688
Album	88,909	48,485,593	545
Track	507,172	119,503,892	235

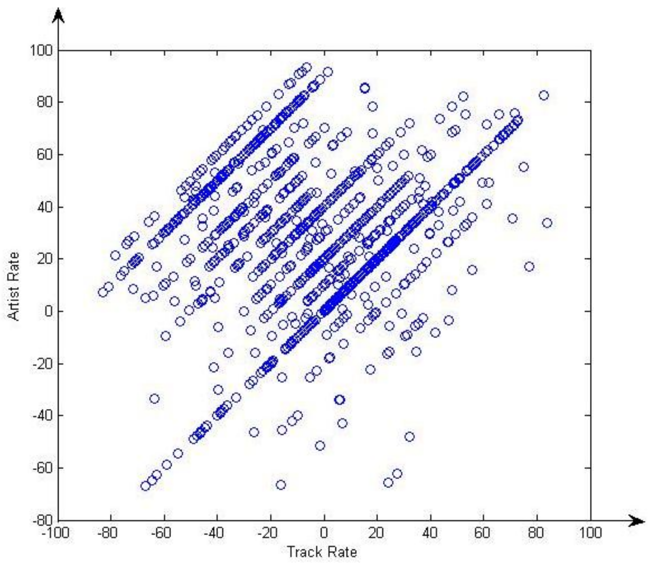
In the following two subsections, we describe techniques for extending both the biases and factors of the matrix factorization model that try to capitalize on such patterns related to taxonomies.

4.1. Taxonomical Bias

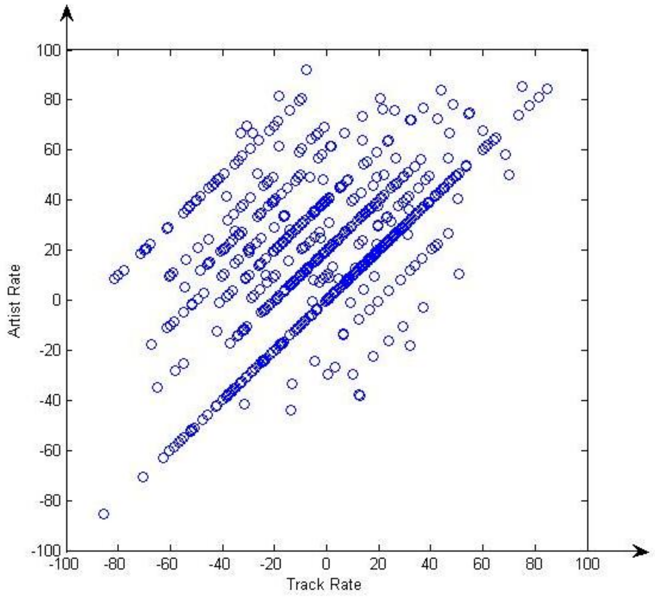
To capture the effect that a user tends to rate different types of items differently, we augment the user bias a_u with 4 additional biases for track, album, artist and genre, which adjusts the predicted rating value based on the type of the item. To exploit the correlation between ratings on artists with ratings on other items, we conducted some further analysis on the data composition. Note that while the training data contains ratings on all four types of items, the validation and test data consists exclusively of ratings on tracks only. Moreover, we also found that for 15% of the ratings on tracks, the user's rating on the track's artist is already known. This provides an opportunity to use the artist rating as another bias when predicting ratings on albums and tracks. In particular, this leads to the following parametrization for predicting track and album ratings:

$$\widehat{r}_{ui} = \sigma\Big(b_i + a_u + a_{u,Cat(i)} + w_i(r_{u,Art(i)} - \bar{b}_u) + p_u^T q_i\Big), \tag{10}$$

where $Cat(i)$ and $Art(i)$ denote the category and the artist of item i respectively. The artist bias $w_i(r_{u,Art(i)} - \bar{b}_u)$ is only used when $r_{u,Art(i)}$ is already known and an additional item specific parameter w_i is learnt to capture the extent to which a rating on item i correlate with the rating on its corresponding artist.



(a)



(b)

Figure 2: Correlation between Artist Ratings and Track Ratings.

4.2. Taxonomical Factorization

In addition to extending the bias component with taxonomy information, we can also capture the correlation between artist ratings and album/track ratings through coupling the latent factor of an artist with those of his albums and tracks. This leads to the following parametrization for predictors on tracks and albums:

$$\widehat{r}_{ui} = \sigma(a_u + b_i + p_u^T(\omega \cdot q_i + (1-\omega) \cdot q_{Art(i)})), \tag{11}$$

where the parameter $\omega \in [0,1]$ controls the relative importance of the target item's factor versus its artist's factor. Using this parametrization, the factors of both the track/album and its artist need to be updated when we process a track/album rating. The artist factor serves as the base feature or prior knowledge for the factors of all the albums and tracks of this artist.

5. Other Extensions

5.1. Integrate Neighborhood Information

The traditional neighborhood methods focus on computing the relationships between items or, alternatively, users. They are most effective at detecting very localized relationships and base predictions on a few very similar neighbors, but they may fall short when there are no or few observed ratings within the neighborhood. In contrast, the latent factor model is effective at capturing global information and has a much better generalization capability due to its capability to more abstractly represent user/items via learnable parameters.

The neighborhood-based prediction model can be easily combined with the matrix factorization model additively:

$$\widehat{r}_{ui} = \sigma\Big(a_u + b_i + p_u^T q_i + |N(u,i;k)|^{-\frac{1}{2}} \sum_{j \in N(u,i;k)} w_{ij}(r_{uj} - \bar{r}_u)\Big). \tag{12}$$

Here the set $N(u,i;k)$ consists of all the items that are selected as k-nearest neighbors of the item i and have been rated by the user u. Traditional neighborhood methods rely on some arbitrary similarity metric such as Cosine or Pearson Correlation Coefficient to define the parameters w_{ij}. In this work, we used a shrunken variant of Pearson Correlation. We treat w_{ij} as free parameters which are learnt together along with the matrix factorization model parameters as first suggested by (Bell and Koren, 2007). During computation, we only need to store and update the parameters for k-nearest neighbors of each item instead of all the item pairs, which results in $k \times n$ parameters where n is number of items. The k-nearest neighbors are precomputed using a map-reduce cluster (Pearson Coefficient as metric). Then we precompute $N(u,i;k)$ on a single machine for each rating record. With the precomputed information, we can efficiently train the model using the method described in Section 7.

The neighborhood-based component of our model can also take time into account by emphasizing the user's more recent ratings with an exponentially decay time weighting function. The idea is shown as follows

$$\begin{aligned}\widehat{r}_{ui} =& \sigma\Big(a_u + b_i + p_u^T q_i \\ &+ |N(u,i;k)|^{-\frac{1}{2}} \sum_{j \in N(u,i;k)} e^{-\alpha_u |\Delta t_j|}(r_{uj} - \bar{b}_u)\Big).\end{aligned} \tag{13}$$

Δt_j is the amount of time between the time of r_{uj} and the prediction time. α_u is initialized with 0 and trained by the data set. This setting can make the recent history contribute more influence over the prediction, and yield better prediction.

5.2. Incorporating Implicit Feedback

In general, implicit feedback can refer to any types of actions users performed on items other than ratings. Implicit feedback is a less precise indicator of user preferences but is generally more abundant and easier to obtain. For music recommendation, ideally the user's listening history such as how many times he listened to a song can be a very useful type of implicit feedback. Unfortunately, such information is not available in the KDD Cup 2011 data set. Instead, we consider another type of simple implicit feedback: whether a user rated an item or not. This type of implicit feedback allows us to utilize the test data, which consists of no ratings, in addition to the training data. To incorporate the information of implicit feedback, we adjust our estimation function as follows:

$$\widehat{r}_{ui} = \sigma\Big(a_u + b_i + (p_u^T + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} y_j^T) q_i\Big), \tag{14}$$

where $R(u)$ denote the set of items that are rated by the user u. Every item j is assigned a feature vector y_j with the same dimension k as the user/item factors p_u, q_i. $|R(u)|^{-\frac{1}{2}}$ is an empirical parameter for normalization of implicit feedback.

Equation 14 shows the implicit feedback extension to basic matrix factorization. We point out that the extension to other models (e.g. neighborhood model) is straightforward. In the restof the paper, we will also show extensions over basic matrix factorization for simplification.

6. Learning with Preference Drift via Importance

Our analysis on the composition of the training and test data of the Yahoo! music data set reveals that all the ratings of a particular user in the test data are dated on or after the last day of their rating in the training and validation data. Furthermore, we also find that the validation data are all dated later than the training data as well. This indicates that the evaluation of this track actually emphasizes predicting each user's latest preferences. Given the lengthy period of time covered by the training data, it is important to inform the model learning strategy to focus more on the latest data rather than equally treating both historical and recent data. It should be noted that the model has already incorporated some mechanisms for supporting date-time awareness, which should be able to remove the global influence of certain non-stationary data characteristics within particular time periods.

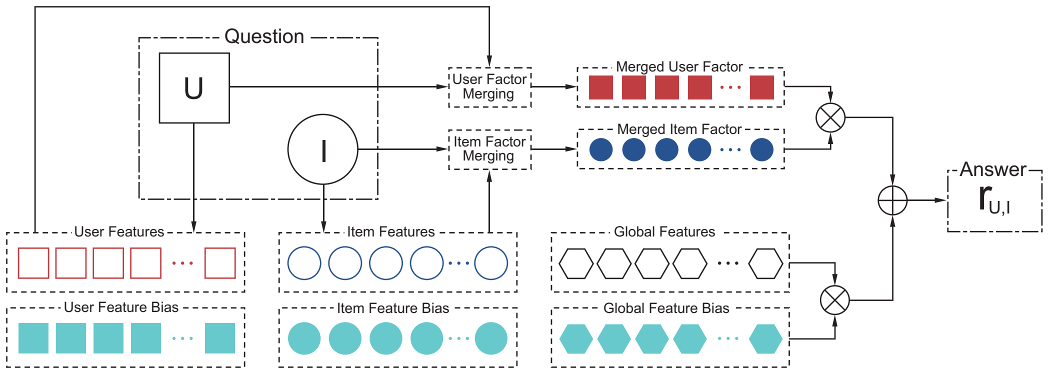


Figure 3: Feature-based matrix factorization

However, our training objective is still to maximize the model's accuracy over all training data. Therefore, we should adapt the training objective to make it *cost-sensitive*.

One common method to implement cost-sensitive model learning is simply to assign a weight to each training rating instance based on its recency and uses a weighted sum of prediction errors in the objective function (Ting, 1998). Intuitively, a user's more recent ratings should be assigned a larger weight whereas his older ratings should have a smaller weight. As incorporating such instance weights does not change the additive form of the objective, the resulting stochastic gradient update rule can be easily adapted by scaling the step length γ with an instance-dependent weight w_{ij}. Unfortunately, we find this simple technique does not work well with the stochastic gradient descent algorithm and does not led to improved performance on the test set.

The basic stochastic gradient descent algorithm can be regarded as randomly sampling a rating instance from a uniform distribution over the available training data in each update step. This inspires us to design an alternative strategy for cost-sensitive learning based on *importance sampling* using a nonuniform distribution over the training data (Zadrozny et al., 2003; Sheng and Ling, 2007). In particular, we let the probability of each rating being sampled be proportional to its recency-based weight, which naturally lets the algorithm more frequently update its parameters based on more recent data. While there exist many possible design choices for the recency weight, we nevertheless find the following simple strategy to work very well. More specifically, we sample the validation data for each user three times more often than the ratings in the training data. Using this simple importance sampling technique, we have achieved RMSE improvement ranging between 0.1 to 0.3 for various different models.

7. Implementation

In implementing the ideas proposed in previous sections, we face two major problems: (1) There are so many variants of models we want to experiment with. (2) The Yahoo! music data set is so big that we must design a scalable solution. In this section, we will discuss how to solve these problems.

We have described many variants of matrix factorization models in the previous sections. Instead of implementing each variant one a time, we design a toolkit to solve the following abstract model in Equation 15

$$\begin{aligned} y(\alpha,\beta,\gamma) = & f\Big(\mu + \left(\sum_j b_j^{(g)}\gamma_j + \sum_j b_j^{(u)}\alpha_j + \sum_j b_j^{(i)}\beta_j\right) \\ & + \left(\sum_j \mathbf{p_j}\alpha_\mathbf{j}\right)^T \left(\sum_j \mathbf{q_j}\beta_\mathbf{j}\right)\Big) \end{aligned} . \tag{15}$$

The input consists of three kinds of features $< \alpha, \beta, \gamma >$, we call α user features, β item features and γ global features. α describes the user aspects that are related to user preference. β describes the item properties. γ describes some global bias effects. Figure 3 shows the idea of the model. We find that most of the models we described in the previous sections can fit into this abstract framework.

A similar idea has been proposed before with factorization machines. Compared with their approach, our model divides the features into three types, while there is no distinction of features in libFM. This difference allows us to include a global feature that does not need to be taken into the factorization part, which is important for bias features such as user day bias and neighborhood-based features. The division of features also gives hints for model design. For global features, we shall consider what aspect may influence the overall rating. For user and item features, we shall consider how to describe user preference and item property better. Basic matrix factorization is a special case of Equation 15. For predicting user/item pair $< u, i >$, we can define

$$\gamma = \emptyset, \alpha_h = \begin{cases} 1 & h = u \\ 0 & h \neq u \end{cases}, \ \beta_h = \begin{cases} 1 & h = i \\ 0 & h \neq i \end{cases}. \tag{16}$$

If we want to integrate neighborhood information, simply redefine γ as Equation 17. Here *index* is a map that maps the possible pairs in the k-nearest neighborhood set to consecutive integers.

$$\gamma_h = \begin{cases} \frac{r_{uj} - \bar{r}_u}{\sqrt{|N(u,i;k)|}} & h = index(i,j), j \in N(u,i;k) \\ 0 & otherwise \end{cases}. \tag{17}$$

We can also include time-dependent user factors by defining new α. Taxonomy information can be integrated into β. We have released the source code of our toolkit[1] [2]. Using the toolkit, we can implement most of the described ideas simply by generating features.

8. Combining Different Models

Different model design choices and parameter settings may have their respective pros and cons. The single model with the best overall performance over a large population of users

1. http://apex.sjtu.edu.cn/apex_wiki/svdfeature
2. We make the ready-to-run experiment scripts for the most effective models available at http://apex.sjtu.edu.cn/apex_wiki/kddtrack1

and items can hardly be the most accurate on every instance. As a result, combining the predictions of multiple models to form a final prediction (aka. ensemble) can often yield much more accurate predictions compared with any individual modelsĖnsemble methods were key to the success of the Netflix prize winning solutions (Jahrer et al., 2010) as well as the winning solutions of most previous KDD Cup contests. In this section, we first describe the basic stacking framework for combining multiple models. We then conduct a series of detailed error analyses to identify a set of conditions under which different models would perform more or less effectively. Based on such findings, we describe a novel information ensemble learning framework which augments the normal stacking model with an additional set of meta-feature.

8.1. Stacking-based Ensemble

To use stacking methods to combine different models, we first build a set of models (i.e., component models) using the original training data and then learn another regression model with the component models' predictions as input features using an additional set of validation data (Breiman, 1996). The training instances for the stacking model is of the format $(<\widehat{r}^0_{ui}, \widehat{r}^0_{ui}, ..., \widehat{r}^n_{ui}>, r_{ui})$, where $\widehat{r}^t_{ui}$ denote the prediction for (u, i) pair by the t-th model. Once the regression model for model combination is learnt, the component models are then retrained using the combined training and validation. In the end, the ensemble regression model is used to combine component models' predictions on the test data to make the final predictions.

8.2. Informative Ensemble with Meta Features

The stacking model presented earlier applies the same set of weights to combine different component models irrespective what the users and items are. However, it would be interesting to investigate whether different models perform more or less effectively under certain conditions. For example, the empirical study carried out by Cremonesi et. al. (Cremonesi et al., 2010) demonstrated that a matrix factorization model with more latent factors tends to perform better on tail items (i.e., items with few ratings) but may be less accurate on popular items due to over-fitting. In this section, we conduct detailed error analysis on the performance of various models and try to identify specific conditions under which different models or parameter settings would systematically work better or worse. We then describe an extension of the stacking model called *informative ensemble*, which introduces a set of additional features to describe the conditions based on which the component models can be more effectively combined.

8.2.1. Effect of Regularization

The regularization controlled by parameter λ is used to guard against overfitting when learning with high dimensional models. We conduct a set of experiments to study if there is any inherent trade off in choosing between weak and strong regularization. In particular, we want to see if these two parameters have different effects on head vs. tail items and heavy vs. light users. We construct user and item segments based on the number of ratings associated with a user or item, and then try to measure the performance within each segment separately as we vary λ. We choose the basic matrix factorization model Equation (1) as the underlying algorithm and the results were plotted in Figure 4 below. In Figure 4-(a), bin 1 contains the set of users with the fewest ratings whereas bin 4 contains those users with the most ratings. In Figure 4-(b), bin 1 contains the set of items with the fewest ratings whereas bin 4 contains those items with the most ratings. From the results, we can clearly see that regularization has a very different effect on different bins. For bin 1, a more strongly regularized model tends to work better whereas for the other bins, a moderate regularization strength tends to produce the optimal performance.

8.2.2. Effect of Taxonomy Awareness

In section 4, we introduced several techniques for exploiting the correlation between user preferences on an artist and his albums/tracks. Since artists are more frequently rated in the Yahoo! Music data set, user preference on artists can often be more reliably predicted. We hypothesize that considering a user's preference for the artist should be most useful when making predictions on tracks and albums with few ratings. To see if taxonomy awareness could have distinct effect on tracks with many or few ratings, we vary the parameter ω in Equation (11) to observe its effect on different item segments, which is plotted in Figure 5. We can see the performance in different bins did exhibit distinct trends with respect to ω. On the least rated items, a larger ω value, which enforces stronger taxonomical correlation, tends to produce better results, whereas on the frequently rated items, performance only gets worse as ω increases.

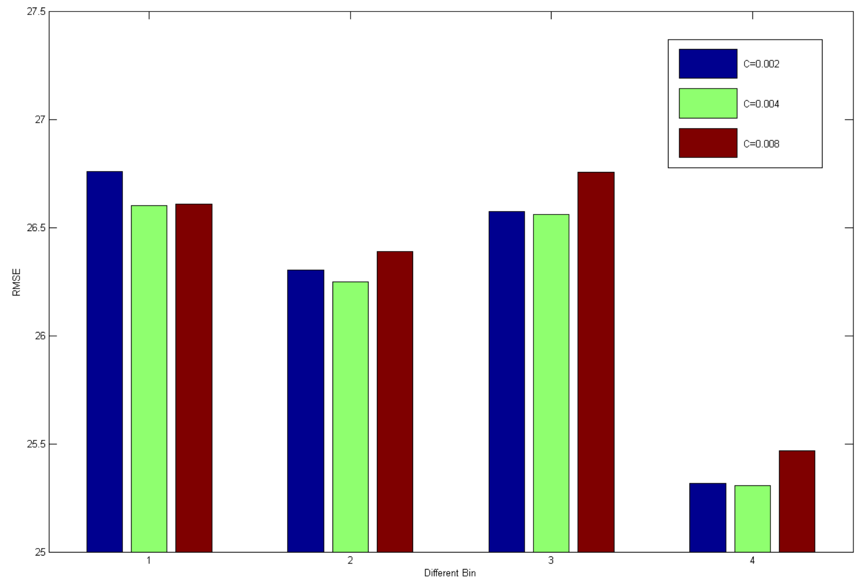
8.2.3. Stacking with Meta Features

As shown in the above experiments, different kinds of models and parameter settings appear to work more or less effectively on users and items under different conditions. Thus it is desirable to have an ensemble method that can adjust how different component models are combined based on the properties of the pair of user and item for which prediction needs to be made. To achieve such a goal, we propose to augment the input representation of training instances for the stacking model with an additional set of *meta features*, each of which describes a certain property of the user or item. So the instances to the ensemble model would be in the form $(< f_1, f_2, ..., f_p, \widehat{r}^0_{ui}, \widehat{r}^0_{ui}, ..., \widehat{r}^n_{ui} >,\ r_{ui})$, where $f_1, f_2, ..., f_p$ are the p meta features. We refer to this stacking framework based on both component model predictions and meta features as an *informative ensemble*. In the ensemble model used in our final solution, there are a total of 16 meta features, which are listed in Table 2. In the experiment section, we will report detailed results on the degree of improvement resulting from including these meta features.

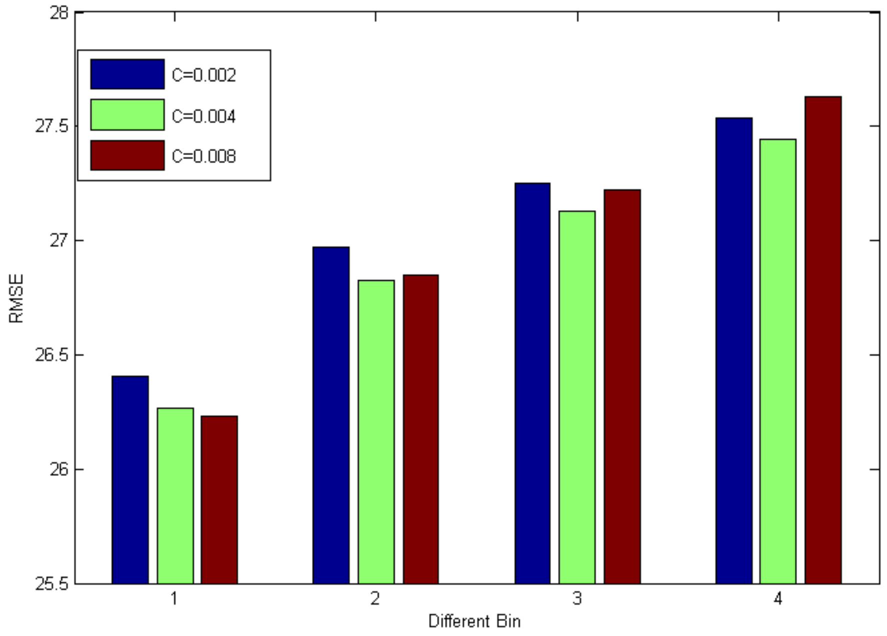
9. Experiments

In this section, we empirically evaluate the effectiveness of various techniques proposed in the paper. We measure the performance in terms of root mean squared error (RMSE). Given a test set S consisting of held out user ratings, the RMSE is computed by:

$$RMSE = \sqrt{\frac{\sum_{(u,i)\in S}(r_{ui} - \widehat{r}_{ui})^2}{|S|}}. \tag{18}$$



(a)



(b)

Figure 4: Effect of Regularization on the number of ratings associated with item and user

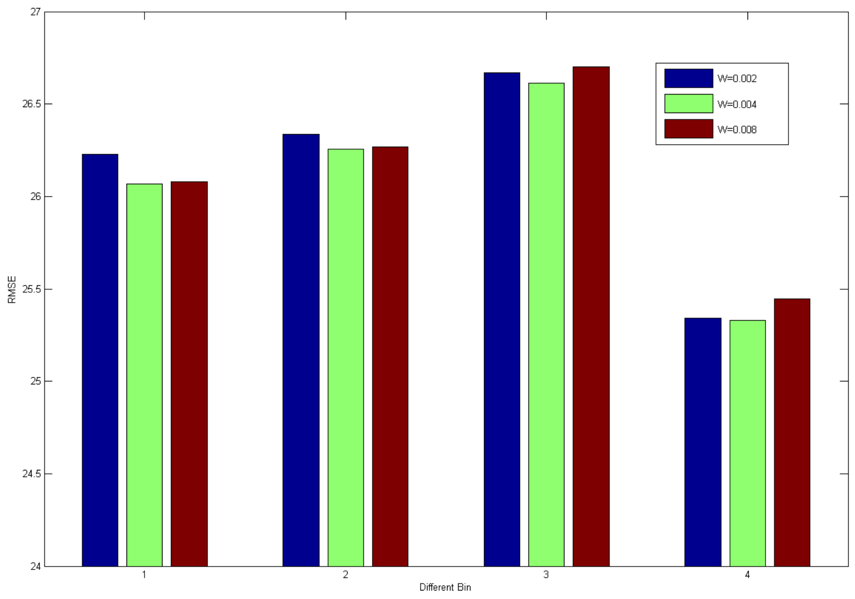


Figure 5: Effect of Taxonomy Awareness on Items Segmented by Number of Ratings

So a lower RMSE value corresponds to higher prediction accuracy. In all the experiments hereafter, the reported RMSE values were measured on the Test1 set provided by the KDD Cup 2011 organizers.

9.1. Single Model Experiments

In this section, we conduct experiments to evaluate the effectiveness of the various extensions to the basic matrix factorization (MF) model in Equation (1) using the Yahoo! Music data. The lowest RMSE obtained by a MF model is 23.4881. For each of the extensions listed in Table 3, we report its best performance as well as the amount of improvement over MF. This allows us to see the usefulness of each of these individual tricks. As can be seen, considering temporal information resulted in the most significant improvement. Date and session features are more useful than time.

Note that the various extensions can be easily combined additively into a single model and the parameters introduced by different tricks could be learnt jointly via the regularized least squares model. We have tried many different combinations of these extensions and found the one shown in the last row of Table 3 to perform best; this is our best single model. For all the single model experiments, we used different regularization constants, and the regularization constant was fixed at 0.004. The learning rate was set as 0.005.

Table 2: Meta Features Used in the Informative Ensemble

ID	Type	Feature Description
1	Numeric	Number of ratings of the user
2	Numeric	Number of ratings of the item
3	Numeric	Mean rating of the user
4	Numeric	Mean rating of the item
5	Numeric	Variance of the ratings of the user
6	Numeric	Variance of the ratings of the item
7	Numeric	Number of days on which the user have ratings in the training data
8	Numeric	Number of days between the user's first and last rating in the training data
9	Binary	Whether the user has any ratings on the artist of the track in the training data
10	Binary	Whether the user has any observed ratings in the session of the target rating
11	Binary	Whether the user has any observed ratings on the day of the target rating
12	Binary	Whether any nearest neighbors of i have been rated by u
13	Categorical	Type of the item i, which may be genre, artist, album and track

Table 3: Summary of Proposed Extensions to Matrix Factorization

ID	Name	Reference	RMSE	% Improvement
A	bin based date bias	Equation (5)	22.89	2.50
B	bin based time bias	Equation (5)	23.21	1.15
C	tensor based date bias (date only)	Equation (6)	22.67	3.46
D	tensor based time bias (time only)	Equation (6)	22.67	3.47
E	date dependent factor	Equation (7)	22.70	3.33
F	time dependent factor	Equation (7)	22.81	2.87
G	date-centered user factor	Equation (8)	23.17	1.35
H	session bias	Equation (9)	23.20	1.22
I	taxonomical bias	Equation (10)	23.35	0.58
J	taxonomical factorization	Equation (11)	23.44	0.18
K	neighborhood model (k=10)	Equation (12)	23.30	0.79
L	time-dependent neighborhood model (k=10)	Equation (13)	23.31	0.73
M	implicit feedback	Equation (14)	22.89	2.51
N	MF+C+D+E+F+H+I+J+M+L		22.09	5.94

Table 4: Models with Different Temporal Resolutions

bin size (date factor)	bin size (time factor)	bin size (date bias)	bin size (time bias)	RMSE
150 days	2 hours	10 days	20 mins	22.40
300 days	4 hours	10 days	20 mins	22.47
600 days	8 hours	10 days	20 mins	22.59
150 days	2 hours	20 days	40 mins	22.44
300 days	4 hours	20 days	40 mins	22.48
600 days	8 hours	20 days	40 mins	22.66
150 days	2 hours	40 days	60 mins	22.49
300 days	4 hours	40 days	60 mins	22.53
600 days	8 hours	40 days	60 mins	22.72

Table 5: Combination of Multi-Resolution Dynamic Models

Scheme	RMSE
LR	22.55
LR with meta feature	22.50
GBRT	22.21
GBRT with meta feature	21.87

9.2. Ensemble Experiments

We also conducted several groups of experiments evaluating various techniques related to ensembles. We have tried both linear regression (LR) and the gradient-boosted regression tree (GBRT) as the learning algorithms used in the stacking model. The LR model linearly combines different component models while the GBRT model combines the input features nonlinearly.

9.2.1. Ensemble of Multi-Resolution Dynamic Models

As we have noted in section 3.4, an important parameter in the tensor-based bias model Equation (6) and the dynamic factor model Equation (7) is the number of bins to divide the time line into, which controls the *temporal resolution* of the model. As we have argued, models with fine and coarse resolutions tend to work better or worse under different conditions, so we suggest combining a collection of models with different temporal resolutions which complement each other. To demonstrate the effectiveness of the idea of *ensemble of multi-resolution dynamic models*, we built a set of models with varying temporal resolutions as listed in Table 4 and measured each individual model's performance. We also compare the different methods for combining these models with results reported in Table 5. It can be seen that while the individual models with different resolutions appeared to have similar performance, combining them nonlinearly via GBRT with meta features reduces the best single model's RMSE by 0.53 or more than 2%.

Table 6: Component Models Used in the Ensemble

ID	Description	RMSE
1	matrix factorization with category and artist bias (100 latent factors)	23.75
2	matrix factorization with category and artist bias (200 latent factors)	23.54
3	1 + category-artist bias+taxonomy aware predictor ($\omega = 0.1$)	23.45
4	1 + category-artist bias+taxonomy aware predictor ($\omega = 0.2$)	23.45
5	2 + category-artist bias+taxonomy aware predictor ($\omega = 0.1$)	23.60
6	2 + category-artist bias+taxonomy aware predictor ($\omega = 0.2$)	22.83
7	2 + tensor date bias (binsize 10 days) and piecewise date linear factor(binsize 150 days)	23.15
8	2 + tensor date bias (binsize 20 days) and piecewise date linear factor(binsize 300 days)	22.71
9	2 + tensor date bias (binsize 40 days) and piecewise date linear factor(binsize 600 days)	22.84
10	7 + implicit feedback	22.75
11	2 + tensor date-time bias (binsize 10 days, 15 mins) and piecewise date-time linear factor(binsize 150 days, 2 hours)	22.83
12	2 + tensor date-time bias (binsize 20 days, 30 mins) and piecewise date-time linear factor(binsize 300 days, 4 hours)	22.70
13	2 + tensor date-time bias (binsize 40 days, 1 hour) and piecewise date-time linear factor(binsize 600 days, 8 hours)	23.59
14	model N in Table 3, 512 latent factors	22.06

Table 7: Combination of Heterogeneous Models

Scheme	RMSE
LR	22.77
LR with meta feature	22.50
GBRT	21.52
GBRT with meta feature	21.26

9.2.2. Ensemble of Heterogeneous Models

In this previous section, we have the effectiveness of homogeneous ensemble, or combination of instances of the same model learnt with different parameters settings. In this section, we further experiment with the combination of heterogeneous models, which yielded the best performance. Our final ensemble model consists of a total 14 models, which are a superset of the models tested in the previous subsection. We tried to create a collection of highly diverse models by deliberately creating models with different subsets of tricks listed in Table 6 as well as including several versions of the same model trained with different parameter settings. Again, we compare different methods of combining models as shown in Table 7. We can see that the nonlinear GBRT model is much more effective than the linear regression model, and incorporating the meta features could further reduce the RMSE by more than 0.2, which is about 1% improvement.

10. Conclusion

In this paper, we describe the design of matrix factorization models and ensemble learning methods for accurate rating prediction on the Yahoo! music data used for KDD Cup 2011. We study different extensions of matrix factorization to handle temporal dynamics and taxonomical information. We also propose a meta-feature based ensemble learning framework for combining multiple models. Experimental results on KDD Cup 2011 data sets demonstrated the effectiveness of the various techniques proposed.

Acknowledgments

The team is supported by grants from NSFC-RGC joint research project HKUST 624/09 and 60931160445. We greatly appreciate the computing support from Professor Lin Gu.

References

R. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *ICDM*, pages 43–52, 2007.

Leo Breiman. Stacked regressions. *Mach. Learn.*, 24:49–64, July 1996. ISSN 0885-6125. doi: 10.1023/A:1018046112532. URL http://portal.acm.org/citation.cfm?id=230972.230977.

Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *RecSys*, pages 39–46, 2010.

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

Michael Jahrer, Andreas Töscher, and Robert Legenstein. Combining predictions for accurate recommender systems. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, pages 693–702, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0055-1.

Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proc. of SIGKDD 2009*, 2009.

Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

Victor S. Sheng and Charles X. Ling. Roulette sampling for cost-sensitive learning. In *ECML*, pages 724–731, 2007.

Kai Ming Ting. Inducing cost-sensitive trees via instance weighting. In *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery*, PKDD '98, pages 139–147, London, UK, 1998. Springer-Verlag. ISBN 3-540-65068-7. URL http://portal.acm.org/citation.cfm?id=645802.669183.

Bianca Zadrozny, John Langford, and Naoki Abe. Cost-sensitive learning by cost-proportionate example weighting. In *Proceedings of the Third IEEE International Conference on Data Mining*, ICDM '03, pages 435–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1978-4. URL http://portal.acm.org/citation.cfm?id=951949.952181.