JMLR: Workshop and Conference Proceedings 18:183–197, 2012 Proceedings of KDD-Cup 2011 competition

Feature Engineering in User's Music Preference Prediction

Jianjun Xie XIE_JJ@YAHOO.COM
Scott Leishman SLEISHMAN@CORELOGIC.COM
Liang Tian LTIAN@CORELOGIC.COM
David Lisuk DLISUK@CORELOGIC.COM
Seongjoon Koo SKOO@CORELOGIC.COM
Matthias Blume MABLUME@CORELOGIC.COM
CoreLogic Inc., 703 Palomar Airport Road, Suite 350, Carlsbad, CA 92011

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

Abstract

The second track of this year's KDD Cup asked contestants to separate a user's highly rated songs from unrated songs for a large set of Yahoo! Music listeners. We cast this task as a binary classification problem and addressed it utilizing gradient boosted decision trees. We created a set of highly predictive features, each with a clear explanation. These features were grouped into five categories: hierarchical linkage features, track-based statistical features, user-based statistical features, features derived from the k-nearest neighbors of the users, and features derived from the k-nearest neighbors of the items. No music domain knowledge was needed to create these features. We demonstrate that each group of features improved the prediction accuracy of the classification model. We also discuss the top predictive features of each category in this paper.

Keywords: KDD Cup, feature engineering, recommender systems, kNN

1. Introduction

The KDD Cup 2011 contest posed a fascinating challenge in recommending music items based on a massive ratings dataset provided by Yahoo! Music (Dror et al., 2012). The competition consisted of two tracks: track one was aimed at predicting scores that users gave to various music items (songs, albums, artists, and genres), and track two targeted separating users' highly rated songs from those they have not rated. In this competition we focused our attention on track two as it has many real world applications but limited prior work.

The task of recommending items to a given user based on their and other users' ratings for other items is known as collaborative filtering (Melville and Sindhwani, 2010). There are two main approaches to the formal collaborative filtering task: memory-based and model-based.

In the memory-based approach (Breese et al., 1998), one first calculates the similarity between users or items from user-item rating data to identify a subset of similar other users or items. A recommendation is then produced by taking a weighted average of the known ratings in this subset. The k-nearest neighbor algorithm (kNN) is a popular approach in this category.

The model-based approach (Bell et al., 2009) constructs a low-complexity representation of the complete ratings matrix. Models based on standard matrix approximation methods such as singular value decomposition (SVD) (Golub and Kahan, 1965), principal component analysis (Pearson, 1901), or non-negative matrix factorization are popular. The model-based approach can handle the problem of data sparsity better and it is more amenable to large data sets.

Track two of the 2011 KDD Cup posed a similar but different problem from the traditional collaborative filtering task. Instead of asking the contestants to predict the user's rating on a given music item (like in track one), it asked the participants to separate user's highly rated songs from those they have not rated. We framed this interesting real world task as a binary classification problem.

Creating effective features is one of the most important steps in solving a supervised learning problem. One way to this end is utilizing the recently emerged latent factor and matrix factorization techniques (Bell et al., 2009). A set of latent user-based features and a set of item-based features can be derived from the user-item ratings using the matrix factorization alogrithms like SVD. However these features require intensive computing cost for a large dataset. Furthermore, it is hard to explain explicitly the meaning of each feature.

In this work, we developed a set of hand crafted features which can be grouped into the following five classes: hierarchical linkage features, item-based statistical features, user-based statistical features, features derived from the k-nearest neighbors of the users, and features derived from the k-nearest neighbors of the items. These features combined with our learning algorithm proved to be quite powerful and led us to a fifth place final ranking for the track two challenge.

The rest of this paper is organized as follows. Section 2 discusses how we set up the problem. The details of feature creation are described in Section 3. Our learning algorithm approach is outlined in Section 4. We show our experimental results in Section 5. Finally we discuss and compare our results with alternative approaches and outline possible improvements in Section 6.

2. Problem Setup

The training dataset for track two consists of 62,551,438 ratings from 249,012 users on 296,111 items. The test dataset consists of 607,032 ratings, in which each user contributes exactly 3 highly rated items and exactly 3 unrated items.

To convert this training data set into a binary classification problem, we needed to create positive and negative exemplars. We employed a similar sampling strategy as was used to prepare the test dataset by the contest organizers. We first extracted all tracks rated 80 or higher from the training dataset and set this aside as sampling pool D. For a given user in the training set, we randomly picked n highly rated tracks by this user from the pool and tagged them as positive samples (which we denoted with a value of 1). D has 9,053,130 ratings and 219,429 distinct tracks. If we randomly pick one item from pool D, the probability of a given item being picked is proportional to its odds of receiving high ratings in the overall population. Here n is chosen as an integer between 3 and 5 depending on the number of highly rated tracks in the training set for this user. All users with less than 3 highly rated tracks in the training set were left out. For users with more than 5

highly rated tracks, we randomly selected up to 5 tracks. This resulted in 491,920 ratings with 101,172 distinct users (the same number of users as in the test dataset). This sampling scheme is to simulate how the test set was created in the contest. We did not explore other imbalanced sampling schemes in the competition.

We then constructed the negative examples from the sampling pool D. For a given user u, we randomly picked n tracks that were not rated by u, and tagged them as negative examples (denoted with value 0). The distribution of the negative examples is therefore the same as the positive ones. The probability of a given item being picked is also proportional to its odds of receiving high ratings in the overall population. Since the training set does not have all the rated items for a given user (each user in the test set has 3 highly rated tracks), we also checked the negative examples against the test set to make sure the tracks tagged as negative for user u were in fact unrated by u. In the end, we were left with a new training set in which each user had n highly rated tracks, and n unrated tracks (which were highly rated by other users). The total number of records in this training set was 983,840. We then built a classification model using this constructed training set to predict which tracks the user rated highly and which ones were never rated by this user.

3. Feature Creation

The performance of a supervised learning model relies on two important factors: predictive features and effective learning algorithms. In this work, we created 5 groups of features:

- Hierarchical Linkage Features
- Track-based Statistical Features
- User-based Statistical Features
- User-based kNN Features
- Item-based kNN Features

3.1. Hierarchical Linkage Features

A distinctive attribute of this year's KDD Cup data set was that user ratings were given to items of four types: tracks, albums, artists, and genres. In addition, the items were linked together to form a hierarchy. Each album was by an artist and consisted of one or more tracks. Additionally each item was associated with one or more genres. This rich hierarchical information was very useful to create predictive features. For example, if a user rated an album highly, there was a good chance that the user also rated the individual tracks of that album highly. Table 1 lists ten hierarchical linkage features extracted for a given user-track pair. The intuition of these features is that if a user rates an item highly, he will rate the linked items highly as well.

We tested the predictiveness of the hierarchical linkage features listed in Table 1. The percentage of records with a tag value of 1 and a tag value of 0 is shown in Table 2, which includes both scenarios of when a feature is triggered (feature value > 0) as well as when a feature is not triggered (feature value ≤ 0). The hit rate is defined as the percentage of records that meet the firing rule. It was seen that variables like $R(u, Al(tr))$, $R(u, Ar(tr))$,

Table 1: Hierarchical Linkage Features for a given user-track (u, tr) pair

Feature notation	Default value	Feature description
$R(u, Al(tr))$	-1	The rating score of user u on the album $Al(tr)$ that contains track tr. -1 means user u never rated album $Al(tr)$.
$R(u, Ar(tr))$	-1	The rating score of user u on the artist $Ar(tr)$ who composed track tr.
$\overline{R}(u, \boldsymbol{G}(tr))$	-1	Average rating score of user u on track tr's genres $\boldsymbol{G}(tr)$.
$\overline{R}(u, \boldsymbol{T}(Al(tr))\backslash tr)$	-1	Average rating score of user u on other tracks that belong to the same album as track tr. $\boldsymbol{T}(Al)$ returns the list of all tracks belonging to album Al.
$\overline{R}(u, \boldsymbol{T}(Ar(tr))\backslash tr)$	-1	Average rating score of user u on other tracks that belong to the same artist as track tr.
$\overline{R}(u, \boldsymbol{I}(\boldsymbol{G}(tr))\backslash tr)$	-1	Average rating score of user u on other items (including track, album and artist) that belong to the same genres as track tr. $\boldsymbol{I}(\boldsymbol{G})$ takes a list of genres $\boldsymbol{G}$ as input and returns the list of all tracks, albums, and artists associated with at least one of the genres.
$N(u, \boldsymbol{G}(tr))$	0	Number of ratings of user u on track tr's associated genres.
$N(u, \boldsymbol{T}(Al(tr))\backslash tr)$	0	Number of ratings of user u on other tracks that belong to the same album as track tr
$N(u, \boldsymbol{T}(Ar(tr))\backslash tr)$	0	Number of ratings of user u on other tracks that belong to the same artist as track tr
$N(u, \boldsymbol{I}(\boldsymbol{G}(tr))\backslash tr)$	0	Number of ratings of user u on other items that belong to the same genre as track tr.

$\overline{R}(u, \boldsymbol{T}(Al(tr))\backslash tr)$ all had over 90% positive tags once triggered. This implied that once the user had rated a track's album (take $R(u, Al(tr))$ as an example), more than 90% of the time the user would rate the track. However, when these variables were not triggered, performance would be reduced significantly. We needed to construct additional features to separate the target when these linkage features did not fire.

Table 2: Predictiveness of the hierarchical linkage features

Feature firing rule	Percentage of tag = 1	Percentage of tag = 0	Hit rate
$R(u, Al(tr)) > 0$	98.53%	1.42%	28.22%
$R(u, Al(tr)) \leq 0$	30.92%	69.08%	71.78%
$R(u, Ar(tr)) > 0$	90.5%	9.45%	41.58%
$R(u, Ar(tr)) \leq 0$	21.16%	78.84%	58.42%
$\overline{R}(u, \boldsymbol{G}(tr)) > 0$	64.47%	35.53%	44.06%
$\overline{R}(u, \boldsymbol{G}(tr)) \leq 0$	38.63%	61.37%	55.94%
$\overline{R}(u, \boldsymbol{T}(Al(tr))\backslash tr) > 0$	90.60%	9.40%	19.06%
$\overline{R}(u, \boldsymbol{T}(Al(tr))\backslash tr) \leq 0$	40.46%	59.54%	80.94%
$\overline{R}(u, \boldsymbol{T}(Al(tr))\backslash tr) > 0$	88.05%	11.95%	27.31%
$\overline{R}(u, \boldsymbol{T}(Al(tr))\backslash tr) \leq 0$	35.72%	64.28%	72.69%

3.2. Track-based Statistical Features

After analyzing the input datasets, we found that individual tracks varied wildly along a number of dimensions. Some were popular and had far more ratings than others. Within the ratings applied by the users, some tracks had a large variance, while others seemed to be consistently rated high or low. To capture some of these properties and further separate the items, we created sixteen statistical features for each track. Table 3 defines each of these features, which describe the statistical properties of ratings a track received across all users. It includes maximum, minimum, mean, different distribution percentiles, the frequency count of high ratings a track received and the probability of this track being picked in the test dataset.

3.3. User-based Statistical Features

Similar to what was seen for tracks, individual users exhibited a variety of behaviors. Some users rated many items whereas others rated just a few. Some users only rated items highly, others had a wide distribution, and some only gave items midrange scores. There was also variation among the types of items each user rated, some focused on tracks, some on albums, and so forth. To quantify and further distinguish between users, we developed a set of statistical features defined in Table 4. In this table x represents either a track, album, artist, or genre, thus there are four sets of user-based statistical features created.

Table 3: Track-based statistical features

Feature notation	Definition
$N(tr)$	Number of ratings for track tr.
$R_{max}(tr)$	Maximum rating of track tr.
$R_{min}(tr)$	Minimum rating of track tr.
$\overline{R}(tr)$	Average rating of track tr.
$R_{\sigma}(tr)$	Standard deviation of ratings on track tr.
$R_{\text{LCLM}}(tr)$	Lower confidence limit for the average rating of track tr.
$R_{\text{UCLM}}(tr)$	Upper confidence limit for the mean rating of track tr.
$R_{i\text{th}}(tr)$	Rating percentiles for track tr. $i \in \{1, 5, 25, 50, 75, 95, 99\}$ and $i = 5$ means 5th percentile.
$N_{R_{\geq 80}}(tr)$	Number of score ≥ 80 ratings for track tr·
$P_{test}(tr)$	Probability that track tr will be picked in test dataset.

Table 4: User-based statistical features. Here x represents a single track, album, artist or genre. Thus there are four sets of these features in total.

Feature notation	Definition
$\overline{R}(u, x)$	User's average rating on x.
$N(u, x)$	User's rating count on x.
$Pct_{R_{\geq 80}}(u, x)$	Proportion of the total of the user's ratings which are highly rated ratings of x.
$Pct(u, x)$	Proportion of the total of the user's ratings which are ratings of x.
$R_{i\text{th}}(u, x)$	ith percentile of user's ratings on x. $i \in \{1, 5, 25, 50, 75, 95, 99\}$.
$R_{\text{LCLM}}(u, x)$	Lower confidence limit for the average rating of user on x.
$R_{\text{UCLM}}(u, x)$	Upper confidence limit for the average rating of user on x.
$R_{\sigma}(u, x)$	Standard deviation of user's ratings on x.
$R_{max}(x)$	Maximum rating of user on x.
$R_{min}(x)$	Minimum rating of user on x.

3.4. kNN Features

Since each user rated a small subset of all possible musical items, the user-items rating matrix was very sparse. To improve prediction accuracy and better approximate the user's ratings for unrated items, a classic collaborative filtering approach involves finding "similar" users who have rated the unrated items. The k-nearest neighbors algorithm (kNN) provides a simple means of finding "similar" users and items, and has been widely studied and applied to recommender systems for over a decade (Konstan et al., 1997).

There are two popular kNN targets for collaborative filtering: users and items. We used both of these methods. User-based kNN features were computed on user vectors in a 296,111-dimensional space with one dimension for each distinct track, album, artist, and genre. Item-based kNN features were computed on item vectors in a 249,012-dimensional space with one dimension for each distinct user. Both the user and item vectors were generally very sparse and this fact was utilized to make the cosine similarity computation described below more efficient. For example, when we calculated the user-user similarity, we were essentially looking for the number of common items rated by both users. We only stored the rated items by each user in a sorted list whose size is usually small. There were only N comparisons to be made in order to find out the overlap between the two sorted lists, where N is the size of the smaller list.

Two important factors must be addressed in the use of kNN: 1) how to set the value for the parameter k (the number of nearest neighbors to consider), and 2) the metric used to measure the distance between each pair of users or items.

An optimal k value is usually obtained by repeated trials, taking the value which minimizes the prediction error. For recommender systems, this is usually computed based on the root mean squared error (RMSE) on rating scores.

Various similarity measures have been explored in the past, including Pearson correlation similarity, cosine similarity, mean squared differences, and so forth (Cremonesi et al., 2010).

For this competition, we did not apply the kNN algorithm directly to classification. Instead, we created two sets of additional features by running kNN first on the similarity between users and then on the similarity between items. These new features were added and used to train our classification model.

3.4.1. User-based kNN features

We used a Cosine similarity between two users u_i and u_j, defined as follows (Breese et al., 1998):

$$S(u_i, u_j) = \frac{\boldsymbol{R_{u_i}} \cdot \boldsymbol{R_{u_j}}}{||\boldsymbol{R_{u_i}}||||\boldsymbol{R_{u_j}}||} \tag{1}$$

where $\boldsymbol{R_{u_i}}$ and $\boldsymbol{R_{u_j}}$ are the rating vectors of u_i and u_j respectively.

Items varied in popularity. Two users who shared ratings for a popular item may not be as similar to one another as two users who share ratings for a rare item. Inspired by the idea of Inverse Document Frequency (Aizawa, 2000) commonly used in information retrieval, we weighted each item based on its popularity, i.e., its frequency count across the whole training dataset. If the frequency count for item i is q_i, the weight for item i is

defined as

$$W_i = \frac{1}{\log_2(3+q_i)}. \tag{2}$$

The Cosine similarity in Equation (1) becomes

$$S(u_i,u_j) = \frac{\sum_{k\in r_i\cap r_j} W_{u_i,k}W_{u_j,k}}{\sqrt{\sum_{m\in r_i} W_m^2}\sqrt{\sum_{n\in r_j} W_n^2}}, \tag{3}$$

where r_i and r_j are the indices of the items rated by u_i and u_j, respectively.

The rating for item i for user u could be predicted using a simple weighted average, as in:

$$R_{pred}(u,i) = \frac{\sum_{j\in K} R(j,i)S(u,j)}{\sum_{j\in K} S(u,j)}, \tag{4}$$

where $R(j,i)$ is the rating of user j on item i. K indexes over users.

Table 5 defines the features created for the user-based kNN. These features describe the average of ratings a track (or its album, artist or genre) received from a user's k nearest neighbors. The number of the nearest neighbors who rated the track as well as the average of similarity scores are also listed as individual features.

3.4.2. Item-based kNN Features

In addition to finding similar users based on the items they rate, we exploited the inverse relationship and found similar items based on how users rated them. We constructed several item-based features by using k-nearest neighbors to find similar items. The similarity measure used is the same as that defined in Equation (1) except users u_i and u_j were replaced by items x_i and x_j. Table 6 gives the definition of these features. For a given user-track pair (u,tr), this set of features describes the ratings received from a user u on a track tr's nearest neighbors. These nearest neighbors can be track, album, artist or genre. The number of ratings and the similarity score are also treated as different features. There are fifteen item-based kNN features created for a given value of k. We used $k = 10$ and $k = 20$ in this competition.

4. Learning Algorithm

Using the complete training dataset as input, we calculated all of the features discussed in Section 3 and fed these results into our classification model. When dealing with a binary classification task, there are a variety of choices for the learning algorithm: logistic regression, neural networks, support vector machines (SVM), decision trees, etc. In this competition we chose gradient boosted regression trees as our classifier due to its implementation simplicity and superior accuracy in many real-world applications (Friedman, 2001; Li et al., 2007; Xie et al., 2009). The gradient boosted regression tree software package we used is called TreeNet, developed by Salford Systems (Salford, 1999). We withheld 25% of the modeling dataset as a validation set and used the remaining 75% as the training set. The parameters used in the final submission of our TreeNet model were set as follows: learning rate = 0.015, number of nodes = 10, number of trees = 1000, minimum child = 200, loss function = Huber.

Table 5: User-based kNN features for a user-track pair (u, tr). $\Pi(u, k)$ represents user u's k nearest neighbors. In this competition we set $k = 10$.

Feature notation	Definition
$\overline{R}(\Pi(u,k), tr)$	Average rating on track tr from user u's k nearest neighbors.
$\overline{R}(\Pi(u,k), Al(tr))$	Average rating on track's album $Al(tr)$ from u's k nearest neighbors.
$\overline{R}(\Pi(u,k), Ar(tr))$	Average rating on track's artist $Ar(tr)$ from u's k nearest neighbors.
$\overline{R}(\Pi(u,k), \boldsymbol{G}(tr))$	Average rating on track's genres $\boldsymbol{G}(tr)$ from u's k nearest neighbors.
$N(\Pi(u,k), tr)$	Number of ratings on track tr from u's k nearest neighbors.
$N(\Pi(u,k), Al(tr))$	Number of ratings on track's album $Al(tr)$ from u's k nearest neighbors.
$N(\Pi(u,k), Ar(tr))$	Number of ratings on track's artist $Ar(tr)$ from u's k nearest neighbors.
$N(\Pi(u,k), \boldsymbol{G}(tr))$	Number of ratings on track's genres $\boldsymbol{G}(tr)$from u's k nearest neighbors.
$\overline{S}(\Pi(u,k), tr)$	Average of similarity score from u's k nearest neighbors who rated track tr.
$\overline{S}(\Pi(u,k), Al(tr))$	Average of similarity score from u's k nearest neighbors who rated track tr's album.
$\overline{S}(\Pi(u,k), Ar(tr))$	Average of similarity score from u's k nearest neighbors who rated track tr's artist.
$\overline{S}(\Pi(u,k), \boldsymbol{G}(tr))$	Average of similarity score from u's k nearest neighbors who rated track tr's genres.

Table 6: Item-based kNN features for a user-track pair (u, tr). We created two sets of item-based kNN features using $k = 10$ and $k = 20$.

Feature Notation	Definition
$\overline{R}(u, \Pi_I(tr, k))$	Average rating from user u on track tr's k nearest neighbor items (can be track, album or artist).
$\overline{R}(u, \Pi_T(tr, k))$	Average rating from user u on track tr's k nearest tracks.
$\overline{R}(u, \Pi_{Al}(tr, k))$	Average rating from user u on track tr's k nearest albums.
$\overline{R}(u, \Pi_{Ar}(tr, k))$	Average rating from user u on track tr's k nearest artists.
$\overline{R}(u, \Pi_G(tr, k))$	Average rating from user u on track tr's k nearest genres.
$N(u, \Pi_I(tr, k))$	Number of ratings from user u on track tr's k nearest neighbor items (can be track, album or artist).
$N(u, \Pi_T(tr, k))$	Number of ratings from user u on track tr's k nearest tracks.
$N(u, \Pi_{Al}(tr, k))$	Number of ratings from user u on track tr's k nearest albums.
$N(u, \Pi_{Ar}(tr, k))$	Number of ratings from user u on track tr's k nearest artists.
$N(u, \Pi_G(tr, k))$	Number of ratings from user u on track tr's k nearest genres.
$\overline{S}(u, \Pi_I(tr, k))$	Average similarity score of track tr's k nearest neighbor items (can be track, album or artist) that are rated by user u.
$\overline{S}(u, \Pi_T(tr, k))$	Average similarity score of track tr's k nearest neighbor tracks that are rated by user u.
$\overline{S}(u, \Pi_{Al}(tr, k))$	Average similarity score of track tr's k nearest neighbor albums that are rated by user u.
$\overline{S}(u, \Pi_{Ar}(tr, k))$	Average similarity score of track tr's k nearest neighbor artists that are rated by user u.
$\overline{S}(u, \Pi_G(tr, k))$	Average similarity score of track tr's k nearest neighbor genres that are rated by user u.

We spent most of our effort generating more predictive features instead of tuning model parameters like learning rate, number of trees with the same set of features, etc. After all feature values were calculated, feature (i.e. variable) selection was performed based on the importance of each feature to overall prediction performance. For each decision tree inside TreeNet, feature importance was calculated by (Breiman et al., 1984):

$$f(x_i, T) = \sum_{t \in T} \Delta I(x_i, T), \tag{5}$$

where $\Delta I(x_i, T) = I(t) -_{pL} I(t_L) -_{pR} I(t_R)$ is the decrease in impurity to the actual or potential split on variable x_i at a node t of an optimally pruned tree T. pL and pR are the proportions of cases sent to the left t_L or right t_R by x_i. The features were entered into the model based on their total contribution to the impurity reduction at each split. The top predective features of each category in the final model are noted and discussed in Section 5.

5. Results

The steps we took to tackle the track two problem follow the order of feature creation described in Section 3. Our first model was built on the ten hierarchical linkage features alone. The output of this model is a probability of the user highly rating the given track. We denote this as $P(u, tr)$. After the model was trained, the test dataset was scored and sorted by descending $P(u, tr)$ value for each user. The top three records for that user (those with the highest $P(u, tr)$) were labeled as 1 and the remaining three were labeled as 0. The error rate for this approach was 10.00% when measured on the 50% of the test set available (called the test1 set) and shown on the leaderboard. The remainder of the test set is called test2 and held out by the contest organizers to determine final ranking.

We proceeded by adding the track-based statistical features to the linkage features and used this to build our second model. This model achieved an error rate of 9.14% on the test1 set.

Table 7 shows the error rates of models with different groups of features. It also lists the top five most predictive features of each model and the top three predictive features in each category. The significance of each feature is calculated by Equation (5) as stated in section 4. From this, it is seen that the hierarchical linkage features dominate the first three models. Only one track based feature $N(tr)$ made it into the list of top five features. None of the top three user-based features made it into the overall top five. This implies that the track-based and user-based statistical features were not as strong as the hierarchical linkage features. Once added into the model, the kNN features became the most predictive. In particular, the item-based kNN features were stronger than the user-based kNN features.

From Figure 1, it is seen that the error rate continued to decrease with the addition of more features. The hierarchical linkage features dominated the first three models. Our best single model result achieved a 3.98% error rate on the test1 set. It has been shown in many data mining competitions (Bell et al., 2008; Guyon et al., 2009; Chapelle and Chang, 2011) that combining predictions from many individual classifiers can further improve the overall accuracy. We bagged 11 boosted tree models in our final ensemble. Each component model had slightly different parameter settings but was trained on the same training set and all five groups of features. Our final model was a simple voting sum of the (1 or 0) prediction

from each component model. For example, if all models predicted 1 for a given user-item pair, then the sum would be 11. The top three tracks that had the highest voting sum for a given user were labeled as 1 and the rest were labeled as 0 in the final result. This bagging process improved our final result to an error rate of 3.87% on the test1 set and 3.80% on test2 set, which placed us fifth on this track of the competition.

Table 7: Results of different models. The error rate is from the leader board on test1 dataset. The top 5 predictive features of each model as well as the top 3 predictive features in each category (2nd row in the same category) are also listed.

Feature Group	Error Rate	Predictive Features
Hierarchical linkage (HL)	10.00%	$R(u, Ar(tr))$, $\overline{R}(u, \boldsymbol{T}(Al(tr))\backslash tr)$, $R(u, Al(tr))$, $\overline{R}(u, \boldsymbol{T})Ar(tr))\backslash tr)$, $\overline{R}(u, \boldsymbol{G}(tr))$
HL + Track-based statistics (TS)	8.97%	$R(u, Ar(tr))$, $R(u, Al(tr))$, $\overline{R}(u, \boldsymbol{T}(Al(tr))\backslash tr)$, $\overline{R}(u, \boldsymbol{T}(Ar(tr))\backslash tr)$, $N(tr)$
		$N(tr)$, $R_\sigma(tr)$, $R_{\text{UCLM}}(tr)$
HL + TS + User-based statistics	7.19%	$R(u, Ar(tr))$, $R(u, Al(tr))$, $\overline{R}(u, \boldsymbol{T}(Al(tr))\backslash tr)$, $N(u, \boldsymbol{I}(\boldsymbol{G}(tr))\backslash tr)$, $\overline{R}(u, \boldsymbol{T}(Ar(tr))\backslash tr)$
		$N(u, x)$, $Pct(u, al)$, $N(u, tr)$
Above + User-kNN	5.78%	$\overline{R}(\Pi(u, k), tr)$, $R(u, Al(tr))$, $R(u, Ar(tr))$, $N(u, \boldsymbol{I}(\boldsymbol{G}(tr))\backslash tr)$, $\overline{R}(u, \boldsymbol{T}(Al(tr))\backslash tr)$
		$\overline{R}(\Pi(u, k), tr)$, $\overline{R}(\Pi(u, k), Ar(ar))$, $N(\Pi(u, k), tr)$
Above + Item-kNN (k=10)	4.39%	$\overline{R}(u, \Pi_I(tr, k))$, $\overline{S}(u, \Pi_I(tr, k))$, $R(u, Ar(tr))$, $R(u, Al(tr))$, $\overline{R}(\Pi(u, k), tr)$
		$\overline{R}(u, \Pi_I(tr, k))$, $\overline{S}(u, \Pi_I(tr, k))$, $\overline{R}(u, \Pi_{Ar}(tr, k))$
Above + Item-kNN (k=20)	3.98%	$\overline{R}(u, \Pi_I(tr, k))$, $\overline{S}(u, \Pi_I(tr, k))$, $R(u, Ar(tr))$, $R(u, Al(tr))$, $N(u, x)$
		$R(u, \Pi_I(tr, k))$, $\overline{S}(u, \Pi_I(tr, k))$, $N(u, \Pi_I(tr, k))$
Final model	3.87%	Final model is a vote average of 11 models that each have all the above categories of features but with slightly different model configurations.

6. Discussion

We have described the detailed steps taken by our team in the KDD Cup 2011 track two competition. We converted each user's music preference prediction into a binary classification problem from the one class training dataset. We created five sets of derived features to characterize the behavior of users and musical items. From this, we found that the item-based kNN features were the strongest among all the sets of features. The hierarchical linkage features were the second-strongest. The predictiveness of these features revealed

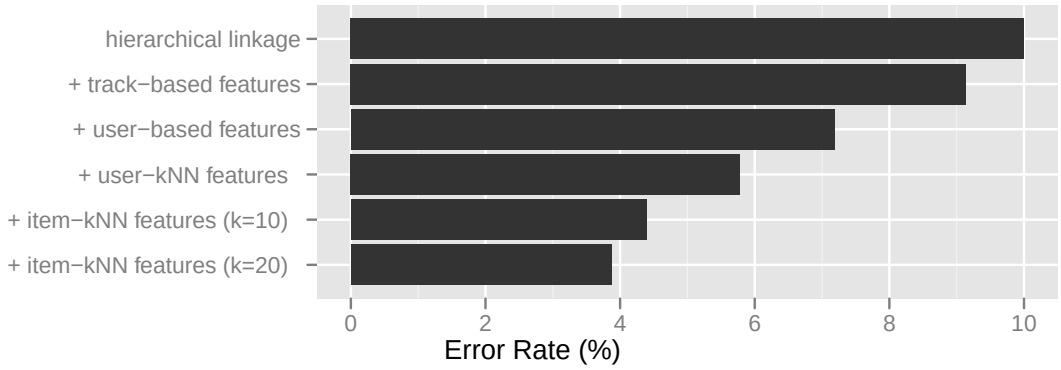

Figure 1: Improvement of model performance with increase of features

the importance of the hierarchical relationships among the musical items. User-based kNN features were weak compared with the above two sets. One reason may have been that the user-based data was sparser (it is very unlikely that a given item has only been rated by 1 or 2 users, however highly possible that a user only rated a handful of items). Another reason is that track two only required the algorithm to rank the items within the same user. As long as the rank of each item within the same user was correct, differing characteristics between users may have had less impact on the result.

Matrix factorization, particularly the SVD approach, has demonstrated superior prediction accuracy than the classic nearest-neighbor techniques for recommender systems (Bell et al., 2009). An alternative approach would have been to use the latent factors generated from the SVD to build the classification model. Since the latent factors are obtained through global optimization on the rating error, the classification model built on top of these factors would likely have better performance. However, these latent features may not have as clear explanations as the features we utilized in this work.

The top three teams on the leaderboard all broke the 2.5% error rate. This improvement may come from the complex blending of many single models (Mckenzie et al., 2012; Lai et al., 2012; Jahrer and Toscher, 2012) rather than the simple voting sum of 11 models used in our work. The post-processing based on the discovery of some data insights (Lai et al., 2012) also further improved the final model results, which our team failed to explore.

In addition to the SVD approaches, further improvements in model performance were possible within the current framework. These areas include optimizing the k value in the kNN features, testing different similarity functions and combining the models built on different samples (we only performed one sampling for the model development dataset in the competition).

Acknowledgments

We would like to thank the organizers of this year's KDD Cup competition as well as Yahoo! for making the dataset available. We would also like to acknowledge our employer CoreLogic Inc. for supporting our endeavor.

References

Akiko Aizawa. The feature quantity: an information theoretic perspective of tfidf-like measures. In *Proceedings of the 23rd annual international ACM SIGIR conference on research and development in information retrieval*, pages 104–111, 2000.

Robert M. Bell, Yehuda Koren, and Chris Volinsky. The BellKor 2008 solution to the Netflix Prize. *Statistics Research Department at AT&T Research*, 2008.

Robert M. Bell, Yehuda Koren, and Chris Volinsky. Matrix factorization techniques for recommender systems. In *IEEE Computer*, volume 42, pages 30–37, 2009. URL http://research.yahoo.com/node/2859.

John S. Breese, David Heckerman, and Carl Myers Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In G. F. Cooper and S. Moral, editors, *Proceedings of the 14^{th} Conference on Uncertainty in Artificial Intelligence*, pages 43–52, 1998.

Leo Breiman, Jermone H. Friedman, R. Olshen, and Charles J. Stone. *Classification and regression trees*. Chapman & Hall/CRC, 1984.

Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In *JMLR Workshop and Conference Proceedings*, volume 14, pages 1–24, 2011.

Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In Xavier Amatriain, Marc Torrens, Paul Resnick, and Markus Zanker, editors, *Proceedings of the 2010 ACM Conference on Recommender Systems, ACM RecSys 2010, Barcelona, Spain, September 26-30*, pages 39–46, 2010. URL http://research.yahoo.com/pub/3500.

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

Jerome H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 28(5):1189–1232, 2001.

Gene Golub and William Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics: Series B, Numerical Analysis*, 2(2):pp. 205–224, 1965. ISSN 0887459X. URL http://www.jstor.org/stable/2949777.

Isabelle Guyon, Vincent Lemaire, Boull Marc, Gideon Dror, and David Vogel. Analysis of the KDD Cup 2009: Fast Scoring on a Large Orange Customer Database. In *JMLR Workshop and Conference Proceedings*, volume 7, pages 1–22, 2009.

Michael Jahrer and Andreas Toscher. Collaborative filtering ensemble for ranking. *Journal of Machine Learning Research W&CP*, 18:in this volume, 2012.

Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. GroupLens: Applying Collaborative Filtering to Usenet News. *Communications of the ACM*, 40:77–87, 1997.

Siwei Lai et al. Hybrid recommendation models for binary user preference prediction problem. *Journal of Machine Learning Research W&CP*, 18:in this volume, 2012.

Ping Li, Christopher J. C. Burges, and Qiang Wu. McRank: Learning to rank using multiple classification and gradient boosting. *Advances in neural information processing systems*, 2007.

Todd G. Mckenzie et al. Novel models and ensemble techniques to discriminate favorite items from unrated ones for personalized music recommendation. *Journal of Machine Learning Research W&CP*, 18:in this volume, 2012.

Prem Melville and Vikas Sindhwani. Recommender systems. In Claude Sammut and Geoffrey Webb, editors, *Encyclopedia of Machine Learning*. Springer-Verlag, 2010.

Karl Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

Salford. Salford Systems - TreeNet. http://www.salford-systems.com/products/treenet/overview.html, 1999.

Jianjun Xie, Viktoria Rojkova, Siddharth Pal, and Stephen Coggeshall. A Combination of Boosting and Bagging for KDD Cup 2009 - Fast Scoring on a Large Database. In *JMLR Workshop and Conference Proceedings*, volume 7, pages 35–43, 2009.