JMLR: Workshop and Conference Proceedings 215–229

Committee Based Prediction System for Recommendation: KDD Cup 2011, Track2

Hang Zhang HZHANG@OPERASOLUTIONS.COM
12230 El Camino Real, Suite 330, San Diego, CA 92130
Eric Riedl EBRIEDL@MATH.HARVARD.EDU
Department of Mathematics, Harvard University
Valery Petrushin VPETRUSHIN@OPERASOLUTIONS.COM
12230 El Camino Real, Suite 330, San Diego, CA 92130
Siddharth Pal SIDDHARTH.PAL@OPERASOLUTIONS.COM
12230 El Camino Real, Suite 330, San Diego, CA 92130
Jacob Spoelstra JSPOELSTRA@OPERASOLUTIONS.COM
12230 El Camino Real, Suite 330, San Diego, CA 92130

Abstract

This paper describes a solution to the 2011 KDD Cup competition, Track2: discriminating between highly rated tracks and unrated tracks in a Yahoo! Music dataset. Our approach was to use supervised learning based on 65 features generated using various techniques such as collaborative filtering, SVD, and similarity scoring. During our modeling stage, we created a number of predictors including logistic regression, artificial neural networks and gradient-boosted decision trees. To further improve robustness and reduce the variance, we used three of our top performing models and took a weighted average for the final submission, which achieved 4.3768% error.

Keywords:
KDD Cup 2011, musical track ranking, ranking prediction, prediction model, artificial neural network, gradient boosting decision tree, committee of predictors

1. Introduction

Recommender systems have revolutionized e-commerce. The ability to accurately predict users' preferences has had far-reaching implications for marketing, on-line communities and e-commerce. Companies such as Amazon and Netflix have been some of the most publicized examples of using recommender systems, but recommender systems pervade the internet and many other aspects of our lives.

The challenge of the KDD Cup 2011 was to build recommendation engines on a Yahoo! Music dataset. The dataset for the competition was sampled from the actual Yahoo! Music community's ratings for various musical items. User ratings are on four different types of items: tracks, albums, artists and genres.

The challenge was divided into 2 separate tasks viz, Track1 - predicting the scores given by users to various items and Track2 - separation of three highly rated (items obtaining score of at least 80 on the scale of 0 to 100) from three other unrated tracks for a user.

Both these tasks were independent and contestants were allowed to compete at either one or on both of the tasks. We focused our attention on Track2.

For Track2, Yahoo! Inc. used a dataset, $\mathbf{\Omega}$, containing user ratings on Yahoo! Music. The set $\mathbf{\Omega}$ was further split into two parts by the competition organizers: a training set $\mathbf{\Omega_L}$ and a test set $\mathbf{\Omega_T}$. The test set $\mathbf{\Omega_T}$ was constructed so that for each user in $\mathbf{\Omega_T}$ there were precisely six tracks, three of which the user had rated highly and three of which the user had not rated. The three highly rated tracks were sampled uniformly from the user's highly rated tracks, while the three unrated tracks were sampled from the user's unrated tracks with probability proportional to the number of high ratings they had received from the general population. Contestants were asked to identify for each user in $\mathbf{\Omega_T}$ the three tracks which were highly rated (predict 1) and the three tracks which had not been rated (predict 0). The actual labels were held back and a subset (Test1) of $\mathbf{\Omega_T}$ was used to determine the position of the contestants on the leaderboard. The ranking of the contestants was based on total number of misclassifications. The results on the remainder (Test2) of $\mathbf{\Omega_T}$ were not disclosed until the end of the competition.

In addition to the rating data, the competition organizers also provided datasets describing the hierarchical relationships among items. Figure 1 shows all the different possible hierarchical relationships among the four types of items. As the figure shows, a track may have a parent album, a parent artist, and multiple parent genres. Similarly, an album may have a single parent artist and multiple parent genres, and an artist may have multiple parent genres. Some items may not have a complete structure of parent items. For instance, some tracks may not have parent album, but still have a parent artist and parent genres.

Track2 of the competition posed several challenges.

- The training data consisted of 224,041 tracks, 52,829 albums, 18,674 artists, 567 genres and 249,012 users. The dimensions of the ratings matrix on tracks alone would be $224,041 \times 249,012$, which is too large to be stored in memory. The size of the ratings matrix made it difficult to directly implement many standard machine learning techniques.

- The rating data was very sparse. On average, a user had rated about 109 tracks out of the total 224,041 tracks, and a track had been rated by 121 users out of the total 249,012 users. Users rated an average of 48 albums, 77 artists and 14 genres each. As the numbers show, most users had only rated a very small fraction of the items.

- The format of the test data was different from that of the training data. The training data had different numbers of ratings for different users, while in the test data, only six items per user were given. Moreover, in the training data, the target value for each user-item pair was a rating, while in the testing data, the target value for a user-item pair was binary. Clearly, these formats are very different both in terms of the type of data entry and the target value.

- Finally, there was no single, obvious way to use the hierarchical data. This was further complicated by the fact that not every item had a complete set of parents.

We organize the paper as follows. In Section 2 we describe our approach to the challenge. In Section 3 we describe the data analysis, feature extraction and generation of the training

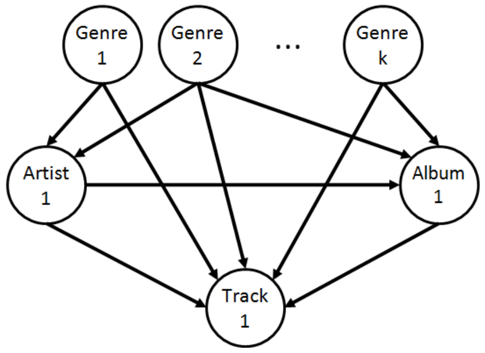

Figure 1: Hierarchical structure of items.

and test datasets. In Section 5 we describe the modeling methodology and in Section 6 we summarize our final submission and how it compared with other teams on the leaderboard.

2. Our Approach

We addressed the challenges posed by Track2 of the competition by translating the problem into a supervised learning framework. We began by sampling out a dataset $\hat{\mathbf{\Omega}}_\mathbf{T}$ from $\mathbf{\Omega}_\mathbf{L}$ that shared the same structure as the test data i.e, it contained six items per user with three items highly rated by the user and three items unrated by the user. The next step was to extract features for $\hat{\mathbf{\Omega}}_\mathbf{T}$ and $\mathbf{\Omega}_\mathbf{T}$ and prepare the modeling datasets viz, $\mathbf{D}_\mathbf{L}$ and $\mathbf{D}_\mathbf{T}$ respectively. We used $\mathbf{D}_\mathbf{L}$ for training various supervised classification algorithms.

By proceeding in this way, we were able to get around working on a huge rating matrix. Extracting features also helped us deal with the sparsity of the rating data, because we could select features which made sense even when we had relatively little rating information on a user. Because we extracted features in a flexible, ad hoc manner, we were able to use the hierarchical data in many different ways, which allowed us to better deal with cases like a track with no album.

3. The Data

Data plays the most crucial role in developing good prediction models. In order to give a sense of the types of rating data available, we present a few statistics. Table 1 presents the total numbers of ratings on tracks, albums, artists, and genres from the general population in $\mathbf{\Omega}_\mathbf{L}$. Table 2 presents the average percentage of each user's ratings which were on the four classes of items in $\mathbf{\Omega}_\mathbf{L}$.

Notice that from Table 1, we see that there were significantly more ratings on tracks than there were on artists. However, from Table 2 we see that on average a much higher percentage of a user's ratings were on artists than were on tracks. This suggests that users with a lot of ratings had more of their ratings on tracks, while users with relatively few ratings had more of their ratings on artists. This is probably part of the reason that our

Table 1: Numbers of ratings on four classes of items from general population in $\mathbf{\Omega_L}$

	# of ratings	% of total ratings
Track	27,167,857	43.9
Album	11,928,316	19.3
Artist	19,289,882	31.1
Genre	3,558,351	5.7

Table 2: Average % (and standard deviation) of a user's ratings on four classes of items in $\mathbf{\Omega_L}$

Tracks	Albums	Artists	Genres
21.1(22.4)	9.5(11.8)	55.2(27.3)	14.2(17.7)

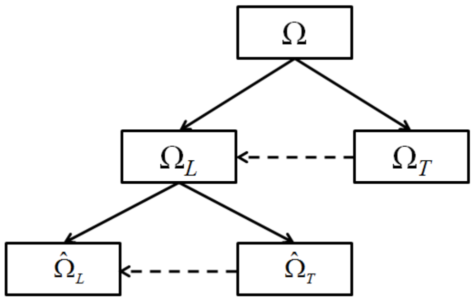

Figure 2: Data splits.

features which took into account statistics about the artist had higher significance in our models.

We split dataset $\mathbf{\Omega_L}$ into two sets $\mathbf{\hat{\Omega}_T}$ and $\mathbf{\hat{\Omega}_L}$ in a manner similar to the way Yahoo! Inc. prepared the training data $\mathbf{\Omega_L}$ and the test data $\mathbf{\Omega_T}$ from $\mathbf{\Omega}$. For each user in $\mathbf{\Omega_L}$ with at least six highly rated tracks, we randomly picked three highly rated tracks and put them into $\mathbf{\hat{\Omega}_T}$. The remaining ratings of the user were put into $\mathbf{\hat{\Omega}_L}$. We then randomly picked three tracks not rated by the user, where the probability of selecting a track was proportional to the number of high ratings it received from the general population, and put them into $\mathbf{\hat{\Omega}_T}$ to constitute the remaining three tracks for the user. Thus, every user in $\mathbf{\hat{\Omega}_T}$ had at least three highly rated tracks in $\mathbf{\hat{\Omega}_L}$, which gave us useful information about the user's preferences. If a user had less than six highly rated tracks in $\mathbf{\Omega_L}$, the user's data was not split and was all put into $\mathbf{\hat{\Omega}_L}$. We ended up with 88186 users in $\mathbf{\hat{\Omega}_T}$.

Figure 2 depicts the dataset partition by the competition organizers and the further splits carried out by us as described above. The solid arrows represent a data split.

4. Feature Extraction and Analysis

Datasets $\hat{\mathbf{\Omega}}_{\mathbf{L}}$ and $\mathbf{\Omega}_{\mathbf{L}}$ acted as reference dictionaries for extracting information about a user in $\hat{\mathbf{\Omega}}_{\mathbf{T}}$ and $\mathbf{\Omega}_{\mathbf{T}}$, respectively. These two datasets contain information about a user's preferences and how other users have rated similar items. These dictionaries are represented as dash-line arrows in Figure 2.

In all we extracted 65 features for the {user,track} pair $\{u_i, t_j\}$. We use al_j, ar_j, and $g_{j1}, \cdots, g_{jk}$ to denote the parent album, artist, and genres of t_j, respectively.

4.1. Six Classes of Features

These 65 features can be categorized into six classes.

- *Forty-seven features based on statistics* - We used various statistics to generate 47 features. Rather than list all of the features, we give a few examples which we hope will give a flavor of the types of statistics we considered.
  - The fractions of the child tracks of album al_j, artist ar_j, and genres g_{j1}, ..., g_{jk} which were rated and rated highly by user u_i. (6 features)
  - The relative frequencies of the parent album, artist, and genres of track t_j being rated or highly rated by user u_i, compared to the total number of ratings the user u_i placed on albums, artists, and genres, respectively. (6 features)
  - The fractions of the general population's ratings on t_j and its parent album, artist, and genres which are high. (4 features)
  - The average ratings over all users on the track t_j or its parent album al_j, artist ar_j and genres g_{j1}, g_{j2}, ..., g_{jk} (4 features)

- *Seven features derived from SVD prediction* - SVD has been widely used in recommendation systems. It has been shown that it can extract latent features from a huge rating matrix, and improve the scalability of the recommendation system by reducing the dimension of the data while still making accurate predictions. Readers are referred to Johnson and Wichern (2002) for details of SVD, and Sarwar et al. (2002) and Vozalis and Margaritis (2007) for its applications in recommendation systems. Because of the large size of the data matrix, we implemented a sparse SVD algorithm, which only required loading the {user, item} rating entries. As a trade-off between learning speed and accuracy, we selected $k = 500$ as the number of components.

  We trained SVD models on the rating data of each of the four hierarchical levels to predict the ratings of user u_i on track t_j, album al_j, artist ar_j and genres $g_{j1}, \cdots, g_{jk}$, and used the predictions at each level (the average of the predictions on the genre level) as features. For each SVD model, we considered only the direct ratings of the corresponding hierarchical level. We also generated three SVD models, one each on the album, artist and genre levels, where we ignored direct ratings and considered ratings on child tracks as ratings on the item. Our sparse SVD algorithm was able to gracefully handle multiple ratings for the same {user,item} pair, which was encounted when we used the ratings on tracks to represent the ratings on parent albums, artists, and genres in the last three SVD models.

- *Four features derived from user-based collaborative filters* - Collaborative filters (CFs) are methods of estimating the ratings on a {user, item} pair $\{u, i\}$ by using ratings on item i from other users similar to user u (user-based CFs), or user u's ratings on items that are similar to item i (item-based CFs). CFs have been widely used in many recommendation systems, such as GroupLens (Resnick et al., 1994), LikeMinds (Greening, 1997), and FireFly (Shardanand and Maes, 1995). We implemented two user-based CFs, LikeMinds and FireFly, to predict user u_i's response to track t_j and its parent album al_j. We expected that the users who had similar rating preferences to u_i and who had rated the item before might give us extra information that was missed in the previous features. Here, we did not include user-based CFs for the parent artist and genres of t_j since the predictive values of the CFs at these two levels did not help improve the performance of our prediction system.

- *Three features derived from item-based CFs* - We also used the predicted ratings from item-based CFs, on t_j, al_j, and ar_j, as features. For user u_i, we calculated the pairwise similarity, the Pearson correlation coefficient, between t_j and each of the other tracks u_i rated in the referred dictionary. The predicted values from the item-based CFs were just the average rating over all other tracks, albums, and artists the user rated, weighted by the similarity with t_j, al_j, and ar_j, respectively. Sarwar et al. (2001) had a detailed description of the implemented item-based CFs.

- *Three features derived from similarity scores* - We used similarity scores for tracks from the test set based on their similarities with the user's highly rated tracks' hierarchies. The idea is to look at the prediction problem as a relevant document retrieval problem (Manning et al., 2008). All rated tracks form a document and their album, artist and genre IDs are words. The set of all highly rated tracks form the user's document space. Each test track is a query, which can get a similarity score that reflects how close the track is to the user's favorive tracks.

  Three similarity scores are explored. Let the user's rating history be represented as a set of 3-tuples (al_i, ar_i, g_i), where $i = 1, \cdots, n$ and al_i is an album ID , ar_i is an author ID and g_i is a genre ID of a track that was rated highly by the user. The test track also is represented as a set of 3-tuples (al, ar, g_j), where $j = 1, \cdots, m$ and al is an album ID, ar is an artist ID and g_j are genre IDs.

  The first similarity score for the track t and user u is calculated using (1). First the weighted score S_{ij} for each combination of 3-tuples from the user history and the test track is calculated. Then we calculated the maximum and average of S_{ij} added them to obtain the final score.

$$\begin{aligned}
&S_{ij} = w_1 \cdot (al_i \equiv al) + w_2 \cdot (ar_i \equiv ar) + w_3 \cdot (g_i \equiv g_j) \\
&S_{\texttt{max}} = \texttt{max}\, S_{ij} \\
&S_{\texttt{avg}} = \texttt{mean}(S_{ij}) \\
&S_1(t, u) = S_{\texttt{max}} + S_{\texttt{avg}}
\end{aligned} \tag{1}$$

  Here $(a \equiv b)$ is equal to 1 if a and b are the same IDs and 0 otherwise. The weights were optimized experimentally and set to $w = (1, 1, 2)$. This means that the similarity at Genre level has higher weights than at Album and Artist levels.

For the second similarity score, the user's space is represented as pairs (item, count), where item is an album, author or genre ID and count is the number of times the user rated the item highly. The similarity score is calculated as a weighted sum of rating frequencies for album, author and genres. The weights were set to $w = (10, 5, 2)$ to emphasize the importance of album and artist ratings.

The third similarity score implements the TF-IDF (term frequency-inverse document frequency) approach with cosine similarity. Each item in the user's space is represented by a weight (2).

$$W_{ij} = \begin{cases} (1 + \log(T_{iu})) \cdot \log(N/D_i) & \texttt{if } T_{ij} > 0 \\ 0 & \texttt{otherwise} \end{cases} \tag{2}$$

where T_{iu} is the number of occurences of the i-th item in user u's top ratings, D_i is the number of users that highly rated i-th item, and N is the number of users (in the training set).

The test track is represented as a vector of weights that correspond to its album, author and genres and similarity is estimated as a dot product with the corresponding vector of weights in the user's space (cosine similarity). Due to scarsity of data, the direct usage of the above similarity scores for prediction gives an error rate in the range $12 - 15\%$, but they served as valuable features for training more sophisticated models.

- *One feature estimating a likelihood ratio* - We computed one feature which attempts to capture the ratio between how likely t_j is drawn from user u_i's highly rated tracks and how likely t_j is drawn from the general population's rating records. Because Yahoo! selected unrated tracks in the test set based on how likely they were to be highly rated by other users, and the highly rated tracks uniformly from the user's own rating history, we hope that this feature helped us account for the difference between these distributions. We put forward the following quantities as estimates of the likelihood that t_j is drawn from u_i's rating records:

$$\begin{aligned} p^0_{al_j} &= \frac{\texttt{\# of child tracks of album } al_j \texttt{ highly rated by user } u_i}{\texttt{\# of tracks highly rated by user } u_i} \\ p^0_{ar_j} &= \frac{\texttt{\# of tracks by artist } ar_j \texttt{ highly rated by user } u_i}{\texttt{\# of tracks highly rated by user } u_i} \\ p^0_{g_{jl}} &= \frac{\texttt{\# of tracks in genre } g_jl \texttt{ highly rated by user } u_i}{\texttt{\# of tracks highly rated by user } u_i} \end{aligned} \tag{3}$$

We see that $p^0_{al_j}$ is the probability that u_i's highly rated tracks have parent album al_j, and the other quantities have similar interpretations. In a similar way, we estimate the probabilities $p^1_{al_j}$, $p^1_{ar_j}$, and $p^1_{g_{j1}}, \cdots, p^1_{g_{jk}}$ that highly rated tracks from the general population have parent al_j, ar_j and $g_{j1}, \cdots, g_{jk}$.

Then the likelihood ratio lr is defined as:

$$lr = \frac{p^0_{al_j} + p^0_{ar_j} + \sum_{\texttt{l=1,2,...,k}} p^0_{g_{jl}}}{p^1_{al_j} + p^1_{ar_j} + \sum_{\texttt{l=1,2,...,k}} p^1_{g_{jl}}} \tag{4}$$

4.2. Feature Analysis

Our features had widely varying correlations with the target value, and widely varying correlations with each other. However, when we attempted to reduce the number of features by eliminating ones with low correlation with the target and ones which had high correlation with another feature, our error rate went up. We suspect that this is because the training data contained a large amount of information, more than could be captured with such a small number of features. Thus, every feature had at least some value. Indeed, one of our greatest challenges was selecting a large set of sufficiently diverse features.

The features we used tended to have a high concentration at the minimum value (usually 0), especially for tracks that a user had not rated. Often, however, a positive value meant that the track was extremely likely to be highly rated by the user. In Figure 3 we show a histogram on a subset of the data for our feature which was most highly correlated with the true value. We suspect that the fact that our features behaved in this way was part of the reason that decision trees were so effective in this problem.

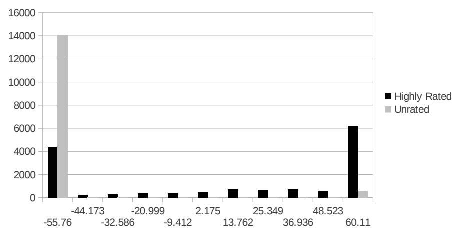

Figure 3: Histograms of the probability (normalized) that ar_j was highly rated, conditioned on it was rated by user u_i

The hierarchical structure information was adequately utilized in the process of feature development. For instance, we built SVD models not only on the $\{u_i,t_j\}$ pairs, but also on all $\{u_i,al_j\}$, $\{u_i,ar_j\}$, and $\{u_i,g_j\}$ pairs, separately. As another instance, we calculated the probabilities that u_i rated al_j, ar_j, and g_j highly and utilized them to predict whether u_i rated t_j highly or not. Readers may notice other instances of how we incorporated the hierarchical structure information in our feature development process when reading the Appendix where we describe the features in more details.

5. MODELING

We tried various supervised learning techniques varying from linear model (logistic regression model) to nonlinear model (artificial neural network, ANN), from single model to committee of multiple experts. Table 5 summarizes the performance of different models we built from the leaderboard after we made the submission. It illustrates how the performance of our recommendation system evolved during the competition.

Table 3: Error rates of different models on Test1

Model	Error Rate (%)
Logistic Regression	6.2
Single ANN	5.09
Committee of 15 ANNs	4.90
Committee of 106 ANNs (106ANNs)	4.80
Gradient Boosted Decision Tree 1 (Tree1)	4.45
Gradient Boosted Decision Tree 2 (Tree2)	4.456
Final Committee of 106ANNs, Tree1 and Tree2	4.376

In this section, we first introduce the logistic regression and ANN models and discuss their performance. Next, we describe the mixture of multiple ANN models built through bagging and show how the performance of the combined models was better than that of a single model. We then describe the stochastic gradient boosted trees and their performance. Finally, we talk about how we built a committee-based prediction system by averaging the outputs of three of our best performing models and how this committee performed on the test set $\mathbf{\Omega}_T$.

Mixture of experts has been shown to be able to boost the performance of the prediction system from a single expert (Bishop, 2006). It has been shown that mixture of experts is able to improve the classification or regression model in terms of stability and classification accuracy. (Polikar, 2006; Hastie et al., 2009; Opitz and Maclin, 1999).

Before describing how we built our prediction system, we first introduce how we produced the outputs of the system. All of our models outputted an estimate of the probability that the given item was highly rated by the user. To convert these numbers into a binary output, for the six tracks of each user in $\hat{\mathbf{\Omega}}_\mathbf{T}$ and $\mathbf{\Omega}_\mathbf{T}$ we labeled the three tracks with the highest probabilities with 1s, meaning highly rated by the user, and the remaining three tracks with 0s, meaning not rated by the user. The error rate of the system is just the total number of mislabeled items divided by the total number of items in $\hat{\mathbf{\Omega}}_\mathbf{T}$ or $\mathbf{\Omega}_\mathbf{T}$.

5.1. Single Logistic Model and ANN Model

Logistic regression was the first model we trained. It assumes a linear relationship between features and the logarithm of the odds ratio between the posterior probabilities of the two classes. This model gave us error rate 6.2% on the leaderboard.

After logistic regression, we trained a ANN model with a single hidden layer and a single output node, in the order to capture complex underlying nonlinear relationships between the features and the target. The target values in the training set were converted to 1 (for highly rated items) or -1 (for unrated items). The optimal number of hidden nodes and learning rate were determined through model performance on the hold out sets. We ended up using 40 hidden nodes and a learning rate of $\lambda = 0.0005$. The activation function was the hyperbolic tangent function. The performance of this ANN model on the leaderboard was 5.09%.

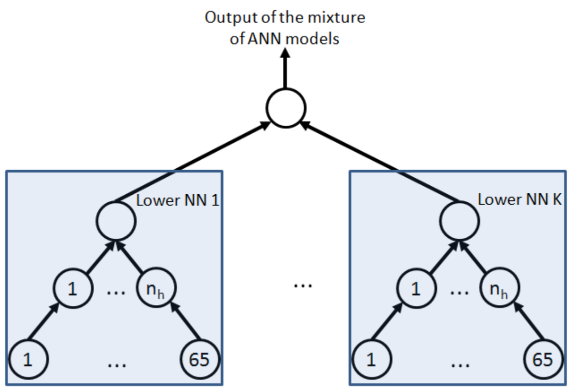

Figure 4: Two-level Neural Network Structure.

The significantly improved performance of the ANN model implied that there did exist complex nonlinear relationships between the features and the targets that was not captured by the logistic model.

5.2. Mixture of Multiple ANN Models through Bagging

We implemented bagging when building a mixture of multiple ANN models. Bagging (bootstrap aggregating) is a widely employed sampling technology in building a mixture of experts (Breiman, 1996). Each expert is trained on a random sample of the full training set. From set $\mathbf{D_L}$, for each sample we randomly selected 70% of the users and put all 6 records of the selected users into the random sample. The reason that we sampled on users instead of on {user,track} pairs was to ensure that in the sampled data, we had the complete six records of each user.

The outputs from all these ANN models were further input to a top-level ANN model, whose output was the final output of the mixture of ANN system. This top-level ANN model has no hidden layer to avoid overfitting the data. The learning rate was 0.0005. The structure of this two-level mixture of ANN models is depicted in Figure 4.

In order to train the weights of the top-level ANN model, we had to ensure that the inputs to it were of the same dataset. So, we passed the whole dataset $\mathbf{D_L}$ through all lower-level ANNs and trained the weights of the top-level ANN model on the outputs of lower-level ANN models. After the top-level ANN model training was completed, the whole two-level ANN system was ready to predict entries in the test set $\mathbf{D_T}$.

Because each lower level ANN model only utilized 70% of the data in $\mathbf{D_L}$, their error rates on $\mathbf{D_T}$ was around 5.61%, worse than the model trained on the whole $\mathbf{D_L}$. However, the two-level ANN system had error rate 4.90% on $\mathbf{D_T}$, even when only 15 lower level ANN models were recruited. It showed that mixture of multiple ANN models through bagging enhanced the performance of the whole prediction system. Encouraged by this, we built 106 lower-level ANN models, using various subsets of the features and data and various numbers of hidden nodes between 40 and 66, and the error rate of the whole two-level ANN

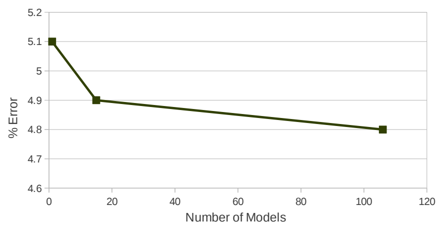

Figure 5: Error on Combinations of Neural Nets.

system on $\mathbf{D_T}$ was further reduced to 4.8%, as depicted in Figure 5. We stopped building more lower-level ANN models due to time limitations.

5.3. Gradient Boosted Decision Trees

The stochastic gradient boosting algorithm (Friedman, 2002) is one of the widely used boosting algorithms. It has been used well in various domains and in data mining competitions (Xie et al., 2009).

We trained a stochastic gradient boosting model using a decision tree as the base classifier. The additive tree models are constructed sequentially to minimize the exponential loss function. The subsequent stage classifiers are trained on examples which were found to be difficult or hard to classify by the previous stage classifier. We started building the best gradient boosted tree models by adjusting the training parameters. The factors we tweaked were the learning rate, the number of trees, and the depth of the tree. The parameters which worked for us were learning rate = 0.06, number of trees = 800 and depth of the tree = 13. Model performance was evaluated on a validation set. We also used the feedback from the leaderboard as a reference. We observed that the validation feedback and the feedback from the leaderboard were not always consistent. There were a few instances where the model did better on the validation data than on the leaderboard. We hypothesized that this could be happening because the some of the highly rated and non-rated items were near the decision boundary. The best performing model gave us the error rate of 4.45% on the leaderboard.

We trained another gradient boosted tree model using the data from the training set where we kept those users and items which weren't classified correctly in the hold out set by the our best performing gradient boosted tree model. This model gave an error rate of 4.456% on the leaderboard. While the leaderboard performance was similar to the early boosted tree model performance, the two models differed on about 0.5% of the records. This further strengthened our belief that some highly rated items and non-rated items were close to the decision boundary.

5.4. Committee-based Prediction System

So far, we have described three different models for classification with satisfactory performances. We decided to mix them together through weighted voting to derive the final prediction on the test set. The weights on these three models were selected by trial-and-error, i.e., we submitted the results on the test set from different combinations of weights to the competition website and picked up the one with the lowest error rate as our final weight vector. The weights that we ended up selecting were proportional to the exponential of the negative error. The final weights on the three models are 26% on the ensemble of ANNs, and 37% on each of the decision trees. The ensemble of ANNs has lower weight than the two decision trees since it has higher error rate on the test data.

6. Conclusion and Discussion

In this paper, we described the committee-based prediction system for recommendation we built to address Track2 task of the KDD Cup 2011 competition. This is a weighted voting system of three classifiers: one ensemble of ANN models, and two decision trees. All three classifiers were built on the feature datasets we extracted from the user's rating history. The committee-based prediction system had error rate 4.376% on the test set, which placed our team seventh on the leaderboard of the competition website.

A crucial part of our prediction system for recommendation is feature extraction. We developed a variety of features to describe various aspects of the data, and to utilize various aspects of the hierarchical structure among items. However, although we were able to use the hierarchical structure in our features in many different ways, we still hope for a more comprehensive and uniform way of utilizing the hierarchical information such that the prediction system can be further enhanced.

Acknowledgments

We would like to acknowledge the contest organizers for organizing such an exciting, inspiring, and fruitful competition. We would also like to thank the analysts and researchers at Opera Solutions LLC. for providing very insightful and constructive advice.

References

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

Leo Breiman. Bagging predictors. *Mach. Learn.*, 24:123–140, August 1996. ISSN 0885-6125. doi: 10.1023/A:1018054314350.

Jerome H. Friedman. Stochastic gradient boosting. *Comput. Stat. Data Anal.*, 38:367–378, February 2002. ISSN 0167-9473. doi: 10.1016/S0167-9473(01)00065-2. URL http://portal.acm.org/citation.cfm?id=635939.635941.

Dan Greening. Building consumer trust with accurate product recommendations. *LikeMinds White Paper*, LMWSWP-210-6966, 1997.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition.* Springer Series in Statistics. Springer, 2nd ed. 2009. corr. 3rd printing 5th printing. edition, September 2009. ISBN 0387848576. URL http://www-stat.stanford.edu/~{}tibs/ElemStatLearn/main.html.

R.A. Johnson and D.W. Wichern. *Applied multivariate statistical analysis.* Prentice Hall, 2002. ISBN 9780131219731.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval.* Cambridge University Press, 2008. ISBN 978-0-521-86571-5.

David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.7341.

R. Polikar. Ensemble Based Systems in Decision Making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, 2006.

Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: an open architecture for collaborative filtering of netnews. In *CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186, New York, NY, USA, 1994. ACM. ISBN 0-89791-689-1. doi: http://doi.acm.org/10.1145/192844.192905.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM. ISBN 1-58113-348-0. doi: http://doi.acm.org/10.1145/371920.372071. URL http://doi.acm.org/10.1145/371920.372071.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth International Conference on Computer and Information Science*, pages 27–28, 2002.

Upendra Shardanand and Patti Maes. Social information filtering: Algorithms for automating "word of mouth". In *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, volume 1, pages 210–217, 1995.

Manolis G. Vozalis and Konstantinos G. Margaritis. Using svd and demographic data for the enhancement of generalized collaborative filtering. *Inf. Sci.*, 177(15):3017–3037, 2007.

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. *Journal of Machine Learning Research*, 7:35–43, 2009.

Appendix A. Detailed Description of Features

In addition to the brief description of the 47 features based on statistics in Section 4, we provide more detailed information for each of them here.

Features f_1-f_3: These three features are the relative frequencies of the parent album, artist, and genres of track t_j being rated by user u_i in the referred dictionary. Equation 5 gives the mathematical definition of f_1.

$$f_1 = \frac{\texttt{\# of ratings on } al_j \texttt{ by user } u_i}{\texttt{\# of ratings on albums by user } u_i} \tag{5}$$

Substituting ar_j for al_j, and "artists" for "albums" in Equation 5, we get the mathematical definition of f_2. Since a single track may have multiple parent genres, f_3 is the summation of the relative frequencies over genres g_{j1}, ..., g_{jk}. The mathematical definition of f_3 is:

$$f_3 = \frac{\sum_{l=1,2,...,k} \texttt{\# of ratings on } g_{jl} \texttt{ by } u_i}{\texttt{\# of genres rated by } u_i} \tag{6}$$

Features f_4-f_6: These are similar to $f_1 - f_3$, but with "rated" in the nominator replaced by "highly rated".

Features f_7-f_9: The relative frequency that the user had rated the corresponding album, artist, or genres highly in the referred dictionaries, under the condition that the user rated the album, artist, or genres before. From this definition, it is clear that:

$$f_{6+i} = \begin{cases} \frac{f_{3+i}}{f_i} & \texttt{if } f_i > 0 \\ 0 & \texttt{if } f_i = 0 \end{cases}, i = 1, 2, 3. \tag{7}$$

Features f_{10}-f_{14}: These five features describe how the general population responded to track t_j. Features f_{10} and f_{11} are the relative frequencies that the general population rated t_j and rated t_j highly, respectively. Feature f_{12} is the relative frequency that t_j was rated highly under the condition that it was rated by the general population. Feature f_{13} is the average rating of t_j over the general population, and feature f_{14} is the total number of ratings t_j received from the general population. The mathematical equations for features $f_{10} - f_{12}$ are:

$$\begin{aligned} f_{10} &= \frac{\texttt{\# of ratings on } t_j}{\texttt{\# of ratings on tracks}} \\ f_{11} &= \frac{\texttt{\# of high ratings on } t_j}{\texttt{\# of ratings on tracks}} \\ f_{12} &= \frac{\texttt{\# of high ratings on } t_j}{\texttt{\# of ratings on } t_j} = \frac{f_{11}}{f_{10}} \end{aligned} \tag{8}$$

Features f_{15}-f_{19} *and* f_{20}-f_{24}: These two sets of features are similar to features f_{10}-f_{14}, except they are on t_j's parent album al_j and artist ar_j.

Features f_{25}-f_{29}: This set of features are similar to features f_{10}-f_{14}, except they are on t_j's parent genres g_{j1}, g_{j2}, ..., g_{jk}. Since t_j may have multiple parent genres, we specifically give the mathematical equations of f_{25}-f_{27} as follows:

$$f_{25} = \frac{\sum_{l=1,2,\ldots,k} \texttt{\# of ratings on } g_{jl}}{\texttt{\# of ratings on genres}}$$

$$f_{26} = \frac{\sum_{l=1,2,\ldots,k} \texttt{\# of high ratings on } g_{jl}}{\texttt{\# of ratings on tracks}} \tag{9}$$

$$f_{27} = \frac{\sum_{l=1,2,\ldots,k} \texttt{\# of high ratings on } g_{jl}}{\sum_{l=1,2,\ldots,k} \texttt{\# of ratings on } g_{jl}} = \frac{f_{26}}{f_{25}}$$

Features f_{30}-f_{32}: These three features describe the rating history of user u_i on the child tracks of album al_j. Features f_{30} and f_{31} are the relative frequencies that user u_i rated and rated highly on the child tracks of al_j in the referred dictionary, respectively. Feature f_{42} is the average rating user u_i placed on tracks of al_j.

Features f_{33}-f_{35}: These three features are similar to f_{30}-f_{32}, except they calculate the statistics of child tracks of artist ar_j.

Features f_{36}-f_{38}: These three features are similar to f_{30}-f_{32}, except they are the statistics of child albums of artist ar_j.

Features f_{39}-f_{41}: Similar to f_{30}-f_{32}. They are the statistics of child tracks of genres g_{j1}, ..., g_{jk}. Since t_j may have multiple parent genres, the mathematical equations for f_{39}-f_{41} are as follows:

$$f_{39} = \frac{\sum_{l=1,2,\ldots,k} \texttt{\# of rated child tracks of } g_{jl} \texttt{ by } u_i}{\sum_{l=1,2,\ldots,k} \texttt{\# of child tracks of } g_{jl}}$$

$$f_{40} = \frac{\sum_{l=1,2,\ldots,k} \texttt{\# of highly rated child tracks of } g_{jl} \texttt{ by } u_i}{\sum_{l=1,2,\ldots,k} \texttt{\# of child tracks of } g_{jl}} \tag{10}$$

$$f_{41} = \frac{\sum_{l=1,2,\ldots,k} \texttt{ratings on child tracks of } g_{jl} \texttt{ by } u_i}{\sum_{l=1,2,\ldots,k} \texttt{\# of rated child tracks of } g_{jl}}$$

Features f_{42}-f_{44}: Similar to f_{39}-f_{41}, just substitute "albums" for "tracks".

Features f_{45}-f_{47}: Similar to f_{39}-f_{41}, just substitute "artists" for "tracks".