Journal of Machine Learning Research 18:199–213, 2012 Proceedings of KDD-Cup 2011 competition

Combining Predictors for Recommending Music: the False Positives' approach to KDD Cup track 2

Suhrid Balakrishnan SUHRID@RESEARCH.ATT.COM
Rensheng Wang RW218J@RESEARCH.ATT.COM
Carlos Scheidegger CSCHEID@RESEARCH.ATT.COM
Angus MacLellan ANGUSM@RESEARCH.ATT.COM
Yifan Hu YIFANHU@RESEARCH.ATT.COM
Aaron Archer AARCHER@RESEARCH.ATT.COM
Shankar Krishnan KRISHNAS@RESEARCH.ATT.COM
David Applegate DAVID@RESEARCH.ATT.COM
Guang Qin Ma GMA@RESEARCH.ATT.COM
S. Tom Au SAU@RESEARCH.ATT.COM
180 Park Ave, Florham Park, NJ, 07932

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

Abstract

We describe our solution for the KDD Cup 2011 track 2 challenge. Our solution relies heavily on ensembling together diverse individual models for the prediction task, and achieved a final leaderboard/Test 1 misclassification rate of 3.8863%. This paper provides details on both the modeling and ensemble creation steps.

1. Introduction

The 2011 KDD Cup featured two music recommendation tracks. This paper will focus on Track 2, where the goal was to design methods to predict whether a user would rate a particular track highly (she likes this track, or a binary label of **1**) or likely not rate this track at all (label **0**). The dataset provided for this challenge was a large partially observed/sparse ratings matrix $\boldsymbol{R}$ (in a 0-100 scale) on a set of users, $\mathcal{U}$, and items, $\mathcal{I}$. The items rated in the training set are music tracks, artists, albums and genres.

Additionally, a reasonably complete taxonomy was provided, which described relations between items. There were two different (overlapping) sources of taxonomy information, one that was track-centric, namely one file which gave relations between tracks and other items (what album a track belongs to, which artist the track is by etc.) and one that was album-centric (in addition to the artist information, which additional genres an album belongs to etc.). Figure 1 schematically depicts these two sources of taxonomy information.

Predictions were solicited for 6 tracks per user, and there were 101,172 users in the test dataset, which results in 607,032 test user-item pairs (there are a large number of "extra" users and items that only appear in the training dataset). Of these test user-item pairs, the contest rules stated that "The three items rated highly by the user were chosen randomly from the user's highly rated items, without considering rating time. The three test items

© 2012 S. Balakrishnan *et al.*

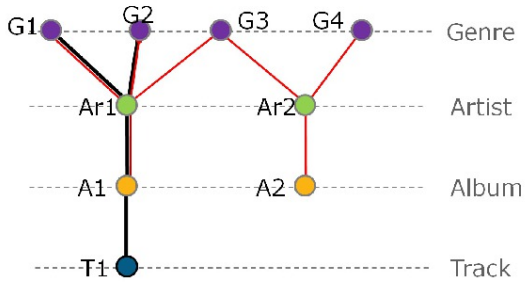

Figure 1: Taxonomy information for items as a graph (best viewed in color). The track-centric information is shown with black edges and album-centric information with red edges. Colors/levels distinguish the different types of items (tracks, albums etc.).

not rated by the user are picked at random with probability proportional to their odds to receive high (80 or higher) ratings in the overall population.". Thus, as set up, this is a binary classification task, with additional known structure in the output, namely that of the 6 items to be rated per user, exactly 3 have label **1** and the remaining three have label **0**.

The evaluation metric was **0/1** misclassification rate (percentage error) and in the style of the Netflix Prize the organizers reported continually updated results on a leaderboard for half of the predictions named Test 1. All the misclassification error rates we report in this paper are based on the data in Test 1 (the other half, "Test 2" was used for the final evaluation). To limit the number of submissions a team could make, submissions were throttled to a rate of one every 8 hours.

2. Our Approach

Our overall approach to the task was to ensemble as many different component strategies as we could reasonably envision and implement. Ensemble-based solutions have performed well in many contests (e.g. the Netflix Prize, many previous KDD Cups and the Yahoo! Learning to Rank Challenge), drawing their strength from excellent generalization performance, easily outperforming any single component model (Jahrer et al., 2010).

Our component models included matrix factorization methods, different classification approaches trained on taxonomy-based features, a large-scale k-nearest neighbor approach and ranking techniques. We also experimented with a few different ensemble techniques. The next sections in the paper delve into details of both these methods and ensembling strategies.

3. Matrix Factorization

Our first approach was to tackle the problem as a modified collaborative filtering task, that is, to use the training data to learn models that predict ratings for all test user item

pairs. In order to get the binary test predictions, we sort the six predicted ratings per user, labeling the top three as **1**s and the bottom three **0**s. Our underlying assumption is that a high value of predicted rating for a track is a strong indication of a preference for that track; recall that the **1** ratings all have ratings values ≥ 80. It is less clear, however, that a low predicted rating value is an indication that the user would not have rated that track at all. We attempt to address this "rated or not" issue by modifying the training data as we explain below.

The advantage of the rating prediction approach is that we can draw from many recently developed methods, such as PMF, BPMF and SVD++, by Salakhutdinov and Mnih (2007, 2008); Koren (2008). One big disadvantage is that these methods have not been developed specifically for the binary prediction task that we are given. Another disadvantage is that in general, these methods do not use the available item taxonomy.

Most of the models we experimented with for rating prediction were based on matrix factorization, mainly due to their scalability and good reported performance on prior ratings prediction tasks like the Netflix prize developed by Koren (2008); Töscher et al. (2009). All of these matrix factorization models estimate k dimensional factors (or parameter vectors) for both users, $j \in \mathcal{U}$ and the items, $i \in \mathcal{I}$. Denoting the user k-vector by $\mathbf{u}_j$ and the item factors by $\mathbf{v}_i$, predictions for the user-item pair i, j are made using:

$$\hat{R}_{ij} = \mathbf{v}_i^{\mathrm{T}} \mathbf{u}_j.$$

The differences in the various techniques we apply are based on one of three criteria. The first criterion is the form of the loss used for training. Most techniques use squared loss between predictions and true ratings. The second criterion is the form of regularization the technique uses. Finally the third criterion is the form of data used for training. One could use all the training data including album and artist items, or just use the tracks. There are choices having to do with whether we treat missing values as **0** or not. Further, there are questions about how to use the taxonomy. We settled on five main forms of the data, each emphasizing a slightly different aspect of information about our final binary classification problem:

A Full - All the provided ratings, on all items and all users. This dataset leverages all the information from tracks/users other than the test tracks/users. On the negative side, this dataset is not optimized for the binary prediction task, or the test tracks and users.

B Full Balanced - The Full dataset A, with additional sampled **0** track ratings (we sample **0** rated tracks in the same way as described in the track 2 problem description). This dataset also borrows strength across non-test users and tracks, but additionally tries to move toward better solutions for the binary prediction task by adding in sampled **0** rated tracks.

C Balanced - This dataset consisted of a subset of A, comprising just high ratings (over all items), and an equal number of sampled zeros. This dataset is smaller than B, and moves solutions even more towards our given task, by only considering high ratings.

D Binary - which is a binary version of the high item rating data. If an item is rated highly, it is present as a **1** in this dataset, and **0** otherwise. This kind of binary rated or not information was found useful in the Netflix Prize.

E Augmented - where we used A. and augmented this with further imputed ratings from the taxonomy (more details in section 3.6). The aim of this dataset was to regularize solutions using information from the taxonomy.

While training, we monitored progress using a validation dataset that we created of the same size as the test dataset. We sample **1** s from the test users high track ratings, and additionally sample an equal number of zeros. In our training datasets, we exclude these validation samples except for the case of about a hundred test users, who happen to have only three high track ratings. In these cases, we included the high ratings in both our validation and training datasets. Although this introduces a small amount of overfitting for these users, it makes the subsequent prediction task straightforward, as opposed to having to perform additional inference for users not present during training (cold-start).

3.1. Probabilistic Matrix Factorization

The main matrix factorization technique we used is probabilistic matrix factorization (PMF), by Salakhutdinov and Mnih (2007). Specifically, we used the PMF implementation from Graphlab, mainly because of Graphlab's automatic parallelization, as described by Low et al. (2010). PMF uses squared loss, and regularizes the factors in an ℓ_2 manner.

We trained PMF models on most forms of data (A, B and C) above. In general we found PMF to be quite resilient to overfitting (we have found that more factors always translated to better performance) and we trained many models with $k = 400$ factors. Our best model in this category was based on dataset C, and had test misclassification error around 8.4%.

3.2. Weighted PMF

One of the drawbacks of the standard PMF model in our setting is that it does not distinguish between tracks and other items like artists, albums etc. Since our test set is composed solely of tracks, we felt it would be beneficial to bias our models towards getting their ratings predictions correct. We implemented this idea by training a model on dataset B with different costs for various items. In particular, we trained using a weighted loss, which per item is of the form:

$$\mathcal{L}_{ij} = w_i(R_{ij} - \mathbf{v}_i^{\mathrm{T}}\mathbf{u}_j)^2.$$

We set relatively high weights (w_i) for highly rated and sampled **0** tracks. We also set low weights on non track ratings as well as low track ratings. Our best weighted PMF model achieved a test error of 8.6%.

3.3. Implicit Feedback

One other matrix factorization approach was to view the task as one of item prediction instead of rating prediction, as Hu et al. (2008). In this approach, the training data consists of only the set of highly rated items per user (dataset D). The model then tries to learn item and user factors that minimize an objective function on this (entire) binary data. The

loss is weighted, and emphasizes predicting the **1** s, but also assigns a tunable low weight to predicting the **0** s. We used the implementation in MyMediaLite [1]. Our best models in this class achieved around 11.2% misclassification rate.

3.4. CofiRank

Another form of matrix factorization that we used was CofiRank, by Weimer et al. (2007). CofiRank uses a maximum margin matrix factorization setup (a trace norm regularization on the user and item factors). Because it uses bundle methods to relax difficult objective functions, it can be trained on complicated losses. In particular, we were interested in normalized discounted cumulative gain (NDCG), a ranking loss. This seemed a more natural fit with the problem, since individual ratings are not as important as the relative ordering for the six items per test user. Notice that using this NDCG loss does not address the ambiguity of low rating values. Our experience with CofiRank was mixed, in that we were not able to get very large models to train in reasonable amounts of time on dataset A, B and C, but the models we did train seemed quite good. Our best model achieved 18.2% misclassification error.

3.5. Pure SVD

We also experimented with (sparse) singular value decomposition, which produce orthonormal factors. Our motivation was recent work by Cremonesi et al. (2010) who found sparse SVD to be competitive at predicting the top-k item recommendations for any user. Once again, since this is more close to ranking this appeared to be a reasonable method to attempt. For this set of models, we mainly used dataset B, and our best Pure SVD model ($k = 500$) scored around 15.6%.

3.6. Graph-based Data Augmentation

Because of the rich taxonomy information in the data set, we considered using this information in training the factor model. We chose to do this by augmenting the ratings with additional imputed ratings data obtained by smoothing the taxonomy information via graph diffusion.

First, we construct a taxonomy graph, treating each item as a node. Two nodes have an edge between them if the relationship between these two nodes is in the taxonomy. For example, if node i_1 is a track, and node i_2 is an album, and that we know that track i_1 is part of album i_2, then there is an edge between i_1 and i_2 in our graph. This gives a graph with 291,858 nodes and 460,543 edges. In forming this graph, we ignored all genre related information. This graph has many connected components, the largest having 168,624 nodes, followed by 802, 700, 447, 434, and 392 nodes.

Given this taxonomy graph, for each user, only a small fraction of the nodes are rated. We impute the rating for nodes that are not rated by the user, but are close (in graph distance) to the rated nodes in the graph. Starting from a rated node, the unrated neighbors of this node are given the average rating of their rated neighbors. This diffusion of rating

1. http://www.ismll.uni-hildesheim.de/mymedialite/index.html

is repeated in a breadth-first fashion for all nodes in the graph. The result is additional imputed user-item ratings.

We point out that although the taxonomy graph is sparse, it is still well connected, with the largest component having a graph diameter of 19. Thus a breadth-first diffusion quickly assigns ratings to all items. To avoid imputing too many ratings, resulting in a very dense rating matrix, we limit the diffusion to items that are at most 2 hops away from rated items. Furthermore, since we are only interested in items that are highly rated, the diffusion is only carried out for ratings over 80.

We augment the training data with these imputed ratings to create dataset E. We then train an implicit feedback style matrix factorization model (Hu et al., 2008) on this dataset. Our highest scoring factor model derived from this graphical approach gives a misclassification error of 30.8%. We ran out of time before we could attempt to run other matrix factorization models such as PMF on this augmented dataset.

4. Taxonomy-based Classification

Our next approach was almost completely taxonomy-driven and based on classification. In summary, we first create a validation dataset using the training ratings and sampled **0**tracks mimicking the test dataset (thus all items in this dataset are tracks). Next, using the taxonomy and the training ratings $\boldsymbol{R}$, we generate features $\mathbf{x}_z$ for each of the user-item pairs chosen in this dataset, $z = 1 \ldots 607,032$ (these features are also generated for the test dataset). This results in a classification dataset, $\mathcal{D}_{tx}$ (features and their labels) on which we can run various classifiers to train predictive models. At test time we use the classifier predictions/scores $\hat{\mathbf{y}}_{test}$ to create a valid submission per user, by sorting appropriately, and setting exactly three out of the six item predictions to be **1**.

More specifically, we create features for each user-item (i, j) pair by using the track-centric taxonomy information. Recall that the track-centric taxonomy information associates each track with its album, an artist and one or more genres (See Figure 1). As a pre-processing step for feature creation, for a given user j, we first extract from $\boldsymbol{R}$ her ratings on all items—call this vector of ratings $\boldsymbol{R}_j$. We create four sets of features, one set for each of related tracks, albums, artists and genres.

Our first set of features is based on related tracks via album information. Given a track i, we use the taxonomy to find all related tracks that are in the same album. The lowest level of Figure 2 shows these related tracks schematically. We then extract all existing related track ratings from the users known ratings $\boldsymbol{R}_j$ (we do not impute missing values). Now, we create four features from the vector of related track ratings by user j—the maximum, minimum, average and number of ratings in this vector.

Similarly, we generate four features based on related albums. We do this by extracting the same four features using the related albums by the same artist and $\boldsymbol{R}_j$ (see the album level of Figure 2 for a schematic of the related albums). Finally, since there may be more than one genre associated with any track, we also generate two sets of four features. The first set of features comes from related artists from the union of the set of genres. The second set of features comes directly from the union of the set of genres.

To these 16 features, we add the album and the artist rating if it exists in $\boldsymbol{R}_j$. These 18 features for each user-item pair $\mathbf{x}$, and their associated label y comprise our classification

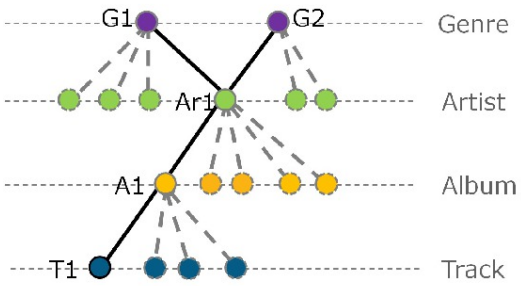

Figure 2: Schematic illustrating related items using the track-centric taxonomy. For a given track (the track node with the solid outline), the lowest level shows related tracks, and higher levels show related albums, artists and genres.

dataset. We correspondingly also create the same 18 features for all test user-item pairs as well. Intuitively, we hoped the features would be informative in at least two ways. First, if a user likes a particular track she would also like related tracks/albums/artists, and this would show up as high value in our features. Second, if a track was sampled (not rated by the user), we also hope to capture this by seeing a lot of missing values in the features we create. In fact, these observations were borne out by a large fraction of data.

We experimented with various classification models based on this data (and made predictions for the test data). We tried random forests as described by Breiman (2001), generalized linear models (binomial link), and support vector machines[2]. We also experimented with various ways to handle missing data, and found that imputing a single value for all missing entries (found via cross-validation) increased classification performance by a small amount. Random forests (200 trees) gave us the best performance of models in this class, with an error rate close to 9.0%. We further improved models by extending the feature sets using user-neighbors (more on these neighbors in section 5), global features for items (features derived from all training users). In our final evaluations, our best model in this class had an error rate around 5.7%.

5. kNN

We also tried a user-based k-nearest neighbors (kNN) approach to the problem. At test time, this requires knowing a 101,172 x 249,012 user similarity matrix, and estimating this matrix required distributed computation. We subdivided the data and performed the computations on a cluster. The test users $\mathcal{U}_{test}$, were partitioned into 405 groups (set A). For each group, the corresponding rating data was extracted from $\boldsymbol{R}$. Similarly, the training users $\mathcal{U}_{train}$ were partitioned into 499 groups (set B) and their ratings data was also extracted from $\boldsymbol{R}$. To compute the similarity matrix, each user in each group A was compared to each user in each group B, resulting in 405*499 = 202,095 group comparison tasks. The exact subdivision was mostly arbitrary; all we required was sufficiently many tasks so that the amount of memory required for each task was small enough to be manageable on the

2. http://svmlight.joachims.org/

available hardware. The 202,095 comparison tasks were spread over 80 Linux OS machines of various capabilities. We used various tools to manage the execution of this Map/Reduce-style workload, including Ningaui by Hume and Daniels (2002). Each of the 405 similarity matrix files generated took approximately 3.5 GB uncompressed, and 700MB compressed. The entire matrix we generated took around 1.4TB uncompressed, and was computed twice during the KDD competition, each time taking approximately 4 days to compute.

The similarity function we computed for our user pairs was Euclidean distance on the common rating vectors. In other words, in order to compute the similarity between users j_1 and j_2, we first find the set of items which both users have rated (we also store the cardinality of this intersection set for all our user pairs). We then compute the Euclidean distance between the rating vectors restricted to this subspace of common items. Unfortunately, we did not have time to explore using the taxonomy for the distance calculation.

After computing the similarity matrix, our kNN predictions were made for test users who did not rate either the album or the artist (in that order) corresponding to a given track. If the user did rate either of these items, this rating was taken to be the predicted rating for that track.

The kNN prediction for a user j item i is a simple average of the ratings of (some number of neighboring) items. Our kNN procedure is sequential and while making predictions we continually keep track of the size of the number of items being used to make our prediction. The idea here is to use all available neighboring item information, but to weigh track information most heavily and add its predictive component first, and then successively add in lower weighted information from albums, artists and genres respectively as needed.

In more detail, a set of k neighboring users (we require that neighbors must additionally overlap on a minimum number of common ratings, which we arbitrarily set to 30) is examined first for ratings of track i. If we have a sufficient number of these track-neighbor ratings we stop and make a prediction as the average of these ratings. If not, we use the taxonomy to then examine album-neighbor ratings (ratings of neighbors of j for the album corresponding to track i, or moving up one level from tracks to albums in Figure 1). If we have enough track-neighbors and album-neighbors, we make our prediction, otherwise we proceed to artist- and genre-neighbor ratings.

Our best kNN method achieved an error of 19.8%, using $k = 25$, and the weights for tracks, albums, artists and genres were set to 0.75, 0.50, 0.25 and 0.10 respectively.

6. Classification Revisited

In an effort to construct more predictive features for the classification approach described in section 4, we also experimented with using features derived from the matrix factor models (in section 3) in combination with the existing taxonomy based features, $\mathcal{D}_{tx}$. More specifically, for dataset C (see section 3), we generated taxonomy-based features $\mathcal{D}_{tx}$ as in section 4. We then concatenated the feature vectors for user-track pairs with additional features derived from trained matrix factorization models. The matrix factorization features we used were the component-wise product of the user and item factors (the k scalar $\mathbf{v}_{ik}\mathbf{u}_{jk}$ values), and additionally the matrix factorization model predicted rating $\hat{R_{ij}}$. This generated $k + 1$ extra features per user-item example (we used both $k = 100$ and $k = 300$).

This then gave us a new, larger set of training examples and associated binary labels. We then trained the same classifiers on this version of the dataset, which provided our best single model. In particular, we trained a linear SVM (7.9% error), a generalized linear model (6.9% error), random forests by Breiman (2001) (5.7% error), boosted decision trees (8.7% error).

7. Collaborative Ranking

As mentioned in section 3.4, a ranking loss appears a better fit to the problem. Inspired by Joachims (2002) and Burges (2010), in the learning to rank community, we also tried an approach where the loss function was rank-based. In other words, learned a function that would produce a score for each of the six test items such that the three **1**s would be higher scoring than the unrated items. The ranking model we consider generates pairwise training data (per user) for models that learn to respect the **0**/**1** ordering, and is indifferent to ordering within the relevance levels.

In our experiments, we created a dataset based on both matrix factorization models and the taxonomy with six ratings per test user (we used one of our validation datasets). There are two advantages of using a constant, small number of entries per user (instead of using every available high rating). First, we get a balanced training set. Second, the number of training pairs grows quadratically with the number of available **0**/**1** ratings, and we wanted to keep that value low. We then trained an SVMrank model described by Joachims (2002) directly on the binary **0**/**1** label pairs as the two 'relevance' levels. The best models we trained in this class had about 6.8% error.

8. Ensemble Creation

Our individual models were reasonable, but by themselves they would not have been ranked very highly in the contest. We used two main techniques to create our ensemble submissions which leverage the diversity of the individual models.

8.1. Ensemble Selection

Because the loss function on Track 2 is misclassification rate and linear ensembling works best on ℓ_2 loss, we were not originally sure how to ensemble our individual models. Our first attempt used Ensemble Selection (ES) (Caruana et al., 2004).

In summary, ES works by creating a non-negative (linear, in our case) combination of component model outputs, via a greedy forward stepwise selection using a validation set. ES has been shown to perform well with a large library of component models, and on complicated error metrics including misclassification error rate, which made it attractive in this setting. We implemented ES for misclassification loss, using a validation dataset. In order to get valid test submissions (three **1**s and three **0**s per user), we performed ensembling of component models on per-user rankings of the six items being tested ($1 \ldots 6$), instead of their binary predictions. The final result of ES was then rounded (sort result, set top three to be **1**s) to produce a valid submission. This ES strategy worked very well in the initial stages of our effort. Unfortunately, we found that ES was prone to overfitting towards later stages. Our best submitted ES solution achieved 5.94% misclassification.

8.2. Leaderboard Ensembling

On further inspection, it became clear that even though the evaluation metric is misclassification rate, since there were only 2 possible outcomes, the loss matrix is identical to that of squared error on outcomes $\{0, 1\}$. This is interesting because for squared error/MSE, it is possible to find a very good approximation to the linear combination of submissions which globally minimizes the distance to the true solution (Töscher et al., 2009). In addition, the technique we discuss here can be used without resorting to a single validation set. Since some of our strategies involved generating new derived training and validation sets, the ability to effectively combine the results from these different sets is attractive.

The main insight is that the leaderboard reveals a fair amount of information about the sufficient statistics required for an optimal linear combination of test solutions. Following the analysis in Section 7 of Team BigChaos Solution to the Netflix Grand Prize (Töscher et al., 2009), for each of our test submissions, $\mathbf{s}$ (the binary solution vector), we first interpret the misclassification rates returned (the leaderboard or Test 1 misclassification rate) as an equivalent squared error from the true test answer, $\mathbf{s}^*$. The idea is that we can view the ensemble task as a linear regression where the predictors are S (a matrix of all our submissions, stored as columns), and the response is the true test answer $\mathbf{s}^*$. This leads to the familiar linear regression estimate of the parameters: $\boldsymbol{\beta} = (S^T S)^{-1}(S^T \mathbf{s}^*)$. In terms of implementation, we use a standard minimum-norm linear least squares solutions based on the SVD of S. This expression cannot be evaluated exactly of course, because the true test solution $\mathbf{s}^*$ is unknown ($S^T S$ is calculable). However, it can be approximated using the identity: $2\langle \mathbf{a}, \mathbf{b} \rangle = ||\mathbf{a} - \mathbf{b}||^2 - \langle \mathbf{a}, \mathbf{a} \rangle - \langle \mathbf{b}, \mathbf{b} \rangle$ for any two vectors $\mathbf{a}$ and $\mathbf{b}$. Since all valid Track 2 submissions (and the true test submission) have $\langle \mathbf{s}, \mathbf{s} \rangle = 1/2$, and the leaderboard result can be scaled to give us an approximate estimate of $||\mathbf{s} - \mathbf{s}^*||^2$, we can estimate $\mathbf{s}^T \mathbf{s}^*$ per submission. This in turn allows us to estimate the $\boldsymbol{\beta}$ parameters which are linear weights for the submissions.

We make ensemble predictions using $S\boldsymbol{\beta}$, which in general does not produce a valid submission. We project to the space of valid solutions by "rounding": we find the binary 6-vector per user that is closest to the relevant entries in $S\boldsymbol{\beta}$. It is easy to show that assigning $\mathbf{0}$ to the three lowest results and $\mathbf{1}$ to the remaining entries is a global minimizer of the ℓ_2 distance from the unconstrained $S\boldsymbol{\beta}$ solution to the space of plausible solutions. Notice, however, that this method offers no guarantees that the linear combination $S\boldsymbol{\beta}$ itself is close to a valid rounding (see Section 9.2 for an attempt to mitigate this).

We also noticed that a small number of test tracks had no high ratings in the training data. If the training and highly rated tracks from the test data was the only rating information used to create the test dataset, this implies that these tracks could not have been sampled for any test user. Consequently these tracks must have been true $\mathbf{1}$s, and so we force them to be $\mathbf{1}$ in our final submission.

We should also point out that since our ensemble technique was based on leaderboard/Test 1 score, we were concerned about overfitting to the Test 1 dataset. Our estimates for how much we were overfitting (or "optimism", see Section 7.2 in (Töscher et al., 2009)) were negligible compared to the gap between the score of our submission and the leaderboard score just below us. Because of that, our final submission used no regularization. This submission achieves a 3.8863% rate on Test 1, our best result and the one we

Weight	Score	Type	Weight	Score	Type	Weight	Score	Type	Weight	Score	Type
-0.2734	5.95	ES	-0.0062	8.97	MF	0.0035	12.62	ES	0.0206	7.23	LE
-0.1210	6.88	GLM	-0.0056	18.96	MF	0.0036	11.22	MF	0.0226	8.42	ES
-0.1124	4.33	LE	-0.0050	8.44	ILP	0.0036	12.67	MF	0.0229	9.72	ES
-0.0643	3.95	LE	-0.0049	5.31	LE	0.0038	9.56	MF	0.0236	9.59	RF
-0.0550	4.65	LE	-0.0048	9.47	RF	0.0039	18.31	SVD	0.0255	8.48	RF
-0.0516	7.40	ES	-0.0047	6.91	RF	0.0043	8.41	RF	0.0265	9.85	RF
-0.0434	6.06	ES	-0.0038	9.88	MF	0.0043	15.63	MF	0.0282	8.96	ES
-0.0315	9.24	RF	-0.0029	17.77	MF	0.0048	15.95	MF	0.0302	4.30	LE
-0.0315	9.24	GLM	-0.0020	16.87	MF	0.0049	9.49	MF	0.0314	5.23	LE
-0.0310	9.19	RF	-0.0019	11.98	GLM	0.0052	5.97	MF	0.0319	4.73	LE
-0.0300	6.48	RF	-0.0018	13.80	ES	0.0052	5.97	MF	0.0347	10.00	MF
-0.0286	8.75	RF	-0.0014	6.80	GLM	0.0076	15.53	MF	0.0373	9.38	RF
-0.0245	8.51	RF	-0.0014	6.80	GLM	0.0083	12.11	MF	0.0383	6.04	LE
-0.0243	7.14	LE	-0.0011	5.97	RF	0.0083	13.79	MF	0.0383	6.85	GLM
-0.0225	9.26	RF	-0.0011	8.80	RF	0.0092	6.79	LE	0.0407	6.02	GLM
-0.0194	8.99	ES	-0.0008	9.24	RF	0.0097	8.79	ES	0.0410	7.70	RF
-0.0186	9.72	GLM	-0.0006	18.53	MF	0.0102	5.05	LE	0.0411	6.59	LE
-0.0182	20.04	GLM	-0.0003	7.70	LE	0.0108	8.75	MF	0.0420	6.09	RF
-0.0176	14.20	MF	0.0002	13.61	MF	0.0114	9.69	MF	0.0484	6.84	ES
-0.0160	17.70	MF	0.0003	15.98	MF	0.0117	8.64	MF	0.0521	4.28	LE
-0.0159	13.64	ES	0.0004	19.83	COFI	0.0119	6.04	RF	0.0528	6.04	RF
-0.0158	4.84	LE	0.0005	19.70	MF	0.0120	4.96	LE	0.0530	6.04	RF
-0.0143	7.88	LE	0.0011	10.87	ES	0.0129	8.62	ES	0.0622	4.09	LE
-0.0140	7.84	RF	0.0019	4.30	LE	0.0147	12.38	MF	0.0653	5.72	LE
-0.0113	9.15	ES	0.0020	8.66	MF	0.0149	11.81	MF	0.0695	5.11	LE
-0.0106	7.72	RF	0.0022	13.04	ES	0.0156	8.85	MF	0.0776	4.28	LE
-0.0103	7.59	ES	0.0026	10.38	ES	0.0176	5.97	RF	0.1387	5.80	RF
-0.0092	9.65	VW	0.0026	13.56	ES	0.0179	8.68	MF	0.1521	5.94	VW
-0.0091	12.18	ES	0.0027	8.57	MF	0.0201	4.72	LE	0.1621	9.94	LE
-0.0073	7.59	ES	0.0029	10.36	ES	0.0204	7.42	ES	0.1880	5.76	RF
									0.1946	3.95	LE

Table 1: Weights and Test 1 Scores (misclassification error percentage) of all submissions from Team False Positives included in the final ensembling. The submissions were weighted and the result rounded to **0**/ **1** as described in the text. Legend: RF = Random Forests (Section 4); LE = Leaderboard Ensembling; ES = Ensemble Selection; GLM = Generalized Linear Model (Section 4); VW = L1 regression using Vowpal Wabbit (Section 4); COFI = CofiRank; SVD = Full sparse SVD.

used for the final ranking among the teams. The set of of weights and the misclassification percentages of each component model used in our final ensemble solution is shown in Table 1. We note that this submission includes previous linear ensembles. This might seem confusing, since it can never help a further linear ensemble to include a previous linear ensemble (since it doesn't change the solution span). Our submissions, however, are the *rounded* linear ensemble, and those do augment the solution span.

9. Ideas To Explore

For completeness, we include in this section a few ideas which we consider interesting in their own right, but did not have time to finish.

9.1. Regression for user label vectors

All the techniques we implemented in the spirit of a general recommendation task, that is, they are all models that can predict scores for arbitrary user-item pairs. However, in the specific case for KDD Cup's track 2, it should be possible to take advantage of the fact that we only needed to predict one of $\binom{6}{3} = 20$ specific binary 6-vectors for each user (call these 6 **0/1** values $\mathbf{y}_j$ for user j). If we could learn a regression model that would take as input a candidate binary 6-vector for a user, $\hat{\mathbf{y}}_j$, and some features $\mathbf{x}_j$ and output a score (low for a 6-vector that is closer to the true answer, w.l.o.g.), then at test time, one could simply generate all 20 possible label vectors, score all of them using the learned model and return the lowest scoring 6-vector.

Here is how we envisioned constructing the features $\mathbf{x}_j$ for this regression task. First, we generate user-item features that are predictive in the classification setting, say for example the taxonomy-based features from section 4. Call these features component features. Then, given a candidate label 6-vector $\hat{\mathbf{y}}_j$, with corresponding component features for the 6 items, consider a single derived feature that is the sum of the component features over only the **1**labeled items. Another derived feature could be the maximum value of a component feature over only the **1**items. One can easily derive a few other such features as well. The observation is that we would expect well-behaved component features (features that are roughly monotonic to the classification label) for the binary classification task to be well behaved for this regression task as well.

A reasonable response value (the regression target) for a 6 item pair with proposed labels, would be for example $2^{|\hat{\mathbf{y}}_j - \mathbf{y}_j^t|}$. Thus, the target would be one if the proposed label matched the true label vector $\mathbf{y}_j^t$, and larger the more errors there were. There are some subtleties involved, in particular, naively creating all 20 possible label vectors would lead to very imbalanced training data. We thought to address this issue by always including in our training set the completely correct and completely incorrect candidate label vectors and further sampling some small number of other incorrect label vectors.

9.2. Ensembles via randomized rounding

The structure of valid test predictions also led us to question whether an ensemble technique could be crafted to take advantage of it.

Suppose that the true test solution is $\mathbf{s}^* \in \mathbb{R}^n$. Let s^i, $i \in [m]$ be the solution vectors we wish to ensemble (where $[m] := \{1, \ldots, m\}$). Let e_i denote the (Test 1) misclassification error of a given solution s^i. This implies $\mathbf{s}^*$ lies on the sphere of Hamming distance ne_i centered at s^i. Since every user has exactly three **1**s and three **0**s, this is equivalent to $(s^i)^T\mathbf{s}^* = \frac{1}{2}n(1 - e_i) =: b_i$. Letting S be the $n \times m$ matrix whose i^{th} column is s^i, we get $S^T\mathbf{s}^* = b$. This motivates us to search for a solution $\hat{x}$ such that $S^T\hat{x} \approx b$.

The basic idea is to find a fractional solution vector x in the affine span of the given solution vectors, then randomly round each user independently to obtain a solution $\hat{x}$. By requiring x to lie in the affine span, we ensure that its weight is 3 for each user. Because we will use x_i as a probability, we must ensure $0 \leq x \leq 1$. Moreover, the 6 components of x corresponding to a single user must be rounded in a dependent fashion, to preserve their sum. We design both our computation of x and our rounding scheme to yield $S^T\hat{x} \approx b$.

We propose two methods to compute x. Both require x to lie in the affine span of the s^i. The first method explicitly requires $0 \leq x \leq 1$. The second requires x to lie in the convex hull of the s^i, which already implies $0 \leq x \leq 1$. In both cases, we minimize $||S^T x - b||_2^2$. To perform these optimizations, we introduce a vector of weights $w \in \mathbb{R}^m$ and express x as Sw, so the objective is convex quadratic in w. In both cases, we include the constraint $\sum_i w_i = 1$ to place x in the affine span. In the second case, we further constrain $w \geq 0$, which restricts to the convex hull. In the first case, we first solve the relaxed problem with the $0 \leq x \leq 1$ bounds omitted, add in the most violated of these constraints, and iterate. In our experience, this constraint generation approach converges rapidly to a solution. Both methods require computing $S^T S$, i.e., all pairwise dot products of the s^i, and in both cases this dominates the computation time.

We now discuss how to round the fractional vector x to a valid integer solution $\hat{x}$. Our aim is to keep $\hat{x}$ close to the affine span of the s^i, while keeping $S^T\hat{x} \approx S^T x$. We achieve both aims by rounding so as to preserve expectations, i.e., $\mathrm{E}[\hat{x}_i] = x_i$ for all i, while making the $\hat{x}_i$ negatively correlated. Negative correlation is desirable because it implies Chernoff-like concentration bounds for every linear combination of the $\hat{x}_i$(Panconesi and Srinivasan, 1997; Gandhi et al., 2006). In particular, this will ensure not only that $S^T\hat{x}$ is concentrated around $S^T x$, but also that $\hat{x}$ is concentrated near the affine span of the s^i.

Let x^u and $\hat{x}^u$ denote the restrictions x and $\hat{x}$ to the 6 items that must be labeled for user u, and let subscripts $i \in [6]$ denote the 6 components for this user. Similarly, define $s_i^u \in \mathbb{R}^6$ to be the vector of labels for user u in solution i, and $S^u \in \mathbb{R}^{6\times m}$ to be the matrix whose columns are s_i^u. Since we will round each user independently of the others, we just need to construct the probability distribution for a single user.

For each subset $I \in \binom{[6]}{3}$ (the collection of 3-element subsets of [6]), let p_I denote the probability that we set $\hat{x}_i^u = 1$ for exactly the items $i \in I$. We determine these p_I by solving a linear program, with linear constraints to preserve the expectations and ensure negative correlation in the $\hat{x}_i$s. A feasible solution to this program guarantees concentration bounds. However, we still have a flexible objective function, which we use to minimize the variance of $(S^u)^T\hat{x}^u$, which objective turns out to be linear in the p_I variables.

This linear program for a single user has just 20 variables and 36 constraints, and is hence trivial to solve. The p_I are bounded above and below by the negative correlation constraints, there are 6 equality constraints to preserve expectations, and $2\binom{6}{2} = 30$ of the negative correlation constraints.

In practice, this procedure achieved an ensembling within the distance constraints of the candidate solutions. Unfortunately, the ensemble itself did not perform better than the individual solutions, the reasons for which we are still investigating.

10. Conclusions

We described our approach to KDD Cup 2011, track 2. In line with reported results Jahrer et al. (2010) found that fairly straightforward but diverse techniques ensemble together very well, giving an effective final score. For discussion, we also outlined a few techniques that we unfortunately did not get a chance to evaluate, but we believe take full advantage of the particular structure of the output of track 2. We thank the organizers and other participants for an exciting contest!

11. Acknowledgments

We would like to acknowledge discussions with Bob Bell, always a source of great ideas and inspiration. We would also like to thank the reviewers for their comments and suggestions.

References

Leo Breiman. Random forests. In *Machine Learning*, volume 45(1), pages 5–32, 2001.

C.J.C. Burges. From ranknet to lambdarank to lambdamart: An overview. In *Microsoft Research Technical Report MSR-TR-2010-82*, 2010.

Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 18–, New York, NY, USA, 2004. ACM. ISBN 1-58113-838-5. doi: http://doi.acm.org/10.1145/1015330.1015432. URL http://doi.acm.org/10.1145/1015330.1015432.

Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, pages 39–46, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-906-0. doi: http://doi.acm.org/10.1145/1864708.1864721. URL http://doi.acm.org/10.1145/1864708.1864721.

Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan. Dependent rounding and its applications to approximation algorithms. *J. ACM*, 53(3):324–360, 2006.

Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *In IEEE International Conference on Data Mining (ICDM 2008*, pages 263–272, 2008.

A. Hume and E. S. Daniels. Ningaui: A linux cluster for business. In *USENIX Annual Technical Conference, in FREENIX Track*, 2002.

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. doi: http://doi.acm.org/10.1145/1835804.1835893. URL http://doi.acm.org/10.1145/1835804.1835893.

Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM. ISBN 1-58113-567-X. doi: http://doi.acm.org/10.1145/775047.775067. URL http://doi.acm.org/10.1145/775047.775067.

Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge*

discovery and data mining, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-193-4. doi: http://doi.acm.org/10.1145/1401890.1401944. URL http://doi.acm.org/10.1145/1401890.1401944.

Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, , and Joseph M. Hellerstein. Graphlab: A new parallel framework for machine learning. In *UAI*, 2010.

Alessandro Panconesi and Aravind Srinivasan. Randomized distributed edge coloring via an extension of the Chernoff-Hoeffding bounds. *SIAM J. Comput.*, 26(2):350–368, 1997.

Ruslan Salakhutdinov and Andriy Mnih. Probabilistic matrix factorization. In *Neural Information Processing Systems (NIPS)*, 2007.

Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *In ICML 2008: Proceedings of the 25th International Conference on Machine Learning*, 2008.

Andreas Töscher, Michael Jahrer, and Robert M. Bell. The BigChaos solution to the netflix grand prize, 2009.

Markus Weimer, Alexandros Karatzoglou, Quoc Viet Le, and Alex Smola. Cofi-rank: Maximum margin matrix factorization for collaborative ranking. In *Neural Information Processing Systems (NIPS)*, 2007.