JMLR: Workshop and Conference Proceedings 18:153–167, 2012 Proceedings of KDD-Cup 2011 competition

Collaborative Filtering Ensemble for Ranking

Michael Jahrer MICHAEL.JAHRER@COMMENDO.AT
Andreas Töscher ANDREAS.TOESCHER@COMMENDO.AT
8580 Köflach, Austria

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

Abstract

This paper provides the solution of the team "commendo" on the Track2 dataset of the KDD Cup 2011 Dror et al.. Yahoo Labs provides a snapshot of their music-rating database as dataset for the competition, consisting of approximately 62 million ratings from 250k users on 300k items. The dataset includes hierachical information about the items. The goal of the competition is to distinguish beteen "High rated" and "Not rated" items of a user. The rating scale is discrete and ranges from 0 to 100, while a "High" rating is a rating≥80. The error measure is the percent of false rated tracks over all users, known as the fractions of misclassifications. The task is to minimize this error rate, hence the ranking should be optimized. Our final submission is a blend of different collaborative filtering algorithms enhanced, with basic statistics. The algorithms are trained consecutively and they are blended together with a neural network. Each of the algorithms optimizes a rank error measure.

Keywords: Collaborative Filtering, Ensemble Learning, KDD Cup, Rating Prediction, Ranking, Music Ratings

1. Introduction

In the KDD Cup 2011 challenge we work with the Yahoo! Music dataset Dror et al.. The datasets of both tracks are disjoint, so we have different users on the Track2. The Track2 dataset is significantly smaller to the one in Track1.

1.1. Notation

We consider the dataset as sparse matrix $\mathbf{R} = [r_{ui}]$, where we use the letter u for users and i for items during this writeup. Bold letters are used for matrices and vectors, non bold for scalars. The set of items is denoted by $\mathbb{I}$ and $|\mathbb{I}|$ stands for the total number of items, $|\mathbb{I}|$=296,111. The set of users is written as $\mathbb{U}$, while $|\mathbb{U}|$ is the total number of users, $|\mathbb{U}|$=249,012. Predictions for user/item pairs are denoted as $\hat{r}_{ui}$. We have no information about the rating date. The set of items, which is rated by user u is $\mathbb{I}(u)$. The set of users, which rate item i is $\mathbb{U}(i)$.

Additional to the rating data there is taxonomy information available. Taxonomy means the relationship between the items within the music domain. For example track x belongs to album y, and album y belongs to artist z. For a more simple and holistic view we calculate item-to-parent and item-to child tables. We use the notation of $\mathbb{P}(i)$ for the parents of item

i and $\mathbb{C}(i)$ for the children of item i. Parents and children are itself items. The total number of children is 1,203,596 (same number of parents).

For the Track2 the goal is to minimize the error rate $E(\theta)$ on the test set. The letter θ indents the trainable weights/parameter of the prediction model.

$$E(\theta) = \frac{100\%}{|\mathbf{R}_{test}|} \sum_{u,i \in \mathbf{R}_{test}} \begin{cases} 0, & r_{ui} \geq 80 \text{ AND } \hat{r}_{ui}(\theta) = Hi \\ 0, & unrated \text{ AND } \hat{r}_{ui}(\theta) = Lo \\ 1, & \text{else} \end{cases} \quad (1)$$

The error assessment is done as follows. For each user we should predict ratings for 6 items. This can be any numeric score, scaling does not matter. Then, the evaluation machine sorts the scores and assigns the 3-highest scores to class "High" and the others to "Low". Equation 1 shows how this is calculated. If we predict all item scores for the particular user correctly, it would result in a 0% error rate. 0% error implies that we separate the "High" rated items perfectly from all others.

Here we can find a sketch of the notation for model parameters. $\mathbf{p}_u$ is a user-dependent feature and $\mathbf{q}_i$ is an item-dependent feature. All features are stored in matrices, e.g. all the user features $\mathbf{p}_u$ are stored in the matrix $\mathbf{p} \in \mathbb{R}^{F \text{x} |U|}$ and the columns are the F-dimensional features. The feature values are typically initialized with small random numbers and they are trained with stochastic gradient descent from the data r_{ui}. As learning rate we use the constant η and λ for L2-regularization. In a basic matrix factorization model a prediction $\hat{r}_{ui}$ is calculated by the dot product of the user and the item feature $\hat{r}_{ui} = \mathbf{p}_u^T \mathbf{q}_i$.

1.2. The Dataset

The provided dataset for Track2 Dror et al. consists of 62,551,438 ratings from 249,012 users on 296,111 items. The range of the rating value is between 0 and 100. An item can be a track, an album, an artist or a genre. Most of the ratings are given to tracks (44%) followed by artists (31%), albums (19%) and genres (6%). The distribution of types within all items are given by 224,041 tracks, 52,829 albums, 18,674 artists and 567 genres. By considering the data as sparse user x item matrix $\mathbf{R} = [r_{ui}]$ the fill-rate of 0.08% is very low value. For example in the Netflix Prize dataset Bennett et al. (2007) the fill-rate of the rating matrix is about 1%.

1.3. The Training of the Models

Our solution is a blended set of predictors. We train each predictor in two steps. In the first step we remove a validation set from the dataset and train the model. Then we store the predictions of the validation set. In the second step we train on all available data with the same meta parameters as in the first step, such as learning rate η, regularization λ, number of epochs etc. and store the predictions of the test set. The validation predictions from the first step are used to train the blender. The target of all models is the minimization of the error rate on the validation set.

predictor	error rate
global mean	50.0%
item count	42.38%
user count	50.0%

Table 1: Baseline predictions. The error rate of global mean and user count is random (50%), because there is no difference in the predicted score for a user. Item count is the number of ratings per item in the dataset and it performs a little better than random.

1.4. Ranking and Sampling

In the KDD Cup 2011 Track2, the key idea for collaborative filtering is to introduce sampling. As mentioned in the introduction, the error measure is the fraction of misclassifications. The ratings are in the range from 0 to 100. In the competition, the "High" rating is defined as rating≥80. For a particular user we should distinguish between "High" rated items and unrated ones. In the test set, the task is to predict scores for 6 items per user. We know that 3 of these items are rated "High", the others are "Not rated" by the user, but sampled with the Item-High probability. Item-High probability means that the drawn item occurs proportional with its number of High-ratings. It is given in equation 2.

$$prob_{Hi}(i) = \frac{1}{|\mathbf{R} \geq 80|} \cdot \sum_{u \in \mathbb{U}(i)} \begin{cases} 1, & r_{ui} \geq 80 \\ 0, & \text{else} \end{cases} \tag{2}$$

In contrast to the high-probability, we can sample an item according to their popularity. Equation 3 denotes this probability.

$$prob(i) = \frac{|\mathbb{U}(i)|}{|\mathbf{R}|} \tag{3}$$

1.5. Drawing the Validation Set

The validation set of KDD Cup 2011 Track1 is given by a separate text file. For Track2 there is no pre-defined validation set available. Therefore, we generate one by ourselves. The generation procedure is based on the public knowledge of the test set generation. We iterate over all users and pick those users, who have ¿3 High ratings on the training set. For those users, 3 random High ratings and 3 un-rated items with the probability of $prob_{Hi}(i)$ are chosen. We end up with 109,360 users and 656,160 ratings being in the validation set. For the final blending stage it is important to use the same validation set for every trained predictor.

For all models in chapter 2 we report error rate values on the validation set. In table 1 we list three baseline error rates.

A low error rate of a single predictor is not necessarily needed in order to blend well within the ensemble. An average user has 250 ratings, an average item 210 ratings.

2. Algorithms

In the following we will describe the collaborative filtering models which are used by our approach. All models are trained on the rating data directly.

All different models get listed here with their corresponding explanation. The formula for predicting one user/item pair is also added to each of the algorithms. For completeness we add the error rate value on the validation set of each of the models with the used meta-parameters. The listed numbers are taken from a predictor which is actually in our final blend.

2.1. Basic Taxonomy

The dataset contains taxonomy information. Items consist of tracks, albums, artists and genres. The information which track belongs to which album, the connection from the albums to the artists and from artists to genres was given. The idea of the basic taxonomy models is to produce predictors which blend well by using taxonomy information which is not captured by the following matrix factorization models.

In total we use 14 different basic taxonomy predictors:

1. $\tilde{r}_{ui}$ = the number of users which gave a high rating to item i (42.42%)
2. $\tilde{r}_{ui}$ = the number of users which rated item i (42.36%)
3. $\tilde{r}_{ui}$ = the number of high ratings the user u has given to tracks included in the album of the given track i (41.05%)
4. $\tilde{r}_{ui}$ = the number of high ratings the user u has given to tracks belonging to the artist(s) of the given track i (36.93%)
5. $\tilde{r}_{ui}$ = the number of high ratings the user u has given to tracks belonging to the genre(s) of the given track i (25.33%)
6. $\tilde{r}_{ui}$ = the number of ratings the user u has given to tracks included in the album of the given track i (38.50%)
7. $\tilde{r}_{ui}$ = the number of ratings the user u has given to tracks belonging to the artist(s) of the given track i (33.45%)
8. $\tilde{r}_{ui}$ = the number of ratings the user u has given to tracks belonging to the genre(s) of the given track i (25.23%)
9. $\tilde{r}_{ui}$ = the number of high ratings the user u has given to a parent (album, artist, genre) of the given track i (20.24%)
10. $\tilde{r}_{ui}$ = the number of ratings the user u has given to a parent (album, artist, genre) of the given track i (18.43%)
11. $\tilde{r}_{ui}$ = the number of high ratings the user u has given to parents (album, artist, genre) and all children of those parents (distance 2) of track i (24.27%)
12. $\tilde{r}_{ui}$ = the number of ratings the user u has given to parents (album, artist, genre) and all children of those parents (distance 2) of track i (25.55%)
13. $\tilde{r}_{ui}$ = the number of high ratings to item i and parents of item i (47.29%)
14. $\tilde{r}_{ui}$ = the number of ratings to item and parents of item i (47.29%)

The final predictions are given by adding 1 in order to avoid a zero and using the logarithm, which leads to:

$$\hat{r}_{ui} = log(\tilde{r}_{ui} + 1) \tag{4}$$

2.2. Matrix Factorization Models

This section is about different matrix factorization techniques. The idea of matrix factorization is to project users and items to a low-dimensional space. So each user and each item is represented by a latent feature vector in $\mathbb{R}^F$. The dot product of them $\mathbf{p}_u^T \mathbf{q}_i$ is the prediction.

2.2.1. SVD (regression)

The SVD model is a simple matrix factorization of the rating matrix. The training is based on regression with a quadratic error function and therefore, it does not directly optimize the ranking score. Predictions of a user/item pair are given by the following formula:

$$\hat{r}_{ui} = \mathbf{p}_u^T \mathbf{q}_i \tag{5}$$

This is one of the most popular models in collaborative filtering since the Netflix Prize Bennett et al. (2007) in 2006. When using stochastic gradient descent as learning algorithm, training time grows linearly with the number of ratings $|\mathbf{R}|$, prediction time of one user/item pair is $O(F)$, hence very fast.

With F=500, η=0.0006, λ=1.5 and 120 epochs we get error rate=15.01% on the validation set.

2.2.2. SVD

The SVD explained here is similar to the SVD (regression) beforehand. The prediction is a dot product, given by

$$\hat{r}_{ui} = \mathbf{p}_u^T \mathbf{q}_i \tag{6}$$

The difference is how to do stochastic gradient descent by simultaneously optimizing the ranking error. Our first approach was to use "zero-sampling", the idea is to train a SVD (regression) on the rating data and on some artificial zero-ratings for randomly sampled items. This was less effective, but better than the regression one.

However, after a few additional experiments we found an effective way to directly learn the ranking from the data. The idea is to learn the rating difference of two items for a given user u. The first item i is selected from the set of items $\mathbb{I}(u)$ rated by user u. The other item i_0 is drawn from all items $\mathbb{I}$, not rated by u. Hence we have two target values r_{ui} and r_{ui_0}. The value of r_{ui_0} can be any constant, we set it for example to 0.

The complete algorithm for training the SVD, which optimized the ranking, is shown in Algorithm 1. The algorithm we use is very similar to the one in Weimer et al. (2008), which is ranking optimization using the ordinal regression loss. Again, we iterate over all elements of the sparse rating matrix $\mathbf{R}$ (line 3). The main difference to standard stochastic gradient descent is that the training is done on pairs, here are item pairs (i, i_0). The second item i_0 (not rated by user u) is sampled according to the Item-High probability from all

items, line 4 in Algorithm 1. The next steps are to predict the ratings $\hat{r}_{ui}$, $\hat{r}_{ui_0}$ (line 5, line 6). The target of the sampled item is set to 0 (line 7). The most important step is to learn the difference between the targets and the actual predictions. The error is shown in line 8. Updates of the features are done feature-wise (line 10 to line 13).

Algorithm 1: Pseudo code for training a SVD for ranking on rating data.

Input: Sparse rating matrix $\mathbf{R} \in \mathbb{R}^{|\mathbb{U}|\text{x}|\mathbb{I}|} = [r_{ui}]$
Tunable: Learning rate η, feature size F

1 Initialize user features $\mathbf{p} \in \mathbb{R}^{F\text{x}|\mathbb{U}|}$ and item features $\mathbf{q} \in \mathbb{R}^{F\text{x}|\mathbb{I}|}$ with small random values
2 **while** *error on validation set decreases* **do**
3 **forall the** $u, i \in \mathbf{R}$ **do**
4 draw an item i_0, not rated by u, from all items $\mathbb{I}$ according to the probability $\text{prob}_{Hi}(i)$
5 $\hat{r}_{ui} \leftarrow \mathbf{p}_u^T \mathbf{q}_i$
6 $\hat{r}_{ui_0} \leftarrow \mathbf{p}_u^T \mathbf{q}_{i_0}$
7 $r_{ui_0} \leftarrow 0$
8 $e \leftarrow (\hat{r}_{ui} - \hat{r}_{ui_0}) - (r_{ui} - r_{ui_0})$
9 **for** $k = 1...F$ **do**
10 $c \leftarrow p_{uk}$
11 $p_{uk} \leftarrow p_{uk} - \eta \cdot (e \cdot (q_{ik} - q_{i_0k}))$
12 $q_{ik} \leftarrow q_{ik} - \eta \cdot (e \cdot c)$
13 $q_{i_0k} \leftarrow q_{i_0k} - \eta \cdot (e \cdot (-c))$
14 **end**
15 **end**
16 **end**

With F=1000, η=0.0002 and 25 epochs we get error rate=5.841% on the validation set with the SVD model.

2.2.3. AFM

This model is called asymmetric factor model because it uses only item-dependent parameters. It was first mentioned by Paterek in Paterek (2007). It learns a different kind of data variability compared to the simple SVD model, hence it blends well within the ensemble. The following equation is the prediction of a user/item pair. In a simplified view, the prediction is again a dot product of the item feature $\mathbf{q}_i$ and the user feature (the right side). But the user feature is given by the set of items $\mathbb{I}(u)$ that user u has rated. Therefore the user is expressed purely by his rated items. The features which are learned from the data are built together in a different way compared to the pure SVD model, hence a linear combination of an SVD and AFM results mostly in a more accurate model.

$$\hat{r}_{ui} = \mu_u + \mu_i + \mathbf{q}_i^T \left(\frac{1}{\sqrt{|\mathbb{I}(u)|}} \sum_{j \in \mathbb{I}(u)} \mathbf{q}_j^{(0)} \right) \tag{7}$$

The two matrices $\mathbf{q}$ and $\mathbf{q}^{(0)}$ have both the size $|\mathbb{I}|$ x F. The term $\frac{1}{\sqrt{|\mathbb{I}(u)|}}$ is used for normalization purpose. User and item biases are μ_u and μ_i.

Training of the AFM with stochastic gradient descent is more effective by iterating over all users $u \in \mathbb{U}$ and make a batch update on the asymmetric part $\mathbf{q}^0$. Algorithm 2 sketches the training process. In line 4 we initialize the update vector $\mathbf{x} \in \mathbb{R}^F$ with zero. Line 16 sums the updates and in line 25 the update is applied to the asymmetric features $\mathbf{q}^{(0)}$. The target value α of the sampled item i_0 can be any number.

Algorithm 2: Pseudo code for gradient descent training an AFM for ranking on rating data.

Input: Sparse rating matrix $\mathbf{R} \in \mathbb{R}^{|\mathbb{U}|\text{x}|\mathbb{I}|} = [r_{ui}]$
Tunable: Learning rate η, sampled target constant α, feature size F

1 Initialize item features $\mathbf{q} \in \mathbb{R}^{F\text{x}|\mathbb{I}|}$ and asymmetric item features $\mathbf{q}^{(0)} \in \mathbb{R}^{F\text{x}|\mathbb{I}|}$ with small random values
2 **while** *error on validation set decreases* **do**
3 **forall the** $u \in \mathbb{U}$ **do**
4 $\mathbf{x} \leftarrow 0$
5 $\mathbf{y} \leftarrow \frac{1}{\sqrt{|\mathbb{I}(u)|}} \cdot \sum_{j \in \mathbb{I}(u)} \mathbf{q}_j^{(0)}$ "virtual user feat."
6 **forall the** $i \in \mathbb{I}(u)$ **do**
7 draw an item i_0, not rated by u, from all items $\mathbb{I}$ according to prob_{Hi}
8 $\hat{r}_{ui} \leftarrow \mu_u + \mu_i + \mathbf{y}^T \mathbf{q}_i$
9 $\hat{r}_{ui_0} \leftarrow \mu_u + \mu_{i_0} + \mathbf{y}^T \mathbf{q}_{i_0}$
10 $r_{ui_0} \leftarrow \alpha$
11 $e \leftarrow (\hat{r}_{ui} - \hat{r}_{ui_0}) - (r_{ui} - r_{ui_0})$
12 $\mu_u \leftarrow \mu_u - \eta \cdot e$
13 $\mu_i \leftarrow \mu_i - \eta \cdot e$
14 $\mu_{i_0} \leftarrow \mu_{i_0} - \eta \cdot e$
15 **for** $k = 1...F$ **do**
16 $x_k \leftarrow x_k + e \cdot (q_{ik} - q_{i_0 k})$
17 **end**
18 **for** $k = 1...F$ **do**
19 $q_{ik} \leftarrow q_{ik} - \eta \cdot (e \cdot y_k)$
20 $q_{i_0 k} \leftarrow q_{i_0 k} - \eta \cdot (e \cdot (-y_k))$
21 **end**
22 **end**
23 **forall the** $i \in \mathbb{I}(u)$ **do**
24 **for** $k = 1...F$ **do**
25 $q_{ik}^{(0)} \leftarrow q_{ik}^{(0)} - \eta \cdot x_k \cdot \frac{1}{\sqrt{|\mathbb{I}(u)|}}$
26 **end**
27 **end**
28 **end**
29 **end**

With F=1000, η=0.00002, α=-50 and 30 epochs we get error rate=6.641% on the validation set.

2.2.4. AFM flipped

Again, we make use of the asymmetric idea, but compared to AFM the roles of users and items are interchanged. Hence, we have a fixed user feature, but the item is expressed by the set of users $\mathbb{U}(i)$ which rated this item. Thus, the item feature is build from a normalized sum of rated user features. User and item biases are μ_u and μ_i.

$$\hat{r}_{ui} = \mu_u + \mu_i + \mathbf{p}_u^T \left(\frac{1}{\sqrt{|\mathbb{U}(i)|}} \sum_{j \in \mathbb{U}(i)} \mathbf{p}_j^{(0)} \right) \tag{8}$$

The two matrices $\mathbf{p}$ and $\mathbf{p}^{(0)}$ have both the size $|\mathbb{U}|$ x F. According to Algorithm 2 we use the same schema, but flipped to the other side (train item-wise, etc.). With F=50, η=0.00005, α=-50 and 50 epochs we get an error rate=7.050% on the validation set.

2.2.5. ASVD

The ASVD model combines the basic SVD approach with the idea from the AFM. The user is described by the user-dependent feature $\mathbf{p}_u$ and the normalized sum of rated items. This model is exactly the same as SVD++ from Y.Koren Koren (2008).

$$\hat{r}_{ui} = \mu_u + \mu_i + \mathbf{q}_i^T \left(\mathbf{p}_u + \frac{1}{\sqrt{|\mathbb{I}(u)|}} \sum_{j \in \mathbb{I}(u)} \mathbf{q}_j^{(0)} \right) \tag{9}$$

The two matrices $\mathbf{q}$ and $\mathbf{q}^{(0)}$ have both the size $|\mathbb{I}|$ x F, matrix $\mathbf{p}$ has the size $|\mathbb{U}|$ x F. Stochastic gradient descent is applied in order to learn the features from the data. Training is similar to Algorithm 2, but the sampling of the second item i_0 is done by picking a random item. With F=1000, η=0.00002, α=0 and 35 epochs we get an error rate=5.949% on the validation set.

2.2.6. ASVD flipped

As in the section above, we apply the asymmetric idea from the ASVD model to the item side. Now, the user is described by $\mathbf{p}_u$ and we have at the item side (right brackets) the item-dependent feature $\mathbf{q}_i$ and the normalized sum of rated user features.

$$\hat{r}_{ui} = \mu_u + \mu_i + \mathbf{p}_u^T \left(\mathbf{q}_i + \frac{1}{\sqrt{|\mathbb{U}(i)|}} \sum_{j \in \mathbb{U}(i)} \mathbf{p}_j^{(0)} \right) \tag{10}$$

The two matrices $\mathbf{p}$ and $\mathbf{p}^{(0)}$ have both the size $|\mathbb{U}|$ x F, matrix $\mathbf{q}$ has the size $|\mathbb{I}|$ x F. As in Algorithm 2 we use the same schema, but flipped to the other side (train item-wise, etc.). With F=2000, η=0.00005, α=-50 and 30 epochs we get an error rate=5.163% on the validation set.

2.2.7. ASVD flipped taxonomy

The ASVD flipped model can be further improved by using the taxonomy information. Items are categorized in 4 different types: tracks, albums, artists and genres. Hierarchical relationships of them are given by the taxonomy info, which comes with the dataset. From the taxonomy we build child and parent tables. $\mathbb{C}(i)$ is the set of related children of item i. $\mathbb{P}(i)$ is the set of parents of the item i. In the model we extend the item part with two new features, the child features $\mathbf{q}^{(1)}$ and the parent features $\mathbf{q}^{(2)}$. We describe the item by its "bag of children" and by the "bag of parents" in the same way as we did it in the asymmetric part $\mathbf{q}^{(0)}$. Here is shown the prediction formula for a user/item pair:

$$\begin{aligned}\hat{r}_{ui} = \mu_u + \mu_i + \mathbf{p}_u^T(\mathbf{q}_i + \frac{1}{\sqrt{|\mathbb{U}(i)|}} \sum_{j \in \mathbb{U}(i)} \mathbf{q}_j^{(0)} + \\ \frac{1}{\sqrt{|\mathbb{P}(i)|}} \sum_{j \in \mathbb{P}(i)} \mathbf{q}_j^{(1)} + \frac{1}{\sqrt{|\mathbb{C}(i)|}} \sum_{j \in \mathbb{C}(i)} \mathbf{q}_j^{(2)})\end{aligned} \tag{11}$$

The matrices $\mathbf{q}$, $\mathbf{q}^{(0)}$, $\mathbf{q}^{(1)}$, $\mathbf{q}^{(2)}$ have the size of $|\mathbb{I}|$ x F, matrix $\mathbf{p}$ has the size $|\mathbb{U}|$ x F. The parameters are learned according to Algorithm 2, but flipped to the other side (train item-wise, etc.). With F=3000, η=0.000025, α=-50 and 40 epochs we get an error rate=4.178% on the validation set. This is our most accurate single model.

2.2.8. ASVD flipped rated taxonomy

Taxonomy information was very helpful in the "ASVD flipped taxonomy" model, furthermore we introduce a new parameter $\mathbf{q}^{(3)}$ which learns structure from rated parents. The set $\mathbb{P}(i)$ is the set of parents, now we use the set $\mathbb{P}_u(i)$ as parents rated by user u.

$$\begin{aligned}\hat{r}_{ui} = \mu_u + \mu_i + \mathbf{p}_u^T(\mathbf{q}_i + \frac{1}{\sqrt{|\mathbb{U}(i)|}} \sum_{j \in \mathbb{U}(i)} \mathbf{p}_j^{(0)} + \\ \frac{1}{\sqrt{|\mathbb{P}(i)|}} \sum_{j \in \mathbb{P}(i)} \mathbf{q}_j^{(1)} + \frac{1}{\sqrt{|\mathbb{C}(i)|}} \sum_{j \in \mathbb{C}(i)} \mathbf{q}_j^{(2)} + \\ \frac{1}{\sqrt{|\mathbb{P}_u(i)|}} \sum_{j \in \mathbb{P}_u(i)} \mathbf{q}_j^{(3)})\end{aligned} \tag{12}$$

With F=1000, η=0.000025, α=-63 and 45 epochs we get an error rate=4.277% on the validation set.

2.3. Neighborhood models

2.3.1. Item-Item KNN

In order to predict the rating $\hat{r}_{ui}$, a k-nearest neighbor algorithm selects K ratings with the highest correlations to the item i. Therefore, we introduce the set of items $\mathbb{I}(u, i, K)$ which consists of rated items from user u with the K-highest correlations to item i. $\mathbb{I}(u, i, K) \subset \mathbb{I}(u)$. Here we have the final prediction formula for the model Item-Item KNN model.

$$\hat{r}_{ui} = \frac{\sum_{j \in \mathbb{I}(u,i,K)} r_{uj} \hat{c}_{ij}}{\sum_{j \in \mathbb{I}(u,i,K)} |\hat{c}_{ij}|} \tag{13}$$

Our item-item neighborhood model uses a sigmoid function to map the correlations c_{ij} to $\hat{c}_{ij}$ by introducing two new parameters σ (scale) and γ (offset).

$$\hat{c}_{ij} = 0.5 \cdot \tanh(\sigma \cdot c_{ij} + \gamma) + 0.5 \tag{14}$$

In general we use the Pearson correlation between two items, by doing this, the model is called "Item-Item KNN Pearson". Another, more effective way of computing the correlation between two items is to use the feature distance. The equation below denotes the correlation c_{ij} between two items i and j. We dub the model "Item-Item KNN SVD feat".

$$c_{ij} = \left(\frac{\sum_{k=1}^{F} (q_{ik} - q_{jk})^2}{\sqrt{\sum_{k=1}^{F} q_{ik}} \sqrt{\sum_{k=1}^{F} q_{jk}}} \right)^{-2} \tag{15}$$

Two models of the Item-Item KNN with SVD features are in the final blend. For those, we calculate the KNN on the residuals of the SVD. We get good results, by using an SVD with F=100 and σ=0.62 γ=-2.16 K=27 the error rate on the validation set is 5.166%.

2.3.2. User-User KNN

The User-User KNN is the flipped version of the Item-Item KNN. The only difference is that users and items are changed. Since we have approximately the same number of items and users, the User-User KNN has similar computational complexity to the item one. The only model, which is in the blend is based on Pearson correlations between users. Hence the model is called "User-User KNN Pearson". With metaparameters σ=23.3 γ=-4.1 and K=507 the error rate on the validation set is 18.35%.

2.4. Restricted Boltzmann Machines

The use of Conditional Restricted Boltzmann Machines for collaborative filtering is described in Salakhutdinov et al. (2007). We use a very similar setup, but adopted the methodology for optimizing the given ranking error measure.

The set of items rated by user u is denoted as $\mathbb{I}(u)$. This leads to the set of unrated items $\mathbb{I} - \mathbb{I}(u)$ for user u. For every user we choose a subset of the unrated items $\tilde{\mathbb{I}}(u) \subset \mathbb{I} - \mathbb{I}(u)$ with the same size as the set of rated items $|\tilde{\mathbb{I}}(u)| = |\mathbb{I}(u)|$. One important thing is to draw the items from the set of unrated items $\tilde{\mathbb{I}}(u)$ according to the probability of an item being rated high.

Every item i is represented via two binary visible units $v_i^{(1)}$ and $v_i^{(2)}$. The unit $v_i^{(2)}$ is 1, if the user rated the item $i \in \mathbb{I}(u)$, otherwise it is 0. The unit $v_i^{(1)}$ represents the unrated items and is 1, if $i \in \tilde{\mathbb{I}}(u)$ otherwise it is 0. For the conditioning of the hidden states we use the items rated with an high rating, this set of items is denoted as $\hat{\mathbb{I}}(u)$ and is a subset $\hat{\mathbb{I}}(u) \subset \mathbb{I}(u)$ of all ratings rated by user u.

The used RBM topology can be seen in Figure 1. The probabilities of the visible and hidden units for being high are given by:

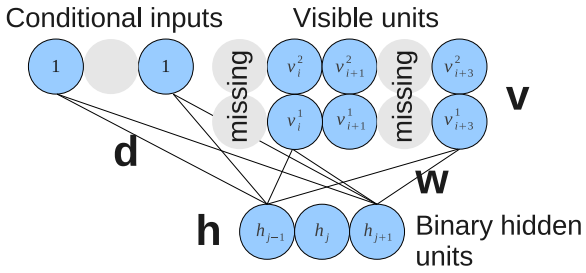

Figure 1: A Conditional Restricted Boltzmann Machine. The hidden units are conditioned on the items rated high by the given user.

$$p(v_i^{(l)} = 1|\mathbf{h}) = \frac{\exp(a_i^{(l)} + \sum_j h_j w_{ij}^{(l)})}{\sum_{l=1}^{2} \exp(a_i^{(l)} + \sum_j h_j w_{ij}^{(l)})} \tag{16}$$

$$p(h_j = 1|\mathbf{v}) = \sigma\left(b_j + \sum_{i \in \mathbb{I}(u) \cup \tilde{\mathbb{I}}(u)} \sum_{l=1}^{2} w_{ij}^{(l)} v_i^{(l)} + \sum_{i \in \hat{\mathbb{I}}(u)} d_{ij}\right) \tag{17}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{18}$$

We initialize the biases of the visible units $a_i^{(l)}$ to the logarithm of the probabilities of the corresponding unit of being high. The biases of the hidden units b_j are initialized to zero. The features w_{ij} and d_{ij} are initialized to values drawn from a zero mean Gaussian distribution with a small standard deviation of 0.001. With 300 hidden units we achieve an error rate=10.11% on the validation set.

For every training case we "activate" only a small set of visible units, the set of rated $\mathbb{I}(u)$ and the sampled set of unrated items $\tilde{\mathbb{I}}(u)$. Every visible unit belonging to an item not included in the set $\hat{\mathbb{I}}(u) = \tilde{\mathbb{I}}(u) \cup \mathbb{I}(u)$ must not be reconstructed for the training of user u. Both sets have the same size $|\tilde{\mathbb{I}}(u)| = |\mathbb{I}(u)|$ and every item is represented via 2 visible units. Hence there is the need to reconstruct $2 \cdot |\hat{\mathbb{I}}(u)|$ visible units, which is less than the total number of items $|\mathbb{I}|$. It is also possible to use a single visible unit for every item being 1 for rated and 0 for unrated. The problem is that this setup would require a reconstruction of every visible unit, which equals $|\mathbb{I}|$ reconstructed visible units for every training user u. On the given dataset, where every user only rated a small fraction of all items, this increases the computational costs. By using two visible units per item it is not needed to reconstruct all items, only the rated items and a subset of unrated items have to be reconstructed.

3. Blending

For track 2 there was no predefined hold out set. In Section 1.5 we describe how we choose our fixed hold out set. This hold out set is designed to resemble the disclosed test set and was fixed for all trained models.

In order to produce one predictor, the hold out set is removed from the data. The algorithm trains the model on the remaining data. Then predictions for the hold out set are produced. Afterwards we insert the hold out set in the training set, which results in the original training set. Now the model is trained again on the original training set and produces predictions for the test set. Hence every predictor $\mathbf{p}_n = [\mathbf{p}_n^{(1)}\mathbf{p}_n^{(2)}]$ consists of predictions on the hold out set $\mathbf{p}_n^{(1)}$ and of predictions for the test set $\mathbf{p}_n^{(2)}$. For the hold out set the true ranking is known. This leads to a supervised learning problem.

We use a single layer neural network with sigmoid units on the hidden layer and a linear readout. The network structure can be seen in Figure 2. The network is trained using pairwise rankings, the idea for this methodology originates in Burges et al. (2005).

The neural network used is visualized in Figure 2. The output is given by:

$$h_{ki} = \tanh\left(\sum_n w_{kn}^{(1)} p_{ni}\right) \tag{19}$$

$$o_i = \sum_k w_k^{(2)} h_{ki} = \sum_k w_k^{(2)} \tanh\left(\sum_n w_{kn}^{(1)} p_{ni}\right) \tag{20}$$

The difference between two outputs o_i and o_j is denoted as

$$o_{ij} = \zeta(o_i - o_j) \tag{21}$$

where ζ is a meta parameter for scaling. In order to train the weights we consider pairwise rankings. We choose two indices i and j, where index i is ranked higher than j. The cost function

$$C_{ij} = -o_{ij} + log\,(1 + e^{o_{ij}}) \tag{22}$$

is described in Burges et al. (2005). It is low, if the outputs of the network predict the correct order, meaning that o_i is higher than o_j.

The derivatives of the pairwise error between the training sample i and j, where i should be ranked higher than j is given by:

$$\frac{\partial C_{ij}}{\partial w_k^{(2)}} = \zeta(h_{ki} - h_{kj})\left(\frac{e^{o_{ij}}}{1 + e^{o_{ij}}} - 1\right) \tag{23}$$

$$\frac{\partial C_{ij}}{\partial w_{kn}^{(1)}} = \zeta w_k^{(2)}\left((1 - h_{ki}^2)p_{ni} - (1 - h_{kj}^2)p_{nj}\right)\left(\frac{e^{o_{ij}}}{1 + e^{o_{ij}}} - 1\right) \tag{24}$$

The training is done using stochastic gradient descent and 32 fold cross validation. The weights are initialized to random values drawn uniformly between -0.4 and 0.4. The parameter ζ is set in the range from 1 to 2, for the final blends we used $\zeta = 1.69$. The number of hidden units was set to 30. Table 2 contains all predictors used in the final blend, which produced an error of 2.4887% on the leaderboard.

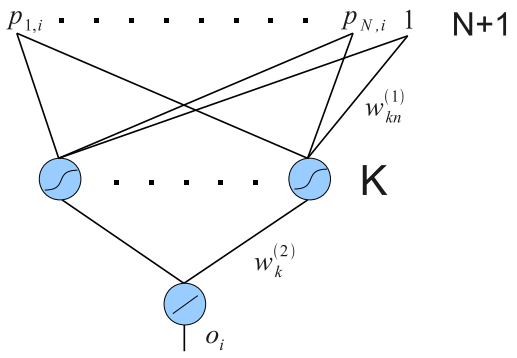

Figure 2: The structure of the neural network used for blending. The N inputs p_{ni} are normalized to mean zero and standard deviation 1. The last input is constantly 1. The hidden layer consists of K sigmoid units, while the readout is purely linear.

4. General thoughts

Sampling is a key idea when optimizing a rank error measure within a collaborative filtering algorithm. Compared to pure stochastic gradient descent it works much better. We found out that regularization is not that important, hence we skip it for most of the models.

During the competition, our goal was to try many different models, which can explain the variability of the data. The accuracy of matrix factorization models can be easily improved by adding more features. We have a much higher relative improvement in the error rate compared to RMSE when doubling the number of features. Hence, some of our best models act with over 1000 features.

We did experiments with sampling items of the same type, e.g. if processing a track then also a track is sampled. However the results only got worse.

In general, taxonomy information helps in getting an accurate collaborative filtering model. Our opinion is that the rating date would also help in further improving the model. Our approach on Track1 shows the effectiveness of integrating the rating date in the models.

References

James Bennett, Stan Lanning, and Netflix Netflix. The Netflix Prize. In *KDD Cup and Workshop in conjunction with KDD*, 2007.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 89–96, New York, NY, USA, 2005. ACM. ISBN 1-59593-180-5. doi: http://doi.acm.org/10.1145/1102351.1102363. URL http://doi.acm.org/10.1145/1102351.1102363.

Gideon Dror, Yehuda Koren, and Markus Weimer. Yahoo! Music Rating Dataset for the KDD Cup 2011 Track2. http://kddcup.yahoo.com/datasets.php.

Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–434. ACM, 2008.

Arkadiusz Paterek. Improving regularized singular value decomposition for collaborative filtering. *Proceedings of KDD Cup and Workshop*, 2007.

Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey E. Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML*, pages 791–798, 2007.

Markus Weimer, Alexandros Karatzoglou, and Alexander J. Smola. Improving maximum margin matrix factorization. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 5211 of *LNAI*, pages 14–14. Springer, 2008. ISBN 978-3-540-87478-2. URL http://www.springerlink.com/index/j4l77t2p7kujg86w.pdf.

Appendix A. Single predictor performance

no	model	error rate Test1	meta-parameters
1	AFM	5.60%	F=3000 epochs=30 η=0.00005 α=-50
2	AFM	6.38%	F=1000 epochs=30 η=0.00002 α=-50
3	AFM flipped	6.98%	F=50 epochs=50 η=0.00005 α=-50
4	ASVD flipped	5.05%	F=200 epochs=30 η=0.00005 α=-50
5	ASVD flipped taxo	4.02%	F=3000 epochs=40 η=0.000025 α=-50
6	ASVD flipped rated taxo	4.40%	F=200 epochs=50 η=0.000025 α=-50
7	ASVD flipped rated taxo	4.11%	F=1000 epochs=45 η=0.000025 α=-50
8	ASVD flipped rated taxo	4.15%	F=3000 epochs=45 η=0.000025 α=-63
9	ASVD flipped rated taxo	4.06%	F=1000 epochs=45 η=0.000025 α=-63
10	ASVD flipped rated taxo	5.08%	F=100 epochs=40 η=0.000025 α=-63
11	ASVD flipped rated taxo	6.88%	F=30 epochs=20 η=0.000025 α=-63
12	ASVD flipped rated taxo	6.98%	F=50 epochs=30 η=0.000025 α=-50, same type in sampling
13	ASVD	5.70%	F=1000 epochs=15 η=0.00005 α=0
14	ASVD	5.63%	F=1000 epochs=35 η=0.00002 α=0
15	Basic Taxonomy (1-14)	18% - 47%	details in Section 2.1
16	RBM discrete	27.7%	nHid=10 η=0.0001 λ=0.02
17	RBM discrete	27.9%	nHid=10 η=0.0001 λ=0.002
18	RBM discrete	12.5%	nHid=100 η=0.0003 λ=0.002
19	RBM discrete	9.77%	nHid=300 η=0.0015 λ=0.0002
20	Item-Item KNN Pearson	15.7%	σ=10.5 γ=-8.2 K=29
21	Item-Item KNN Pearson	13.3%	σ=19.2 γ=-13.2 K=289
22	Item-Item KNN SVD feat	6.72%	σ=0.62 γ=-2.16 K=27 (SVD: F=100 η=0.0001 epochs=30)
23	Item-Item KNN SVD feat	4.92%	σ=1.15 γ=-1.66 K=67 (SVD: F=100 η=0.00003 epochs=50)
24	User-User KNN Pearson	17.8%	σ=23.3 γ=-4.1 K=506
25	SVD (regression)	15.2%	F=100 epochs=120 η=0.0004 λ=1.5
26	SVD (regression)	14.7%	F=500 epochs=120 η=0.0006 λ=1.5
27	SVD	6.74%	F=100 epochs=30 η=0.0002
28	SVD	5.71%	F=1000 epochs=25 η=0.0002

Table 2: Predictors in the final blend. The Neural-Network blender achieves an error rate of 2.617% on the validation set and an error rate of 2.4887% on the KDD Cup 2011 Track2 leaderbord. Test1 is a 50% subset of the leaderboad set, it contains ratings from 101172 users. Each user has exactly 6 ratings in Test1. Our final solution has an error rate of 2.492% on Test1 (and 2.444% on Test2).