The BigChaos Solution to the Netflix Prize 2008

Andreas Töscher and Michael Jahrer

commendo research & consulting
Neuer Weg 23, A-8580 Köflach, Austria
{*andreas.toescher,michael.jahrer*}*@commendo.at*

November 25, 2008

Abstract

The team "BellKor in BigChaos" is a combined team of team BellKor and BigChaos. The solution with a RMSE of 0.8616 is created by a linear blend of the results from both teams. In the following paper we describe the results of BigChaos.

1 Preface

During the last 2 years of research we tried a variety of different collaborative filtering algorithms. In the following we describe all methods which turned out to be useful for the Netflix Prize competition.

In sections 4 to 18 we describe all algorithms used to produce our predictions, listed in section 20. We minimized the RMSE of each individual predictor on the probe set. Whenever possible, we therefore take a simple automatic parameter tuner described in section 3 to tune the meta parameters. Details are reported in the section describing the individual algorithm, and for each predictor, the meta parameters used are reported in section 20.

If an algorithm has been trained on the residuals of another, we note that explicitly; others are based on raw ratings. Exact details can again be found in the predictor listing in section 20. Here we note for every predictor, if it was based on the residuals of another algorithm.

2 Notation

Throughout the document we keep the following notation. A user is denoted with u and item/movie with i. The rating given by user u to item i is denoted by r_{ui}. Predictions for the rating r_{ui} are denoted with $\hat{r}_{ui}$. $N(i)$ stands for the set of users who voted the item i, and $N(u)$ is the set of items voted by the user u. $N(u,i)$ is the set of the K most similar items to item i which were rated by user u and is therefore a subset of $N(u)$. The learning rate is called η and the regularization λ. Constants like $\alpha, \beta, \gamma, \delta\, \theta, \vartheta, ...$ can have different meanings for every algorithm.

3 Parameter Tuner

In several algorithms like in Section 5, there is a need for tuning parameters, in order to minimize the RMSE of a particular predictor. We use two algorithms to automatically tune them. The target is to adjust N parameters p_i, to minimize the RMSE on the probe set with the error of a given set of parameters being denoted as $E(p_1, p_2, ..., p_N)$.

In general, finding good parameters is a mixture of good manual setting and some automatic fine tuning. Both methods described below, are sensitive to the initialization values of p_i. They are not guaranteed to find a global optimum, and can easily get stuck in local optima. So starting with a random initialization of the parameters does not guarantee to get the best solution. To ensure that all our results are reproducible, we report all found parameter values for each individual predictor in section 20. Note that the reported values are these, which we found and used. For most algorithms there may be a much better parameterization.

3.1 Automatic Parameter Tuner 1 (APT1)

This is a simple random search method. The basic idea is to randomly change one parameter, check whether the RMSE gets better, and keep the new value if so. In detail this works as follows:

Randomly select i from {1,2,...,N} and draw a new parameter value from a normal distribution $p_i^{new} = \mathcal{N}(\mu, \sigma^2)$ with $\mu = p_i$ and $\sigma^2 = 0.1 \cdot |p_i|$. If $E^{new} < E$ then $p_i \leftarrow p_i^{new}$. Loop until E is minimal.

For values of p_i, which are close to zero, the variance σ^2 approaches zero too. This results in parameters which can get stuck around zero. To overcome this problem we ensure that $\sigma^2 > 0.001$. With this modification the parameters can change the sign during the training.

3.2 Automatic Parameter Tuner 2 (APT2)

An efficient way of minimizing the error function is a structured coordinate search, as described here. The major advantage is the searching speed. $10^2 ... 10^3$ epochs are in general enough to set ten free parameters. Here, an epoch is the evaluation of the error function E with the new selected parameter p_i. The drawback is that the search can get stuck in a local optima in the N-dimensional search space. Also the located minima can be dependent on the order of p_i's. The algorithm searches for new parameters p_i as follows.

Initialize the N search exponents $e_i = 0.8$. Select i sequentially from {1,2,..,N,1,2..} and try a new parameter by $p_i^{new} = p_i \cdot e_i$ or ${p_i}^{new} = p_i \cdot e_i^{-1}$. Go in the direction per parameter, where $E^{new} < E$. After 5 trials per parameter, increase i by 1. If the error E increases, there is a need for smaller step size, respectively increase the search exponent to get closer to 1.0 by $e_i^{new} = {e_i}^{0.9}$. The algorithm runs until maximum of epochs are reached. After several hundred epochs of search, the fluctuations on the error E are marginal and the search has finished.

4 Basic Predictors (BASIC)

In order to get a baseline prediction, we tried several simple statistics. Only the user mean rating was useful for our overall blend. Other simple effects were captured by the rest of the ensemble.

5 Weekday Model (WDM)

This model predicts ratings on the basis of weekday means (average ratings from Monday to Sunday). We calculate weekday averages per user, movie and globally. The index $w = \{1, 2, ..., 7\}$ is the particular day of the rating, $i = \{1, ..., M\}$ is the movie index and $u = \{1, ..., U\}$ is the user index. In the formula below, $\bar{\mu}_{uw}$ denotes the average rating of user u on weekday w. n_{uw} is the number of ratings used to calculate the average (called support). $\bar{\mu}_{iw}$ describes the item mean on weekday w and the global average on the weekday w is given as $\bar{\mu}_w$. The prediction formulas are motivated by blending local estimates to global ones, based on the number of ratings. The relationship on the support variables $n_{uw} < n_{iw} < n_w$ is true in most cases. To give the prediction model more flexibility, we add non negative exponents ν, ϵ and δ. The non negative parameters α, β and γ are used to control the influence of the weekday estimates based on the support.

$$\bar{r}_{ui} = \frac{\bar{\mu}_{uw} \cdot n_{uw}^{\nu}}{n_{uw}^{\nu} + \alpha} \tag{1}$$

$$\tilde{r}_{ui} = \frac{\bar{\mu}_{iw} \cdot n_{iw}^{\epsilon} + \bar{r}_{ui} \cdot \beta}{n_{iw}^{\epsilon} + \beta} \tag{2}$$

$$\hat{r}_{ui} = \frac{\bar{\mu}_{w} \cdot n_{w}^{\delta} + \tilde{r}_{ui} \cdot \gamma}{n_{w}^{\delta} + \gamma} \tag{3}$$

The parameters α, β, γ, δ, ϵ and ν are set by APT2 (section 3.2). In the predictor listing (section 20) we report the exact values found by the automatic parameter tuner. Typically we apply this model on the residuals of 1 global effect.

6 Basic SVD (BasicSVD)

A SVD computes a rank K approximation $\mathbf{R}' = \mathbf{A}\mathbf{B}^T$ of the rating matrix $\mathbf{R}$. The objective is to minimize the following error function on the training list $\mathcal{L} = \{(u_1, i_1), \ldots, (u_L, i_L)\}$.

$$E(\mathbf{A}, \mathbf{B}, \mathcal{L}) = \sum_{(u,i)\in\mathcal{L}} (r_{ui} - \hat{r}_{ui}^{MF})^{\alpha} + \frac{\lambda}{2}(\|\mathbf{A}\|_F^2 + \|\mathbf{B}\|_F^2) \tag{4}$$

The best results are received, for α between 1 and 2. Finding good parameters for α and λ for large dimensions of K is very time consuming and therefore it is impractical to use an automatic parameter tuner. So all these values were found by manual parameter tuning. Refer to the predictor listing (section 20) to see the exact values we used to train the individual predictors.

All our BasicSVD models are trained using a stochastic gradient descent, where all features are updated simultaneously. For the non negative versions, we ensure that the features always stay positive during the training.

7 SVD Adaptive User Factors (SVD-AUF)

This model extension to SVD was proposed by team BellKor in progress prize paper 2007 [1]. The basic idea is to weight equations on the user side according to a movie-movie similarity measure. Before adaptive user factors can be calculated, movie and user features must exist. They are trained by a SVD with alternating least squares. For non negative features, we use the solver from [1]. For every prediction the user features are recalculated. For a similarity measure, we use the time between ratings of user u. The similarity between two items i and j (from user u) is given by $s_{ij} = (|t_i - t_j| + 1)^{-\alpha} + \beta$. The adjustable parameters in the SVD-AUF algorithm α and β are non negative real numbers. They are set by the automatic parameter tuner APT2 (section 3.2).

8 SVD Alternating Least Squares (SVD-ALS)

Training of a SVD model with alternating least squares is described in [1]. In our approach, we also used of the non negative version described in the paper.

9 Time SVD (TimeSVD)

Collaborative filtering with a SVD model is very efficient in terms of speed and accuracy. It is hard to directly incorporate time into a SVD model. The TimeSVD model adds time information by adding a feature offset when calculating the inner feature product. Given a user feature $\mathbf{p_u}$ and a movie feature $\mathbf{q_i}$ vector, a rating of user u and movie i is predicted by $\hat{r}_{ui} = \sum_{k=1}^{K} p_{u,(k+d_{ui})} \cdot q_{i,(k+d_{ui})}$. This means that features are shifted, depending on the date of the rating. The discrete time offsets d_{ui} are precalculated before the training starts. We divide the rating time span into T time slots per user. The first slot is left side open and the last slot is right side open. For example if a user u gives votes on 50 different days and $T = 10$, then 5 days point to 1 slot. The offset d_{ui} can take values of $d_{ui} = \{0, 1, .., T-1\}$.

Training is done with a normal stochastic gradient descent like in 6. In some models we use the time offsets d_{ui}, either on movie or on user side. We refer to the predictor listing in 20 for details. Learn rate η and regularization λ are reported per predictor.

10 Neighborhood Aware Matrix Factorization (NAMF)

This method is described in [6]. Parameters and training method are used exactly as reported in the paper. That means we use a Pearson Correlation on 10 global effects, to select the 50 best correlating users and movies. The matrix factorizations have 300 and 600 features. For tuning the parameters to combine the MF and neighborhood models we use the APT1 (3.1).

11 Restricted Boltzmann Machines

We use RBMs for collaborative filtering as described in [4]. The conditional RBM with multinomial visible units is called RBMV3 in our implementation. Factorization and minipatches as described in the paper are not used. A variation of this algorithm uses Gaussian visible units, instead of the multinomial ones (RBMV5). In the RBMV3 and RBMV5 a user is represented as a bag of movies, which means that the RBM has visible units for every movie in the dataset. Additionally, we use a flipped version where a movie is represented as a bag of users. A Gaussian visible unit is applied to every customer. In our implementation this algorithm is called RBMV6.

As reported in [2], all variants of RBMs are very insensitive to parameter settings. In the parameter listing (section 20) the learning rate is denoted by η and the regularization with λ.

12 Movie KNN

Neighborhood based approaches are commonly used for collaborative filtering. The main idea is to make a prediction based on neighbor ratings. There are two different perspectives, the customer based and the movie based. The customer based approach uses customer-customer distances. For the Netflix data set the customer based approach is impractical. Due to the large amount of customers, it is not possible to precompute the customer-customer distance matrix (the matrix has a size of 1000 GB in single precision float). The next problem with the customer-customer approach is that two arbitrary customers have few common ratings. Therefore, distances are computed out of very few common ratings. For the Netflix dataset the movie-movie approach works better. On average, there are 30 times more datapoints to calculate a movie-movie distance and the precomputed distance matrix fits into main memory.

We tried out a variety of different KNN models and similarity measures. The following similarities worked best:

- Pearson Correlation
- Set Correlation: $\rho_{ij} = \frac{|N(i) \bigcap N(j)|}{min(|N(i)|,|N(j)|)}$

$N(i)$ describes the set of users who voted for movie i. We shrink the correlation ρ_{ij} to zero based on the support $n_{ij} = |N(i) \bigcap N(j)|$. Negative correlations are set to zero.

$$c_{ij} = \frac{\rho_{ij} \cdot n_{ij}}{n_{ij} + \alpha} \tag{5}$$

The parameter α is a positive constant and typically set in the range from 200 up to 9000. α and all the parameters mentioned in the following KNN versions were set using APT1 (section 3.1). Exact values for the parameters can be found in the predictor listing (section 20).

12.1 Basic Pearson KNN (KNN-Basic)

This is the simplest form of a neighborhood based model. We weight the K best correlating neighbors $N(u,i)$, based on their correlation c_{ij}.

$$\hat{r}_{ui} = \frac{\sum_{j \in N(u,i)} c_{ij} r_{uj}}{\sum_{j \in N(u,i)} c_{ij}} \tag{6}$$

12.2 KNNMovie

This is an extension of the basic model. We use a sigmoid function to rescale the correlations c_{ij} in order to achieve lower probe RMSEs [6].

$$c_{ij}^{new} = \sigma(\delta \cdot c_{ij} + \gamma) \tag{7}$$

$$\sigma(x) = \frac{1}{1 + exp(-x)} \tag{8}$$

$$\hat{r}_{ui} = \frac{\sum_{j \in N(u,i)} c_{ij}^{new} r_{uj}}{\sum_{j \in N(u,i)} c_{ij}^{new}} \tag{9}$$

Rescaling parameters δ and γ are very sensitive. We use the APT1 described in 3.1 for tuning these. All the α, δ and γ values used for the individual predictors are reported in the predictor listing in section 20.

12.3 KNNMovieV3

In this model we extend the KNNMovie model by the date of the ratings. The basic idea is to give recent ratings a higher weight than the old ones.

$$c_{ij}^{date} = \sigma\left(\delta \cdot c_{ij} \cdot \exp\left(\frac{-\mid \triangle t \mid}{\beta}\right) + \gamma\right) \tag{10}$$

$$\sigma(x) = \frac{1}{1 + exp(-x)} \tag{11}$$

$$\hat{r}_{ui} = \frac{\sum_{j \in N(u,i)} c_{ij}^{date} r_{uj}}{\sum_{j \in N(u,i)} c_{ij}^{date}} \tag{12}$$

$\mid \triangle t \mid$ stands for days between the rating r_{ui}, which we are going to predict, and the past rating r_{uj}. As above we use APT1 to tune this model.

12.4 KNNMovieV4

This is an implementation of the BellKor KNN with jointly derived neighborhood interpolation weights, as described in [1]. In their paper, they used a shrinked Pearson Correlation as in equation 5 to select the K best neighbors. We also tried the Set Correlation defined above to select the K best neighbors, and got slightly better results in comparison to the Pearson Correlation (e.g., compare the predictors #44 and #45).

In the predictor listing we clearly state which correlation measure we used to find the K best neighbors.

12.5 KNNMovieV6

This model does not use Pearson or Set correlations; instead we use the length of the common substring between two movies and their production year, in order to get weighting coefficients. We denote the length of the longest common substring between the movies i and j as s_{ij}. The production year of a movie i is represented by d_i.

$$c_{ij}^{sub} = \sigma(\delta \cdot s_{ij}^{\xi} + \gamma) \cdot \exp\left(\frac{-(d_i - d_j)^2}{\beta}\right) \tag{13}$$

$$\sigma(x) = \frac{1}{1 + exp(-x)} \tag{14}$$

$$\tilde{r}_{ui} = \frac{\sum_{j \in N(u)} c_{ij}^{sub} r_{uj}}{\sum_{j \in N(u)} c_{ij}^{sub}} \tag{15}$$

$$\hat{r}_{ui} = \frac{\tilde{r}_{ui} \cdot \sum_{j \in N(u)} c_{ij}^{sub} + \vartheta \cdot \bar{r}_i}{\sum_{j \in N(u)} c_{ij}^{sub} + \vartheta} \tag{16}$$

As opposed to the former models we make use of every voted movie, with a similarity $c_{ij}^{sub} > \theta$, not only the K best. To improve prediction accuracy we softly blend to the movie average $\bar{r}_i$, based on the sum of correlations. Like in above models, we tune all parameters with APT1.

It turned out that movie-movie correlations, based on movie titles and production years, are not very helpful. The predictor #48 improved the results of predictor #86 by 0.0002. In the blend the improvement is even smaller.

12.6 KNNMovieV7

This algorithm is similar to KNNMovieV3. The main difference is that we do not choose the K best correlating movies. Instead we take every movie where the correlation c_{ij} is higher than a threshold θ. In the end we softly blend to the movie average, based on the sum of correlations (as in KNNMovieV6, equation 16).

Typical values of θ are around 0.02. Check section 20.12 for more details, and the complete listing of all predictors and parameters.

13 Regression on Similarity (ROS)

Regression on similarity models are discussed in [6]. Three different models were reported in the predictor listing section. The first one is based on the full movie-movie similarity matrix and a factorized version on movie-movie and user-user side. In the listing section, we specify the learnrate η, regularization λ and the feature size k (in the factorized models).

14 Asymmetric Factor Model (AFM)

We make use of the AFM model of last year's progress prize. The models are exactly the same as what is being referred to as [SIGMOID1] in [2]. In the result section we note k as the number of features. Stochastic gradient descent is used to simultaneously train the features.

15 Global Effects (GE)

Global effects are simple models, where one effect is trained on the residual of the previous effect. A detailed description can be found in [1]. We use the extension as described in [6] for up to 14 global effects. In some algorithms, we list 0 GE as preprocessor, meaning that the model is trained on residuals of global rating mean. The parameter α in the table below is exactly the same as in the paper [1].

Nr.	Name	RMSE probe	RMSE train	α
0	Overall mean	1.1296	1.0845	NA
1	Movie effect	1.0526	1.0105	22
2	User effect	0.9840	0.9176	7.5
3	User x Time(user)	0.9802	0.9110	435
4	User x Time(movie)	0.9778	0.9060	125
5	Movie x Time(movie)	0.9760	0.9041	4100
6	Movie x Time(user)	0.9752	0.9032	420
7	User x Average(movie)	0.9711	0.8938	68
8	User x support(movie)	0.9682	0.8854	76
9	Movie x average(user)	0.9671	0.8842	140
10	Movie x support(user)	0.9659	0.8836	1e10
11	Movie x avgMovieProductionYear(user)	0.9635	0.8802	380
12	User x movieProductionYear(user)	0.9623	0.8762	170
13	User x standardDeviation(movie)	0.9611	0.8725	130
14	Movie x standardDeviation(user)	0.9604	0.8717	3700

The order of the global effects influences the result, so if one applies them in a different order, the result will be different as well. We always use the ordering from the table above. That means, if we refer to "x GE", we mean the first "x global effects" applied exactly in the same order.

We have not experimented with the ordering of global effects. A different ordering might give a lower error.

16 Global Time Effects (GTE)

The model Global Time Effects combines the idea of global effects with time dependency. In general, the algorithm is similar to GE; here distinctive global rating effects are cancelled on movie or on user

side, centered with time averages of that effect. We apply each effect on residuals of the previous one. The following table gives an overview on different kinds of effects.

Nr.	Name	RMSE probe	RMSE train	λ	σ	ϵ	α	β
0	global Avg	1.1271	1.0785	4.67e-4	4.81e-1	1.38e-4	NA	4.52e-8
1	movie Effect	1.0473	1.0033	2.29e1	1.69e1	1.28e-2	1e3	1.25e1
2	user Effect	0.9717	0.9074	2.76e-1	1.32	1.70e-1	8.94	6.91
2.1	movie Effect	0.9705	0.9057	1.01e1	2.40e1	4.07e-5	1.47e4	1.04e-7
2.2	user Effect	0.9686	0.9057	1.62	8.23e1	1.14e-2	1.15e1	1.14e2
3	user x time(user)	0.9679	0.9049	1.37e2	1.33e4	2.54e-3	9.36e-1	3.66e1
4	user x time(movie)	0.9666	0.9014	3.52	2.46e2	3.35e-4	1.06e-2	1.56e2
5	movie x time(movie)	0.9665	0.9012	1.27	2.07e1	6.3e-13	2.08e1	6.72e-8
6	movie x time(user)	0.9653	0.9002	1.88e2	6.23e1	7e-2	9.5	2.27e1
7	user x avg(movie)	0.9619	0.8919	2.70e-3	1.89e2	4.35e-4	1.07e-2	8.94e1
8	user x support(movie)	0.9601	0.8860	4.29e9	1.70e2	1.98e-4	2.05e-3	9.86e1
9	movie x avg(user)	0.9591	0.8852	1.23	1.43e5	1.95e-4	4.58e1	8.50e-2
10	movie x support(user)	0.9581	0.8846	1.12e1	3.06e4	9.18e-3	4.1e-2	1.31
11	movie x avgMovieYear(user)	0.9554	0.8815	7.12e3	3.96e1	1.13e-6	4.1e-4	2.05
12	user x year(movie)	0.9541	0.8775	2.51e2	9.58e3	3.13e-4	1.56	1.25e2
13	user x stddev(movie)	0.9530	0.8739	4.81e-1	6.62e1	1.91e-2	4.1e-3	6.37e1
14	movie x stddev(user)	0.9523	0.8731	9.01e1	1.5e1	4.27e-3	3.56e-6	4.96e-8
15	movie x percentSingleVotes (user)	0.9515	0.8725	7.8e-12	1.59e1	2.31e-8	3.8e-2	7.19e2
16	movie x ratingDateDensity (user)	0.9514	0.8724	3.3e6	9.46	8.6e-11	3.36e1	7.19e-1
17	user x stringlengthMovie Title(movie)	0.9513	0.8713	4.99e4	1.43e2	2.04e-3	2.45e-4	3.14e-6
18	movie x avgStringlenTitle (user)	0.9510	0.8708	6.33e4	4.54e1	3.32e-1	1.09	4.1e-2
19	movie x percentMovieWith NumberInTitle(user)	0.9509 (qual 0.9450)	0.8710	1.92e2	2.53e1	1.03e-4	3.15e-1	3.12

To clarify how global time effects work, we demonstrate them with effect number 7, user x avg(movie). The goal is to predict $\hat{r}_{ui}$ for user u and movie i. The corresponding rating date is t_{ui}. The set $N(u)$ consists of the rated movies by user u. The residual is the integer rating subtracted by the result of the previous effect, $r_{ui}^{res} = r_{ui}^{Int} - r_{ui}$. In the equations below, μ_i denotes the average rating of movie i, and $\bar{\mu}_{ui}$ is the time dependent average at time t_{ui}. Now a prediction with global time effects is explained in the following way.

$$\bar{r}_{ui} = \bar{r}_{ui} + (\mu_i - \bar{\mu}_{ui})\, x_0 + x_1 \tag{17}$$

$$\bar{\mu}_{ui} = \frac{\sum_{j \in N(u)} \mu_j \cdot k(t_{ui}, t_{uj})}{\sum_{j \in N(u)} k(t_{ui}, t_{uj})} \tag{18}$$

The prediction in equation (17) is based on the previous effect r_{ui} (baseline), enhanced by scaled time effect $(\mu_i - \bar{\mu}_{ui})x_0$ with offset correction x_1. We calculate the time-dependent movie average $\bar{\mu}_{ui}$ by doing a convolution in time (18) with a kernel $k(t_0, t_1)$. We found that a negative exponential kernel, such as $k(t_0, t_1) = exp(-|t_0 - t_1|/\sigma)$, works well. Additionally, we use a threshold ϵ for the kernel to limit it to a constant positive value. $\mathbf{x} = [x_0, x_1]$ comes from solving an overdetermined equation system $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$. $\mathbf{A}$ is a matrix with $|N(u)|$ rows and two columns. The first column of $\mathbf{A}$ consists of $\mu_i - \bar{\mu}_{ui}$ and the second column is constant $A_{i,2} = 1$. Target values in $\mathbf{b}$ are the residuals r_{ui}^{res}. The solution $\mathbf{x}$ is calculated by applying ridge regression $\mathbf{x} = \left(\mathbf{A}\mathbf{A}^T + \lambda \mathbf{I}\right)^{-1} \mathbf{A}^T \mathbf{b}$. To refine the prediction per effect, we blend the local rating estimate $\bar{r}_{ui}$ with a prior p (user mean for user side effects, movie mean otherwise) and a global mean g in the following way.

$$\hat{r}_{ui} = \frac{|N(u)| \frac{\bar{r}_{ui}|N(u)| + p\alpha}{|N(u)| + \alpha} + g\beta}{|N(u)| + \beta} \tag{19}$$

The calculation of the other effects could be done analogous to this example. Per effect, the non-negative parameters σ, ϵ, λ, α and β are set by APT2 (section 3.2). As for the global effects (GE) the ordering influences the result. We always used the global time effects exactly in the same order as in the table above. When we refer to having used the "x GTE", we mean applying the first "x global time effects" in the order from above.

17 Time Dep Model

The basic idea is to allow an over time changing rating average of a user. In order to predict this time dependent average, we use a simple convolution over time.

In the equation below, t_{ui} denotes the time (days since the first rating in the dataset) of the rating r_{ui}, and $N(u)$ stands for the set of items voted by user u.

$$\tilde{r}_{ui} = \frac{\sum_{j \in N(u)} k(t_{ui}, t_{uj}) \cdot r_{uj}}{\sum_{j \in N(u)} k(t_{ui}, t_{uj})} \tag{20}$$

$$\tau_{ui} = \sum_{j \in N(u)} k(t_{ui}, t_{uj}) \tag{21}$$

In our models we make use of 3 different time kernels:

- linear: $k(t_{ui}, t_{uj}) = \{ {}^{1 \; if \; |t_{ui} - t_{uj}| < \gamma}_{0 \; if \; |t_{ui} - t_{uj}| \geq \gamma}$
- Gauss: $k(t_{ui}, t_{uj}) = exp\left(-\frac{(t_{ui} - t_{uj})^2}{\gamma}\right)$
- near Gauss: $k(t_{ui}, t_{uj}) = exp\left(-\frac{|t_{ui} - t_{uj}|}{\gamma}\right)$

The ratings of a user are in general not equally distributed over time. By predicting a rating with no others in the near past or future, the denominator τ_{ui} gets very small, leading to badly defined averages. The smaller the time window γ gets, the more often there is an insufficient number of ratings in the neighborhood. For this reason, we softly blend to the customer average and to the global average for small values of τ_{ui}.

We calculate the user averages μ_u and softly blend to the global rating average μ based on $|N(u)|$ the number of ratings.

$$\tilde{\mu}_u = \frac{\mu_u \cdot |N(u)| + \alpha \cdot \mu}{|N(u)| + \alpha} \tag{22}$$

For small values of τ_{ui} we adhere to $\tilde{\mu}$:

$$\hat{r}_{ui} = \frac{\tilde{r}_{ui} \cdot \tau_{ui} + \beta \cdot \tilde{\mu}_u}{\tau_{ui} + \beta} \tag{23}$$

Finding good parameters for α, β and γ by hand is very difficult, so we use a simple random search method as described in section 3.1, to minimize the RMSE on the probe set. The exact values for all meta parameters used to create our predictions can be found, in the predictor listing (section 20). We refer to this method as [CTD]. We also use a flipped version of this to cover changing movie averages [MTD].

18 Neural Network (NN)

A neural network for rating prediction is discussed in [5]. We adopted the same idea, showing that it is very suitable for accurate and fast prediction. Training is done by applying the backprop rule for weight updates. As discussed in [5], the neural network architecture is equivalent to SVD, when using a linear activation function per neuron. By using tanh as activation function, the net is able to manage saturation effects on the output. In general, the NN architecture for rating prediction has around 0.5 million input neurons (480189 + 17770) and 17770 output neurons. In the case of rating prediction there is only one user and one movie active, so training complexity reduces to SVD complexity. Therefore, in the hidden layer, large numbers of neurons are manageable. We also tried multilayer nets, but they are

not working as well on the probe set, as the 1-layer ones in terms of RMSE. In order to receive a non negative version, negative weights are simply set to 0. For regularization we use weight decay, which is equivalent to L2-regularization in SVD models. The weight update rule is:

$$\mathbf{w}^{new} = \mathbf{w} - \eta \left(\frac{\partial E(\mathbf{w}, \mathbf{x})}{\partial \mathbf{w}} + \lambda \cdot \mathbf{w} \right) \tag{24}$$

$\mathbf{w}$ are the weights before update and $\mathbf{w}^{new}$ are the new weights; $\mathbf{x}$ is the input vector. The backprop algorithm [3] calculates the partial error derivative $\frac{\partial E(\mathbf{w}, \mathbf{x})}{\partial \mathbf{w}}$. η is the learning rate and λ is the regularization coefficient. Common values are $\eta = 0.01$ and $\lambda = 0.01$; we report the exact values in the listing section (20).

19 NN Blending (NNBlend)

In general a neural network is a function approximator from a N-dimensional input space to a M-dimensional output space $\mathbb{R}^N \rightarrow \mathbb{R}^M$. The structure in a neural network is layer-oriented. It consists of input, hidden and an output layer. The calculation of the outputs is performed layerwise from inputs to outputs. It is also called feedforward net.

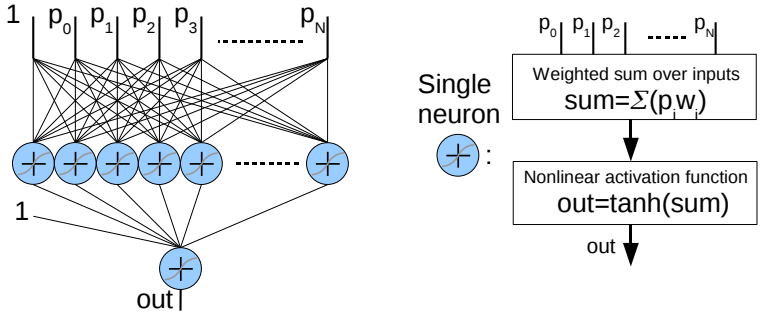

Figure 1: Schema of predictor blending with a 1-layer neural network. The net takes a constant bias and the N predictions as input. The output is given by the evaluation of the neural network with the trained weights.

We use a net with one hidden layer, the schematic diagram is given by Figure 1. At a first stage, all of our predictors, listed in section 20, are trained without using the probe set. The training list of the NN Blending consists of 1408395 ratings, which is the size of the probe set. Training of the net is done with the well known backpropagation algorithm (see [3]). Stochastic gradient descent minimizes the training error on the whole training set. We are not using a testset or regularization techniques like weight decay. The training stops when no significant progress (Δ_{RMSE}=3e-7) of the error between two sequent epochs is observed. Learning rate is set to η=5e-4 and is subtracted by a constant 3e-7 every training epoch. It is important to train more runs with different initial weights. Our suggestion is to train 20-40 different initial weight sets. We call this a training session. A marginal improvement (0.0001 in the qualifying score) is obtained by training two different sessions (for example with 12 and 13 neurons in the hidden layer, taking the best weights of each session and combining them in a robust way). Best outcomes from two training sessions are $\mathbf{s_0}$ and $\mathbf{s_1}$. We combine them by linear regression on the probe set to get the final prediction $\mathbf{p} = w_0 \cdot \mathbf{s_0} + w_1 \cdot \mathbf{s_0}$.

The prediction of the qualifying set is done by evaluating the net with the retrained predictions. Retraining means to insert the probe set into the training set and let all the algorithms run with same learning parameters found on the probe set (learn rate η, regularization λ, number of training epochs, feature size k, ...).

A commonly used technique of combining algorithms is estimating blending weights w_i by linear regression on the probe set (see section "Combining multiple results" in 2007 progress prize paper [2]) to get the final prediction by $\mathbf{p} = \sum_{i=1}^{N} \mathbf{p_i} w_i$. N is the number of predictions in the blend. For comparison to linear blending, the NNBlend improves the final score by 0.0020 on our set of predictions.

20 Predictor Listing

We obtained the RMSE=0.8616 solution by using a linear combination of all individual predictors created by teams BellKor and BigChaos. The predictors created by BellKor and a description of the linear combination methodology can be found in the manuscript "*The BellKor 2008 Solution to the Netflix Prize*". Below we list all predictors by team BigChaos.

20.1 Basic Predictors

1. rmse=1,0676
BASIC user mean rating

20.2 Weekday model

2. rmse=0.9888
WDM based on 1 GE, $\delta = 0.13$, $\epsilon = 9.6e-12$, $\nu = 1.28$, $\alpha = 5.7$, $\beta = 7.8$, $\gamma = 1.1e16$

20.3 Basic SVD

3. rmse=0,9028
BasicSVD k=300, $\eta = 0.001$, $\lambda = 0.011$, $\alpha = 1.9$, on residuals of 1 GE

4. rmse=0,9066
BasicSVD k=300, $\eta = 0.001$, $\lambda = 0.011$, $\alpha = 1.9$, on residuals of MTD (#87)

5. rmse=0,9028
BasicSVD k=300, $\eta = 0.001$, $\lambda = 0.011$, $\alpha = 1.9$, find probe minima on users with more than 500 votes, on residuals of 1 GE

6. rmse=0,9045
BasicSVD k=380, $\eta = 0.001$, $\lambda = 0.02$, $\alpha = 2.0$, nonNegWeights

7. rmse=0,9275
BasicSVD k=200, η=0.002, $\lambda = 0.012$, $\alpha = 1.0$, on residuals of 1 GE

8. rmse=0,9170
BasicSVD k=200, η = 2e-4, $\lambda = 0.005$, $\alpha = 2.0$, nonNegWeights

9. rmse=0,9143
BasicSVD k=300, $\eta = 0.001$, $\lambda = 0.008$, $\alpha = 2.0$, on 0 GE

20.4 SVD adaptive user factors

10. rmse=0.9592
SVD-AUF based on SVD-ALS with k=30, $\eta = 0.005$, $\lambda = 0.045$, nonNegSolver, time similarity (α=0.5, β=0.01)

20.5 SVD alternating least squares

11. rmse=0.9038
SVD-ALS k=128, $\eta = 0.004$, $\lambda = 0.045$, nonNegSolver

20.6 Time SVD

12. rmse=0.9842
TimeSVD non negative features, k=45, T=6, $\eta = 0.002$, $\lambda = 0.2$, no movie time offset

13. rmse=0.9968
TimeSVD on 0GE(global mean), k=91, T=10, η=0.005, λ=0.1

14. rmse=0.9378
TimeSVD on 0GE, k=91, T=10, , η=0.003, λ=0.0125, no user time offset

15. rmse=0.9688
TimeSVD on 0GE, k=91, T=10, η=0.005, λ=0.1, no movie time offset

16. rmse=1.0419
TimeSVD k=200, T=100, η=0.005, λ=0.02, no movie time offset

20.7 Neighborhood Aware Matrix Factorization

17. rmse=0,8988
NAMF 300 dim, η=0.004, $\lambda = 0.02$, bestCorr=50 [Pearson on 10GE], on residuals of 1 GE

18. rmse=0,8981
NAMF 600 dim, η=0.002, $\lambda = 0.02$, bestCorr=50 [Pearson on 10GE], on residuals of 1 GE

19. rmse=1,0614
NAMF, due to a an error in the code this predictor is wrong (probe RMSE=0.9275), accidentally this predictor was included in one NNBlend (#98)

20.8 RBMV3

20. rmse=0,9229
RBMV3 with 30 hidden units, $\eta = 0.002$, $\lambda = 0.0002$

21. rmse=0,9057
RBMV3 with 150 hidden units, $\eta = 0.001$, $\lambda = 0.0002$

22. rmse=0,9162
RBMV3 with 50 hidden units, $\eta = 0.002$, $\lambda = 0.0002$

23. rmse=0,9461
RBMV3 with 10 hidden units, $\eta = 0.002$, $\lambda = 0.0002$

24. rmse=0,9101
RBMV3 with 100 hidden units, $\eta = 0.002$, $\lambda = 0.0002$

25. rmse=0,9063
RBMV3 with 150 hidden units, $\eta = 0.001$, $\lambda = 0.00025$

20.9 RBMV5

26. rmse=0,9070
RBMV5 with 200 hidden units, $\eta = 0.0002$, $\lambda = 0.0006$, on residuals of AFM (#68)

27. rmse=0,9051
RBMV5 with 300 hidden units, $\eta = 0.0003$, $\lambda = 0.0007$, on residuals of 6 GE

28. rmse=0,9117
RBMV5 with 50 hidden units, $\eta = 0.0001$, $\lambda = 0.0004$, on residuals of 10 GE

29. rmse=0,9056
RBMV5 with 200 hidden units, $\eta = 0.0002$, $\lambda = 0.0006$, on residuals of 8 GE

30. rmse=0,9039
RBMV5 with 300 hidden units, $\eta = 0.0002$, $\lambda = 0.0007$, on residuals of 10 GTE

31. rmse=0,9044
RBMV5 with 300 hidden units, $\eta = 0.0003$, $\lambda = 0.0007$, on residuals of MTD (#87)

32. rmse=0,9045
RBMV5 with 300 hidden units, $\eta = 0.0003$, $\lambda = 0.0007$, on residuals of 18 GTE

33. rmse=0,9054
RBMV5 with 200 hidden units, $\eta = 0.0001$, $\lambda = 0.0006$, on residuals of 6 GE

34. rmse=0,9041
RBMV5 with 300 hidden units, $\eta = 0.0002$, $\lambda = 0.0007$, on residuals of 15 GTE

35. rmse=0,9045
RBMV5 with 100 hidden units, $\eta = 0.0004$, $\lambda = 0.0004$, on residuals ROS (#67)

36. rmse=0,9066
RBMV5 with 200 hidden units, $\eta = 0.0002$, $\lambda = 0.0005$, on residuals of 10 GE

20.10 RBMV6

37. rmse=0,9008
RBMV6 with 50 hidden units, $\eta = 0.002$, $\lambda = 0.00035$, on residuals of RBMV3 (#25)

20.11 Basic Pearson KNN

38. rmse=0,9229
KNN-BASIC with k=30, $\alpha = 350$, on residuals of 14 GE

39. rmse=0,9013
KNN-BASIC with k=30, $\alpha = 412$, on NN with 50 neurons (#92)

20.12 Movie KNN

40. rmse=0,8958
KNNMovie k=80, $\alpha = 791$, $\gamma = -2.534$, $\delta = 11.4$, Pearson Correlation, on residuals of BasicSVD (#3)

41. rmse=0,9151
KNNMovieV7 k=inf, $\alpha = 337$, $\beta = 1429$, $\gamma = -2.2$, $\delta = 8.1$, $\xi = 1.0$, $\vartheta = 0.3$, $\theta = 0.0235$, Pearson Correlation, on residuals of 10 GE

42. rmse=0,9013
KNNMovieV3, k=55, $\alpha = 854$, $\beta = 495$, $\gamma = -2.47$, $\delta = 13.2$, Pearson Correlation, on residuals of RBMV5 (#31)

43. rmse=0,8942
KNNMovie, $\alpha = 707$, $\gamma = -2.272$, $\delta = 12.34$, Pearson Correlation, on residuals of RBMV5 (#28)

44. rmse=0,9042
KNNMovieV4, k=25, $\alpha = 777$, Pearson Correlation, on residuals of RBMV3 (#22)

45. rmse=0,9000
KNNMovieV4, k=35, $\alpha = 8500$, Set Correlation, on residuals of RBMV3 (#22)

46. rmse=0,8832
KNNMovieV3, k=122, $\alpha = 1784$, $\beta = 63.3$, $\gamma = -3.21$, $\delta = 1.65$, Set Correlation, on residuals of GTE (#81)

47. rmse=0,8934
KNNMovieV3, k=72, $\alpha = 380$, $\beta = 65$, $\gamma = -3.33$, $\delta = 14.8$, Pearson Correlation, on residuals of NN

with 1000 neurons

48. rmse=0,8843
KNNMovieV6, k=inf, $\gamma = -2.0$, $\beta = 28.4$, $\delta = 0.02$, $\xi = 1.6$, $\theta = 1.97$, $\vartheta = 2.6$, on residuals of MTD (#86)

49. rmse=0,8905
KNNMovieV3, k=55, $\alpha = 1013$, $\beta = 408$, $\gamma = -2.28$, $\delta = 13.6$, Pearson Correlation, on residuals of RBMV6 (#37)

50. rmse=0,8987
KNNMovieV3, k=34, $\alpha = 292$, $\beta = 788$, $\gamma = -2.48$, $\delta = 7.8$, Pearson Correlation, on residuals of AFM (#69)

51. rmse=0,8900
KNNMovieV3, k=55, $\alpha = 854$, $\beta = 495$, $\gamma = -2.47$, $\delta = 13.2$, Pearson Correlation, on residuals of RBMV3 with 150 hidden units

52. rmse=0,9102
KNNMovieV3, k=55, $\alpha = 644$, $\beta = 1171$, $\gamma = -2.55$, $\delta = 8.1$, Pearson Correlation, on residuals of 19 GTE (#80)

53. rmse=0,8852
KNNMovieV3, k=231, $\alpha = 471$, $\beta = 67$, $\gamma = 0.12$, $\delta = 4.5$, Set Correlation, on residuals of MTD (#86)

54. rmse=1,0241
KNNMovieV3, k=220, $\alpha = 450$, $\beta = 71$, $\gamma = 0.12$, $\delta = 4.5$, Set Correlation, on raw ratings

55. rmse=0,9948
KNNMovieV3, k=55, $\alpha = 910$, $\beta = 451$, $\gamma = -2.28$, $\delta = 13.6$, Pearson Correlation, on raw ratings

56. rmse=0,8915
KNNMovieV3, k=49, $\alpha = 851$, $\beta = 491$, $\gamma = -2.47$, $\delta = 13.2$, Pearson Correlation, on residuals of RBMV3 (#22)

57. rmse=0,8929
KNNMovieV3, k=42, $\alpha = 861$, $\beta = 495$, $\gamma = -2.47$, $\delta = 13.2$, Pearson Correlation, on residuals of RBMV3 (#20)

58. rmse=0,9112
KNNMovieV4, k=50, $\alpha = 777$, Pearson Correlation, on residuals of 10 GE

59. rmse=0,9166
KNNMovieV3, k=62, $\alpha = 600$, $\beta = 507$, $\gamma = -2.38$, $\delta = 10.8$, Pearson Correlation, on residuals of 18 GTE (#77)

20.13 Regression on Similarity

60. rmse=0,9311
ROS factorized, on movie-side, k=20, η=2e-3, λ=1e-4, on 14 GE

61. rmse=0,8975
ROS factorized, on user-side, k=50, η=1e-4, λ=1e-2, on NN with 500 neurons

62. rmse=0.9300
ROS untrained, similarity matrix initalized by pearson correlation, additional unknown ratings are those from 10 GE, η=1, λ =2e-4, on 10 GE

63. rmse=0,9304
ROS untrained, similarity matrix initalized by pearson correlation, additional unknwon ratings are taken from (#62), η=1, λ =2e-4, on 10 GE

64. rmse=0,9390
ROS factorized, on user-side, k=10, η=1e-4, λ=1e-2, on 10 GE

65. rmse=0,8970
ROS, η=1, λ =1e-4, on NN with 50 neurons

66. rmse=0,9423
ROS factorized, on movie-side, k=20, η=2e-3, λ=1e-4, on 1 GE

67. rmse=0,9226
ROS, η=1, λ =1e-4, on 14 GE

20.14 Asymmetric Factor Model

68. rmse=0,9462
AFM with k=5, η=0.01, λ=0.005, on 6 GE

69. rmse=0.9366
AFM with k=10, η=0.002, λ=0.01, on 10 GE

70. rmse=0,9301
AFM with k=30, η=0.005, λ=0, on 19 GTE (#80)

20.15 Global Effects

71. rmse=0,9715
GE the first 6 global effects

72. rmse=0,9616
GE the first 10 global effects

73. rmse=0,9818
GE the first 2 global effects

20.16 Global Time Effects

74. rmse=0,8850
GTE 4 effects on RBMV3+KNN (#51)

75. rmse=1,0475
GTE 1 effects

76. rmse=0,8849
GTE 5 effects on RBMV3+KNN (#51)

77. rmse=0,9482
GTE 18 effects, without leaving out the rating itself in prediction of training ratings (results in narrow higher RMSE)

78. rmse=0,9559
GTE 9 effects, without leaving out the rating itself in prediction of training ratings

79. rmse=0,9550
GTE 10 effects, without leaving out the rating itself in prediction of training ratings

80. rmse=0,9450
GTE 19 effects

81. rmse=0,8835
GTE 16 effects on RBMV3+KNN (#51)

20.17 Customer Time Dep Model

82. rmse=0,8872
CTD Near Gaussian Weight, $\alpha = 354.7$, $\beta = 9.2$, $\gamma = 11.8$, on residuals of KNNMovie on RBMV5 (200 hidden units)

83. rmse=0,8861
CTD Linear Weight, $\alpha = 30.4$, $\beta = 85.4$, $\gamma = 102.9$, on residuals of CTD (#84)

84. rmse=0,8865
CTD Near Gaussian Weight, $\alpha = 302.7$, $\beta = 11.1$, $\gamma = 7.7$, on residuals of KNNMovieV3 on RBMV3 (100 hidden units)

85. rmse=0,8834
CTD Near Gaussian Weight, $\alpha = 6415.6$, $\beta = 36.7$, $\gamma = 17.5$, on residuals of GTE (#81)

20.18 Movie Time Dep Model

86. rmse=0,8845
MTD Near Gaussian Weight, $\alpha = 0.18$, $\beta = 89.3$, $\gamma = 285.1$, on residuals of CTD (#83)

87. rmse=0,9695
MTD Gaussian Weight, $\alpha = 0.056$, $\beta = 227.3$, $\gamma = 52.8$, on residuals of CTD on raw ratings

20.19 Neural Network

88. rmse=0,9424
NN, 2-layer (10,10 neurons), η=0.075, λ=6.66e-4

89. rmse=0,9130
NN 1-layer (30 neurons) based on a chain of algorithms, first 6GE, second NN non negative 1-layer(5 neurons), third ROS, η=0.015, λ=6.66e-3

90. rmse=0,9300
NN, 2-layer (10,10 neurons), η=0.075, η_{inner}=0.015 (learnrate for inner connection weights), λ=6.66e-4

91. rmse=0,9178
NN, 1-layer nonNegative (50 neurons), η=0.015, λ=6.66e-3

92. rmse=0,9041
NN, 1-layer (50 neurons), η=0.015, λ=6.66e-3

93. rmse=0,9085
NN, 1-layer nonNegative (100 neurons), η=0.015, λ=6.66e-3

94. rmse=0,9097
NN, 1-layer (20 neurons), η=0.015, λ=5e-3

95. rmse=0,9166
NN, 1-layer (10 neurons), η=0.015, λ=5e-3

20.20 NN Blending

96. rmse=0.8638
NNBlend, all results (#1 .. #18, #20 .. #95 + 17 results from BellKor)

97. rmse=0.8657
NNBlend, all results (#1 .. #18, #20 .. #95)

98. rmse=0.8658
NNBlend, all results (#1 .. #95)

99. rmse=0.8681
NNBlend, 17 results from BellKor

100. rmse=0.8691
NNBlend, 15 results from BellKor

21 Appendix

Detailed results from the algorithm Global Time Effects (GTE) are reported on raw ratings (see section 16). Here we list the parameters after the optimization per effect with APT2 on residuals of a discrete RBM, postprocessed by a movie KNN (result #51) with probe RMSE 0.8970. The parameter β from equation (19) is set to 0. Following predictions from the listing correspond to this table: #74 = Nr.4, #76=Nr.5, #81=Nr.16.

Nr.	Name	RMSE probe	λ	σ	ϵ	α
0	global Avg	0.8967	1.17e8	5e-2	1.02e-10	NA
1	movie Effect	0.8955	1.3e1	2.94e1	7.58e-3	3.46e-3
2	user Effect	0.8933	2.56e1	2.95e-1	1.34e-2	3.43e-1
2.1	movie Effect	0.8932	1.43	2.79e1	2.39e-1	1.13e-8
2.2	user Effect	0.8930	4.72e1	3.54e1	4.88e-8	9.69e-2
3	user x time(user)	0.8925	3.79e2	6.26e4	9.04-e3	6.9e-5
4	user x time(movie)	0.8924	3.06e2	4.09e2	2.42e-2	1.84e-5
5	movie x time(movie)	0.8922	4.82	2.38e1	3.71e-5	7.61e-3
6	movie x time(user)	0.8921	8.76e2	6.69e1	8.45e-2	4.1e-4
7	user x avg(movie)	0.8917	3.01e1	1.75e2	1.76e-4	2.29e-6
8	user x support(movie)	0.8916	7.98e11	1.77e3	7.94e-3	2.35e-2
9	movie x avg(user)	0.8914	5.51e-3	6.12e3	2.87e-2	1.67e-4
10	movie x support(user)	0.8913	9.98e2	4.88e1	8.95e-4	3.65e-5
11	movie x avgMovieYear(user)	0.8911	4.76e3	2.25e3	2.76e-4	5.7e-3
12	user x year(movie)	0.8910	3.41e4	5.69e2	3.13e-3	1.1e-3
13	movie x stddev(user)	0.8909	2.78e2	1.03	1e-4	3.33e-9
14	movie x percentSingleVotes (user)	0.8908	1.15e1	1.85e1	2.38e-3	1.168e-4
15	movie x ratingDateDensity (user)	0.8908	4.73e6	3.17	8.4e-8	1.3e-6
16	movie x avgStringlenTitle (user)	0.8907	2.05e4	4.24e-1	1.15e-1	3.32e-7

References

[1] R. Bell and Y. Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *IEEE International Conference on Data Mining*. KDD-Cup07, 2007.

[2] R. M. Bell, Y. Koren, and C. Volinsky. The BellKor solution to the Netflix prize. Technical report, AT&T Labs - Research, October 2007.

[3] Y. LeCun, L. Bottou, G. Orr, and K. Muller. Efficient backprop. In G. Orr and M. K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998.

[4] R. Salakhutdinov, A. Mnih, and G. E. Hinton. Restricted boltzmann machines for collaborative filtering. In *ICML*, pages 791–798, 2007.

[5] G. Takács, I. Pilászy, B. Németh, and D. Tikk. On the gravity recommendation system. In *KDD Cup Workshop at SIGKDD 07*, pages 22–30, August 2007.

[6] A. Töscher, M. Jahrer, and R. Legenstein. Improved neighborhood-based algorithms for large-scale recommender systems. In *KDD Workshop at SIGKDD 08*, August 2008.