The BellKor Solution to the Netflix Grand Prize

Yehuda Koren
August 2009

I. Introduction

This article describes part of our contribution to the "BellKor's Pragmatic Chaos" final solution, which won the Netflix Grand Prize. The other portion of the contribution was created while working at AT&T with Robert Bell and Chris Volinsky, as reported in our 2008 Progress Prize report [3]. The final solution includes all the predictors described there. In this article we describe only the newer predictors.

So what is new over last year's solution? First we further improved the baseline predictors (Sec. III). This in turn improves our other models, which incorporate those predictors, like the matrix factorization model (Sec. IV). In addition, an extension of the neighborhood model that addresses temporal dynamics was introduced (Sec. V). On the Restricted Boltzmann Machines (RBM) front, we use a new RBM model with superior accuracy by conditioning the visible units (Sec. VI). The final addition is the introduction of a new blending algorithm, which is based on gradient boosted decision trees (GBDT) (Sec. VII).

II. Preliminaries

The Netflix dataset contains more than 100 million date-stamped movie ratings performed by anonymous Netflix customers between Dec 31, 1999 and Dec 31, 2005 [4]. This dataset gives ratings about $m = 480,189$ users and $n = 17,770$ movies (aka, items).

The contest was designed in a training-test set format. A Hold-out set of about 4.2 million ratings was created consisting of the last nine movies rated by each user (or fewer if a user had not rated at least 18 movies over the entire period). The remaining data made up the training set. The Hold-out set was randomly split three ways, into subsets called Probe, Quiz, and Test. The Probe set was attached to the training set, and labels (the rating that the user gave the movie) were attached. The Quiz and Test sets made up an evaluation set, which is known as the Qualifying set, that competitors were required to predict ratings for. Once a competitor submits predictions, the prizemaster returns the root mean squared error (RMSE) achieved on the Quiz set, which is posted on a public leaderboard (`www.netflixprize.com/leaderboard`). RMSE values mentioned in this article correspond to the Quiz set. Ultimately, the winner of the prize is the one that scores best on the Test set, and those scores were never disclosed by Netflix. This precludes clever systems which might "game" the competition by learning about the Quiz set through repeated submissions.

Compared with the training data, the Hold-out set contains many more ratings by users that do not rate much and are therefore harder to predict. In a way, this represents real requirements for a collaborative filtering (CF) system, which needs to predict new ratings from older ones, and to equally address all users, not just the heavy raters.

We reserve special indexing letters to distinguish users from movies: for users u, v, and for movies i, j. A rating r_{ui} indicates the preference by user u of movie i. Values are ranging from 1 (star) indicating no interest to 5 (stars) indicating a strong interest. We distinguish predicted ratings from known ones, by using the notation $\hat{r}_{ui}$ for the predicted value of r_{ui}.

The scalar t_{ui} denotes the time of rating r_{ui}. Here, time is measured in days, so t_{ui} counts the number of days elapsed since some early time point. About 99% of the possible ratings are missing, because a user typically rates only a small portion of the movies. The (u,i) pairs for which r_{ui} is known are stored in the *training set* $\mathscr{K} = \{(u,i) \mid r_{ui} \text{ is known}\}$. Notice that $\mathscr{K}$ includes also the Probe set. Each user u is associated with a set of items denoted by $\mathrm{R}(u)$, which contains all the items for which ratings by u are available. Likewise, $\mathrm{R}(i)$ denotes the set of users who rated item i. Sometimes, we also use a set denoted by $\mathrm{N}(u)$, which contains all items for which u provided a rating, even if the rating value is unknown. Thus, $\mathrm{N}(u)$ extends $\mathrm{R}(u)$ by also considering the ratings in the Qualifying set.

Models for the rating data are learned by fitting the previously observed ratings (training set). However, our goal is to generalize those in a way that allows us to predict future, unknown ratings (Qualifying set). Thus, caution should be exercised to avoid overfitting the observed data. We achieve this by regularizing the learned parameters, whose magnitudes are penalized. The extent of regularization is controlled by tunable constants. Unless otherwise stated, we use L2 regularization.

This is a good place to add some words on the constants controlling our algorithms (including step sizes, regularization, and number of iterations). Exact values of these constants are determined by validation on the Probe set. In all cases but one (to be mentioned below), such validation is done in a manual greedy manner. That is, when a newly introduced constant needs to get tuned, we execute multiple runs of the algorithms and pick the value that yields the best RMSE on the Netflix Probe set [4]. This scheme does not result in optimal settings for several reasons. First, once a constant is set we do not revisit its value, even though future introduction of other constants may require modifying earlier settings. Second, we use the same constants under multiple variants of the same algorithm (e.g., multiple dimensionalities of a factorization model), whereas a more delicate tuning would require a different setting for each variant. We chose this convenient, but less accurate method, because our experience showed that over tuning the accuracy of a single predictor does

Y. Koren is with Yahoo! Research, Haifa, ISRAEL. Email: `yehuda@yahoo-inc.com`

not deliver a real contribution after being incorporated within the overall blend.

III. Baseline predictors

Collaborative filtering models try to capture the interactions between users and items that produce the different rating values. However, many of the observed rating values are due to effects associated with either users or items, independently of their interaction. A prime example is that typical CF data exhibit large user and item biases – i.e., systematic tendencies for some users to give higher ratings than others, and for some items to receive higher ratings than others.

We will encapsulate those effects, which do not involve user-item interaction, within the *baseline predictors*. Because these predictors tend to capture much of the observed signal, it is vital to model them accurately. This enables isolating the part of the signal that truly represents user-item interaction, and subjecting it to more appropriate user preference models.

Denote by μ the overall average rating. A baseline prediction for an unknown rating r_{ui} is denoted by b_{ui} and accounts for the user and item effects:

$$b_{ui} = \mu + b_u + b_i \tag{1}$$

The parameters b_u and b_i indicate the observed deviations of user u and item i, respectively, from the average. For example, suppose that we want a baseline estimate for the rating of the movie Titanic by user Joe. Now, say that the average rating over all movies, μ, is 3.7 stars. Furthermore, Titanic is better than an average movie, so it tends to be rated 0.5 stars above the average. On the other hand, Joe is a critical user, who tends to rate 0.3 stars lower than the average. Thus, the baseline estimate for Titanic's rating by Joe would be 3.9 stars by calculating $3.7 - 0.3 + 0.5$.

A way to estimate the parameters is by decoupling the calculation of the b_i's from the calculation of the b_u's. First, for each item i we set

$$b_i = \frac{\sum_{u \in \mathrm{R}(i)} (r_{ui} - \mu)}{\lambda_1 + |\mathrm{R}(i)|} . \tag{2}$$

Then, for each user u we set

$$b_u = \frac{\sum_{i \in \mathrm{R}(u)} (r_{ui} - \mu - b_i)}{\lambda_2 + |\mathrm{R}(u)|} . \tag{3}$$

Averages are shrunk towards zero by using the regularization parameters, λ_1, λ_2, which are determined by validation on the Probe set. We were using: $\lambda_1 = 25, \lambda_2 = 10$. Whenever this work refers to baseline predictors estimated in this decoupled fashion, they are denoted by $\tilde{b}_{ui}$.

A more accurate estimation of b_u and b_i will treat them symmetrically, by solving the least squares problem

$$\min_{b_*} \sum_{(u,i) \in \mathscr{K}} (r_{ui} - \mu - b_u - b_i)^2 + \lambda_3 \Big(\sum_u b_u^2 + \sum_i b_i^2\Big) . \tag{4}$$

Hereinafter, b_* denotes all user and item biases (b_us and b_is). The first term $\sum_{(u,i) \in \mathscr{K}} (r_{ui} - \mu + b_u + b_i)^2$ strives to find b_u's and b_i's that fit the given ratings. The regularizing term, $\lambda_3 (\sum_u b_u^2 + \sum_i b_i^2)$, avoids overfitting by penalizing the magnitudes of the parameters. This least square problem can be solved fairly efficiently by the method of stochastic gradient descent. In practice, we were using more comprehensive versions of (4), to which we turn now.

A. *Time changing baseline predictors*

Much of the temporal variability in the data is included within the baseline predictors, through two major temporal effects. The first addresses the fact that an item's popularity may change over time. For example, movies can go in and out of popularity as triggered by external events such as the appearance of an actor in a new movie. This is manifested in our models by treating the item bias b_i as a function of time. The second major temporal effect allows users to change their baseline ratings over time. For example, a user who tended to rate an average movie "4 stars", may now rate such a movie "3 stars". This may reflect several factors including a natural drift in a user's rating scale, the fact that ratings are given in the context of other ratings that were given recently and also the fact that the identity of the rater within a household can change over time. Hence, in our models we take the parameter b_u as a function of time. This induces a template for a time sensitive baseline predictor for u's rating of i at day t_{ui}:

$$b_{ui} = \mu + b_u(t_{ui}) + b_i(t_{ui}) \tag{5}$$

Here, $b_u(\cdot)$ and $b_i(\cdot)$ are real valued functions that change over time. The exact way to build these functions should reflect a reasonable way to parameterize the involving temporal changes.

A major distinction is between temporal effects that span extended periods of time and more transient effects. We do not expect movie likeability to fluctuate on a daily basis, but rather to change over more extended periods. On the other hand, we observe that user effects can change on a daily basis, reflecting inconsistencies natural to customer behavior. This requires finer time resolution when modeling user-biases compared with a lower resolution that suffices for capturing item-related time effects.

We start with our choice of time-changing item biases $b_i(t)$. We found it adequate to split the item biases into time-based bins, using a constant item bias for each time period. The decision of how to split the timeline into bins should balance the desire to achieve finer resolution (hence, smaller bins) with the need for enough ratings per bin (hence, larger bins). In fact, there is a wide variety of bin sizes that yield about the same accuracy. In our implementation, each bin corresponds to roughly ten consecutive weeks of data, leading to 30 bins spanning all days in the dataset. A day t is associated with an integer $\mathrm{Bin}(t)$ (a number between 1 and 30 in our data), such that the movie bias is split into a stationary part and a time changing part:

$$b_i(t) = b_i + b_{i,\mathrm{Bin}(t)} \tag{6}$$

While binning the parameters works well on the items, it is more of a challenge on the users' side. On the one hand, we would like a finer resolution for users to detect very short lived temporal effects. On the other hand, we do not expect enough ratings per user to produce reliable estimates for isolated bins. Different functional forms can be considered for parameterizing temporal user behavior, with varying complexity and accuracy.

One simple modeling choice uses a linear function to capture a possible gradual drift of user bias. For each user u, we denote the mean date of rating by t_u. Now, if u rated a movie on day t, then the associated time deviation of this rating is defined as

$$\text{dev}_u(t) = \text{sign}(t - t_u) \cdot |t - t_u|^\beta .$$

Here $|t - t_u|$ measures the number of days between dates t and t_u. We set the value of β by validation on the Probe set; in our implementation $\beta = 0.4$. We introduce a single new parameter for each user called α_u so that we get our first definition of a time-dependent user-bias:

$$b_u^{(1)}(t) = b_u + \alpha_u \cdot \text{dev}_u(t) \tag{7}$$

This simple linear model for approximating a drifting behavior requires learning two parameters per user: b_u and α_u.

The linear function for modeling the user bias meshes well with *gradual drifts* in the user behavior. However, we also observe *sudden drifts* emerging as “spikes” associated with a single day or session. For example, we have found that multiple ratings a user gives in a single day, tend to concentrate around a single value. Such an effect need not span more than a single day. This may reflect the mood of the user that day, the impact of ratings given in a single day on each other, or changes in the actual rater in multi-person accounts. To address such short lived effects, we assign a single parameter per user and day, absorbing the day-specific variability. This parameter is denoted by b_{ut}.

In the Netflix data, a user rates on 40 different days on average. Thus, working with b_{ut} requires, on average, 40 parameters to describe each user bias. It is expected that b_{ut} is inadequate as a standalone for capturing the user bias, since it misses all sorts of signals that span more than a single day. Thus, it serves as an additive component within the previously described schemes. The user bias model (7) becomes

$$b_u^{(3)}(t) = b_u + \alpha_u \cdot \text{dev}_u(t) + b_{ut} . \tag{8}$$

The discussion so far leads to the baseline predictor

$$b_{ui} = \mu + b_u + \alpha_u \cdot \text{dev}_u(t_{ui}) + b_{u,t_{ui}} + b_i + b_{i,\text{Bin}(t_{ui})} . \tag{9}$$

If used as a standalone predictor, its resulting RMSE would be 0.9605.

Another effect within the scope of baseline predictors is related to the changing scale of user ratings. While $b_i(t)$ is a user-independent measure for the merit of item i at time t, users tend to respond to such a measure differently. For example, different users employ different rating scales, and a single user can change his rating scale over time. Accordingly, the raw value of the movie bias is not completely user-independent. To address this, we add a time-dependent scaling feature to the baseline predictors, denoted by $c_u(t)$. Thus, the baseline predictor (9) becomes

$$b_{ui} = \mu + b_u + \alpha_u \cdot \text{dev}_u(t_{ui}) + b_{u,t_{ui}} + (b_i + b_{i,\text{Bin}(t_{ui})}) \cdot c_u(t_{ui}) . \tag{10}$$

All discussed ways to implement $b_u(t)$ would be valid for implementing $c_u(t)$ as well. We chose to dedicate a separate parameter per day, resulting in: $c_u(t) = c_u + c_{ut}$. As usual, c_u is the stable part of $c_u(t)$, whereas c_{ut} represents day-specific variability.

Adding the multiplicative factor $c_u(t)$ to the baseline predictor (as per (10)) lowers RMSE to 0.9555. Interestingly, this basic model, which captures just main effects disregarding user-item interactions, can explain almost as much of the data variability as the commercial Netflix Cinematch recommender system, whose published RMSE on the same Quiz set is 0.9514 [4].

B. *Frequencies*

It was brought to our attention by our colleagues at the Pragmatic Theory team (PT) that the number of ratings a user gave on a specific day explains a significant portion of the variability of the data during that day. Formally, denote by F_{ui} the overall number of ratings that user u gave on day t_{ui}. The value of F_{ui} will be henceforth dubbed a “frequency”, following PT’s notation. In practice we work with a rounded logarithm of F_{ui}, denoted by $f_{ui} = \lfloor \log_a F_{ui} \rfloor$.[1]

Interestingly, even though f_{ui} is solely driven by user u, it will influence the item-biases, rather than the user-biases. Accordingly, for each item i we introduce a term b_{if}, capturing the bias specific for the item i at log-frequency f. Baseline predictor (10) is extended to be

$$b_{ui} = \mu + b_u + \alpha_u \cdot \text{dev}_u(t_{ui}) + b_{u,t_{ui}} + (b_i + b_{i,\text{Bin}(t_{ui})}) \cdot c_u(t_{ui}) + b_{i,f_{ui}} . \tag{11}$$

We note that it would be sensible to multiply $b_{i,f_{ui}}$ by $c_u(t_{ui})$, but we have not experimented with this.

The effect of adding the frequency term to the movie bias is quite dramatic. RMSE drops from 0.9555 to 0.9278. Notably, it shows a baseline predictor with a prediction accuracy significantly better than that of the original Netflix Cinematch algorithm.

Here, it is important to remind that a baseline predictor, no matter how accurate, cannot yield personalized recommendations on its own, as it misses all interactions between users and items. In a sense, it is capturing the portion of the data that is less relevant for establishing recommendations and in doing so enables deriving accurate recommendations by subjecting other models to cleaner data. Nonetheless, we included two of the more accurate baseline predictors in our blend.

Why frequencies work?: In order to grasp the source of frequencies contribution, we make two empirical observations. First, we could see that frequencies are extremely powerful for a standalone baseline predictor, but as we will see, they contribute much less within a full method, where most of their benefit disappears when adding the user-movie interaction terms (matrix factorization or neighborhood). Second is the fact that frequencies seem to be much more helpful when used with movie biases, but not so when used with user-related parameters.

[1]Notice that F_{ui} is strictly positive whenever it is used, so the logarithm is well defined.

Frequencies help in distinguishing days when users rate a lot in a bulk. Typically, such ratings are given not closely to the actual watching day. Our theory is that when rating in a bulk, users still reflect their normal preferences. However, certain movies exhibit an asymmetric attitude towards them. Some people like them, and will remember them for long as their all-time favorites. On the other hand, some people dislike them and just tend to forget them. Thus, when giving bulk ratings, only those with the positive approach will mark them as their favorites, while those disliking them will not mention them. Such a behavior is expected towards most popular movies, which can be either remembered as very good or just be forgotten. A similar phenomenon can also happen with a negative approach. Some movies are notoriously bad, and people who did not like them always give them as negative examples, indicating what they do not want to watch. However, for the other part of the population, who liked those movies, they are not going to be remembered long as salient positive examples. Thus, when rating in bulk, long after watching the movie, only those who disliked the movie will rate it.

This explains why such biases should be associated with movies, not with users. This also explains why most of the effect disappears when adding the interaction terms, which already “understand” that the user is of the type that likes/dislikes the movie. In other words, we hypothesize that high frequencies (or bulk ratings) do not represent much change in people’s taste, but mostly a biased selection of movies to be rated – some movies are natural candidates as “bad examples”, while others are natural “good examples”. We believe that further validating our hypothesis bears practical implications. If, indeed, frequencies represent biased selection, they should be treated as capturing noise, which needs to get isolated out when making recommendations.

Finally, we should comment that a movie renter such as Netflix, might have additional data sources that complement frequencies. For example, data on time pased since actual watching date, or on whether ratings were entered in response to a given questionnaire or initiated by the user.

C. Predicting future days

Our models include day-specific parameters. We are often asked how these models can be used for predicting ratings in the future, on new dates for which we cannot train the day-specific parameters? The simple answer is that for those future (untrained) dates, the day-specific parameters should take their default value. In particular for (11), $c_u(t_{ui})$ is set to c_u, and $b_{u,t_{ui}}$ is set to zero. Yet, one wonders, if we cannot use the day-specific parameters for predicting the future, why are they good at all? After all, prediction is interesting only when it is about the future. To further sharpen the question, we should mention the fact that the Netflix Qualifying set includes many ratings on dates for which we have no other rating by the same user and hence day-specific parameters cannot be exploited.

To answer this, notice that our temporal modeling makes no attempt to capture future changes. All it is trying to do is to capture transient temporal effects, which had a significant influence on past user feedback. When such effects are identified they must be tuned down, so that we can model the more enduring signal. This allows our model to better capture the long-term characteristics of the data, while letting dedicated parameters absorb short term fluctuations. For example, if a user gave many higher than usual ratings on a particular single day, our models discount those by accounting for a possible day-specific good mood, which does not reflects the longer term behavior of this user. This way, the day-specific parameters accomplish a kind of data cleaning, which improves prediction of future dates.

D. What’s in the blend?

The RMSE=0.9555 result of model (10) is included in the blend. To learn the involved parameters, b_u, α_u, b_{ut}, b_i, $b_{i,\text{Bin}(t)}$, c_u, and c_{ut} one should minimize the regularized squared error on the training set. Learning is done by a stochastic gradient descent algorithm running for 30 iterations. We use separate learning rate (step size) and regularization (weight decay) on each kind of learned parameter, by minimizing the cost function

$$\min_{b_*,c_*,\alpha_*} \sum_{(u,i)\in\mathscr{K}} \Big(r_{ui} - \mu - b_u - \alpha_u \cdot \text{dev}_u(t_{ui}) - b_{u,t_{ui}} - \\ (b_i + b_{i,\text{Bin}(t_{ui})}) \cdot (c_u + c_{u,t_{ui}})\Big)^2 + \lambda_a b_u^2 + \lambda_b \alpha_u^2 + \\ \lambda_c b_{u,t_{ui}}^2 + \lambda_d b_i^2 + \lambda_e b_{i,\text{Bin}(t_{ui})}^2 + \lambda_f (c_u - 1)^2 + \lambda_g c_{u,t_{ui}}^2 . \tag{12}$$

Actual values of the learning rates and regularization constants $(\lambda_a, \lambda_b, \ldots, \lambda_g)$ are as follows:

	b_u	b_{ut}	α_u	b_i	$b_{i,\text{Bin}(t)}$	c_u	c_{ut}
lrate $\times 10^3$	3	25e-1	1e-2	2	5e-2	8	2
reg $\times 10^2$	3	5e-1	5000	3	10	1	5e-1

Notice that regularization shrinks parameters towards zero, with one exception. The multiplier c_u is shrunk towards 1, i.e., we penalize $(c_u - 1)^2$, rather than c_u^2. Similarly, all learned parameters are initialized to zero, except c_u that is initialized to 1.

The blend also includes the result of the more accurate baseline predictor (11). In fact, this is the only case where we resorted to an automatic parameter tuner (APT) to find the best constants (learning rates, regularization, and log basis). Specifically, we were using APT1, which is described in [13]. The reason we used APT here is twofold. First, this baseline predictor component is embedded in our more comprehensive models (described later). Therefore, it is worthwhile to highly optimize it. Second, this is a small quickly-trained model. So we could easily afford many hundreds of automatic executions seeking optimal settings. Still, it is worth mentioning the benefit of APT was an RMSE reduction of (only) 0.0016 over our initial manual settings.

The parameters of the RMSE=0.9278 result of model (11) were learned with a 40-iteration stochastic gradient descent process, with the following constants governing the learning of each kind of parameter:

	b_u	b_{ut}	α_u	b_i	$b_{i,\text{Bin}(t)}$	c_u	c_{ut}	$b_{i,f_{ui}}$
lrate $\times 10^3$	2.67	2.57	3.11e-3	.488	.115	5.64	1.03	2.36
reg $\times 10^2$	2.55	.231	395	2.55	9.29	4.76	1.90	1.10e-6

The log basis, a, is set to 6.76. Later, we refer to this model as [PQ1].

IV. Matrix factorization with temporal dynamics

Matrix factorization with temporal dynamics was already described in last year's Progress Report [3], or with more detail in a KDD'09 paper [8]. The major enhancement for this year is the incorporation of the improved baseline predictors described in Sec. III.

The full model, which is known as timeSVD++ [8] is based on the prediction rule

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(p_u(t_{ui}) + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right) . \qquad (13)$$

Here, the exact definition of the time-dependent baseline predictor, b_{ui}, follows (10).

As is typical for a SVD++ model [7], we employ two sets of static movie factors: $q_i, y_i \in \mathbb{R}^f$. The first set (the q_is) is common to all factor models. The second set (the y_is) facilitates approximating a user factor through the set of movies rated by the same user, using the normalized sum $|\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j$. Different normalizations of the form $|\mathrm{N}(u)|^{-\alpha}$ could be employed. Our choice of $\alpha = \frac{1}{2}$ attempts at fixing the variance of the sum (see also [7] for more intuition on this choice.)

User factors, $p_u(t) \in \mathbb{R}^f$ are time-dependent. We modeled each of the components of $p_u(t)^T = (p_{u1}(t), \dots, p_{uf}(t))$ in the same way that we treated user biases. In particular we have found modeling after (8) effective, leading to

$$p_{uk}(t) = p_{uk} + \alpha_{uk} \cdot \mathrm{dev}_u(t) + p_{ukt} \quad k = 1, \dots, f . \qquad (14)$$

Here p_{uk} captures the stationary portion of the factor, $\alpha_{uk} \cdot \mathrm{dev}_u(t)$ approximates a possible portion that changes linearly over time, and p_{ukt} absorbs the very local, day-specific variability.

We were occasionally also using a more memory efficient version, without the day-specific portion:

$$p_{uk}(t) = p_{uk} + \alpha_{uk} \cdot \mathrm{dev}_u(t) \quad k = 1, \dots, f \qquad (15)$$

The same model was also extended with the aforementioned frequencies. Since frequency affects the perception of movies, we tried to inject frequency awareness into the movie factors. To this end we created another copy of the movie factors, for each possible frequency value. This leads to the model

$$\hat{r}_{ui} = b_{ui} + (q_i^T + q_{i,f_{ui}}^T) \left(p_u(t_{ui}) + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} .y_j \right) \qquad (16)$$

Here the definition of b_{ui} is frequency-aware following (11). Notice that while the transition to frequency-aware biases was measurably effective, the introduction of frequency-dependent movie factors was barely beneficial.

A. What's in the blend?

We included multiple variations of the matrix factorization models in the blend. All models are learned by stochastic gradient descent applied directly on the raw data, no pre- or post-processing are involved. In other words, all parameters (biases, user-factors and movie-factors) are simultaneously learned from the data. Constants (learning rates and regularization to be specified shortly) are tuned to reach lowest RMSE after 40 iterations. (Practically, one can give or take around ten iterations without a meaningful RMSE impact). However, for blending we have found that over-training is helpful. That is, we often let the algorithm run far more than 40 iterations, thereby overfitting the train data, which happens to be beneficial when blending with other predictors.

The first model is the one using rule (13), together with the more memory efficient user-factors (15). The settings controlling the learning of bias-related parameters are as described in Sec. III-D. As for learning the factors themselves (q_i, p_u and y_j), we are using a learning rate of 0.008 and regularization of 0.0015, where the learning rate decays by a multiplicative factor of 0.9 after each iteration. Finally, for α_{uk} the learning rate is 1e-5 and the regularization is 50. These same settings remain the same throughout this section. The three variants within our blend are:

1) $f = 20$, #iterations=40, RMSE=0.8914
2) $f = 200$, #iterations=40, RMSE=0.8814
3) $f = 500$, #iterations=50, RMSE=0.8815

The next model still employs rule (13), but with the more accurate user-factor representation (14). This adds one type of parameter, p_{ukt}, which is learned with a learning rate of 0.004 and regularization of 0.01. The two variants within the blend were both heavily over-trained to overfit the training data:

1) $f = 200$, #iterations=80, RMSE=0.8825
2) $f = 500$, #iterations=110, RMSE=0.8841

Finally we have our most accurate factor model, which follows (16). While main novelty of this model (over the previous one) is in the bias term, we also added the frequency-specific movie-factors $q_{i,f_{ui}}$. Their respective learning rate is 2e-5, with regularization of 0.02. The blend includes six variants:

1) $f = 200$, #iterations=40, RMSE=0.8777
2) $f = 200$, #iterations=60, RMSE=0.8787
3) $f = 500$, #iterations=40, RMSE=0.8769
4) $f = 500$, #iterations=60, RMSE=0.8784
5) $f = 1000$, #iterations=80, RMSE=0.8792
6) $f = 2000$, #iterations=40, RMSE=0.8762

Later, we refer to the model with $f = 200$ and #iterations=40 as [PQ2].

V. Neighborhood models with temporal dynamics

The most common approach to CF is based on neighborhood models. While typically less accurate than their factorization counterparts, neighborhood methods enjoy popularity thanks to some of their merits, such as explaining the reasoning behind computed recommendations, and seamlessly accounting for new entered ratings. The method described in this section is based on Sec. 5 of our KDD'09 paper [8].

Recently, we suggested an item-item model based on global optimization [7], which will enable us here to capture time dynamics in a principled manner. The static model, without temporal dynamics, is centered on the following prediction rule:

$$\hat{r}_{ui} = b_{ui} + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - \tilde{b}_{uj}) w_{ij} + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} c_{ij} \tag{17}$$

Here, the n^2 item-item weights w_{ij} and c_{ij} represent the adjustments we need to make to the predicted rating of item i, given a rating of item j. It was proven greatly beneficial to use two sets of item-item weights: one (the w_{ij}s) is related to the values of the ratings, and the other disregards the rating value, considering only which items were rated (the c_{ij}s). These weights are automatically learned from the data together with the biases. The constants $\tilde{b}_{uj}$ are precomputed according to (2)–(3).

When adapting rule (17) to address temporal dynamics, two components should be considered separately. First, is the baseline predictor portion, $b_{ui} = \mu + b_i + b_u$, which explains most of the observed signal. Second, is the part that captures the more informative signal, dealing with user-item interaction $|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - \tilde{b}_{uj}) w_{ij} + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} c_{ij}$. For the baseline part, nothing changes from the factor model, and we make it time-aware, according to either (10), or (11). The latter one adds frequencies and is generally preferred. However, capturing temporal dynamics within the interaction part requires a different strategy.

Item-item weights (w_{ij} and c_{ij}) reflect inherent item characteristics and are not expected to drift over time. The learning process should make sure that they capture unbiased long term values, without being too affected from drifting aspects. Indeed, the time-changing nature of the data can mask much of the longer term item-item relationships if not treated adequately. For instance, a user rating both items i and j high in a short time period, is a good indicator for relating them, thereby pushing higher the value of w_{ij}. On the other hand, if those two ratings are given five years apart, while the user's taste (if not her identity) could considerably change, this is less evidence of any relation between the items. On top of this, we would argue that those considerations are pretty much user-dependent – some users are more consistent than others and allow relating their longer term actions.

Our goal here is to distill accurate values for the item-item weights, despite the interfering temporal effects. First we need to parameterize the decaying relations between two items rated by user u. We adopt an exponential decay formed by the function $e^{-\beta_u \cdot \Delta t}$, where $\beta_u > 0$ controls the user specific decay rate and should be learned from the data. This leads to the prediction rule

$$\begin{aligned} \hat{r}_{ui} = & b_{ui} + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} e^{-\beta_u \cdot |t_{ui} - t_{uj}|} c_{ij} + \\ & |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} e^{-\beta_u \cdot |t_{ui} - t_{uj}|} ((r_{uj} - \tilde{b}_{uj}) w_{ij}) . \end{aligned} \tag{18}$$

The involved parameters are learned by minimizing the associated regularized squared error. Minimization is performed by stochastic gradient descent for 20–30 iterations. The model is applied directly to the raw data, so all parameters (biases and movie-movie weights) are learned simultaneously. Learning of bias-related parameters is governed by the same constants discussed in Sec. III. As for the movie-movie weights (both w_{ij} and c_{ij}), their learning rate is 0.005 with regularization constant of 0.002. Finally, the update of the exponent β_u, uses a particularly small step size of 1e-7, with regularization constant equaling 0.01.

We also experimented with other decay forms, like the more computationally-friendly $(1 + \beta_u \Delta t)^{-1}$, which resulted in the same accuracy, with an improved running time. (No need to change meta-parameters.)

As in the factor case, properly considering temporal dynamics improves the accuracy of the neighborhood model. The RMSE decreases from 0.9002 [7] to 0.8870 (see next subsection). To our best knowledge, this is significantly better than previously known results by neighborhood methods. To put this in some perspective, this result is even better than those reported [1, 2, 11, 15] by using hybrid approaches such as applying a neighborhood approach on residuals of other algorithms. A lesson is that addressing temporal dynamics in the data can have a more significant impact on accuracy than designing more complex learning algorithms.

A. *What's in the blend?*

We ran the time-aware neighborhood model, with biases following (10) for 20, 25, and 30 iterations of stochastic gradient descent. The resulting RMSEs were 0.8887, 0.8885 and 0.8887, respectively. The results with 20 and 30 iterations are in the blend.

We also tried extending (18) with a non-normalized term. This involved adding a third set of movie-movie weights, d_{ij}, as follows:

$$\begin{aligned} \hat{r}_{ui} = & b_{ui} + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} e^{-\beta_u \cdot |t_{ui} - t_{uj}|} c_{ij} + \\ & |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} e^{-\beta_u \cdot |t_{ui} - t_{uj}|} ((r_{uj} - \tilde{b}_{uj}) w_{ij}) + \\ & \sum_{j \in \mathrm{R}(u)} e^{-\gamma_u \cdot |t_{ui} - t_{uj}|} ((r_{uj} - \tilde{b}_{uj}) d_{ij}) . \end{aligned}$$

Here, we also tried to emphasize the very adjacent ratings made by the user. Therefore, the new decay-controlling constants, the γ_us, were initialized with a relatively high value of 0.5 (compared to initializing β_u with zero.) In addition, for d_{ij} we used a slower learning rate of 1e-5. Learning was done by 25 iterations of stochastic gradient descent. The result with RMSE=0.8881 is included in the blend. In retrospect, we believe that such a miniscule RMSE reduction does not justify adding a third set of movie-movie weights.

Finally, we ran the time-aware neighborhood model, with biases following (11) for 20 iterations of stochastic gradient descent. (The third set of movie-movie weights was not used.) The result of RMSE=0.8870 is included in the blend. Note that the RMSE reduction from 0.8885 to 0.8870 is solely attributed to the frequency bias term. Later, we refer to this model as [PQ3].

VI. Extensions to Restricted Boltzmann Machines

A. RBMs with conditional visible units

We extended the Restricted Boltzmann Machines (RBM) model suggested by [12]. The original work showed a significant performance boost by making the hidden units conditional on which movies the current user has rated. We have found that a similarly significant performance boost is achieved by conditioning the visible units.

Intuitively speaking, each of the RBM visible units corresponds to a specific movie. Thus their biases represent movie-biases. However, we know that other kinds of biases are more significant in the data. Namely, user-bias, single-day user bias, and frequency-based movie bias. Therefore, we add those biases to the visible units through conditional connections, which depend on the currently shown user and date.

Let us borrow the original notation [12], which uses a conditional multinomial distribution for modeling each column of the observed visible binary rating matrix V:

$$p(v_i^k = 1|h) = \frac{\exp(b_i^k + \sum_{j=1}^F h_j W_{ij}^k)}{\sum_{l=1}^5 \exp(b_i^l + \sum_{j=1}^F h_j W_{ij}^l)} \tag{19}$$

We extend this conditional distribution as follows. Let the current instance refer to user u and date t. We add a user/date bias term:

$$p(v_i^k = 1|h,u,t) = \frac{\exp(b_{ut}^k + b_u^k + b_i^k + \sum_{j=1}^F h_j W_{ij}^k)}{\sum_{l=1}^5 \exp(b_{ut}^l + b_u^l + b_i^l + \sum_{j=1}^F h_j W_{ij}^l)} \tag{20}$$

where b_u^k is a user-specific parameter, and b_{ut}^k is a user×date-specific variable. The learning rule is

$$\Delta b_u^k = \varepsilon_1(\langle v_i^k \rangle_{data} - \langle v_i^k \rangle_T), \quad \Delta b_{ut}^k = \varepsilon_2(\langle v_i^k \rangle_{data} - \langle v_i^k \rangle_T)\,.$$

We have found it convenient here to deviate from the mini-batch learning scheme suggested in [12], and to learn b_u^k and b_{ut}^k in a fully online manner. That is, we update each immediately after observing a corresponding training instance. The used learning rates are: $\varepsilon_1 = 0.0025$ and $\varepsilon_2 = 0.008$. Notice that unless otherwise stated, we use the same weight decay suggested in the original paper, which is 0.001.

Considering the significant effect of frequencies, we can further condition on them here. Let the current instance refer to user u and date t with associated frequency f. The resulting conditional distribution is as follows:

$$p(v_i^k = 1|h,u,t,f) = \frac{\exp(b_{if}^k + b_{ut}^k + b_u^k + b_i^k + \sum_{j=1}^F h_j W_{ij}^k)}{\sum_{l=1}^5 \exp(b_{if}^l + b_{ut}^l + b_u^l + b_i^l + \sum_{j=1}^F h_j W_{ij}^l)} \tag{21}$$

where b_{if}^k is a movie×frequency-specific variable. Its learning rule will be

$$\Delta b_{if}^k = \varepsilon_3(\langle v_i^k \rangle_{data} - \langle v_i^k \rangle_T)\,.$$

where $\varepsilon_3 = 0.0002$. Online learning is used as well.

When using frequencies we also employed the following modification to the visible-hidden weights, which was brought to our attention by our colleagues at the Pragmatic Theory team. Instead of using the original weights W_{ij}^k, we will use frequency-dependent weights W_{ijf}^k, which are factored as

$$W_{ijf}^k = W_{ij}^k \cdot (1 + C_{fj})\,.$$

We use online learning for the new parameters C_{fj}, with learning rate of 1e-5.

As it turned out, this extension of the weights barely improves performance when frequency-biases are already present, while being somewhat onerous in terms of running time. Thus, we are unlikely to recommend it. Still, it is part of our frequency-aware RBM implementation.

B. RBMs with day-specific hidden units

[2]Motivated by the day-specific user factor (14), we also tried to create day-specific RBM hidden units. On top of the F hidden units, we also add G day-specific units. For a user that rated on r different days, we create r *parallel* copies of the G day-specific units. All those parallel copies share the same hidden-visible weights, hidden biases, and conditional connections. Also, each parallel copy is connected only to the visible units corresponding to the ratings given in its respective day.

To put this formally, for a day-specific hidden unit indexed by j, with a corresponding rating date t, we use the indicator vector $r^t \in \{0,1\}^n$ to denote which movies the current user rated on date t. Then, the Bernoulli distribution for modeling hidden user features becomes

$$p(h_j = 1|V, r_t) = \sigma(b_j + \sum_{i=1}^n \sum_{k=1}^5 r_i^t v_i^k W_{ij}^k + \sum_{i=1}^n r_i^t D_{ij})\,. \tag{22}$$

In our implementation we used this model together with the frequency-biased RBM. All parameters associated with the day-specific units were learned in mini-batches, as their non day-specific counterparts, but with a learning rate of 0.005 and a weight decay of 0.01. Results were not encouraging, and further refinement is still needed. Still a single variant of this scheme contributes to the blend.

C. What's in the blend?

First a note on over-training. Our parameter setting made the RBM typically converge at lowest Quiz RMSE with 60–90 iterations. However, for the overall blend it was beneficial to continue overfitting the training set, and let the RBM run for many additional iterations, as will be seen in the following.

We include in the blend four variants of the RBM model following (20):

1) $F = 200$, #iterations=52, RMSE=0.8951
2) $F = 400$, #iterations=62, RMSE=0.8942
3) $F = 400$, #iterations=82, RMSE=0.8944
4) $F = 400$, #iterations=100, RMSE=0.8952

There are also two variants of the RBM with frequencies (21):

1) $F = 200$, #iterations=90, RMSE=0.8928
2) $F = 200$, #iterations=140, RMSE=0.8949

[2]An idea developed together with Martin Piotte

Later, we refer to these two models as [PQ4] and [PQ5].

Interestingly, the RMSE=0.8928 result is the best we know by using a pure RBM. If our good experience with postprocessing RBM by kNN [2] is repeatable, one can achieve a further significant RMSE reduction by applying kNN to the residuals. However, we have not experimented with this.

Finally, there is a single predictor RBM with 50 hidden units and 50 day-specific hidden units, which ran 70 iterations to produce RMSE=0.9060. Later, we refer to this model as [PQ6].

VII. GBDT Blending

A key to achieving highly competitive results on the Netflix data is usage of sophisticated blending schemes, which combine the multiple individual predictors into a single final solution[3]. This significant component was managed by our colleagues at the Big Chaos team [14]. Still, we were producing a few blended solutions, which were later incorporated as individual predictors in the final blend.

Our blending techniques were applied to three distinct sets of predictors. First is a set of 454 predictors, which represent all predictors of the BellKor's Pragmatic Chaos team for which we have matching Probe and Qualifying results [14]. Second, is a set of 75 predictors, which the BigChaos team picked out of the 454 predictors by forward selection [14]. Finally, a set of 24 BellKor predictors for which we had matching Probe and Qualifying results. Details of this set are given at the end of this section.

A. Gradient Boosted Decision Trees

While major breakthroughs in the competition were achieved by uncovering new features underlying the data, those became rare and very hard to get. As we entered the final 30 days of the competition ("last call for grand prize period"), we realized that individual predictors, even if novel and accurate, are unlikely to make a difference to the blend. We speculated that the most impact during a short period of 30 days would be achieved by exploring new blending techniques or improving the existing ones. Blending offers a lower risk path to improvement in a short time. First, unlike individual predictors, better blending is directly connected to the final result. Second, blending simultaneously touches many predictors, rather than improving one at a time. This led to the idea of employing Gradient Boosted Decision Trees, which was raised together with Michael Jahrer and Andreas Töscher. Eventually, it did indeed make a contribution to the blend, though we hoped for a more significant impact.

Gradient Boosted Decision Trees (GBDT) are an additive regression model consisting of an ensemble of trees, fitted to current residuals in a forward step-wise manner. In the traditional boosting framework, the weak learners are generally shallow decision trees consisting of a few leaf nodes. GBDT ensembles are found to work well when there are hundreds of such decision trees. Standard references are [5, 6], and a known implementation is Treenet [16].

[3]While we use here the generic term "blending", the more accurate term would be "stacked generalization".

GBDT combine a few advantages, including an ability to find non-linear transformations, ability to handle skewed variables without requiring transformations, computational robustness (e.g., highly collinear variables are not an issue) and high scalability. They also naturally lend themselves to parallelization. This has made them a good choice for several large scale practical problems such as ranking results of a search engine [9, 17], or query-biased summarization of search results [10]. In practice we had found them, indeed, very flexible and convenient. However, their accuracy lags behind that of Neural Network regressors described in [13].

There are four parameters controlling GBDT, which are: (1) number of trees, (2) size of each tree, (3) shrinkage (or, "learning rate"), and (4) sampling rate. Our experiments did not show much sensitivity to any of these parameters (exact choices are described later.)

Since GBDT can handle very skewed variables, we added to the list of predictors four additional features: user support (number of rated movies), movie support (number of rating users), frequency and date of rating (number of days passed since earliest rating in the dataset).

We applied GBDT learning on the aforementioned sets of 454 and 75 predictors. The Probe set is used for training the GBDT, which is then applied on the Qualifying set. Parameter settings are: #trees=200, tree-size=20, shrinkage=0.18, and sampling-rate=0.9. The results, which are included in the blend, are of RMSE=0.8603 (454 predictors) and RMSE=0.8606 (75 predictors).

When working with the much smaller set of 24 BellKor predictors, we used the settings: #trees=150, tree-size=20, shrinkage=0.2, and sampling-rate=1.0. The result of RMSE=0.8664 was included in the blend.

It is also beneficial to introduce a clustering of users or movies, which will allow GBDT to treat all users (or movies) of a certain kind similarly. In the past [2], we touted splitting users into bins based on their support, and applying an equal blending strategy for all users in the same bin. This is already addressed in the GBDT implementation described above, thanks to adding the user support variable to the blended features. However we can introduce additional kinds of user relationships to the scheme. For example, a matrix factorization model computes a short vector characterizing each user (a user factor). Like-minded users are expected to get mapped to similar vectors. Hence, adding such vectors to the blended feature sets will effectively allow GBDT to slice and dice the user base into subsets of similar users on which the same blending rules should be applied. The same can be done with movies.

We included in the blend three forms of this idea, all applied on the set of 24 BellKor predictors. First we added to the blended predictors features from the timeSVD++ model (16) of dimensionality $f = 20$. This way, all individual bias terms were added as features. In addition, for each movie-user pair $u-i$, we added the 20-D movie factor $(q_i + q_{i,f_{ui}})$, and the 20-D user factor $p_u(t_{ui}) + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j\in\mathrm{R}(u)} y_j$. This resulted in RMSE=0.8661.

Second, we used the 20 hidden units of an RBM as a 20-D user representation (in lieu of the timeSVD++ user representation). The movie representation is still based on the timeSVD++ model. The resulting RMSE is also 0.8661.

Finally, we added k-NN features on top of the timeSVD++ features. That is, for each $u-i$ pair, we found the top 20 movies most similar to i, which were rated by u. We added the movie scores, each multiplied by their respective similarities as additional features. Similarities here were shrunk Pearson correlations [1]. This slightly reduces the RMSE to 0.8660.

Another usage of GBDT is for solving a regression problem per movie. For each user we computed a 50-D characteristic vector formed by the values of the 50 hidden units of a respective RBM. Then, for each movie we used GBDT for solving the regression problem of linking the 50-D user vectors to the true user ratings of the movie. The result, with RMSE=0.9248, will be denoted as [PQ7] in the following description.

B. *List of BellKor's Probe-Qualifying pairs*

We list the 24 BellKor predictors which participated in the GBDT blending. Notice that many more of our predictors are in the final blend of Qualifying results (as mentioned earlier in this article). However, only for those listed below we possess corresponding Probe results, which require extra computational resources to fully re-train the model while excluding the Probe set from the training set.

Post Progress Prize 2008 predictors
Those were mentioned earlier in this document:

1) PQ1
2) PQ2
3) PQ3
4) PQ4
5) PQ5
6) PQ6
7) PQ7

Progress Prize 2008 predictors
The following is based on our notation in [3]:

8) SVD++$^{(1)}$ $(f = 200)$
9) Integrated $(f = 100,\ k = 300)$
10) SVD++$^{(3)}$ $(f = 500)$
11) First neighborhood model of Sec. 2.2 of [3] (RMSE=0.9002)
12) A neighborhood model mentioned towards the end of Sec. 2.2 of [3] (RMSE=0.8914)

Progress Prize 2007 predictors
The following is based on our notation in [2]:

13) Predictor #40
14) Predictor #35
15) Predictor #67
16) Predictor #75
17) NNMF (60 factors) with adaptive user factors
18) Predictor #81
19) Predictor #73
20) 100 neighbors User-kNN on residuals of all global effects but the last 4
21) Predictor #85
22) Predictor #45
23) Predictor #83
24) Predictor #106

One last predictor with RMSE=0.8713 is in the final blend. It is based on the blending technique described in page 12 of [3]. The technique was applied to the four predictors indexed above by: 2, 9, 12, and 13.

VIII. Concluding remarks

Granting the grand prize celebrates the conclusion of the Netflix Prize competition. Wide participation, extensive press coverage and many publications all reflect the immense success of the competition. Dealing with movies, a subject close to the hearts of many, was definitely a good start. Yet, much could go wrong, but did not, thanks to several enabling factors. The first success factor is on the organizational side – Netflix. They did a great service to the field by releasing a precious dataset, an act which is so rare, yet courageous and important to the progress of science. Beyond this, both design and conduct of the competition were flawless and non-trivial. For example, the size of the data was right on target. Much larger and more representative than comparable datasets, yet small enough to make the competition accessible to anyone with a commodity PC. As another example, I would mention the split of the test set into three parts: Probe, Quiz, and Test, which was essential to ensure the fairness of the competition. Despite being planned well ahead, it proved to be a decisive factor at the very last minute of the competition, three years later.

The second success factor is the wide engagement of many competitors. This created positive buzz, leading to further enrollment of many more. Much was said and written on the collaborative spirit of the competitors, which openly published and discussed their innovations on the web forum and through scientific publications. The feeling was of a big community progressing together, making the experience more enjoyable and efficient to all participants. In fact, this facilitated the nature of the competition, which proceeded like a long marathon, rather than a series of short sprints.

Another helpful factor was some touch of luck. The most prominent one is the choice of the 10% improvement goal. Any small deviation from this number, would have made the competition either too easy or impossibly difficult. In addition, the goddess of luck ensured most suspenseful finish lines in both 2007 Progress Prize and 2009 Grand Prize, matching best sports events.

The science of recommender systems is a prime beneficiary of the contest. Many new people became involved in the field and made their contributions. There is a clear spike in related publications, and the Netflix dataset is the direct catalyst to developing some of the better algorithms known in the field. Out of the numerous new algorithmic contributions, I would like to highlight one – those humble baseline predictors (or biases), which capture main effects in the data. While the literature mostly concentrates on the more sophisticated algorithmic aspects, we have learned that an accurate treatment of main effects is probably at least as significant as coming up with modeling breakthroughs.

Finally, we were lucky to win this competition, but recognize the important contributions of the many other contestants,

from which we have learned so much. We would like to thank all those who published their results, those who participated in the web forum, and those who emailed us with questions and ideas. We got our best intuitions this way.

References

[1] R. Bell and Y. Koren. Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights. *IEEE International Conference on Data Mining (ICDM'07)*, pp. 43–52, 2007.

[2] R. Bell, Y. Koren and C. Volinsky. The BellKor Solution to the Netflix Prize. 2007.

[3] R. Bell, Y. Koren and C. Volinsky. The BellKor 2008 Solution to the Netflix Prize. 2008.

[4] J. Bennett and S. Lanning. The Netflix Prize. *KDD Cup and Workshop*, 2007. `www.netflixprize.com`

[5] J. H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, Vol. 29 (2001), pp. 1189-1232.

[6] J. H. Friedman. Stochastic Gradient Boosting. *Computational Statistics & Data Analysis*, Vol. 38 (2002), pp. 367–378.

[7] Y. Koren. Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model. *Proc. 14th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'08)*, pp. 426–434, 2008.

[8] Y. Koren. Collaborative Filtering with Temporal Dynamics. *Proc. 15th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'09)*, pp. 447–456, 2009.

[9] P. Li, C. J. Burges and Q. Wu. Mcrank. Learning to Rank Using Multiple Classification and Gradient Boosting. *Proc. 21st Proc. of Advances in Neural Information Processing Systems*, 2007.

[10] D. Metzler and T. Kanungo. Machine Learned Sentence Selection Strategies for Query-Biased Summarization. *SIGIR Learning to Rank Workshop*, 2008.

[11] A. Paterek. Improving Regularized Singular Value Decomposition for Collaborative Filtering. *Proc. KDD Cup and Workshop*, 2007.

[12] R. Salakhutdinov, A. Mnih and G. Hinton. Restricted Boltzmann Machines for Collaborative Filtering. *Proc. 24th Annual International Conference on Machine Learning*, pp. 791–798, 2007.

[13] A. Töscher and M. Jahrer. The BigChaos Solution to the Netflix Prize 2008, 2008.

[14] A. Töscher and M. Jahrer. The BigChaos Solution to the Netflix Grand Prize, 2009.

[15] A. Töscher, M. Jahrer and R. Legenstein. Improved Neighborhood-based Algorithms for Large-Scale Recommender Systems. *KDD'08 Workshop on Large Scale Recommenders Systems and the Netflix Prize*, 2008.

[16] Treenet. Salford Systems. `www.salfordsystems.com/TreeNet.php`.

[17] Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen and G. Sun. A General Boosting Method and its Application to Learning Ranking Functions for Web Search. *Proc. 21st Proc. of Advances in Neural Information Processing Systems*, 2007.