The BellKor 2008 Solution to the Netflix Prize

Robert M. Bell
AT&T Labs - Research
Florham Park, NJ

Yehuda Koren
Yahoo! Research
Haifa, Israel

Chris Volinsky
AT&T Labs - Research
Florham Park, NJ

BellKor@research.att.com

1. Introduction

Our RMSE=0.8643[2] solution is a linear blend of over 100 results. Some of them are new to this year, whereas many others belong to the set that was reported a year ago in our 2007 Progress Prize report [3]. This report is structured accordingly. In Section 2 we detail methods new to this year. In general, our view is that those newer methods deliver a superior performance compared to the methods we used a year ago. Throughout the description of the methods, we highlight the specific predictors that participated in the final blended solution. Nonetheless, the older methods still play a role in the blend, and thus in Section 3 we list those methods repeated from a year ago. Finally, we conclude with general thoughts in Section 4.

2. New Methods

The foundations of our progress during 2008 are laid out in the KDD 2008 paper [4]. The significant enhancement of the techniques reported in that paper is accounting for temporal effects in the data. In the following we briefly review the techniques described in the paper [4], while giving extra details on how those methods can address temporal effects, and some other variants that we tried. For a deeper treatment and general background, please refer to the original paper. We assume a good familiarity with our notation at [4] and with last year's Progress Prize Report [3]. All methods described in this section are trained using standard stochastic gradient descent, which became a preferred framework for analyzing the Netflix dataset. This algorithm requires setting two constants – step size (aka, learning rate) and regularization coefficient (aka, weight decay). We derived the values of these constants manually, seeking to minimize RMSE on the Probe set. A description of the learning equations and proper constant settings are given in the original papers [4,5].

2.1 Factor models

In the paper [4] we give a detailed description of three factor models. The first one is a simple SVD with biases model as in Eq. (12) of the paper:

[2] All root mean squared error (RMSE) mentioned in this article are measured on the Netflix Quiz set.

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^T q_i$$

This model is now widely used among Netflix competitors, as evident by Netflix Prize Forum posts, and is formally described by others [6, 7]. Hereinafter, we will refer to this model as "SVD", in accordance with the terminology at [4].

The second model delivers a similar accuracy, while offering several practical advantages, as described in Eq. (13) of the paper:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(|\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

As in [4], we will refer to this model as "Asymmetric-SVD". Interestingly, it can be shown that this is a factorized neighborhood model in disguise [5]. Thus, this model bridges neighborhood and factor models.

Finally, the more accurate factor model, to be named "SVD++", is as described in Eq. (15) of [4]:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

These models are learnt using stochastic gradient descent. The variable μ is constant (mean rating in training data, e.g., $\mu = 3.7$). However, the user- and movie-biases b_u, b_i, are usually learnt from the data to improve prediction accuracy.

> A single solution in our blend is based on the SVD++ model with 60 factors. In this case, user- and movie-biases were fixed as constants, which reduces prediction accuracy and is equivalent to running SVD++ on residuals of double-centered data. This leads to RMSE=0.8966.

Accounting for temporal effects

We identify three strong temporal effects in the data:

1. Movie biases – movies go in and out of popularity over time. Several events can cause a movie to become more or less favorable. This is manifested in our models by the fact that movie bias b_i is not a scalar but a function that changes over time. This effect is relatively easy to capture, because such changes span extended amounts of time. That is, we do not expect a movie likeability to hop on a daily basis, but rather to change over more extended periods. Further, we have

relatively many ratings per movie, what allows us to model these effects adequately.

2. User biases – users 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 means that in our models we would like to take the parameter b_u as a function that changes over time. Such effects can stem from many reasons. For example, it is related to a natural drift in a user's rating scale, to the fact that ratings are given in relevance to other ratings that were given recently and also to the fact that the identity of the rater within a household can change over time. Importantly, this effect is characterized with two properties that make it hard to be captured. First, we observe the effect even at the resolution of a single day, which is the finest resolution available within the Netflix data. In other words, the effective user bias on a day can be significantly different than the user bias on the day earlier or the day after. Second difficulty stems from the fact that users are usually associated with only a handful of ratings, especially when focusing on their ratings within a single day.
3. User preferences – users change their preferences over time. For example, a fan of the "psychological thrillers" genre may become a fan of "crime dramas" a year later. Similarly, humans change their perception on certain actors and directors. Part of this effect is also related to the fact that several people may rate within the same household. This effect is modeled by taking the user factors (the vector p_u) as a function that changes over time. Once again, we need to model those changes at the very fine level of a daily basis (after all, at each new session we may receive the ratings from a different person at the household), while facing the built-in scarcity of user ratings. In fact, these temporal effects are the hardest to capture, because preferences are not as pronounced as main effects (user-bias), but are split over many factors.

Now, let us describe how those temporal effects were inserted into our models. We focus on the more accurate model, which is "SVD++". The general framework is:

$$\hat{r}_{ui}(t) = \mu + b_u(t) + b_i(t) + q_i^T \left(p_u(t) + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

Here, we predict a rating at time (or, day) *t*. Notice that the relevant parameters are now structured as time-dependent functions, which are defined as described shortly. To increase accuracy, it is important that all parameters will be learnt from the data simultaneously. In other words, biases, movie-factors and user-factors are jointly learned from the data.

As mentioned earlier, temporal effects of movie biases are easier to catch since we do not need the finest resolution there, and since there are many ratings associated with a single movie. Thus, an adequate decision would be to split the movie biases into time-based bins. We are using 30 bins, spanning all days in the dataset: from Dec 31, 1999 till Dec 31, 2005, such that each bin corresponds to about 10 consecutive weeks of data. This effectively increases the number of parameters required for describing movies biases by a

factor of 30. Each day, *t*, is associated with an integer between 1 to 30 called *Bin(t)*, such that:

$$b_i(t) = b_{i,Bin(t)}$$

While binning the parameters works well on the movies, it is more of a challenge on the user side. On 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 having enough ratings per user to produce reliable estimates for isolated bins. Different function forms can be considered for modeling temporal user behavior. Their prediction accuracy is related to the number of involved parameters. We concentrate on two simple extreme choices, as we have found that their sum, gave us almost as good results as we could get by other options that we tried. We start dealing with user biases (b_u's); user preferences (p_u's) will be treated analogously.

The first modeling choice is very concise in number of parameters, and requires adding only a single parameter per user bias. Let us first introduce some new notation. For each user *u*, let us 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:

$$dev_u(t) = \text{sign}\left(t - t_u\right) \cdot \left|t - t_u\right|^{\beta}$$

We set the value of β by cross validation to 0.4. Then, for each user we center all those time deviations, and work with the centered variables $\widehat{dev_u(t)}$. Notice that those variables are constants that are derived directly from the training data.

Now, in order to define a time dependent bias, we introduce a single new parameter for each user called α_u and get our first definition of a time-dependent user-bias:

$$b_u^{(1)}(t) = b_u + \alpha_u \cdot \widehat{dev_u(t)}$$

This offers a simple linear model that does not require adding many new parameters, but at the same time is quite limited in its flexibility. Therefore, we also resort to another extreme, the most flexible model, where we assign a single parameter per user and day such that the user biases become:

$$b_u^{(2)}(t) = b_{u,t}$$

This way, on each day a user bias is captured by an independent parameter. In the Netflix data, a user rates on 40 different days on average. Thus, working with $b_u^{(2)}(t)$ requires, on average, 40 parameters to describe each user bias (unlike $b_u^{(1)}(t)$ that required two parameters per user bias). In fact, $b_u^{(2)}(t)$ is inadequate as a standalone for capturing the user bias, since it misses all sorts of signal that span more than a single day. Thus, in practice we add it to the other kind of time-dependent user bias, obtaining:

$$b_u^{(3)}(t) = b_u^{(1)}(t) + b_u^{(2)}(t)$$

The same way we treat user biases we can also treat each component of the user preferences $p_u(t)^T = \left(p_{u1}(t), p_{u2}(t), \ldots, p_{uf}(t) \right)$. Either as:

$$p_{uk}^{(1)}(t) = p_{uk} + \alpha_{uk} \cdot \widehat{dev_u(t)} \qquad k = 1, \ldots, f$$

Or:

$$p_{uk}^{(3)}(t) = p_{uk}^{(1)}(t) + p_{uk,t} \qquad k = 1, \ldots, f$$

Notice that for the Netflix data, taking the user factors as $p_{uk}^{(3)}(t)$ requires, on average, about 42 parameters per component. This can lead to tremendous space requirements, and would render this particular variant less attractive under many real life situations. In fact, we came up with more concise models of almost the same accuracy. However, for the sake of the Netflix competition, the most elaborate description of user factors was found useful. It is interesting to comment that a simple regularized stochastic gradient descent algorithm was enough to avoid overfitting, and quite incredibly allowed us to fit many billions of parameters to the data.

We implemented the most elaborated time-dependent SVD++ model, which we henceforth dub "SVD++$^{(3)}$":

$$\hat{r}_{ui}(t) = \mu + b_u^{(3)}(t) + b_i(t) + q_i^T \left(p_u^{(3)}(t) + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

As stated earlier, all involved parameters (biases, user- and movie-factors) are learnt simultaneously, such that the model is trained directly on the raw data, without any kind of pre-processing; see [4]. Prediction accuracy slowly improves, with increasing number of factors, as shown in the following table:

f – dimensionality of factor vectors	RMSE
20	0.8893
50	0.8831
100	0.8812
200	0.8806
500	0.8801
1000	0.8798
2000	0.8795

Table 1. Accuracy of time dependent SVD++ model

The solution included in the blend is based on 2000 factors yielding RMSE=0.8795. We also implemented "lighter" variants of time-dependent SVD++, which required far less parameters, such as:

$$\hat{r}_{ui}(t) = \mu + b_u^{(1)}(t) + b_i(t) + q_i^T \left(p_u^{(1)}(t) + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

Henceforth, we will name this model "SVD++$^{(1)}$". We applied it with 100 factors and got RMSE=0.8879 (not included in the final blend). Then, we took the residuals

of this model, and smoothed them by a movie-movie neighborhood model, as described in our ICDM'2007 paper [1] (or, KDD-Cup'2007 paper [2]). This neighborhood model is denoted as [kNN] in Sec. 3 (or in [3]). We used 30 neighbors, and the RMSE of the result was 0.8842 (included in the final blend). We also included in the blend another related variant, where the inner product matrix was estimated over the residuals of an RBM, what lowered the RMSE to 0.8822.

2.2 Neighborhood models

We implemented various variants of the neighborhood model described in Sec. 3 of [4]. The basic model is based on Eq. (9) there:

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

The result of RMSE=0.9002 was included in the final blend.

Other variants of the model would use only a subset of the neighbors, and can be applied on residuals of other methods. The general formula based on Eq. (10) in [4] is:

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

Here, $\tilde{b}_{ui}$ is the prediction for the rating by user u and movie i, as estimated by some other method, and k is the number of neighbors. We will refer to this model as "GlobalNgbr".

This way we applied the kNN model (with full set of neighbors; k=17,770), to residuals of SVD++ with 60 factors (and fixed biases). The result, with RMSE=0.8906 is included in the final blend.
We also applied this method with k=2000 on residuals of global effects to obtain RMSE=0.9067 (also, within the blend).

The previous prediction rule can be easily extended to address time-dependent user biases:

$$\hat{r}_{ui}(t) = \tilde{b}_{ui} + b_u^{(3)}(t) + |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^k(i;u)} c_{ij}$$

Within the final blend, such a model (with k=35) was applied to residuals of a Restricted Boltzmann Machine (100 hidden units) to obtain an RMSE of 0.8893.

Similarly, the basic model (which works on raw ratings) can be enhanced to account for time dependent user- and movie-biases:

$$\hat{r}_{ui}(t) = \mu + b_u^{(3)}(t) + b_i(t) + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} c_{ij}$$

Another slight improvement is obtained by decaying neighbors that were rated distantly in time, by adding another term to the prediction rule:

$$\hat{r}_{ui}(t) = \mu + b_u^{(3)}(t) + b_i(t) + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} c_{ij} + \sum_{j \in \mathrm{R}(u)} \exp(\gamma \cdot \left| t - t_{uj} \right|) d_{ij}$$

Here, t_{uj} is the day in which user *u* rated item *j*, and $\left| t - t_{uj} \right|$ is the number of days between the rating of item *i* and that of item *j*. The constant γ was set to 0.5.

The result of this neighborhood model, as included in the final blend, is of RMSE=0.8914.

An alternative version of the neighborhood model used the Sigmoid function in order to aggregate the movie-movie weights, as follows:

$$\hat{r}_{ui} = \tilde{b}_{ui} + \lambda \cdot \sigma \left(\sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + \sum_{j \in \mathrm{N}^k(i;u)} c_{ij} \right)$$

The function $\sigma(x) = \left(1 + \exp(-x)\right)^{-1} - 0.5$ maps *x* to (-0.5,0.5). The parameter λ is learnt from the data together with all other parameters.

Results of this method are inferior to those of the methods described in [4] and earlier in this subsection. Nonetheless, three related results are used within the final blend. First, we applied the method to residuals of global effects with *k*=17770 (full dense set of neighbors) to obtain RMSE=0.9200. When limiting the number of neighbors using *k*=200, the resulting RMSE increases to 0.9230. The last variant was applied to residuals of RBM (200 hidden units) to yield RMSE=0.8931.

2.3 Integrated models

As we explain in [4] (Sec. 5), one can achieve better prediction accuracy by combining the neighborhood and factor models. In particular, the neighborhood model described in the previous subsection allows a symmetric treatment, where neighborhood parameters and factor parameters are learnt simultaneously. The basic model follows Eq. (16) of [4]:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T \left(p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

$$+ |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^k(i;u)} c_{ij}$$

We will later refer to this model as "Integrated". It can be enhanced to account for temporal effects, as we did with the factor models. We start with the more concise models, which require a modest addition of parameters to achieve:

$$\hat{r}_{ui}(t) = \mu + b_u^{(1)}(t) + b_i(t) + q_i^T \left(p_u^{(1)}(t) + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

$$+ |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^k(i;u)} c_{ij}$$

The result of this model with f=750 and k=300, as included in the final blend, yields RMSE=0. 8827.

In order to further improve accuracy, we employ a more elaborated temporal model for the user biases:

$$\hat{r}_{ui}(t) = \mu + b_u^{(3)}(t) + b_i(t) + q_i^T \left(p_u^{(1)}(t) + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

$$+ |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^k(i;u)} c_{ij}$$

Once again, we use a limited neighborhood size (k=300), as neighborhood models better complement factor models when they are well localized. Prediction accuracy very slowly improves when increasing the dimensionality of the factor model, as shown in the following table:

f	RMSE
200	0.8789
500	0. 8787
750	0. 8786
1000	0. 8785
1500	0. 8784

Table 2. Accuracy of an integrated model

The result with f=1500 (RMSE=0. 8784) is included in the final blend.
We also tried to integrate the neighborhood model with other factor models. Two related results are in the final blend. First, we added the neighborhood model (k=300) to an Asymmetric-SVD model (f=60), with no temporal effects. The achieved RMSE was 0.8959. Second, we added the neighborhood model (k=300) to an RBM with Gaussian visible units and 256 hidden units. The resulting RMSE was 0.8943 (once again, temporal effects were not addressed here).

We should note that we have not tried (yet) the supposedly most powerful integrated model, which addresses full temporal effects also for user preferences, by replacing $p_u^{(1)}(t)$ with $p_u^{(3)}(t)$, as follows:

$$\hat{r}_{ui}(t) = \mu + b_u^{(3)}(t) + b_i(t) + q_i^T \left(p_u^{(3)}(t) + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

$$+ |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij} + |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}^k(i;u)} c_{ij}$$

2.3 Other methods

There are two additional developments during the last year, with a very modest contribution to the final blend.

2.3.1 Shrinking towards recent actions

A possible way for accounting for temporal effects is by overweighting the more recent user actions. Indeed, this is inferior compared to the more principled approach described earlier, which could provide a full modeling of how user behavior is changing over time. Nonetheless, when holding prediction sets that have been previously computed without accounting for temporal effects, a simple correction as described below is effective.

In the following we assume that we want to correct $\hat{r}_{ui}$, which is the predicted rating for user u on item i at day t (=t_{ui}). We would like to shrink $\hat{r}_{ui}$ towards the average rating of u on day t. The rationale here is that the single day effect is among the strongest temporal effects in the data. To this end we compute several magnitudes related to the actions of user u on day t:

- n_{ut} - the number of ratings u gave on day t
- $\overline{r}_{ut}$ - the mean rating of u at day t
- V_{ut} - the variance of u's ratings at day t

This allows us to compute a confidence coefficient, related to how *u* tended to concentrate his/her rating on day *t:*

- $c_{ut} = n_{ut} \cdot \exp(-\alpha \cdot V_{ut})$

Accordingly, we shrink our estimate towards $\overline{r}_{ut}$ controlled by the confidence coefficient, so that the corrected prediction is:

$$\frac{\beta \cdot \hat{r}_{ui} + c_{ut} \cdot \overline{r}_{ut}}{\beta + c_{ut}}$$

The participating constants were determined by cross validation to be: $\alpha = 8, \beta = 11$.

We used this correction with a single solution in the final blend. First, we combined two solutions. The first one is 50 neighbors kNN on 100-unit RBM with RMSE 0.8888 (see predictor #40 in last year's Progress Prize Report [3]). Second result is by the SVD++[(1)] model with *f*=200 that yields RMSE=0.8870. In order to combine the models, we split the predictions into 15 bins based on their support and compute a separate linear combination within each bin; see [3]. Such a combination leads to an RMSE of 0.8794. Finally we correct for a single day effect to achieve RMSE=0.8788.

A stronger correction accounts for periods longer than a single day, and also tries to characterize the recent user behavior on *similar* movies. To this end we compute pairwise similarities between all movies, denoted by s_{ij}, which are defined as the square of the Pearson correlation coefficient among the ratings of the two respective movies. Now, we weight the influence between movie *i* and all other movies rated by *u*. Those weights reflect both the similarity between the movies and the time proximity between the corresponding rating events, as follows:

$$w_{ij}^{u} = s_{ij} \cdot \exp\left(-\theta \left| t_{ui} - t_{uj} \right|\right)$$

Here, we are using θ=0.2. Then we compute the following three magnitudes:

- $n_{ui} = \sum_{u \text{ rated } j} w_{ij}^{u}$
- $\overline{r}_{ui} = \dfrac{\sum_{u \text{ rated } j} w_{ij}^{u} \cdot r_{uj}}{\sum_{u \text{ rated } j} w_{ij}^{u}}$
- $V_{ui} = \dfrac{\sum_{u \text{ rated } j} w_{ij}^{u} \cdot \left(r_{uj}\right)^2}{\sum_{u \text{ rated } j} w_{ij}^{u}} - \left(\overline{r}_{ui}\right)^2$

As done previously, we use the weighted support and variance to compute a confidence coefficient:

- $c_{ut} = n_{ui} \cdot \exp(-\alpha \cdot V_{ui})$

Here, we use $\alpha = 5$. This way, the corrected score is:

$$\frac{\hat{r}_{ui} + c_{ui} \cdot \overline{r}_{ui}}{1 + c_{ui}}$$

We used this correction with three solutions in the final blend, as follows:

1. Post-process predictor #83 in last year's Progress Prize Report [3] to lower the RMSE from 0.9057 to 0.9037.
2. We applied 30 neighbors kNN on residuals of NSVD2 (200 factors) to obtain RMSE=0.8948. Then, by correcting the score the RMSE decreased to 0.8924.
3. We used a variant of the previously described neighborhood model, with mild temporal biases, as follows:

   $$\hat{r}_{ui}(t) = \mu + b_u^{(1)}(t) + b_i(t) + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} c_{ij}$$

   The resulting RMSE of 0.8964 was improved to 0.8935 by applying the correction.

2.3.2 Blending multiple solutions

Our basic scheme for blending multiple predictors is based on a linear regression model as described in [3]. Also, occasionally we blend two predictors by partitioning the ratings into 15 bins based on user- and movie-support, to allow a separate linear combination within each bin [3]. This year we added new methods for blending predictors, to which we turn now.

Assume that we have a series of *s* predictors: $r^{(k)} = \left\{ r_{ui}^{(k)} \right\}_{u,i}, k = 1, \ldots, s$. We would like to combine the *s* predictors into a blended predictor $\hat{r}$. Taking a simple linear combination turns into solving a regression problem seeking optimal values of the coefficients $a^{(1)}, \ldots, a^{(s)}$, which will minimize the Probe RMSE of $\hat{r} = \sum_{k=1}^{s} a^{(k)} \cdot r^{(k)}$. However, such an approach will assign a single, global weight to each predictor, without differentiating between the abilities of certain predictors to better model certain users or movies. Thus, we suggest introducing more coefficients: for each predictor *k* and movie *i*, we introduce the coefficient $b_i^{(k)}$. Likewise, for predictor *k* and user *u*, we introduce the coefficient $c_u^{(k)}$. Now, the combination of the *s* predictors is defined through:

$$\hat{r}_{ui} = \sum_{k=1}^{s} \left(a^{(k)} + b_i^{(k)} + c_u^{(k)} \right) \cdot r_{ui}^{(k)}$$

We train the model over the Probe set. The parameters are regularized, and globally optimal solution of the associated least squares problem can be obtained using a least squares solver. We used stochastic gradient descent (learning rate=$2 \cdot 10^{-6}$, weight decay=10^{-3}) for the training.

Within the final blend, this scheme was used once. We combined two of last year's predictors. One was NNMF (60 factors) with adaptive user factors (RMSE=0.8973). The other was an RBM (100 hidden units; RMSE=0.9087). The combined predictor has an RMSE of 0.8871.

An issue with the above combination scheme is that it requires a separate set of parameters for each user, while in the Probe set (which is the training set in this context), there are very few ratings per user, making learning those parameters unreliable. In order to avoid this, we need to borrow information across users. One way to achieve this is by assuming that the user support (number of associated ratings in the full data set) determines the relative success of a single predictor on a user. Thus, we refrain from directly parameterizing users, but refer to them through their support. As for the movies, we have more information on them in the Probe data, so we still use a separate parameter per movie. Though, in order to borrow information across movies, we additionally address movies through their support.

Let n_u be the number of ratings associated with user *u* in the training data. We perform a log-transformation setting $m_u = \log n_u$. Finally, we center the resulting values, working with $\widehat{m_u}$. We follow the same procedure for movies: Let n_i be the number of ratings associated with movie *i* in the training data. We use a log-transformation setting $m_i = \log n_i$. Finally, we center the resulting values, working with $\widehat{m_i}$. The combination of the *s* predictors is defined as:

$$\hat{r}_{ui} = \sum_{k=1}^{s} \left(a^{(k)} + b_i^{(k)} + c^{(k)} \cdot \widehat{m_i} + d^{(k)} \cdot \widehat{m_u} \right) \cdot r_{ui}^{(k)}$$

The values of the parameters are learnt by stochastic gradient descent with weight decay on the Probe data.

This blending technique is used twice within the final blend:

1. We generate an RMSE=0.8771 predictor by combining four basic predictors: (i) SimuFctr (60 factors; RMSE=0.9003), (ii) RBM (100 hidden units; RMSE=0.9087), (iii) 50 neighbors kNN on 100-unit RBM (RMSE=0.8888), (iv) SVD++[(1)] (*f*=200; RMSE=0.8870).
2. We generate an RMSE=0.8855 predictor by combining five basic predictors, which were trained *without* including the Probe set in the training data even when generating the Quiz results: (i) SVD++[(3)] (*f*=50; RMSE=0.8930), (ii) NNMF (60 factors; RMSE=0.9186), (iii) Integrated (*f*=100, *k*=300), (iv) RBM (100 hidden units; RMSE=0.9166), (v) GlobalNgbr (k=500; RMSE=0.9125).

3. Older Methods

Besides using the newer techniques described in the previous section, our solution also includes the following predictors that are based on techniques in the 2007-Progress Prize report [3]. In general, we believe that most of those techniques are inferior to the newly developed ones when considering both accuracy and efficiency.

Asymmetric factor models

1. *rmse*=0. 9286
   SIGMOID2 with k=40
2. *rmse*=0. 9383
   NSVD2 with k=40
3. *rmse*=0.9236
   NSVD1 with k=200
4. *rmse*=0.9259,
   NSVD1 with k=150
5. *rmse*=0. 9260
   NSVD1 with k=40
6. *rmse*=0. 9225
   SIGMOID1 with k=100

Regression models

7. *rmse*=0. 9223
   BIN-SVD3 based on 40 vectors
8. *rmse*=0.9212
   PCA based on top 50 PCs
9. *rmse*=0.9241
   PCA based on top 40 PCs
10. *rmse*=0.9335
    BIN-SVD-USER based on 256 vectors
11. *rmse*=0.9290
    BIN-SVD3-USER based on 65 vectors
12. *rmse*=0. 9437
    BIN-SVD-USER based on 196 vectors
13. *rmse*=0. 9610
    BIN-SVD-USER based on 100 vectors, but here we regressed residuals of double centering rather than the usual residuals of global effects
14. *rmse*=0. 9414
    BIN-SVD3-USER based on 40 vectors
15. *rmse*=0. 9067
    20 neighbors Corr-kNN on residuals of BIN-SVD-USER (60 vectors)
16. *rmse*=0.9030
    50 neighbors kNN on residuals of BIN-SVD-USER (100 vectors)

17. *rmse*=0.9269,
    PCA-USER, based on top 40 PCs
18. *rmse*=0.9302,
    STRESS with 40 coordinates per movie

Restricted Boltzmann Machines with Gaussian visible units

19. *rmse*=0.9052
    800 hidden units
20. *rmse*=0.9044
    400 hidden units
21. *rmse*=0.9056
    256 hidden units
22. *rmse*=0.9429
    100 hidden units, applied on raw data (no normalization/centering)
23. *rmse*=0.9074
    100 hidden units, on residuals of full global effects
24. *rmse*=0.9267
    256 hidden units, without conditional RBM, on residuals of full global effects

Restricted Boltzmann Machines

We use conditional RBMs as described in [5].
25. *rmse*=0.9029
    256-unit RBM
26. *rmse*=0.9029
    200-unit RBM
27. *rmse*=0.9087
    100-unit RBM
28. *rmse*=0.9093
    100-unit RBM (learning rate= .15 decaying by 0.9 each iteration)

Using RBM as a pre-processor:
29. *rmse*=0.8960
    Postprocessing residuals of 100-unit RBM with factorization
30. *rmse*=0.8905
    50 neighbors kNN on 200-unit RBM
31. *rmse*=0.8904
    40 neighbors kNN on 150-unit RBM

Matrix factorization
32. *rmse*=0. 8992
    IncFctr (80 factors), adaptive user factors by [MseSim]
33. *rmse*=0. 9070
    [Corr-kNN] applied to residuals of SimuFctr (40 factors)

34. *rmse*=0. 9050
    SimuFctr (40 factors), adaptive user factors with $s_{ij}=MSE(i,j)^{-12}$
35. *rmse*=0.9026
    Cor-kNN on residuals of NNMF (60 factors)
36. *rmse*=0.8963
    NNMF (90 factors), adaptive user factors by [MseSim]
37. *rmse*=0.8986
    NNMF (90 factors), adaptive user factors by naive [SuppSim] (where $x_{ij}=n_i \cdot n_j/n$)
38. *rmse*=0.9807
    NNMF (90 factors), adaptive user factors by $s_{ij}=MSE(i,j)^{-12}$
39. *rmse*=0.8970
    NNMF (90 factors), adaptive user factors by [SuppSim]
40. *rmse*=1.1561
    NNMF (128 factors), adaptive user factors by [MseSim], but adaptive user factors where computed with Lasso regularization, rather than Ridge regularization
41. *rmse*=0.9039
    NNMF (128 factors)
42. *rmse*=0.8955,
    NNMF (128 factors), adaptive user factors by [MseSim]
43. *rmse*=0.9072
    NNMF (60 factors)
44. *rmse*=0.9018
    NNMF (40 factors, adaptive user factors by [EditSim]
45. *rmse*=0.9426
    LassoNNMF (30 factors)
46. *rmse*=0.9327
    LassoNNMF (30 factors), adaptive user factors by [SuppSim]
47. *rmse*=0.8998
    Start with SimuFctr 60 factors, then a single GaussFctr iterations on movie side followed by many GaussFctr iterations on user side
48. *rmse*=0.9070
    Start with NNMF 90 factors, followed by many GaussFctr iterations on user side
49. *rmse*=0.9098
    Start with SimuFctr 40 factors, followed by many GaussFctr iterations on user side

Neighborhood-based model (k-NN)

Some k-NN results were already mentioned. Here, we report the rest.

50. *rmse*=0.9309
    50 neighbors Fctr-kNN on residuals of full global effects. Weights based on 10 factors computed on binary matrix
51. *rmse*=0.9037
    75 neighbors Slow-kNN on residuals of SimuFctr (50 factors)

52. *rmse*=0. 8953
    30 neighbors kNN on residuals of NNMF (180 factors)
53. *rmse*=0. 9105
    50 neighbors kNN on residuals of all global effects except the last 4
54. *rmse*=0.9496
    25 neighbors kNN on raw scores (no normalization)
55. *rmse*=0.8979
    60 neighbors kNN on residuals of NNMF (60 factors)
56. *rmse*=0.9215
    50 neighbors Bin-kNN on residuals of full global effects, neighbor selection by [CorrSim]
57. *rmse*=0.9097
    25 neighbors Fctr-kNN on residuals of NNMF (60 factors). Weights based on 10 NNMF factors
58. *rmse*=0.9290
    50 neighbors Fctr-kNN on raw scores. Weights based on 10 factors computed on binary matrix
59. *rmse*=0.9097
    100 neighbors User-kNN on residuals of NNMF (60 factors)
60. *rmse*=0.9112
    100 neighbors User-kNN on residuals of SimuFctr (50 factors)
61. *rmse*=0. 9248
    30 neighbors User-MSE-kNN on residuals of full global effects
62. *rmse*=0.9170
    Corr-kNN on residuals of full global effects
63. *rmse*=0.9079
    Corr-kNN on residuals of IncFctr (80 factors),
64. *rmse*=0.9237
    MSE-kNN on residuals of full global effects
65. *rmse*=0.9085
    Supp-kNN on residuals of SimuFctr (50 factors).
66. *rmse*=0.9110
    Supp-kNN on residuals of IncFctr (80 factors)
67. *rmse*=0.9440
    Supp-kNN on residuals of full global effects. Here, we used the more naïve similarities where xij=ni*nj/n
68. *rmse*=0.9335
    Supp-kNN on residuals of full global effects

Combinations:
Each of the following results is based on mixing two individual results. Before mixing we split the user-movie pairs into 15 bins based on their support. For each bin we compute unique combination coefficients based on regression involving the Probe set.

69. *rmse*=0.8876
    Combination of [3]'s #36 with <NNMF (60 factors) adaptive user factors by MseSim>
70. *rmse*=0.8977
    Combination of #59 with #55
71. *rmse*=0.8906
    Combination of [3]'s #45 with [3]'s #73
72. *rmse*=0.9078
    Combination of #62 with <User-kNN on raw scores>
73. *rmse*=0.8967
    Combination of [3]'s #45 with [3]'s #50
74. *rmse*=0.8957
    Combination of [3]'s #45 with with <NNMF (60 factors) adaptive user factors by MseSim>
75. *rmse*=0.9017
    Combination of #53 with <User-kNN on residuals of all global effects except last 4>
76. *rmse*=0.8937
    Combination of [3]'s #45 with #54
77. *rmse*=0.8904
    Combination of [3]'s #45 with <30 neighbors kNN on residuals of SimuFctr (50 factors)>

Imputation of Qualifying predictions:
We had predictions for the Qualifying set with RMSE of 0.8836. Then, we inserted the Qualifying set into the training set, while setting unknown scores to the RMSE= 0.8836 predictions. We tried some of our methods on this enhanced training set:

78. *rmse*=0.8952
    MSE-kNN on residuals of SimuFctr (20 factors)
79. *rmse*=0.9057
    SimuFctr (50 factors)
80. *rmse*=0.9056
    SimuFctr (20 factors), Probe set is excluded from training set
81. *rmse*=0.9093
    IncFctr (40 factors), adaptive user factors by [SuppSim]. Probe set is excluded from training set
82. *rmse*=0.9005
    MSE-kNN on residuals of IncFctr (40 factors)
83. *rmse*=0.9082
    50 neighbors kNN on residuals of global effects

Specials:

84. *rmse*=1.1263
    Take binary matrix (rated=1, not-rated=0), and estimate it by 40 factors. Using these factors, construct predictions for the Probe and Qualifying set and center the

predictions for each set. Consequently, using the probe set we learn how to regress centered true ratings on these predictions, and do the same on the Qualifying set.

5. Discussion

During the two years of analyzing the Netflix data, we have learnt several interesting lessons, which apparently are not reflected well in the prior literature. In the following we briefly discuss some of them.

Collaborative filtering methods address the sparse set of rating values. However, much accuracy is obtained by also looking at other features of the data. First is the information on which movies each user chose to rate, regardless of specific rating value ("the binary view"). This played a decisive role in our 2007 solution, and reflects the fact that the movies to be rated are selected deliberately by the user, and are not a random sample. Second important feature, which played a very significant role in our progress through 2008, is accounting for temporal effects and realizing that parameters describing the data are not static but dynamic functions. At the same time, we should mention that not all data features were found to be useful. For example, we tried to benefit from an extensive set of attributes describing each of the movies in the dataset. Those attributes certainly carry a significant signal and can explain some of the user behavior. However, we concluded that they could not help at all for improving the accuracy of well tuned collaborative filtering models.

Beyond selecting which features of the data to model, working with well designed models is also important. It seems that models based on matrix-factorization were found to be most accurate (and thus popular), as evident by recent publications and discussions on the Netflix Prize forum. We definitely agree to that, and would like to add that those matrix-factorization models also offer the important flexibility needed for modeling temporal effects and the binary view. Nonetheless, neighborhood models, which have been dominating most of the collaborative filtering literature, are still expected to be popular due to their practical characteristics - being able to handle new users/ratings without re-training and offering direct explanations to the recommendations. During our work we have found that the known heuristic-based neighborhood methods can be replaced with more profound methods (as those in Sec 2.2), which deliver much improved accuracy while retaining the useful properties of general neighborhood methods.

We were quite surprised by how many parameters can be added to a single model, while still improving prediction accuracy on the test set. In the chart below each curve corresponds to a matrix-factorization model, with an increasing number of parameters. It is evident that accuracy improves as we add more parameters to a single model, or as we move to models richer in parameters. Notice that accuracy improves even when fitting over 10 billion parameters to the dataset, which is somewhat surprising considering that the dataset contains just 100 million ratings. This indicates a complex multifaceted nature of user-movie interaction. We should remark that it is possible to build more memory

efficient models that achieve almost the same accuracy as the most complex models, but this is beyond the scope of this document.

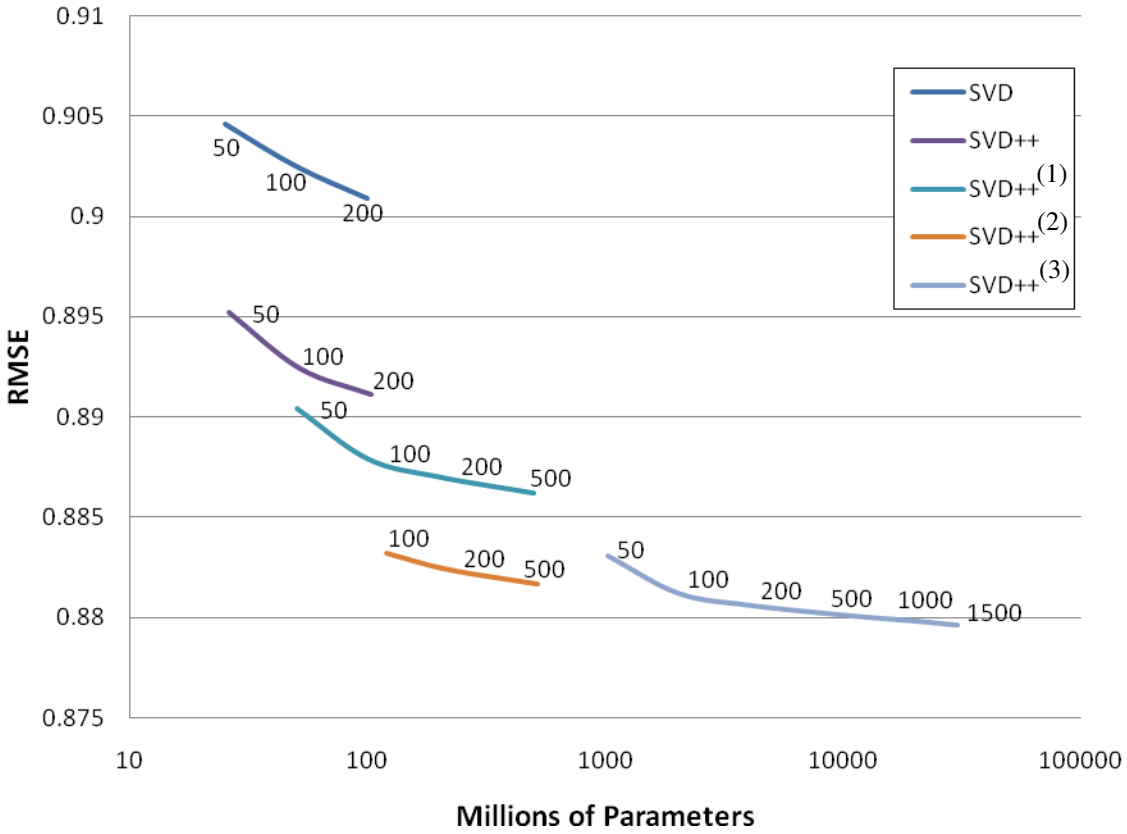

Figure 1. Matrix factorization models – error vs. #parameters. The plot shows how the accuracy of each of five individual factor models improves by increasing the number of involved parameters (which is equivalent to increasing the dimensionality of the factor model, denoted by numbers on the charts). In addition, the more complex factor models, whose descriptions involve more distinct sets of parameters, are the more accurate ones.

Finally, using increasingly complex models is only one way of improving accuracy. An apparently easier way to achieve better accuracy is by blending multiple simpler models. The chart in Fig. 2 shows how the accuracy of our final solution improves with increasing the number of blended predictors. As expected, the first few predictors have a decisive contribution to improving accuracy, while the rest have a marginal contribution. A lesson here is that having lots of models is useful for the incremental results needed to win competitions, but practically, excellent systems can be built with just a few well-selected models.

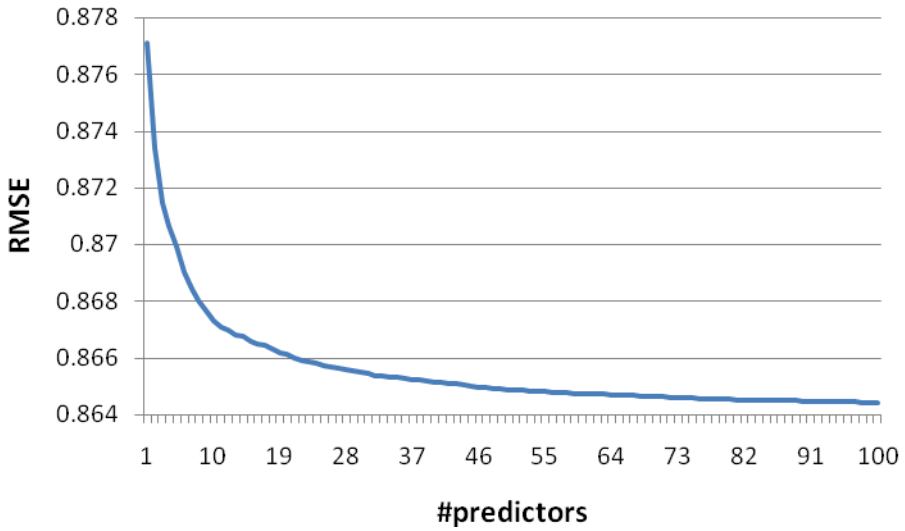

Figure 2. The plot above shows RMSE as a function of the number of methods used. By blending five predictors one can achieve RMSE=0.8699. As more predictors are added accuracy slowly improves, till reaching RMSE=0.8643 with 100 predictors.

References

1. R. Bell and Y. Koren, "Scalable Collaborative Filtering with Jointly Derived Neighborhood Interpolation Weights", *IEEE International Conference on Data Mining (ICDM'07)*, IEEE, 2007.
2. R. Bell and Y. Koren, "Improved Neighborhood-based Collaborative Filtering", *KDD-Cup and Workshop*, ACM press, 2007.
3. R. Bell and Y. Koren, and C. Volinsky, "The BellKor solution to the Netflix Prize", http://www.netflixprize.com/assets/ProgressPrize2007_KorBell.pdf, 2007.
4. Y. Koren, "Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model", *Proc. 14th ACM Int. Conference on Knowledge Discovery and Data Mining (KDD'08)*, ACM press, 2008.
5. Y. Koren, "Factor in the Neighbors: Scalable and Accurate Collaborative Filtering", http://public.research.att.com/~volinsky/netflix/factorizedNeighborhood.pdf, submitted.
6. A. Paterek, "Improving Regularized Singular Value Decomposition for Collaborative Filtering", *KDD-Cup and Workshop*, ACM press, 2007.
7. G. Takacs, I. Pilaszy, B. Nemeth and D. Tikk, "Major Components of the Gravity Recommendation System", *SIGKDD Explorations*, **9** (2007), 80-84.

Appendix A. Combining with BigChaos

We combined our solution with the one produced by the BigChaos team in order to further improve accuracy. The combined solution, of RMSE=0.8616, is a linear combination of 207 predictor sets. Each predictor set was centered to have mean zero before the combination and an estimate of the Quiz set mean was added back after the combination. Any final predictions outside the range [1, 5] were clipped. The predictor sets include 100 BigChaos predictors, including neural net blends (see accompanying progress report document); 84 predictors described in the 2007 progress report (listed in Section 3 of this document), and 23 predictors described in Section 2 of this document.

Coefficients for the linear combination were computed via an approximate linear regression on the Quiz set. We utilize the fact that a linear regression requires knowing only sufficient statistics that can be estimated from RMSEs of the Quiz predictors and other known quantities. Due to the large number of blended predictors, many of which are close to collinear, some of the estimated coefficients are very unstable without regularization. Consequently, we used ridge regression, with ridge parameter (or, "regularization constant") equaling 0.001. Computation is done by a standard least squares solver library (LAPACK).