Inside the One Million Dollars Algorithm

*July 16, 2019

Yuncong Cui
118010045

Xinyi, Zhang
118010422

Bote, Zheng
117010408

***Abstract*—Recommender systems provide users with personalized suggestions for products or services. At the same time, Netflix, an American media-services provider, is one of the most famous companies using it to improve their service. Netflix also hold a competition called Netflix Prize in order to increase by 10 percent accuracy comparing to the Cinematch, which is the current recommendation algorithm. In this work, we introduce the new algorithm from the winning team called BellKor Pragmatic Chaos to the old one from scratch. And compare it with the basic neighborhood algorithm in the text book using simulation and equation deduction. This work contains four parts, overview of data set, short answer, long answer and simulation. The short answer contains the literature review of the winning team of Netflix Prize from 2007 to 2009. The long answer contains formula deductions of major methods and comparison between the basic collaborative filtering models and the advanced ones.**

***Index Terms*—Netflix Price, neighborhood method, baseline predictor, collaborative filtering**

I. Introduction

In 2006 Netflix announced a data mining / machine learning competition in search of a model that could outperform their own "best" collaborative filtering algorithm that had been developed internally called Cinematch. The competition became known as the "Netflix Prize" and gained a lot of media attention during the two years that it ran for, with many teams from around the world competing for the grand prize – beating the own algorithm of Netflix and the allure of a $1-million-dollar prize money. It is important to recognize that this competition was years before machine learning gained the hype that we see today, years before deep learning had any profound and even before the role of the data scientist had even been coined. The Netflix Prize showcased an innovative way of open source collaboration and to bring the power of machine learning to the forefront of the business world.

II. Short Answer

In June 2009, the new team, BellKor's Pragmatic Chaos, became the first team to achieve more than 10% improvement, beating Cinematch by 10.06%, on the quiz set. At the end of this period, two teams reached the target by more than 10% on the quiz set: BellKor's Pragmatic Chaos had an RMSE of 0.8554, and The Ensemble had an RMSE of 0.8553, slightly better. The final winner was to be declared by comparing their RMSEs on the test set. Both teams beat Cinematch by more than 10% on the test set, and actually got the same RMSE on that set: 0.8567. But BellKor Pragmatic Chaos submitted their algorithm 20 minutes earlier, and thus became the winner of the grand prize. A world-class science race lasting almost three years concluded with a 20-minute differential.

In fact, the solution of the winning team was combined by many methods, with hundreds of ingredient algorithms blended together and thousands of model parameters fine-tuned specifically to the training set provided by Netflix.

A. Dataset

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. This dataset gives ratings about m = 480189 users and n = 17770 movies. A Hold-out set of about 4.2 million ratings was created consisting of the last nine movies rated by each user. 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 were attached. The Quiz and Test sets are known as the Qualifying set, that competitors were required to predict ratings for. Once a competitor submits predictions, the prize-master returns the root mean squared error (RMSE) achieved on the Quiz set. Ultimately, the winner of the prize is the one that scores best on the Test set.

The text file "qualifying.txt" is the Quiz set. It is used to test the submitted algorithm. The text file "probe.txt" is for competitors to test algorithms of themselves. Each text file in the folder train has all the rating record of each user, each rating record includes user number, movie number, rating and time.

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.

B. The Solution in 2007

These solutions are from the first paper of the winning team, which is quiet intuitive. Just like what we have learned in the class. They kept the naive baseline predictor. In the training process, they tried five ways to minimize the RMSE.

1) K-NN: First, the main algorithm is based on k-NN, which is the neighborhood method. k-NN needs to extract structures in the network of interactions among movies and users. The neighborhood method is one of the main approaches. It relies on pairwise statistical correlation.

$User - user\ correlation$: Two users with similar ratings of movies in the training set are row-wise "neighbors:" they are likely to have similar ratings for a movie in the quiz set. For example, if one of the users rates movie i with 4 stars, the other user is likely to rate it with 4 stars too.

$Movie - movie\ correlation$: Two movies that got similar ratings from users in the training set are column-wise "neighbors:" they are likely to have similar ratings by a user in the quiz set. For example, if one of the movies is rated 4 stars by user u, the other movie is likely to be rated 4 stars by this user too.

Both arguments sound intuitive. The focus is on the movie-movie correlation. Since there are so many $(u; i)$ pairs in the training data, of course we want to leverage more than just one neighbor. "Neighbor" here refers to closeness in movies' styles. Given a movie i, pick its L nearest neighbors, where "distance" among movies is measured by a similarity metric. A standard similarity metric is the cosine coefficient, where each entry is the Euclidean space of each pair of columns in the error matrix as two vectors.

Now, for a given movie i, rank all the other movies. Then the top L movies are picked as the neighbors that will count in neighborhood modeling. L is an integer between 1 and the number of movies.The predicted rating is simply the baseline predictor plus a weighted sum of the ratings from these neighbor movies (and normalized by the weights). There can be many choices for these weights. This gives out the final neighborhood prediction matrix.

2) A Factorization Model: Second, they tried factorization model. The essence of this model is alternating between computing all movie factors and all user factors, by optimally solving regularized least squares problems; also known as alternating least squares. An additional accuracy boost they used was achieved by making all factors to be nonnegative. Another addition was to use Lasso regression. In practice, they used a rich Gaussian prior when computing use factors based on their experience and experiment.

3) Restricted Boltzmann Machines: The third one was called the Restricted Boltzmann Machines. At first glace, we were confused about the name. Generally speaking, it was somehow like a reduction of dimension. Further materials will be discussed in longer answer. There was one modification that replace the multinational visible units with Gaussian ones.

4) Asymmetric Factor Models: The fourth method was the asymmetric factor models. Comparing to the factorization model with a symmetric view of users and movies. This method involved a clear redundancy, as user parameter are dependent on the movie-parameters, and vice-versa. An interesting family of models differs from the factorization model by parameterizing only the movies, which have higher support in the training data. Consequently, no explicit use factors associated with the movies liked by that user. This aggregate is often a plain sum of the respective movie factors, transformed by a function that accounts for the deviations in the number of summed values. Normalize this sum by the square root of the support of the respective user. Other than those, they introduced two more efficient related models. One was a weighted scores model and the other one was weighing with residuals. Both of them will be further discussed in the long answer.

5) Regression Models: The fifth one was the regression model. Regression can be performed in two symmetric ways: a user-centric approach and a movie-centric approach.

6) Combinations: In the end, the algorithm combines with multiple results and got the best RMSE = 0.8712.

C. The Solution in 2008

Based on the previous research, the achievement in this year complements some new ideas that can largely improve the performance.

1) Factor Models: The first factor model is a simple SVD with biases model. This model is widely used among Netflix competitors, as evident by Netflix Prize. The second model delivers a similar accuracy, while offering several practical advantages. We will refer to this model as "Asymmetric-SVD". Interestingly, it can also be shown that this is a factorized neighborhood model. Thus, this model bridges neighborhood and factor models. Finally, the more accurate factor model is named "SVD++".

These models are learnt using stochastic gradient descent. The variable μ is constant (mean rating in training data μ = 3.7). However, the user and movie biases 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. This leads to RMSE = 0.8966.

2) Global Neighbor: The new model is "Global Neighbor". In this way the k-NN model (with full set of neighbors; k = 17,770) can be applied to residuals of SVD++ with 60 factors (and fixed biases). The result, with RMSE = 0.8906 is included in the final blend. The previous prediction rule can be easily extended to address time-dependent user biases.

3) Integrated Models: 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.

In order to further improve accuracy, a more elaborated temporal model for the user biases is employed. The result with f = 1500 (RMSE = 0.8784) is included in the final blend. Two related results are in the final blend. First, the neighborhood model is added to an Asymmetric-SVD model, with no temporal effects. The achieved RMSE is 0.8959.

Second, the neighborhood model is added to an RBM with Gaussian visible units and 256 hidden units. The resulting RMSE is 0.8943 (once again, temporal effects were not addressed here).

D. The Solution in 2009

1) Baseline Predictor: Comparing to previous baseline predictor, the new baseline improved a lot. But let us start from the beginning. The purpose of the collaborative filtering models is to capture the interactions between users and items that produce the different rating values. However, we found that many of the observed rating values are due to effects associated with either users or items, independently of their interaction. In other words, systematic tendencies for some users to give higher ratings than others, and for some items to receive higher than others. Thus, the fundamental baseline predictors will not involve user-item interaction. It is important to keep it as an accurate model because these predictors tend to capture much of the observed signal. This enables isolating the part of that truly represents use-item interaction, and subjecting it to more appropriate user preference models.

Based on the fundamental one, this team did some improvement. The first one is to make it related to time. They found that 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 bi 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, some parameters were transferred to functions of time.

What's more, they also use the "Frequency" method. The meaning is 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. In order to grasp the source of frequencies contribution, they made two empirical observations. First, they could see that frequencies are extremely powerful for a standalone baseline predictor but 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.

They also predicted future days. The main idea is that for those future (untrained) dates, the day-specific parameters should take their default value. However, here is one more question that the Netflix Qualifying set includes many ratings on dates for which they have no other rating by the same user and hence day-specific parameters cannot be exploited. To solve this, the 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 their 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, their 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.

2) Neighborhood Models with Temporal Dynamics: This method is an extension of the neighborhood model. It is an item-item model based on global optimization, which will enable us here to capture time dynamics in a principled manner. The item-item weights represent the adjustments that needs to be made to the predicted rating of item i, given a rating of item j. 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 weights. 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, those considerations are pretty much user-dependent – some users are more consistent than others and allow relating their longer-term actions.

3) Matrix Factorization Model: The major enhancement in 2009 is the incorporation of the improved baseline predictors, which will be described in Long Answer.

4) Restricted Boltzmann Machines (RBM): Restricted Boltzmann Machines (RBM) model showed a significant performance boost by making the hidden units conditional on which movies the current user has rated. A similarly significant performance boost is achieved by conditioning the visible units. Intuitively speaking, each of the RBM invisible units corresponds to a specific movie. Add Namely, user-bias, single-day user bias, and frequency-based movie bias to the visible units through conditional connections, which depend on the currently shown user and date.

5) Gradient Boosted Decision Trees (GBDT): 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. GBDT ensembles are found to work well when there are hundreds of such decision trees. 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. This is already addressed in the GBDT implementation described above, thanks to adding the user support variable to the blended features. 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.

III. Long Answer

This part covers formula deductions of two main methods of collaborative filtering, which are neighborhood model and singular value decomposition (SVD). And compare baseline predictor and two models mentioned above with the lecture notes.

A. Explicit and Implicit Feedback

Define that R(u) is the explicit feedback and N(u) is the implicit feedback. The explicit feedback means that the set of users who rated item i. For movie, the implicit feedback would be movie rental history, which tells us about user preferences without requiring them to explicitly provide their ratings. However, according to the training data mentioned above, Netflix did not provide us of the rental history. Nonetheless, the data they gave us also had a deeper implication. The data implies movies that users rate, regardless of how they rated these movies. That means that the data implicitly tell us the preferences of users on the movies.

Both R(u), N(u) are binary matrices, where "1" stands for rated and "0" for not rated.

B. Baseline Predictor

1) Basic: Start from the classic baseline predictor. Denote by μ the overall average rating.

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

The parameter b_u and b_i indicate the observed deviation of user u and item i from the average. A way to estimate parameters is by decoupling the of the b_i's from the calculation of the b_u's.

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

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

Averages are shrunk towards zero by using the regularization parameters λ_1 and λ_2. It is like a penalty. It only matters if R(u) or R(i) is small. Both of them are learned from the training set. According to team BellKor, they set $\lambda_1 = 25$ and $\lambda_2 = 10$. Therefore, the least squares equation is:

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

2) Time Changing Baseline Predictors: As we mentioned above in dataset part, Netflix provided us of time data. It is a useful element to be used in prediction. There are two reasons to support it. One is that popularity of a movie is changing with time. The other one is that users change their baseline ratings over time. For example, the four stars 10 years ago is not as high as what it should be today. It maybe five stars nowadays, just like inflation. Therefore, both bias b_i and b_u should be changed to a function changing over time.

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

Let us analyze it in detail. For the term of b_i, the bias of item. First they split the timeline into time-based bins. During each time period corresponding to a single bin which has distinct item bias. In the movie rating case, movies are not expected to fluctuate on a daily basis, but rather to change over more extended periods. On the other hand, user effects can change on a daily basis, reflecting inconsistencies natural to customer behavior. This requires finer time resolution when modeling user-biases compared to a lower resolution that suffices for capturing item-related time effects. Based on their experiment, the each bin corresponds to roughly ten consecutive weeks of data, leading to an overall number of 30 bins spanning all days in the dataset.

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

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, they did not have enough ratings per user to produce reliable estimates for isolated bins. Therefore, they have

$$\mathrm{dev}_u(t) = \mathrm{sign}(t - t_u) \cdot |t - t_u|^\beta \tag{7}$$

Where the t_u is the mean date. β is the hyper parameter that need to be set by cross validation. The 2009 paper set it as 0.4.

$$b_u(t) = b_u + \alpha_u \cdot \mathrm{dev}_u(t) \tag{8}$$

Where the α_u and b_u are unique for each user. Both of them are learned from training set. Here, we call it a simple linear model for approximating a drift behavior. So far we have discussed smooth functions for modeling the user bias, which mesh well with gradual concept drift. However, in many applications there are sudden drifts emerging as "spikes" associated with a single day or session. For example, in the movie rating dataset we have found that multiple ratings a user gives in a single day, tend to concentrate around a single value. Such an effect does 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, they assign a single parameter per user and day, absorbing the day-specific variability. This parameter is denoted by $b_{u,t}$. Notice that in some applications the basic primitive time unit to work with can be shorter or longer than a day. In the Netflix movie rating data, a user rates on 40 different days on average. Thus, working with $b_{u,t}$ requires, on average, 40 parameters to describe each user bias. It is expected that $b_{u,t}$ 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.

$$b_u(t) = b_u + \alpha_u \cdot \mathrm{dev}_u(t) + b_{u,t} \tag{9}$$

Similarly, we can use the same idea to predict periodic one.

$$b_u(t) = b_u + \alpha_u \cdot \mathrm{dev}_u(t) + b_{u,t} + b_{u,\text{ period }(t)} \tag{10}$$

Generally speaking, in this case, we use the equation(9). So, we have

$$b_{ui}(t) = \mu + b_u + \alpha_u \cdot \mathrm{dev}_u(t) + b_{u,t} + b_i + b_{i,\mathrm{Bin}(t)} \tag{11}$$

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 $c_u(t)$

$$b_{ui} = \mu + b_u + \alpha_u \cdot \mathrm{dev}_u(t_{ui}) + b_{u,t_{ui}} + \left(b_i + b_{i,\mathrm{Bin}(t_{ti})}\right) \cdot c_u(t_{ui}) \tag{12}$$

3) Frequencies: 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. It means 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.

$$\begin{aligned} b_{ui} &= \mu + b_u + \alpha_u \cdot \mathrm{dev}_u(t_{ui}) + b_{u,t_{ui}} \\ &+ \left(b_i + b_{i,\mathrm{Bin}(t_{ti})}\right) \cdot c_u(t_{ui}) + b_{i,f_{ui}} \end{aligned} \tag{13}$$

Where $F_u i$ the overall number of ratings that user u gave on day t_{ui}. The value of $F_u i$ will be henceforth dubbed a "frequency", following PT's notation. In practice we work with a rounded logarithm of $F_u i$, denoted by $f_{ui} = \lfloor \log_a F_{ui} \rfloor$

4) Square Error of BP:

$$\begin{aligned} \min_{b_*, c_*, \alpha_*} \sum_{\mathcal{K}} & \left(r_{ui} - \mu - b_u - \alpha_u \cdot \mathrm{dev}_u(t_{ui}) - b_{u,t_{ui}} - \right. \\ & \left.\left(b_i + b_{i,\mathrm{Bin}(t_{ui})}\right) \cdot (c_u + c_{u,t_{ui}})\right)^2 + \lambda_a b_u^2 + \lambda_b \alpha_u^2 + \\ & \lambda_c b_{u,t_{ui}}^2 + \lambda_d b_i^2 + \lambda_e b_{i,\mathrm{Bin}(t_{ui})}^2 + \lambda_f (c_u - 1)^2 + \\ & \lambda_g c_{u,t_{ui}}^2 \end{aligned} \tag{14}$$

Therefore, we can learn b_u, α_u, b_{ut}, b_i, b_i, $\mathrm{Bin}(t)$, c_u and c_{ut} using gradient descent.

C. Neighborhood Models with Temporal Dynamics

1) Neighborhood Model: The most common approach to CF is based on neighborhood models. Let's start from basic neighborhood model. Further explanation is on text book 4.2.4. in this case, we use item-item correlation now.

$$d_{ij} = \frac{\mathbf{r}_i^T \mathbf{r}_j}{\|\mathbf{r}_i\|_2 \|\mathbf{r}_j\|_2} = \frac{\sum_u \tilde{r}_{ui} \tilde{r}_{uj}}{\sqrt{\sum_u (\tilde{r}_{ui})^2 \sum_u (\tilde{r}_{uj})^2}} \tag{15}$$

Then, we introduce a Pearson correlation coefficient to you:

$$\rho_{ij} = \frac{\sum_{u \in U_{ij}} (r_{ui} - \mu_i)(r_{uj} - \mu_j)}{\sqrt{\sum_{u \in U_j} (r_{ui} - \mu_i)^2} \cdot \sqrt{\sum_{u \in U_j} (r_{uj} - \mu_j)^2}} \tag{16}$$

Then add a penalty. Accordingly, similarities based on a greater user support are more reliable. An appropriate similarity measure, denoted by s_{ij} , would be a shrunk correlation coefficient:

$$\boldsymbol{s}_{ij} = \frac{n_{ij}}{n_{ij} + \lambda_2} \boldsymbol{\rho}_{ij} \tag{17}$$

where The variable n_{ij} denotes the number of users that rated both i and j and λ is learned from the training set.
Then the $\hat{r}_{ui}$

$$\hat{r}_{ui} = b_{ui} + \frac{\sum_{j \in \mathrm{S}^k(i;u)} s_{ij} (r_{uj} - b_{uj})}{\sum_{j \in \mathrm{S}^k(i;u)} s_{ij}} \tag{18}$$

Where we identify the k items rated by u, which are most similar to i. This set of$k$ neighbors is denoted by$S^k_{(i;u)}$. However, they questioned the suitability of a similarity measure that isolates the relations between two items, without analyzing the interactions within the full set of neighbors. Then they changed it as follows:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in \mathrm{S}^k(i;u)} \theta_{ij}^u (r_{uj} - b_{uj}) \tag{19}$$

Where $\left\{\theta_{ij}^u | j \in \mathrm{S}^k(i;u)\right\}$ is calculated by globally solving a suitable optimization problem, this simultaneous interpolation accounts for the many interactions between neighbors leading to improved accuracy. In order to facilitate global optimization, they abandoned such user-specific weights in favor of global weights independent of a specific user. The weight from j to i is denoted by w_{ij} and will be learnt from the data through optimization.

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) w_{ij} \tag{20}$$

Comparing with the statistical equation above, it is more like a machine learning way to solve the problem. Then,we consider the interpretation of the weights. Usually the weights in a neighborhood model represent interpolation coefficients relating unknown ratings to existing ones. Here, it is useful to adopt a different viewpoint, where weights represent offsets to baseline estimates. Now, the residuals, $r_{uj} - b_{uj}$, are viewed as the coefficients multiplying those offsets. For two related items i and j, we expect w_{ij} to be high. Thus, whenever a user u rated j higher than expected ($r_{uj} - b_{uj}$ is high), we would like to increase our estimate for u 's rating of i by adding $(r_{uj} - b_{uj})w_{ij}$ to the baseline estimate. Likewise, our estimate will not deviate much from the baseline by an item j that u rated just as expected ($r_{uj} - b_{uj}$ is around zero), or by an item j that is not known to be predictive on i (w_{ij} is close to zero). Generally speaking, it decides users' personal preference.
Add implicit feedback:

$$\hat{r}_{ui} = b_{ui} + \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj})\, w_{ij} + \sum_{j \in \mathrm{N}(u)} c_{ij} \tag{21}$$

Much like the w_{ij}'s, the c_{ij}'s are offsets added to baseline estimates. For two items i and j, an implicit preference by u to$j$ lead us to modify our estimate ofr_{ui} by c_{ij} , which is expected to be high if j is predictive on i. Viewing the weights as global offsets, rather than as user-specific interpolation coefficients, emphasizes the influence of missing ratings. In other words, the opinion of a user is formed not only by what he rated, but also by what he did not rate. For example, suppose that a movie ratings dataset shows that users that rate $The\ Avengers$ 3 high also gave high ratings to $The\ Avengers$ 1 and $The\ Avengers$ 2. This will establish high weights from $The\ Avengers$ 1 and $The\ Avengers$ 2. Now, if a user did not rate $The\ Avengers$ 1 and $The\ Avengers$ 2 at all, his predicted rating for $The\ Avengers$ 3 will be penalized, as some necessary weights cannot be added to the sum.

$$\hat{r}_{ui} = \mu + b_u + b_i + \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj})\, w_{ij} + \sum_{j \in \mathrm{N}(u)} c_{ij} \tag{22}$$

A characteristic of the current scheme is that it encourages greater deviations from baseline estimates for users that provided many ratings (high $|\mathrm{R}(u)|$) or plenty of implicit feedback (high $|\mathrm{N}(u)|$). In general, this is a good practice for recommender systems. We would like to take more risk with well modeled users that provided much input. For such users we are willing to predict quirkier and less common recommendations. On the other hand, we are less certain about the modeling of users that provided only a little input, in which case we would like to stay with safe estimates close to the baseline values. However, based on experiment, it shows that the current model somewhat overemphasizes the dichotomy between heavy raters and those that rarely rate. Better results were obtained when they moderated this behavior, replacing the prediction rule with:

$$\begin{aligned}\hat{r}_{ui} =& \mu + b_u + b_i + |\mathrm{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj})\, w_{ij} \\ &+ |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} c_{ij}\end{aligned} \tag{23}$$

And here is the regularized least squares problem:

$$\begin{aligned}\min_{b_*, w_*, c_*} \sum_{(u,i) \in \mathcal{K}} & (r_{ui} - \mu - b_u - b_i - |\mathrm{N}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathbb{N}^k(i,u)} c_{ij} \\ & - |\mathrm{R}^k(i;u)|^{-\frac{1}{2}} \sum_{j \in \mathbb{R}^k(i;u)} (r_{uj} - b_{uj}) w_{ij})^2 \\ & + \lambda_4 (b_u^2 + b_i^2 + \sum_{j \in \mathbb{R}^k(i;u)} w_{ij}^2 + \sum_{j \in \mathbb{N}^k(i;u)} c_{ij}^2)\end{aligned} \tag{24}$$

2) Temporal Dynamics Part: After we have a overall understanding of the kNN model,we add the temporal dynamic part to it.

$$\begin{aligned}\hat{r}_{ui} =& b_{ui} + |\mathbf{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbb{N}(u)} e^{-\beta_u \cdot |t_{ui} - t_{uj}|} c_{ij} + \\ & |\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} e^{-\beta_u \cdot |t_{ui} - t_{uj}|} \left(\left(r_{uj} - \tilde{b}_{uj} \right) w_{ij} \right)\end{aligned} \tag{25}$$

Likewise, the least square problem:

$$\begin{gathered}\hat{r}_{ui} = b_{ui} + |\mathbf{N}(u)|^{-\frac{1}{2}} \textstyle\sum_{j \in \mathbb{N}(u)} e^{-\beta_u \cdot |t_{ui} - t_{uj}|} c_{ij} + \\ |\mathbf{R}(u)|^{-\frac{1}{2}} \textstyle\sum_{j \in \mathbb{R}(u)} e^{-\beta_u \cdot t_{ui} - t_{uj}|} \left(\left(r_{uj} - \tilde{b}_{uj} \right) w_{ij} \right) + \\ \textstyle\sum_{j \in \mathbb{R}(u)} e^{-\gamma_u \cdot |t_{ui} - t_{uj}|} \left(\left(r_{uj} - \tilde{b}_{uj} \right) d_{ij} \right)\end{gathered} \tag{26}$$

D. Matrix Factorization with Temporal Dynamics

1) SVD: Each user u is associated with a user-factors vector $p_u \in \mathbb{R}^f$, and each item i with an item-factors vector $q_u \in \mathbb{R}^f$. Prediction is done by the rule:

$$\hat{r}_{ui} = b_{ui} + p_u^T q_i \tag{27}$$

2) Asymmetric SVD: Extend the model by considering also implicit information.

$$\begin{aligned}\hat{r}_{ui} = b_{ui} + q_i^T (& |\mathbf{R}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{R}(u)} (r_{uj} - b_{uj}) x_j \\ & + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j)\end{aligned} \tag{28}$$

Typically the number of users is much larger than the number of products. Thus, exchanging user- parameters with item-parameters lowers the complexity of the model.

3) SVD++: In fact, as far as integration of implicit feedback is concerned, we change basic SVD model a little bit:

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j \right) \tag{29}$$

SVD++ does not offer the previously mentioned benefits of having less parameters, conveniently handling new users and readily explainable results. This is because we abstract each user with a factors vector. However, SVD++ is clearly advantageous in terms of prediction accuracy. Actually, to our best knowledge, its results are more accurate than all previously published methods on the Netflix data.

4) timeSVD++: We add some temporal dynamics in SVD++:

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(p_u\left(t_{ui}\right) + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbb{N}(u)} y_j \right) \tag{30}$$

$$p_{uk}(t) = p_{uk} + \alpha_{uk} \cdot \mathrm{dev}_u(t) + p_{ukt} \quad k = 1, \ldots, f \tag{31}$$

Where p_{ukt} absorbs the very local, day-specific variability. Also, here is without the day-specific portion

$$p_{uk}(t) = p_{uk} + \alpha_{uk} \cdot \mathrm{dev}_u(t) \quad k = 1, \ldots, f \tag{32}$$

5) timeSVD++ with Frequency:

$$\hat{r}_{ui} = b_{ui} + \left(q_i^T + q_{i,f_{ui}}^T\right) \left(p_u\left(t_{ui}\right) + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathbb{N}(u)} \cdot y_j \right) \tag{33}$$

Notice that while the transition to frequency-aware biases was measurably effective, the introduction of frequency-dependent movie factors was barely beneficial.

E. Difference between 2009 and lecture notes

1) Baseline: In the more accurate model, they add time and frequency factors. Both of them lead to a higher precision.

2) K-NN: First, introduce explicit and implicit feedback to it. Second, use machine learning idea to train parameters from training set. Third, add temporal dynamics to it.

3) Matrix Factorization: SVD, Asymetric SVD, SVD++, Time SVD++, Time SVD++ with frequency. The accuracy boast a lot. By the way, the SVD model is also popular among other area, which is need to know.

IV. Simulation

We first simulated the neighborhood method. We used the small dataset in the example of the textbook. As expected, the result was 0.54 as the same as it in the textbook. Even for the small dataset, the RMSE was not convincing, but the prediction of k-NN was very close to the practical rating sample. So we simulated the algorithms begin with the k-NN.

For the specific algorithms that the winning team used. We simulated 6 methods with python to calculate the RMSE. The methods were k-NN, k-NN with regularization, general matrix factorization, SVD, SVD++ and timeSVD++ (all has the same baseline formula: baseline with temporal dynamics). Also, all datasets we used has more than 1000 users and 2000 movies to make the results as close as possible. For example, in the dataset1 we had 1640 users and 2071 movies.

First, this was the basic k-NN algorithm. The neighborhood method behaved like the simplest predictor with the RMSE around 0.95.

Second, this was the k-NN algorithm with regularization. For our program, we chose the $\lambda = 1$. As the advanced material in the textbook mentioned, $\lambda = 1$ was the best parameter for the test dataset. This optimization was not complicated. But unexpectedly, for the practical data, this method was very effective, even better than some deep learning method

Then, we used Matrix Factorization to decompose the rating matrix. For the vectors and the matrix we got, we could regard them as the flavors of users and bias of movies. Then we added these to the formula to make the prediction. As a result, the RMSE could be reduce to less than 0.90. In addition, we also used SVD, SVD++ and timeSVD++ to optimize the matrix factorization, and made the RMSE much smaller.

In the Fig.1, we can see the SVD in the algorithm is very useful. In fact, the matrix factorization with SVD was the most important in BellKor's algorithm, which was the key improvement of the optimization.

Simultion Results:
F is the parameter of frequency.

K-NN:
dataset1: RMSE = 0.9550
dataset2: RMSE = 0.9435
dataset3: RMSE = 0.9548
dataset4: RMSE = 0.9498

K-NN with Regularization:
dataset1: RMSE = 0.9533
dataset2: RMSE = 0.9385
dataset3: RMSE = 0.9445
dataset4: RMSE = 0.9415

PMF:
dataset1: F = 200, #iterations = 90, RMSE = 0.9099
dataset2: F = 200, #iterations = 90, RMSE = 0.8914
dataset3: F = 200, #iterations = 90, RMSE = 0.8999
dataset4: F = 200, #iterations = 90, RMSE = 0.9032

SVD:
dataset1: F = 200, #iterations = 90, RMSE = 0.8343
dataset2: F = 200, #iterations = 90, RMSE = 0.8443

dataset3: F = 200, #iterations = 90, RMSE = 0.8433
dataset4: F = 200, #iterations = 90, RMSE = 0.8321

SVD++:
dataset1: F = 200, #iterations = 90, RMSE = 0.8014
dataset2: F = 200, #iterations = 90, RMSE = 0.8232
dataset3: F = 200, #iterations = 90, RMSE = 0.8174
dataset4: F = 200, #iterations = 90, RMSE = 0.8261

timeSVD++:
dataset1: F = 200, #iterations = 90, RMSE = 0.7832
dataset2: F = 200, #iterations = 90, RMSE = 0.7931
dataset3: F = 200, #iterations = 90, RMSE = 0.7956
dataset4: F = 200, #iterations = 90, RMSE = 0.8101

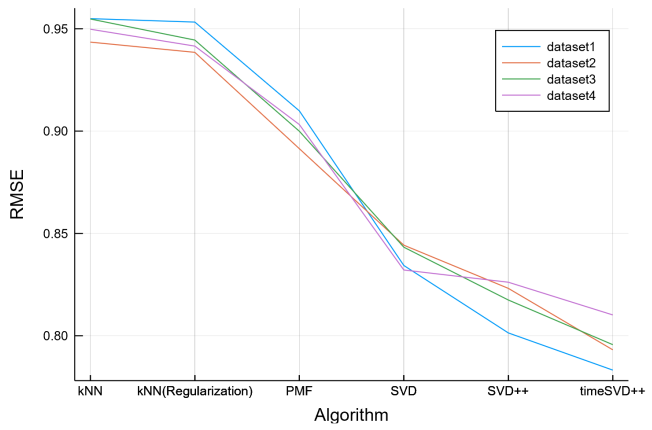

Fig. 1. The scale of datasets is smaller than the standard quiz dataset that Netflix provides. Although the RMSE cannot stand for the user data in the real world, the trend of the figure can still demonstrate the merit of SVD algorithm.

References

[1] M, Chiang. Networked Life: 20 Questions and Answers. Cambridge University Press, 2012.
[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] R. Bell, Y. Koren, The Bellkor Solution to the Netflix Grand Prize. Netflix prize documentation, 2009.
[5] Y. Koren, Factorization meets the neighborhood: a multifaceted collaborative filtering model. Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2008.
[6] Y. Koren, Collaborative filtering with temporal dynamics. Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2009.
[7] Y. Koren, B. Robert, and V. Chris. Matrix factorization techniques for recommender systems. Computer 8 (2009): 30-37.