CS426 Processing Massive Data: Project 2 Movie Recommendation Report

Zuoming Zhang, Weichen Li, Peiyuan Liu, Yanping Xie
5120309626, 5120309662, 5120309394, 5120309300
Department of Computer Science and Engineering, Shanghai Jiao Tong University, China
237632270@qq.com; eizo.lee@sjtu.edu.cn; ms_green@sjtu.edu.cn; Rea.s.xyp@gmail.com

Abstract

In this project, we are going to implement a movie recommendation system and give predictions.
...

CONTENTS

I. Introduction

A. Recommendation Systems

There is an extensive class of Web applications that involve predicting user responses to options. Such a facility is called a *recommendation system*. We shall begin this chapter with a survey of the most important examples of these systems. However, to bring the problem into focus, two good examples of recommendation systems are:

1) Offering news articles to on-line newspaper readers, based on a prediction of reader interests.
2) Offering customers of an on-line retailer suggestions about what they might like to buy, based on their past history of purchases and/or product searches.

Recommendation systems use a number of different technologies. We can classify these systems into two broad groups.

- *Content-based systems* examine properties of the items recommended. For instance, if a Netflix user has watched many cowboy movies, then recommend a movie classified in the database as having the "cowboy" genre.
- *Collaborative filtering* systems recommend items based on similarity measures between users and/or items. The items recommended to a user are those preferred by similar users. There are some new algorithms that have proven effective for recommendation systems.

1) Content-Based Filtering: A pure *Content-Based Recommender System* makes recommendations for a user based solely on the profile built up by analyzing the content of items which that user has rated in the past.

For example, the user may have watched "Harry Potter and the Sorcerer's Stone". Then the recommender system may suggest "Harry Potter and the Chamber of Secrets" and "Polar Express" to the user. (Shown in Fig 1)

The *Content-Based Recommender* has several features:

- Has its root in *Information Retrieval* (IR)
- It is mainly used for recommending *text-based products* (web pages, usenet news messages) –products for which you can find a textual description.
- The items to recommend are "described" by their associated *features* (e.g. keywords).
- The *User Model* can be structured in a "similar" way as the content: for instance the features/keywords more likely to occur in the preferred documents. Then, for instance, text documents can be recommended based on a comparison between their content (words appearing in the text) and a user model (a set of preferred words).

Fig. 1. Movie recommendation example

- The user model can also be a *classifier* based on whatever technique (e.g., Neural Networks, Naive Bayes, C4.5).

2) Collaborative Filtering: In general, collaborative filtering is the process of filtering for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc.

Applications of collaborative filtering typically involve very large data sets. Collaborative filtering methods have been applied to many different kinds of data including: sensing and monitoring data, such as in mineral exploration, environmental sensing over large areas or multiple sensors; financial data, such as financial service institutions that integrate many financial sources; or in electronic commerce and web applications where the focus is on user data, etc. The remainder of this discussion focuses on collaborative filtering for user data, although some of the methods and approaches may apply to the other major applications as well.

Collaborative filtering systems have many forms, but many common systems can be reduced to two steps:

1) Look for users who share the same rating patterns with the active user (the user whom the prediction is for).
2) Use the ratings from those like-minded users found in step 1 to calculate a prediction for the active user.

This falls under the category of user-based collaborative filtering. A specific application of

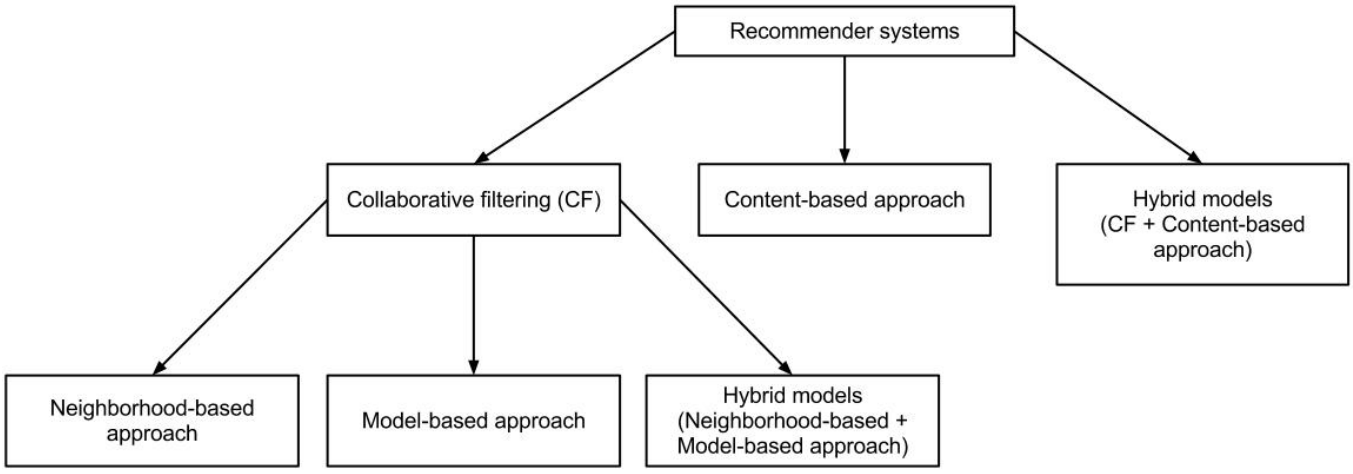

Fig. 2. Collaborative Filtering in Recommender Systems

this is the user-based Nearest Neighbor algorithm.

Alternatively, item-based collaborative filtering (users who bought x also bought y), proceeds in an item-centric manner:

1) Build an item-item matrix determining relationships between pairs of items.
2) Infer the tastes of the current user by examining the matrix and matching that user's data.

An example of collaborative filtering in recommender systems is shown in Fig 2.

B. The Netflix Challenge

Netflix, Inc. is an American provider of on-demand Internet streaming media available to viewers. It has a movie recommendation system called CineMatch.

The movie recommendation of Netflix is shown in Fig 3.

A significant boost to research into recommendation systems was given when NetFlix offered a prize of $1,000,000 to the first person or team to beat their own recommendation algorithm, CineMatch, by 10%. After over three years of work, the prize was awarded in September, 2009.

The NetFlix challenge consisted of a published dataset, giving the ratings by approximately half a million users on (typically small subsets of) approximately 17,000 movies. This data was selected from a larger dataset, and proposed algorithms were tested on their ability to predict the ratings in a secret remainder of the larger dataset. The information for each (user, movie) pair in the published dataset included a rating (1-5 stars) and the date on which the rating was made.

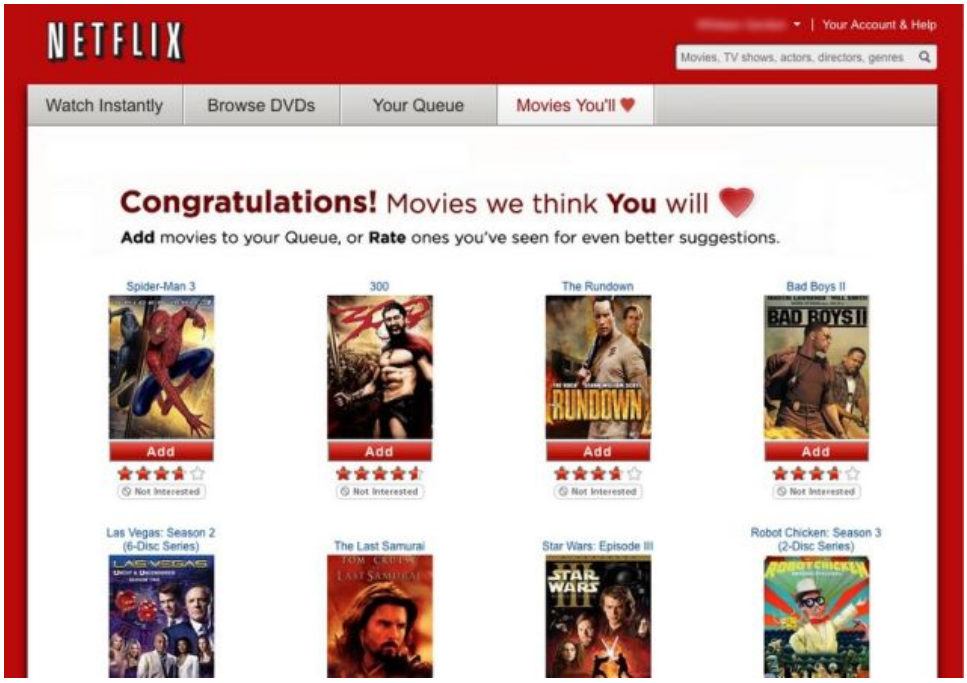

Fig. 3. Movie recommendation system of Netflix

The RMSE was used to measure the performance of algorithms. CineMatch has an RMSE of approximately 0.95; i.e., the typical rating would be off by almost one full star. To win the prize, it was necessary that your algorithm have an RMSE that was at most 90% of the RMSE of CineMatch.

The bibliographic notes for this chapter include references to descriptions of the winning algorithms. Here, we mention some interesting and perhaps unintuitive facts about the challenge.

- CineMatch was not a very good algorithm. In fact, it was discovered early that the obvious algorithm of predicting, for the rating by user u on movie m, the average of:
  1) The average rating given by u on all rated movies and
  2) The average of the ratings for movie m by all users who rated that movie.

  was only 3% worse than CineMatch.

- The UV-decomposition algorithm was found by three students (Michael Harris, Jeffrey Wang, and David Kamm) to give a 7% improvement over CineMatch, when coupled with normalization and a few other tricks.

- The winning entry was actually a combination of several different algorithms that had been developed independently. A second team, which submitted an entry that would

have won, had it been submitted a few minutes earlier, also was a blend of independent algorithms. This strategy – combining different algorithms – has been used before in a number of hard problems and is something worth remembering.

- Several attempts have been made to use the data contained in IMDB, the Internet movie database, to match the names of movies from the NetFlix challenge with their names in IMDB, and thus extract useful information not contained in the NetFlix data itself. IMDB has information about actors and directors, and classifies movies into one or more of 28 genres. It was found that genre and other information was not useful. One possible reason is the machine-learning algorithms were able to discover the relevant information anyway, and a second is that the entity resolution problem of matching movie names as given in NetFlix and IMDB data is not that easy to solve exactly.

- Time of rating turned out to be useful. It appears there are movies that are more likely to be appreciated by people who rate it immediately after viewing than by those who wait a while and then rate it. "Patch Adams" was given as an example of such a movie. Conversely, there are other movies that were not liked by those who rated it immediately, but were better appreciated after a while; "Memento" was cited as an example. While one cannot tease out of the data information about how long was the delay between viewing and rating, it is generally safe to assume that most people see a movie shortly after it comes out. Thus, one can examine the ratings of any movie to see if its ratings have an upward or downward slope with time.

The winner of the Netflix Prize, comes up with an algorithm called *BellKor Recommendation System*, which contains three parts(Shown in Fig 4):

- *Global*: Overall deviations of users/movies.
- *Factorization*: Addressing "regional" effects.
- *Collaborative Filtering*: Extract local patterns.

Our recommendation system is based on *BellKor Recommendation System*, and we have made some modifications and improvements to ensure that the prediction is more accurate.

II. Division of Labor

Our group is composed of four members: Zuoming Zhang, Weichen Li, Peiyuan Liu, Yanping Xie, and Zuoming Zhang is the team leader.

...

III. Theoretical Introduction

Firstly, we obtain a matrix R called *Utility Matrix*, shown as below:

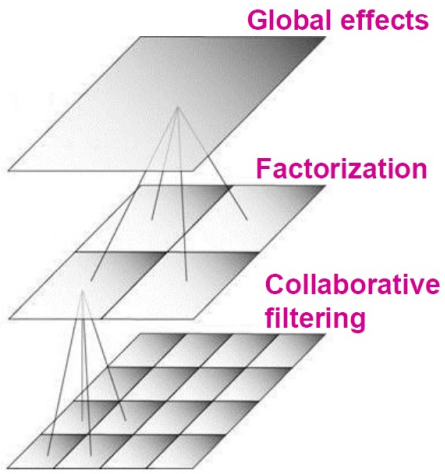

Fig. 4. BellKor Recommendation System

$$R = \underbrace{\left(\begin{array}{ccccc} r_{1,1} & & \dots & & r_{1,X} \\ r_{2,1} & r_{2,2} & \dots & & \\ \dots & \dots & \dots & \dots & \dots \\ & r_{I-1,2} & \dots & r_{I-1,X-1} & \\ r_{I,1} & r_{I,2} & \dots & & r_{I,X} \end{array}\right)}_{X \text{ users}} \Bigg\} I \text{ items}$$

$r_{x,y}$ is the rating given by user y to the movie x. It is realistic that some movies are not rated by some users, so the corresponding positions of those ratings in R remain empty.

Since our goal is to predict certain potential movie ratings that certain users may give, we need to make use of relevant algorithms to optimize the prediction.

We use **SVD**, **SVD++**, **timeSVD++** with biases model and regularization as our algorithms.

Both algorithms will decompose R into the production of two matrices Q and P^T, shown as below:

$$R = \begin{bmatrix} r_{1,1} & & \dots & & r_{1,X} \\ r_{2,1} & r_{2,2} & \dots & & \\ \dots & \dots & \dots & \dots & \dots \\ & r_{I-1,2} & \dots & r_{I-1,X-1} & \\ r_{I,1} & r_{I,2} & \dots & & r_{I,X} \end{bmatrix} \approx \begin{bmatrix} q_{1,1} & q_{1,2} & \dots & q_{1,K} \\ q_{2,1} & q_{2,2} & \dots & q_{2,K} \\ \dots & \dots & \dots & \dots \\ q_{I,1} & q_{I,2} & \dots & q_{I,K} \end{bmatrix} \times \begin{bmatrix} p_{1,1}^T & p_{1,2}^T & \dots & p_{1,X}^T \\ p_{2,1}^T & p_{2,2}^T & \dots & p_{2,X}^T \\ \dots & \dots & \dots & \dots \\ p_{K,1}^T & p_{K,2}^T & \dots & p_{K,X}^T \end{bmatrix}$$

A. *Simple SVD(RSVD) with Biases Model*

In **SVD** with biases model we utilize the equations below:

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

$$e_{ui} = r_{ui} - \hat{r}_{ui}$$

$\hat{r}_{ui}$ is the predicted rating from user u to item i, μ is the overall mean rating, b_i is the bias for movie i, b_u is the bias for user u.

e_{ui} is the difference between the real rating and the predicted rating.

And our goal is to minimize *SSE*:

$$SSE = \frac{1}{2} \sum_{(i,u) \in R} e_{ui}^2 = \frac{N}{2} \times (RMSE)^2$$

N is the total number of ratings in R.

Moreover, in order to avoid overfitting, it is necessary to use regularization method to penalize models with extreme parameter values as follows:

$$SSE = \frac{1}{2} \sum_{(i,u) \in R} e_{ui}^2 + \frac{1}{2}\lambda \sum_u |p_u|^2 + \frac{1}{2}\lambda \sum_i |q_i|^2 + \frac{1}{2}\lambda \sum_u |b_u|^2 + \frac{1}{2}\lambda \sum_i |b_i|^2$$

SVD with regularization is called **RSVD**.

Then, by *gradient descent*, we can iteratively lower the SSE and update b_u, b_i, p_{uk}, q_{ki}.

In each iteration, updates are shown below:

$$b_u := b_u + \eta(e_{ui} - \lambda b_u)$$

$$b_i := b_i + \eta(e_{ui} - \lambda b_i)$$

$$p_{uk} := p_{uk} + \eta(e_{ui} q_{ki} - \lambda p_{uk})$$

$$q_{ki} := q_{ki} + \eta(e_{ui} p_{uk} - \lambda q_{ik})$$

η is the learning rate, λ is a constant for penalty.

After each iteration, a new and lower $RMSE$ is generated.

When all iterations are finished, we can obtain the final $RMSE$.

B. *SVD++*

Compared with **SVD**, implicit feedback is added in **SVD++**.

We change $\hat{r}_{ui}$ as below:

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

Here $N(u)$ represents the behavior record of user u. y_j is an attribute of the items.

We update q_{ki} and y_j as below:

$$q_{ki} := q_{ki} + \eta(e_{ui}(p_{uk} + \frac{1}{\sqrt{|N(u)|}}\sum_{j\in N(u)} y_{jk}) - \lambda q_{ik})$$

$$y_{jk} := y_{jk} + \eta\left\{\left[\sum_{j\in N(u), i\in R(u)} e_{ui}q_{ki}|N(u)|^{-\frac{1}{2}}\right] - \lambda y_{jk}\right\}$$

Apart from these two modifications, all the other parts of *SVD++* are the same as *SVD*.

C. *timeSVD++*

Three strong temporal effects are identified in the data:

- Movie biases: movies go in and out of popularity over time.
- User biases: users change their baseline ratings over time.
- User preferences: users change their preferences over time.

Now, let's describe how those temporal effects were inserted into our models. The general framework is:

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

The definitions of those new parameters are:

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

$$b_u(t) = b_u + \alpha_u \cdot \widehat{dev}_u(t)$$

$$p_{uk}(t) = p_{uk} + \alpha_{uk} \cdot \widehat{dev}_u(t)$$

$$dev_u(t) = sign(t - t_u) \cdot |t - t_u|^\beta$$

where $\widehat{dev}_u(t)$ is the centered variables of $dev_u(t)$, and the value of β is set by cross validation to 0.4. Notice that those variables $\widehat{dev}_u(t)$ are constants that are derived directly from the training data.

In this project, we use 55 bins for 5115 days in total.

With regularization, SSE for **timesvd++** is:

$$\begin{aligned} SSE &= \frac{1}{2}\sum_{(i,u)\in R} e_{ui}^2 + \frac{1}{2}\lambda\sum_u |p_u|^2 + \frac{1}{2}\lambda\sum_{j\in N(u)} |y_u|^2 + \frac{1}{2}\lambda\sum_i |q_i|^2 + \frac{1}{2}\lambda\sum_u |b_u|^2 + \frac{1}{2}\lambda\sum_i |b_i|^2 \\ &+ \frac{1}{2}\lambda_1\sum_u |\alpha_u|^2 + \frac{1}{2}\lambda_2\sum_i |b_{i,Bin(t)}|^2 + \frac{1}{2}\lambda_3\sum_u\sum_k |\alpha_{uk}|^2 \end{aligned}$$

where λ_1, λ_2, λ_3 are three constants for penalty which are different from λ.

We update α_u, $b_{i,Bin(t)}$, α_{uk} as below:

$$\alpha_u := \alpha_u + \eta_1(e_{ui}(t)\widehat{dev}_u(t) - \lambda_1\alpha_u)$$

$$b_{i,Bin(t)} := b_{i,Bin(t)} + \eta(e_{ui}(t) - \lambda_2 b_{i,Bin(t)})$$

$$\alpha_{uk} := \alpha_{uk} + \eta_1(e_{ui}(t)q_{ki}\widehat{dev}_u(t) - \lambda_3\alpha_{uk})$$

$$q_{ki} := q_{ki} + \eta(e_{ui}(t)(p_{uk} + \alpha_{uk}\widehat{dev}_u(t) + \frac{1}{\sqrt{|N(u)|}}\sum_{j\in N(u)} y_{jk}) - \lambda q_{ik})$$

All the other parts of **timeSVD++** are the same as **SVD++**.

IV. timeSVD++ Implementation

A. Operating System Environment

The recommendation of operating system is *Ubuntu 14.04 LTS*, and the programming language is C++.

B. Parameter Setting

There are several important parameters in our algorithm: iteration number ($iter$), dimension (dim), learning rate ($alpha$ / $alpha_{betau}$ / $alpha_{bibin}$ / $alpha_{betauk}$), regularization parameter λ ($lambda$ / $lambda_{betau}$ / $lambda_{bibin}$ / $lambda_{betauk}$). Value of these parameters plays an important role in the performance of timeSVD++. Here we define the performance as the test result provided by the course website. We conducted a series of experiments to locate the best setting of the parameters.

First, we experimented with iteration number. The larger the iteration number, the better training objective value we got. But the converging speed of the objective value slowed down significantly with the increase of iteration times. So the best practice is to set $iter$ as 50, a moderate value that balances the objective value and the training time.

We then tested with dimension number. We ran SVD, SVD++ and timeSVD++ with different values of dim. Results are shown in Fig.5. We can see from the result that, in general, timeSVD++ performs better than SVD and SVD++. And all the three methods

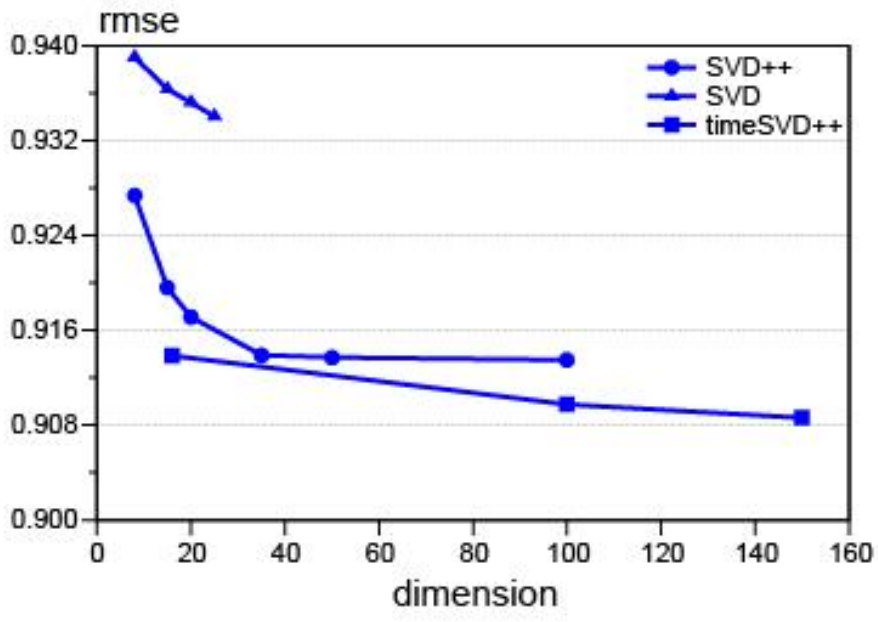

Fig. 5. Dimension

perform better with larger dim, but the performance will converge to some limit in the end. It will take much more computation time for larger dim, so similar to the setting of $iter$, we needed to choose a moderate value of dim for the trade-off. And we took 150 in our final implementation.

$alpha$, $alpha_{betau}$, $alpha_{bibin}$ and $alpha_{betauk}$ are the learning rates. We obtained the reference value of global learning rate $alpha$ from the internet, and tested around the reference value. Further, we introduced local learning rates $alpha_{betau}$, $alpha_{bibin}$ and $alpha_{betauk}$ for time components $betau$, $bibin$ and $betauk$ respectively, components that represent time effect, to precisely tune the influence of different training components. We experimented with different local learning rates, and found the optimal setting $alpha = 0.008$, $alpha_{betau} = alpha_{bibin} = alpha_{betauk} = 0.0032$.

$lambda$, $lambda_{betau}$, $lambda_{bibin}$ and $lambda_{betauk}$ are parameters used in regularization method to avoid overfitting. Similar to the learning rates, we obtained the reference value of global $lambda$ from the internet, and introduced local $lambda_{betau}$, $lambda_{bibin}$ and $lambda_{betauk}$ for time components $betauk$, $bibin$ and $betau$ respectively. We first experimented with different combinations of $lambda_{betau}$, $lambda_{bibin}$ and $lambda_{betauk}$. By testing the performance of applying only one of the three time components, with the same $lambda-$parameter however, we found that $betauk$ did better than $betau$ and $betau$ did better than $bibin$. This indicated that $bibin$ has greater possibility of overfitting than $betau$ and $betau$ greater than $betauk$. So we decided that $lambda_{bibin}$ should be

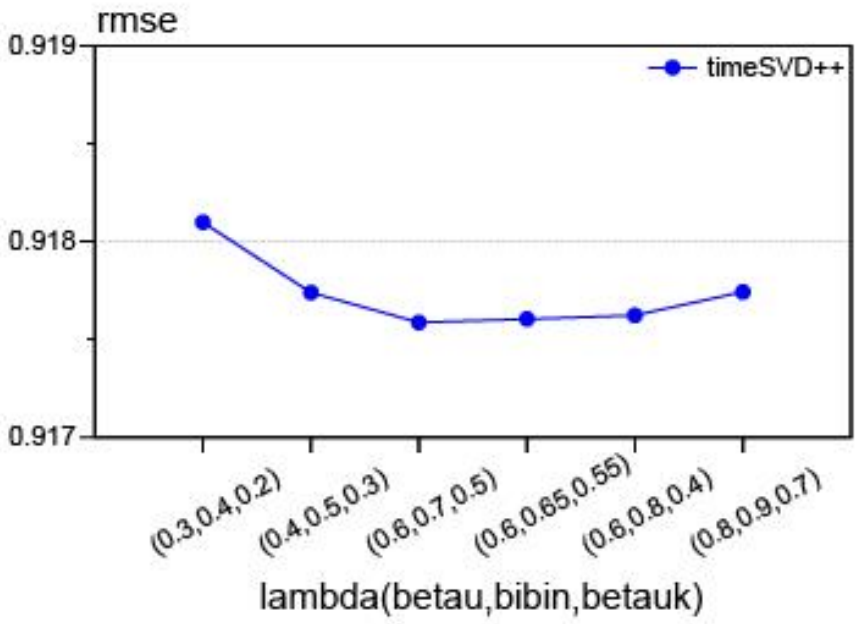

Fig. 6. lambda(betau,bibin,betauk)

larger than $lambda_{betau}$, and $lambda_{betau}$ larger than $lambda_{betauk}$. We tested with various combinations of $lambda_{betau}$, $lambda_{bibin}$ and $lambda_{betauk}$ in the pattern $lambda_{bibin} > lambda_{betau} > lambda_{betauk}$, and the result is shown in Fig.6. We chose the optimal setting $lambda_{betau} = 0.6$, $lambda_{bibin} = 0.7$ and $lambda_{betauk} = 0.5$.

Then we tuned global $lambda$, around the reference value. The setting of $lambda_{betau}$, $lambda_{bibin}$ and $lambda_{betauk}$ was fixed at (0.6, 0.7, 0.5). The experimentation result is shown in Fig.7. We set $lambda = 0.03$ in final implementation.

C. Final Result

After finding out the best results using each single method, we added them up with different weights. The final best result is $RMSE = 0.907924$.

D. Analysis

V. Conclusion & Additional Finding

A. Conclusion

B. Additional Finding

References

[1] Bell R M, Koren Y, Volinsky C. The bellkor 2008 solution to the netflix prize[J]. KorBell Team's Report to Netflix, 2008.

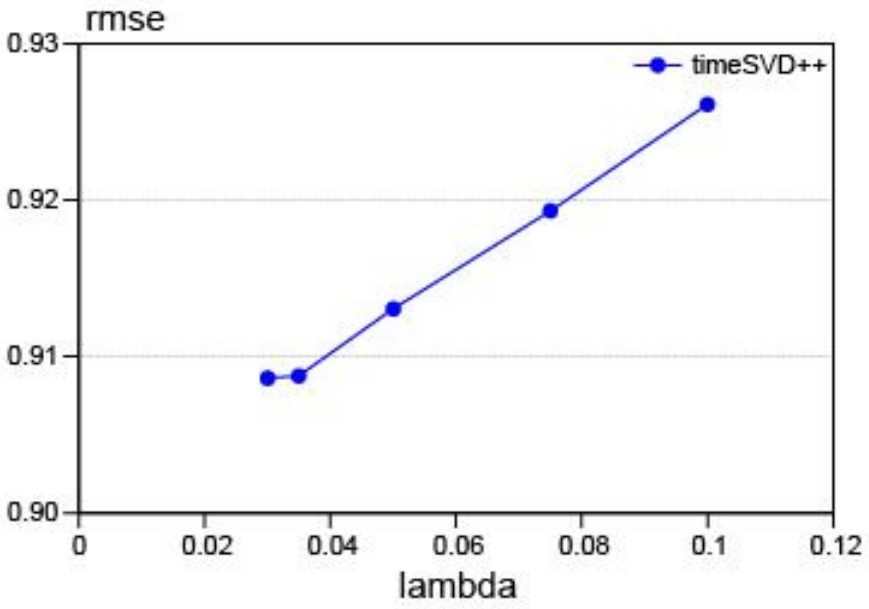

Fig. 7. lambda

[2] Koren Y. Collaborative filtering with temporal dynamics[J]. Communications of the Acm, 2010, 53(4):89-97.
[3] Koren Y. The BellKor solution to the Netflix Grand Prize[J]. Netflix Prize Documentation, 2009.
[4] T02scher A, Jahrer M, Bell R M. The BigChaos Solution to the Netflix Grand Prize[J]. Netflix Prize Documentation, 2009.
[5] Jure Leskovec, Anand Rajarman, Jeffrey D. Ullman. Mining of Massive Datasets. 2010.
[6] Francesco Ricci. Content-Based Filtering and Hybrid Systems.
[7] Collaborative filtering. http://en.wikipedia.org/wiki/Collaborative_filtering#Methodology
[8] Netflix http://en.wikipedia.org/wiki/Netflix