

SVD++ with an improved bias calculation step

Martin Ivanov Gregory Banfi Dhivyabharathi Ramasamy
Group magrdh, Department of Computer Science, ETH Zurich, Switzerland

Abstract—Recommender systems play a vital role in today’s online purchase systems by providing personalized suggestions for products. These systems often use *Collaborative Filtering* (CF) where recommendations are made based only on past user behaviour. In this work we present a CF approach based on a latent factor model that uses an improved bias calculation step to achieve higher prediction accuracy. Furthermore our solution incorporates both explicit and implicit user feedback and was tested on a subset of the Netflix data. Our experiments show that our method achieves qualitatively good results with the only drawback being the high execution time of the algorithm.

I. INTRODUCTION

In our modern world consumers are overwhelmed with choices as retailers offer a vast variety of products with a large number characteristics. Providing users with sensible recommendations is a complicated endeavour which can greatly contribute to user satisfaction. As a result recommender systems are a crucial component in online purchase systems that recommend products that users are likely to express interest in [1]. Recommender systems are often based on *Collaborative Filtering* (CF) [2] which takes into consideration only past user behaviour and does not require explicit user profiling. Two of the most popular approaches to CF are *Singular Value Decomposition* (SVD) [3] and *Item-item Collaborative Filtering*[4].

In this paper, we present an improved *Singular Value Decomposition* approach that takes into consideration both implicit and explicit data and uses an alternative approach for calculating baseline predictions. In section II we outline *SVD++ with an improved bias calculation step*.

We provide results from benchmarking and experiments in section III which illustrates that our approach significantly improves the prediction accuracy compared to other baseline solutions.

In section IV we present the advantages and disadvantages of our solution in terms of execution time and prediction accuracy.

In section V we present future work and possible additional improvements to our methods. Especially we focus on a hybrid solution based on both *Singular Value Decomposition* and *Item-item Collaborative Filtering*. Finally in section VI we summarize our results and conclude our findings.

II. SVD++ WITH AN IMPROVED BIAS CALCULATION STEP

The dataset used to evaluate our algorithm is a subset of the Netflix data [5] and is provided by the *Computational Intelligence Lab* course. The training set consists of 10000 users and 1000 movies with a total of 694054 ratings. Ratings range from 1 to 5 where 1 is the lowest and 5 is the highest rating. The quality of the algorithm is measured by the root mean squared error (RMSE) accuracy metric:

$$(RMSE) : \sqrt{\sum_{(u,i) \in TestSet} (r_{ui} - \hat{r}_{ui})^2 / |TestSet|} \quad (1)$$

Furthermore the accuracy of our solution was benchmarked on a validation set via an online submission system and will get a final ranking based on a test set RMSE measurement.

Our solution is an extension to the popular SVD++ [6] method and offers an alternative user and item bias calculation step. SVD++ is a latent factor model approach to *Collaborative Filtering* which incorporates both explicit and implicit feedback to achieve better prediction accuracy. The model is induced by *Singular Value Decomposition* on the user-item ratings matrix and connects each user u with a user-factor vector p_u and each movie i with an item-factor vector q_i . A prediction is done by calculating the inner product $\hat{r}_{ui} = q_i^T p_u$.

This simple prediction can further be altered to take into consideration a baseline estimate[7] $b_{ui} = \mu + b_u + b_i$. The b_u and b_i are user and item biases respectively and μ is the overall rating average. The reasoning behind taking biases into consideration is the fact that some users give higher ratings than others and some movies tend to get higher ratings compared to the global average. For example if we want to get a baseline estimate for the movie *Interstellar* by user Bob we will do the following: lets say that the global mean is 3.0 stars, *Interstellar* has better than average ratings by 1.0 stars and Bob rates movies 0.5 stars above the average, then we propose that Bob will give $3.0 + 1.0 + 0.5 = 4.5$ stars to *Interstellar*.

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \quad (2)$$

In our solution we decided to take into consideration an alternative approach to calculating the baseline estimate. As the training dataset is very sparse it could happen that a particular movie has only a single rating. According to the previously outlined baseline estimate in this case the

average movie rating will be the only rating we observed for this movie which is generally a bad prediction. Instead we can view a single observation as being drawn from a true probability distribution of averages. Then we can calculate a better mean estimate by taking into consideration a linear combination between the observed mean and the apriori mean where the blending ratio is the ratio of variances. As outlined by Simon Funk [8] instead of calculating the blending ratio we can use a constant which we can tune via cross-validation. In our solution we used $K = 20$ as a blending ration. The final movie bias prediction can be defined as follows:

$$\frac{K \cdot \mu' + \mu_i \cdot |R(i)|}{K + |R(i)|} \quad (3)$$

Where μ' is the mean of movie averages, μ_i is the average of a particular movie and $|R(i)|$ is the number of ratings of the movie.

The same principle applies for calculating the user biases where we used the exact same blending ratio.

$$\frac{K \cdot \sigma' + \sigma_i \cdot |R(u)|}{K + |R(u)|} \quad (4)$$

In this calculation σ' is the mean of all user offset values, σ_i is the offset of a particular user and $|R(u)|$ is the number of ratings a particular user made.

This alternative approach to baseline estimation significantly improved our results.

Incorporating implicit feedback [9] into the current solution can further increase the accuracy of our predictions. An example of suitable implicit feedback for the Netflix dataset could be rental history and user attributes which allow us to indirectly infer user preferences. Our training data did not provide us with such implicit data, however we can still take into consideration whether a user has rated a specific item. Fundamentally a user informs us about his/hers preference by deciding to give a rating regardless of the actual rating value. Even though this source of implicit data does not contribute vastly to the overall prediction, it still allows us to better capture user preferences and achieve a slight accuracy improvement if we run SVD for enough iterations. A user that showed preference is now characterized by a new vector which is normalized according to the number of items the user has rated.

$$|N(u)|^{-0.5} \sum_{j \in N(u)} y_j \quad (5)$$

The final SVD++ prediction model incorporates all of the above mentioned techniques and is defined as follows:

$$\hat{r}_{ui} = b_{ui} + q_i^T (p_u + |N(u)|^{-0.5} \sum_{j \in N(u)} y_j) \quad (6)$$

To learn appropriate vectors for p_u, q_i, b_u, b_i, y_i we need to minimize the squared error on the set of known ratings:

$$\begin{aligned} \min_{b_*, q_*, p_*, y_*} \sum_{(u,i) \in K} (r_{ui} - \mu - b_u - b_i \\ - q_i^T (p_u + |N(u)|^{-0.5} \sum_{j \in N(u)} y_j))^2 \\ + \lambda (b_u^2 + b_i^2 + \|p_u\|^2 + \|q_i\|^2 + \sum_{j \in N(u)} \|y_j\|^2) \end{aligned} \quad (7)$$

We used a *Stochastic Gradient Descent* (SGD) [8] to minimize the above equation. Furthermore we compute a prediction for a given rating and calculate the prediction error as follows:

$$\epsilon_{ui} \stackrel{\text{def}}{=} r_{ui} - \mu - b_u - b_i - q_i^T (p_u + |N(u)|^{-0.5} \sum_{j \in N(u)} y_j) \quad (8)$$

We then alter the parameters by a magnitude determined by the learning rate in the opposite direction of the gradient.

- $b_u \leftarrow b_u + \alpha_1 \cdot (\epsilon_{ui} - \lambda_1 \cdot b_u)$
- $b_i \leftarrow b_i + \alpha_1 \cdot (\epsilon_{ui} - \lambda_1 \cdot b_i)$
- $q_i \leftarrow q_i + \alpha_2 \cdot (\epsilon_{ui} \cdot (p_u + |N(u)|^{-0.5} \sum_{j \in N(u)} y_j) - \lambda_2 \cdot q_i)$
- $p_u \leftarrow p_u + \alpha_2 \cdot (\epsilon_{ui} \cdot q_i - \lambda_2 \cdot p_u)$
- $\forall j \in N(u) :$
- $y_j \leftarrow y_j + \alpha_2 \cdot (\epsilon_{ui} \cdot |N(u)|^{-0.5} \cdot q_i - \lambda_2 \cdot y_j)$

Here α_1 and α_2 are the learning rates that determine the step size or how far we move at each iteration. λ_1 and λ_2 are the regularization terms used to prevent over-fitting of the data. Both the learning rates and the regularization parameters are picked via cross-validation. In our solution we used the following values for these parameters: $\alpha_1 = 0.002$, $\alpha_2 = 0.025$, $\lambda_1 = 0.002$, $\lambda_2 = 0.1$. Another parameter picked via cross-validation is the number of latent factors. For our solution 5 latent factors gave the best performance. We run SGD until convergence which in *Collaborative Filtering* is often just a fixed number of iterations.

III. RESULTS

Even though the provided training set is considerably smaller than the full Netflix data our method achieves relatively low RMSE on the validation set. As we outlined in section II we used cross-validation to tune the parameters in our solution and experiment with different iteration counts. Running *Stochastic Gradient Descent* for 30 iterations worked well for us.

We compared our implementation to 4 main baseline solutions: *Conventional SVD* [10], *SVD with no bias calculation step* [11], *SVD with normal bias calculation step* and *SVD with an improved bias calculation step* [8]. Figure 1 and Figure 2 outline the achieved validation error of the baselines and of our implementation respectively. Both figures illustrate the RMSE, execution time and the number of iterations used for each respective method.

Our solution was implemented in *Matlab* and runs in approximately 815 seconds for 30 iterations on an Intel(R)

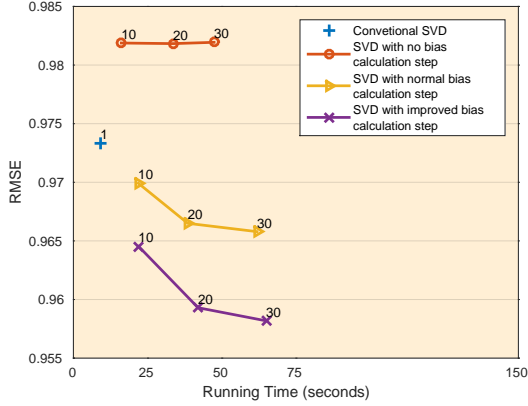


Figure 1. RMSE (lower is better) for each of the 4 baseline solutions on the validation dataset (submission system). Accuracy improves when the number of iterations (denoted as numbers on the chart) of SGD increases. Execution time also increases with the number of iterations.

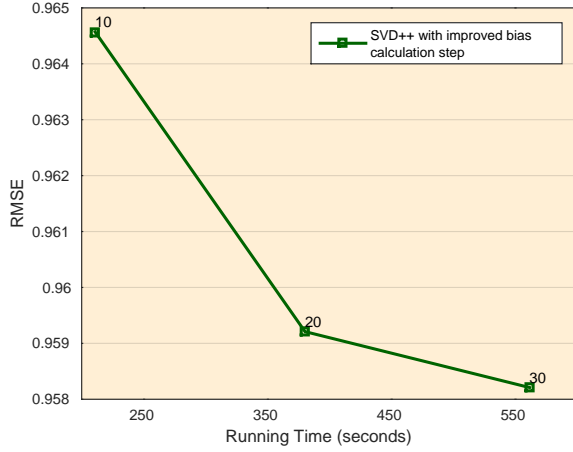


Figure 2. RMSE for SVD++ with an improved bias calculation step on the validation dataset (submission system). Accuracy and execution time increases with the number of iterations.

Core(TM) i5-4210U CPU @ 1.70GHz with 8GB RAM. This being said the execution time on the validation data is much smaller due to the fact that an external submission system was used to execute and benchmark our code.

IV. DISCUSSION

In this section we look at the strengths and weaknesses of our approach and compare it to the baseline solutions outlined in section III.

The main idea of our method is to combine an alternative approach to user and item bias calculation with a *Stochastic Gradient Descent SVD* that incorporates both implicit and explicit data. The approach produces satisfactory results and its' only drawback is the execution time which significantly

increases due to the involvement of further computation steps needed to calculate the implicit factors.

Due to the very limited form of implicit feedback and the relatively small training set the accuracy improvements are negligible and does not justify the running time of the solution. Nevertheless we are convinced that addressing different aspects of the data can greatly improve the quality of a recommender system and incorporating different implicit data sources can contribute to lowering the prediction error.

The alternative bias calculation step proposed in our solution greatly increased the accuracy of our predictions and incorporating it with a *Stochastic Gradient Descent SVD* produced our best results when both RMSE and execution time are taken into consideration. Based on the RMSE metric this solution is very close to the implicit feedback version and runs much faster due to the fewer computations at each iteration step. Thus we will use this approach as a final submission on the ranking system.

Unlike *Conventional SVD* which relies on imputing missing ratings to make the matrix dense our solution is modeled directly on the observed ratings. The main issue with imputing ratings is computational complexity. Our solution does not suffer from this drawback and uses regularization to avoid overfitting.

As a final remark we would like to point out that our method can be merged with *Item-item Collaborative Filtering* to produce a hybrid recommender system which can further contribute to accuracy improvements. We briefly outline a proposal for such a hybrid in section V.

V. FUTURE WORK

In this section we propose a technique that could improve the performance of our implementation, namely *Item-to-item Collaborative Filtering*.

The main idea is when we are predicting the rating of a movie to use the ratings of similar movies to improve the quality of the prediction. This will allow us to give more personalized recommendations by constructing a neighbourhood of similar movies.

Item-to-item Collaborative Filtering is a well known technique and is outlined by Michael Ekstrand and John Riedl[4]. The main component of this algorithm is the similarity function, which is used to compute a value that represents how similar are two particular movies. A popular choice for a similarity function is the *cosine similarity*.

$$s(i, j) = \frac{i \cdot j}{\|i\|_2 \cdot \|j\|_2} \quad (9)$$

Here i and j are vectors containing ratings for movie i and j respectively. In order to use the similarities in the prediction step a similarity matrix must be computed in advance to compare all pairs of movies.

Similarity values and ratings of similar movies are then used to improve the prediction.

$$p_{u,i} = \frac{\sum_{j \in S} s(i,j) \cdot r_j}{\sum_{j \in S} |s(i,j)|} \quad (10)$$

As i and j are vectors of ratings, r_j is the rating that the user gave to the similar movie j and S is the set containing all similar movies with respect to movie i .

Since our solution is based on *Stochastic Gradient Descent* the item-to-item approach has to be adapted, a possible implementation can be found in Koren's paper [6]. The similarity matrix is used to find the k most similar movies that intersect with the set of rated movies by that specific user. The resulting set is $R^k(i, u)$, which contains all rated similar movies with respect to movie i and user u . We then iterated over R in order to get a bias which we can add to our prediction.

$$\begin{aligned} \hat{r}_{ui} &= b_{ui} + q_i^T (p_u + |N(u)|^{-0.5} \sum_{j \in N(u)} y_j) \\ &+ |R^k(i, u)|^{-0.5} \cdot \sum_{j \in R^k(i, u)} (r_{uj} - b_{uj}) \cdot w_{ij} \end{aligned} \quad (11)$$

For each similar movie we can compute the difference between its rating and baseline. The difference is then multiplied with the value of w_{ij} . The matrix w contains the weights of the differences and will then be updated according to the prediction error. Finally we normalize the sum of all the weighted differences with the squared root of the total number of rated similar movies.

When minimizing the error w is updated as follows:

- $\forall j \in R^k(i, u) :$
 $w_{ij} \leftarrow w_{ij} + \alpha_3 \cdot (|R^k(i, u)|^{0.5} \cdot \epsilon_{ui} \cdot (r_{uj} - b_{uj}) - \lambda_8 \cdot w_{ij})$

VI. SUMMARY

The conducted experiments outline that our solution achieves low RMSE if enough iterations of *Stochastic Gradient Descent* are performed. Taking into consideration our current training set and the limited implicit feedback, incorporating implicit data into our method was not beneficial due to the increased execution time. Nevertheless we are convinced that if additional implicit feedback sources are present the accuracy improvements will be sufficient enough to justify the running time of the solution. Thus making the solution a viable option when various types of data are present for training the algorithm.

The alternative bias calculation proved to be a vital component of our solution and had a positive impact by lowering the overall error rate. Our method further outlines that simpler solutions to *Collaborative Filtering* can very often give good results when the amount of training data is small and limited to a single type.

REFERENCES

- [1] G. Adomavicius and A. Tuzhilin, "Towards the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," pp. 734–749, 2005. [Online]. Available: <http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1423975>
- [2] D. Goldberg, D. Nichols *et al.*, "Using collaborative filtering to weave an information tapestry," pp. 61–70, 1992. [Online]. Available: <http://dl.acm.org/citation.cfm?id=138867>
- [3] B. Sarwar, G. Karypis, J. Konstan *et al.*, "Incremental singular value decomposition algorithms for highly scalable recommender systems," pp. 2–3, 2002. [Online]. Available: http://files.grouplens.org/papers/sarwar_SVD.pdf
- [4] M. Ekstrand, J. Riedl, and J. Konstan, "Collaborative filtering recommender systems," pp. 95–101, 2010. [Online]. Available: <http://files.grouplens.org/papers/FnT%20CF%20Recsys%20Survey.pdf>
- [5] J. Bennet and S. Lanning, "The netflix prize," 2007. [Online]. Available: <http://www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf>
- [6] Y. Koren, "Factorization meet the neighborhood: a multifaceted collaborative filtering model," p. 6, 2008. [Online]. Available: <http://www.research.yahoo.com/files/kdd08koren.pdf>
- [7] S. Grower, "Netflix prize and svd," pp. 3–4, 2014. [Online]. Available: <http://buzzard.ups.edu/courses/2014spring/420projects/math420-UPS-spring-2014-gower-netflix-SVD.pdf>
- [8] S. Funk, "Netflix update: Try this at home," 2006. [Online]. Available: <http://sifter.org/~simon/journal/20061211.html>
- [9] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," pp. 45–46, 2009. [Online]. Available: <http://www2.research.att.com/~volinsky/papers/ieeecomputer.pdf>
- [10] K. Baker, "Singular value decomposition tutorial," 2005. [Online]. Available: https://www.ling.ohio-state.edu/~kbaker/pubs/Singular_Value_Decomposition_Tutorial.pdf
- [11] A. Yeung, (2010) Matrix factorization: A simple tutorial and implementation in python. [Online]. Available: <http://www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python/>



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

SVD++ WITH AN IMPROVED BIAS CALCULATION
STEP

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):

IVANOV
RAMASAMY
BAUFI

First name(s):

MARTIN
DHIVYA
GREGORY

With my signature I confirm that


- I have committed none of the forms of plagiarism described in the "Citation etiquette" information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

ETH ZURICH 18.06.2015

Signature(s)


Dheeraj
BauFI Gregory

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.