# Giving Matrix Factorization A Boost

Daniel Yu, Lea Auf der Maur, Luca Lanzendörfer
Team Pendulum, Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—Collaborative Filtering methods have become widely used in consumer oriented e-commerce through different matrix factorization methods. We demonstrate how Collaborative Filtering methods based on matrix factorization can be further improved by boosting many specialized SVD and Neural Network approaches to obtain a competitive score on the CIL dataset.**

## I. Introduction

Recommender Systems and especially Collaborative Filtering methods have become widely used methods in the growing consumer oriented online marketplace. Recommender Systems have in general increased in popularity over the last decade for a variety of different topics such as news, books, music and movies. The two main areas of research are focused around collaborative filtering and content-based filtering. The former focuses on recommending items based on the similarity of a user's preference compared to others whereas the latter uses characteristics of an item to recommend similar items.

The Netflix Challenge [1] gave rise to a new wave of research into Recommender Systems. Up until then the state-of-the-art collaborative filtering methods were based on neighborhood searches and clustering algorithms [2]. The Netflix Challenge concluded with an interesting new insight, that the unknown ratings could be estimated much more reliably using matrix factorization approaches rather than neighborhood methods [3].

In Section II we will discuss the related work which has contributed to the state-of-the-art as well as some of the new approaches using Neural Networks. Our approach builds on these matrix factorization techniques by constructing a variety of different SVD models as well as Neural Networks. We then combine these methods through boosting [4] to further reduce the bias and variance of our final prediction. These models and methods will be discussed in more detail in Section III as well as the boosting framework in Section IV. Section V will conclude this paper by discussing the results and the impact on further research.

## II. Related Work

The field of Recommender Systems has been studied since the early 1990's [5]. Content-based Filtering and Collaborative Filtering methods quickly became widespread as they were closely coupled to the needs of the emerging e-commerce market. Up until the Netflix Challenge state-of-the-art methods used clustering [6], neighborhood and dimensionality reduction methods [7] although it had not yet been fully understood why low dimensional data resulted in better recommendations.

The first team to win the annual prize during the Netflix Challenge blended together 107 results [8] consisting of kNN, factorization models and restricted Boltzmann Machines, realizing that it was better to focus on combining and blending very different approaches rather than refining a single method.

For many years the strongest Collaborative Filtering methods were based on latent factor models using methods such as SVD and SGD to recommend items [9]. However, in recent years Neural Networks have taken the lead and have become the state-of-the-art in recommending items [10] [11].

## III. Methods and Models

We tested different matrix factorization techniques including different SVD methods based on Koren et al. [2] as well as non-negative matrix factorization approaches described by Zhang et al. [12]. Additionally, we implemented different neural network configurations with varying degrees of success.

To be able to achieve a competitive score we used these methods to obtain different base predictors. These predictors were then combined in a boosting framework to obtain a better score than any single predictor was capable of. In the following we will discuss these base predictors in more detail.

### A. SVD Movie Mean

The *SVDMM* predictor extracts a mean per row of the input matrix according to (1). The i-th movie mean is then copied into the i-th row, replacing each unknown entry. We then compute the SVD on this matrix and keep only the $k$ largest singular values. Multiplying these matrices together again we obtain the final matrix which we use as a predictor. This predictor is not capable of producing competitive scores on its own as we obtained a validation error of around 1.0088.

$$movie\_mean_i = \frac{1}{|\Omega_i|} \sum_{(i,j)\in\Omega_i} X_{i,j} \tag{1}$$

where $\Omega_i$ contains all input pairs for the i-th movie and $X$ is the sparse matrix containing all input values.

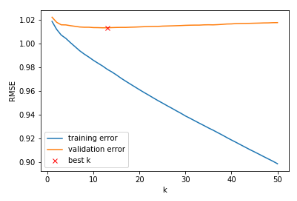

Figure 1: SVD movie mean

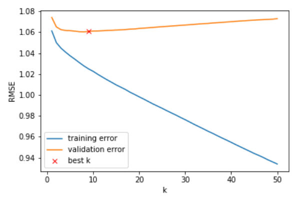

Figure 2: SVD user mean

We select the largest $k$ singular values using cross validation with an 80-20 training and validation split. The validation results can be seen in Figure 1. This is done since the largest singular values correlate to the Eigenvectors carrying most of the information whereas the smaller singular values mostly only correlate to noise [13].

## B. SVD User Mean

The *SVDUM* employs the same idea as *SVDMM* but computes a mean for every user (3) and replaces all columns according to their user means.

$$user\_mean_j = \frac{1}{|\Omega_j|} \sum_{(i,j)\in\Omega_j} X_{i,j} \tag{2}$$

where $\Omega_j$ contains all input pairs for the j-th user and $X$ is the sparse matrix containing all input values.

Using cross validation to find the $k$ most suitable singular values we find that this predictor is also not competitive on its own with a validation error of about 1.0504. Figure 2 shows the validation error for different singular values.

## C. SVD Movie/User Mean Iterative

To improve the result of *SVDMM* and *SVDUM*, we created an iterative version. In each step, the known values are inserted into the matrix and the SVD with rank $k$ is computed on the matrix. We repeat this step $p$ times. The pseudocode can be seen below.

**Algorithm 1** SVD Movie/User Mean Iterative

```
1: matrix ← Initialize matrix with movie/user mean
2: for $i \in \{1, \dots, p\}$ do
3:     matrix ← Insert the know values into matrix
4:     matrix ← Perform SVD with rank $k$ on the matrix
5: return matrix
```

This optimization lowers the score to 0.9905 for *SVDMMI* and 0.9908 for *SVDUMI*.

## D. SVD Combined

Combining the ideas of the previous two methods we created the *SVDC* method. This method uses a pre-processed matrix where each unknown entry contains a linear combination of the user (u), movie (m) and the global (g) mean. The coefficients for the linear combination were found using a grid search over $[-1, 1] \times [-1, 1] \times [-1, 1]$ with 0.1 step sizes. Additionally, we searched over $k = [8, 13]$ to find the $k$ best singular values. The best results are shown in Table I.

| Validation RMSE | Train RMSE | u | m | g | k |
|---|---|---|---|---|---|
| 0.994560 | 0.961807 | -1.0 | -0.8 | 0.8 | 10 |
| 0.994593 | 0.954939 | -1.0 | -0.8 | 0.8 | 13 |
| 0.994606 | 0.962235 | -0.8 | -0.6 | 0.6 | 10 |
| 0.994631 | 0.957193 | -1.0 | -0.8 | 0.8 | 12 |
| 0.994647 | 0.955369 | -0.8 | -0.6 | 0.6 | 13 |

Table I: best coefficients and singular values

## E. SVD on SVD Iterative

Similar to *SVDC* we define a base predictor called *SoSI* which uses the *SVDMMI* as well as *SVDUMI* methods. The idea is that we define two matrices $X_m$ and $X_u$ where the former contains the movie and the latter the user means. Additionally, we added a blending parameter $p$ [14] which is used to calculate more reliable means. The entries are computed as $(\bar{u} + \bar{g}p)/(\#u + p)$ where $\bar{u}$ is the user mean, $\bar{g}$ is the global mean and $\#u$ is the number of rated movies by user $u$. This blending is used to compensate for sparse data since the true mean often lies closer to the apriori mean rather than the observed mean. We set $p = 3$ in all our experiments. Figure 3 shows the *SoSI* pipeline.

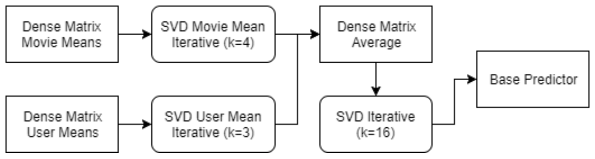

Figure 3: SVD on SVD Iterative pipeline

$X_m$ and $X_u$ are then factorized and reduced to a lower dimensionality as explained in Subsection III-C. These dimensionality reduced matrices $\hat{X}_m$ and $\hat{X}_u$ are then merged through averaging $\bar{X}_{mu} = (\hat{X}_m + \hat{X}_u)/2$. This matrix is then factorized again and only the best $k = 16$ singular values are kept, the result is a predictor which scores around 0.9836 and which is then boosted with the other base predictors to obtain the final prediction.

## F. SVD Offset

*SVDO* follows the idea, that each entry of the matrix can be decomposed into three parts:

$$\hat{r}_{i,j} = p_i^T q_j + bm_i + bu_j \tag{3}$$

where $\hat{r}_{i,j}$ is the estimated rating for movie $i$ and user $j$, $p_i^T q_j$ describes the matrix factorization, $bm_i$ is the bias for movie $i$ and $bu_j$ is the user bias for user $j$.

The pure matrix factorization $p_i^T q_j$ can only describe the interaction between movie $i$ and user $j$, but different users usually have different rating ranges, which is why we need a notion of bias for a user. The same idea applies towards movies. We use this idea to initialize our matrix before applying SVD.

To replicate the movie bias $mb$, which should describe the offset of how much the movie rating deviates from the global mean, we measured the mean of the deviation for each movie. For the user bias $ub$, we measured the mean of the deviation from the global mean for each user, where the movie mean has already been subtracted. To initialize the matrix we subtract the user bias and movie bias respectively from the global mean and perform an SVD to approximate the matrix factorization part and estimate $p$ and $q$. The pseudocode can be seen below:

**Algorithm 2** SVD Offset

```
1: $g\_mean \leftarrow$ Calculate the mean of all known ratings
2: $m\_matrix \leftarrow \left\{ \begin{array}{l} r_{i,j} - g\_mean, \text{ if } r_{i,j} \text{ known} \\ 0, \text{else} \end{array} \right\}_{i,j}$
3: $mb \leftarrow$ mean of known values per row for $m\_matrix$
4: $u\_matrix \leftarrow \left\{ \begin{array}{l} r_{i,j} - g\_mean - mb_j, \text{ if } r_{i,j} \text{ known} \\ 0, \text{else} \end{array} \right\}_{i,j}$
5: $ub \leftarrow$ mean of known values per column for $u\_matrix$
6: $X \leftarrow \{g\_mean + mb_i + ub_j\}_{i,j}$
7: $matrix \leftarrow$ Perform SVD with rank $k$ on $X$
8: return $matrix$
```

### G. SVD Post Processed

*SVDPP* is a continuation of *SoSI*. The idea is to translate the output of the SVD into actual ratings, which could behave in a non-linear way. We split the training set 90 to 10 where the first 90 percent of the data is used to make a prediction using *SoSI*. We fit a third order polynomial through the other 10 percent using ridge regression. Using this approach we obtain a viable validation score of around 0.9860.

### H. SVD Neighborhood Aware

*SVDNA* is also a continuation of *SoSI*. The idea is to do a K-means with the learned movie vectors from the SVD. This method is usually referred to as the item-oriented approach. The central piece of a good item-oriented approach is to have a good working distance/similarity measure of two movie vectors. The paper by Sarwar et al. [15] discusses different distance/similarity functions. For our method we used the *shrunk correlation coefficient* [16] denoted by $s_{i,j}$, which is based on the Pearson correlation:

$$s_{i,j} = 1 - d_{i,j} = \frac{n_{i,j}}{n_{i,j} + 100} p_{i,j} \tag{4}$$

where $n_{i,j}$ is the number of users that rated both movie $i$ and $j$, $p_{i,j}$ is the Pearson correlation between movie $i$ and $j$ and $d_{i,j}$ denotes the distance measurement, which is the complement of the similarity measurement.

To predict an unobserved rating $\hat{r}_{i,u}$ from user $u$, we also consider the top $k$ nearest movie neighbors of movie $i$. We denote the set of the nearest movie neighbors as $S^k(u,i)$. The predicted value is a weighted average of the estimated value and the values of his k nearest neighbors.

$$\hat{r}_{i,u} = \frac{r_{i,u} + c \sum_{j \in S^k_{i,u}} r_{i,j} s_{i,j}}{1 + c \sum_{j \in S^k_{i,j}} s_{i,j}} \tag{5}$$

where $r_{i,j}$ is the estimated rating from the *SoSI* method as preprocessing and $c$ is the damping factor for the neighborhood weights. We obtain a result of 0.9833, using $k = 4$ and $c = 0.01$.

### I. Neural Network

To create the input for our NN we used word embeddings (*WE*) to map user-movie pairs into a vector [17]. The size of the *WE* was found using a grid search over $[10, 100]$ with a step size of 10. We found that a *WE* of size 80 and 30 for the movie and user embedding gave the best results.

Since our dataset does not contain information related to time we could not reproduce the state-of-the-art deep neural network [10]. However, we tried different configurations using the pyramid and monolith schemes as well as dropout layers [18] to avoid overfitting. We found that using a NN with three to five hidden layers and no dropout gave the most promising results. We also found that the difference among *sigmoid*, *tanh* and *ReLU* activation functions were negligible in our case. However, the different optimization functions contributed strongly as *adam* [19] achieved the lowest validation error within 5 epochs whereas *adadelta* [20] needed up to 80 epochs but pushed the error slightly lower. The best results in terms of units per layer were obtained with a monolith scheme where the input layer size is a concatenation of the movie and user *WE*. The results for our monolith scheme with varying embedding sizes and hidden layers can be seen in Table II.

| **Validation RMSE** | **m** | **u** | **d** | **l** |
|---|---|---|---|---|
| 0.993277 | 80 | 30 | 10 | 4 |
| 0.994049 | 70 | 40 | 10 | 5 |
| 0.994158 | 70 | 40 | 10 | 3 |
| 0.994212 | 40 | 40 | 70 | 4 |
| 0.994217 | 60 | 30 | 90 | 4 |

Table II: best parameters using *sigmoid* & *adadelta*

where $m$ and $u$ are the movie and user word embedding size, $d$ is the number of units per hidden layer and $l$ is the number of hidden layers.

In addition to the word embedding input we also implemented an alternate NN where the neural network receives additional side information. For an input consisting of a user-movie pair we additionally added the user and movie mean, the number of times the user rated and the movie was rated, the minimum and maximum ratings given by the user and received by the movie as well as the variance in the ratings given by the user and received by the movie. The idea for these context-based inputs was taken from Strub et al. [21]. We found using these additional inputs coupled with a pyramid scheme gave us only a small improvement, with the lowest validation error being around 0.9976.

## IV. Boosting

To improve the result to 0.9815, we considered the methods described above as weak learners. Our competitive submission is a polynomial combination of these weak learners. To train the whole stack of learners, we split the data into four sets: train data level 1, test data level 1, train data level 2 and test data level 2. The weak learners receive the train data level 1 set for training and we validated their score with the test data level 1. For finding the weights of the polynomial combination we used the train data level 2 and validated the score with the test data level 2. Figure 4 gives an overview of the pipeline.

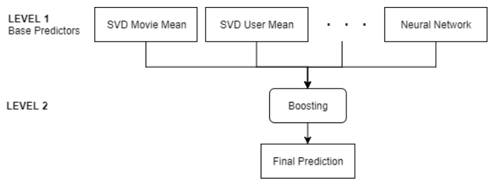

Figure 4: boosting pipeline

For our submission we used a polynomial combination of degree 3 as well as all the configurations of weak learners discussed in Section III.

## V. Results and Discussion

We divide our base methods into three groups: SVD intialization, SVD postprocessing and Neural Network methods.

*SVD initialization*

*SVDMM, SVDUM, SVDC* and *SVDO* are different methods on how to initialize missing values in the user-movie matrix. We can see that compared to the *Global Mean* baseline, just by replacing the missing values with the movie and user mean improves the score by 0.0577 and 0.0153. Combining both means gives an improvement of 0.0702. The best initialization we found was by calculating the offset which gives us an improvement of 0.0781.

*SVD postprocessing*

*SVDMMI*, *SVDUMI*, *SoSI*, *SVDPP* and *SVDNA* are different methods on how to improve the matrix after calculating the SVD with some initialization. *SVDMMI* and *SVDUMI* are built on top of *SVDUM* and *SVDMM*, which improves the score by 0.0596 and 0.0183. By combining these two methods and adding another SVD (*SoSI*) we improve the score by another 0.0069. Unfortunately, *SVDPP* could not improve the score, but *SVDNA*, which is also built on top of *SoSI*, was able to achieve a small score improvement of around 0.0003. Note that iterative methods have a train RMSE of 0 because we are always re-inserting the values we already know.

*Neural Network*

*Neural Network* is a class of itself as it does not use any SVD techniques. Compared to the *SVD Global Mean* it has a score improvement of 0.0681.

| Predictor | Train RMSE | Validation RMSE |
|---|---|---|
| SVD Global Mean | 1.0386 | 1.0657 |
| SVD Movie Mean | 0.9818 | 1.0088 |
| SVD User Mean | 1.0184 | 1.0504 |
| SVD Combined | 0.9620 | 0.9955 |
| SVD Offset | 0.9427 | 0.9876 |
| SVD Movie Mean Iterative | 0.000 | 0.9905 |
| SVD User Mean Iterative | 0.000 | 0.9908 |
| SVD on SVD Iterative | 0.9200 | 0.9836 |
| SVD Post Processed | 0.9277 | 0.9860 |
| SVD Neighborhood Aware | 0.9211 | 0.9833 |
| Neural Network | 0.9527 | 0.9976 |
| Boosting | 0.9780 | 0.9815 |

Table III: comparison between base predictors

As we can see from Table III the largest improvements arise from good initializations, the score can then be further improved through additional postprocessing but the competitive score is only achieved through the final boosting stage which brings the validation error to around 0.9815.

All results were generated using Python with scikit-learn [22] as well as Keras [23] for the Neural Networks.

## VI. Conclusion

Collaborative Filtering has become an essential part in many areas of e-commerce with the Netflix Challenge bringing a new wave of research into the field.

In this paper we have shown a variety of different initialization and postprocessing methods for SVD. We show that blending different weak learners is an approach which manages to produce competitive scores on the CIL dataset by reducing the underlying bias and variance of each weak learner. We have combined these weak learners through a 2-level boosting architecture enabling us to easily add or remove base predictors as well as being able to easily test additional boosting methods.

From our experience it is clear that the initialization is the most fundamental part of being able to obtain good predictions, especially on non-iterative models. We believe that this area could be a basis for future work as well as an extensive search on the configuration space of more unique weak learners.

## References

[1] J. Bennett, S. Lanning *et al.*, "The netflix prize," in *Proceedings of KDD cup and workshop*, vol. 2007. New York, NY, USA, 2007, p. 35.

[2] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, 2009.

[3] Y. Koren, "The bellkor solution to the netflix grand prize," *Netflix prize documentation*, vol. 81, pp. 1–10, 2009.

[4] R. E. Schapire, "The strength of weak learnability," *Machine learning*, vol. 5, no. 2, pp. 197–227, 1990.

[5] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," *Communications of the ACM*, vol. 35, no. 12, pp. 61–70, 1992.

[6] L. H. Ungar and D. P. Foster, "Clustering methods for collaborative filtering," in *AAAI workshop on recommendation systems*, vol. 1, 1998, pp. 114–129.

[7] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Analysis of Recommendation Algorithms for e-Commerce," in *Proceedings of the 2Nd ACM Conference on Electronic Commerce*, ser. EC '00. New York, NY, USA: ACM, 2000, pp. 158–167. [Online]. Available: http://doi.acm.org/10.1145/352871.352887

[8] R. M. Bell, Y. Koren, and C. Volinsky, "The bellkor 2008 solution to the netflix prize," *Statistics Research Department at AT&T Research*, 2008.

[9] Y. Koren and R. Bell, "Advances in collaborative filtering," in *Recommender systems handbook*. Springer, 2015, pp. 77–118.

[10] H. Wang, N. Wang, and D.-Y. Yeung, "Collaborative deep learning for recommender systems," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1235–1244.

[11] C. Du, C. Li, Y. Zheng, J. Zhu, C. Liu, H. Zhou, and B. Zhang, "Collaborative Filtering with User-Item Co-Autoregressive Models," *arXiv preprint arXiv:1612.07146*, 2016.

[12] S. Zhang, W. Wang, J. Ford, and F. Makedon, "Learning from incomplete ratings using non-negative matrix factorization," in *Proceedings of the 2006 SIAM International Conference on Data Mining*. SIAM, 2006, pp. 549–553.

[13] M. W. Berry, S. T. Dumais, and G. W. OBrien, "Using linear algebra for intelligent information retrieval," *SIAM review*, vol. 37, no. 4, pp. 573–595, 1995.

[14] S. Funk, "Netflix Update: Try This at Home," http://sifter.org/simon/journal/20061211.html, 2006.

[15] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 285–295.

[16] Y. Koren, "Factorization meets the neighborhood: a multifaceted collaborative filtering model," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 426–434.

[17] O. Barkan and N. Koenigstein, "Item2vec: neural item embedding for collaborative filtering," in *Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on*. IEEE, 2016, pp. 1–6.

[18] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[19] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *CoRR*, vol. abs/1412.6980, 2014. [Online]. Available: http://arxiv.org/abs/1412.6980

[20] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," *CoRR*, vol. abs/1212.5701, 2012. [Online]. Available: http://arxiv.org/abs/1212.5701

[21] F. Strub, J. Mary, and R. Gaudel, "Hybrid Collaborative Filtering with Autoencoders," *arXiv preprint arXiv:1603.00806*, 2016.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[23] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.