# The effectiveness of Bayesian Factorization Machines

Riccardo De Santi, Federico Gossi, Antonio Lopardo, Luca Malagutti
Team name: SpaghettiCode
Department of Computer Science, ETH Zürich, Switzerland

***Abstract*—Collaborative Filtering (CF) is a method that leverages preference and taste data collected on many users, to make automatic predictions on the interests of a given user, assuming that similar users will tend to have similar tastes. In this report, we illustrate our work on a CF algorithm in the context of the Computational Intelligence Lab (CIL) CF project competition. We extend and improve Bayesian SVD++, a state-of-the-art CF algorithm, adding a form of Empirical Bayes for hyperprior initialization and custom weight scaling during Monte Carlo sampling as well as a form of ensemble learning for the final submission.**

## I. Introduction

Recommender systems have been a practical application of machine learning from the inception of the field. Since the wider adoption of the internet and the push for personalized user experiences on its largest websites, these systems have become more and more relevant. Netflix, in particular, has made use of such methods extensively, and in an effort to challenge the wider public to improve on its own recommendation system, in 2006 it published anonymized ratings data and offered prizes to the best performing submissions. Many of the solutions originally developed for the Netflix Prize challenge were based on Matrix Factorization (MF) or Singular Value Decomposition (SVD). After more than a decade, despite the significant improvements in the field of Machine Learning, methods heavily inspired by MF still prove quite competitive, especially when dealing with limited computing power and smaller training sets. We tested some of these methods, namely FunkSVD [1], SVD++ [2], and BayesianSVD++ [3], inspired by the results and arguments from [4], and we found them fast, easy to use, and performant on the dataset of the CIL CF competition. This dataset, see Figure 1, has a similar format compared to the Netflix Prize data, though it is smaller, ten times less sparse and its ratings don't carry the date on which they were submitted to the service. We also experimented with other models: some combine Neural Networks and MF-based methods like NNMF [5], others make use of AutoEncoders like DeepRec [6], while others are based on Graph Neural Networks (GNNs) [7] like LightGCN [8] and IGMC [9]. All three of these approaches looked promising with impressive benchmarks and an extensive literature behind them, but when tested on the CIL CF dataset they performed worse than BayesianSVD++, even after significant adaptations and experimentations.

$$\begin{array}{c} \\ U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_n \end{array} \begin{array}{c} \begin{array}{ccccc} I_1 & I_2 & I_3 & \cdots & I_m \end{array} \\ \begin{bmatrix} 0 & 0 & 3 & \cdots & 1 \\ 0 & 2 & 0 & \cdots & 0 \\ 5 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 4 & 1 & \cdots & 0 \end{bmatrix} \end{array}$$

Fig. 1. Zero filled rating matrix format of the CIL CF-2021 competition dataset

TABLE I
Baselines validation and submission results

| Model | Validation Mean | Public Test score | std |
|---|---|---|---|
| LightGCN | 0.9995 | 0.9997 | $1.3 \cdot 10^{-4}$ |
| DeepRec | 0.9815 | 0.9826 | $7.8 \cdot 10^{-4}$ |
| FunkSVD | 1.0228 | 1.0241 | $4.0 \cdot 10^{-3}$ |
| NNMF | 0.9946 | 0.9909 | $3.1 \cdot 10^{-3}$ |
| SVD++ | **0.9792** | **0.9777** | $7.1 \cdot 10^{-5}$ |

## II. Methods

In this section we present the baselines to which we compare our final submission. The results of each baseline are summarized in Table I. Every baseline model was trained 5 times on runs with different seeds. We report the mean of the validation scores and their standard deviation, together with the submission score on the public test set for the model with the lowest validation score out of the 5 we trained.

### A. Graph Neural Networks

During the first part of our work on this project we directed our attention on the recent developments in the sub-field of Collaborative Filtering with GNNs.

*a) LightGCN:* LightGCN [8] is a recent model for collaborative filtering based on Graph Neural Networks. LightGCN learns user and item embeddings by iteratively aggregating the features of neighboring nodes on the user-item graph. It is a simpler and more lightweight version of a previous GNN-based architecture [10], from which it removes both feature transformation matrices and non-linear activation functions from the graph convolutions, keeping only the most essential features of GNNs for the problem of recommendation prediction. Each final model prediction is simply defined as

the inner product of the learned user and item representations. In its original implementation, LightGCN uses the Bayesian Personalized Ranking (BPR) loss [11], and is evaluated on the task of Top-N ratings prediction using recall and normalized discounted cumulative gain (ndgc) as metrics on a binary rating matrix. To adapt this architecture to the CIL dataset on the task of collaborative filtering, we rewrote its official Pytorch implementation completely from scratch, replacing the BPR loss with an RMSE loss, and adapting the graph convolution matrix to make it possible to handle ratings distributed between 1 and 5. Our final model converges in around 100 iterations, but its training time is over 10 times longer than the other baselines' while achieving only sub-optimal results. Despite the somewhat unsatisfactory outcome, we still believe that a better performing ranking prediction model based on GNNs could be useful to generate latent representations which encode graph feature information for both users and items. These embeddings could then be added as additional input to other models.

### B. Autoencoders

As tools that easily achieve dimensionality reduction, Autoencoders have been widely used to perform Collaborative Filtering [12] [13].

*a) DeepRec:* The DeepRec architecture [6], in particular, uses an unconstrained deep autoencoder for rating prediction. This model is trained completely end-to-end using SELU (scaled exponential linear units) activation units and high dropout percentages. It takes as inputs the user vectors of the original rating matrix and tries to estimate all the missing ratings by encoding and decoding the input vectors to and from a latent space. The predicted ratings are then optimized using an MSE loss masked on the training data of each user. For our baseline results, we used a model with 6 layers, 3 for the encoder and 3 for the decoder. Our experimental findings suggest that for the competition dataset, the best encoder (and decoder) architecture consists of three layers, the second one of which is smaller than the other two. For instance, in all the reported baseline runs, an encoder and decoder with layer sizes of (128, 32, 128) were used, achieving very satisfactory results for a relatively under-optimized model architecture. These valid results initially encouraged us to try to improve the DeepRec architecture as our project contribution. Further details on these attempts can be found in the Experiments section of this report.

### C. Matrix Factorization

Matrix Factorization is a class of algorithms that are wildly used for CF [14]. They work by learning a decomposition of the rating matrix into the product of two lower-rank rectangular matrices, which contain latent space representations of users and items. We present three baselines based on this kind of model: FunkSVD, NNMF and SVD++.

*a) FunkSVD:* A relatively simple yet effective algorithm developed as part of the Netflix Prize competition is FunkSVD. This method decomposes the rating matrix R in a matrix of users by latent factors and an items by latent factors matrix. When multiplied, the two matrices return the reconstructed R matrix. The main idea behind FunkSVD is to randomly initialize the two matrices and update their values with SGD using the known elements of R as training data. We can also insert some bias terms for the global rating mean and the average offsets from the mean of each user and item.

$$\widehat{r}_{ui} = \mu + b_u + b_i + U_u V_{\mathrm{i}}^T \tag{1}$$

$U_u$ is the latent factor vector of the user, $V_i^T$ is the latent factor vector of the item, $b_u$ and $b_i$ are the bias terms for the user and the item while $\mu$ is the global rating bias.

*b) SVD++:* SVD++ [2] is an extension to FunkSVD that takes into account implicit feedback. The implicit feedback usually consists of side information that indicates the preference of a user towards an item. When we only have access to rating information, the implicit feedback is based on the set of items $R(u)$ rated by the user $u$:

$$\widehat{r}_{ui} = \mu + b_u + b_i + V_i \left( U_u^T + |R(u)|^{-\frac{1}{2}} \sum_{j \in R(u)} r_{uj} \right) \tag{2}$$

As shown by the results of SVD++ in table I, implicit feedback is particularly useful in our task. Since defining $R(u)$ in LibFM does not require the exact rating [3, Section 4.1.3], we can construct $R(u)$ by taking the items rated by a user from both the training set and the test set. This is additional information that is not utilized when working on the training set only.

*c) NNMF:* As a promising extension to vanilla Matrix Factorization models, we also considered Neural Network Matrix Factorization [5] as a baseline. In this model, the rating matrix is still split into two low-rank factors, but the inner product, usually used to combine the learned user and the item latent vectors for each rating prediction, is replaced by a learnable arbitrary function. This function is encoded by a multi-layer perceptron, which is trained together with the rest of the model. In particular, the learning process involves alternating between fixing the latent features and optimizing the network and optimizing the latent features while keeping the MLP fixed. NNMF is trained with RMSProp and early stopping and performs gradient descent without the use of batches.

## III. Results

We now briefly introduce the concept of Factorization Machine (FM), which is central in our final model.
Consider a setting with a set $S$ of tuples $(\boldsymbol{x}, y)$, where $\boldsymbol{x} \in \mathbb{R}^p$ is a feature vector and $y$ is its corresponding target. FMs model all nested interactions up to order $d$ between the $p$ input variables in $\boldsymbol{x}$ using interaction parameters. The factorization machine (FM) model of order $d = 2$ is defined as:

$$\hat{y}(\boldsymbol{x}) = w_0 + \sum_{j=1}^{p} w_j x_j + \sum_{j=1}^{p} \sum_{j'=j+1}^{p} x_j x_{j'} \sum_{f=1}^{k} v_{jf} v_{j'f} \tag{3}$$

In [3, Section 4.1] the authors have shown that the matrix factorization model SVD++ can be seen as a particular instance of a FM. As a consequence, the fully Bayesian treatment that is given to FMs can be adopted for SVD++ as well, thus leading to a model called BayesianSVD++. In particular, as detailed in [3, Section 3.4], priors and hyperpriors on the model parameters are introduced, and Bayesian inference is done by sampling the posterior distributions through a collapsed Gibbs sampler.

### A. Adding New Features

*a) kNN features:* In CF, neighborhood-based approaches (also known as kNN, for short) identify pairs of items that tend to be rated similarly or by users with a similar history of rating [15]. In order to estimate the unknown rating $r_{ui}$ with the weighted average of the neighbors' ratings, we need the set of users $N(u, i)$ that tend to rate similarly to $u$ and rated the item $i$. Alternatively, we need the set of items $N(i, u)$ that tend to be rated similarly to $i$ and were rated by the user $u$. As described in [3], for every entry $(u, i, r_{u,i})$ we added features $r_{u,i1}/m, \ldots, r_{u,im}/m$ for each rating that the user $u$ has given to other items $i_1, \ldots, i_m$, and features $r_{u1,i}/n, \ldots, r_{un,i}/n$ for each rating that the item $i$ received from other users $u_1, \ldots, u_n$. This approach turned out to perform worse than the baseline, probably because the information about other ratings of each user-item pair is already modeled by the 2nd-order-interaction parameters $v$ in equation 3.

*b) 3rd order features:* Factorization machines were extended to higher-order feature combinations in[16]. And in [17], the implementation of higher-order factorization machines performs better with order 3, this suggests that BayesianSVD++ could also benefit from modeling 3rd order interactions. For this implementation, we used the code from [18] to get the predictions of a 3rd order FM, and we added these predictions as features for each entry $(u, i, r_{u,i})$, but these additional features did not have much effect on the score. This could be due to the low quality of the predictions of the 3rd order FM (validation RMSE was greater than 0.99), or to the fact that the 3rd order FM models uses also 1st and 2nd order interactions, which are already modeled by BayesianSVD++.

### B. Simulated Annealing

We have introduced a simulated annealing process in order to reduce the random-walk behavior during the first iterations of the Gibbs sampling algorithm when applied on the posteriors of the model parameters. In short, we have attempted to make the posterior distributions arbitrarily uniform by altering their entropy through a temperature parameter $t$. More specifically, exploiting the linear relationship between entropy and variance in the Gaussian distribution, we have altered the variance of the posterior $\mathcal{N}(\mu,\, t\sigma^2)$ by initializing $t$ at a value strictly bigger than 1, and then decreasing it to 1 through a geometric cooling schedule, namely $t_{i+1} \leftarrow \alpha t_i$. We have tried several cooling schedules and parameters, but the Gibbs sampler convergence dynamics did not seem to improve. This may be due to the fact that the original sampler is a collapsed Gibbs sampler, which should have less burn-in and random-walk behavior than a vanilla Gibbs sampler. The results in II are obtained with $t_0 = 1.2$, $\alpha = 0.99$, $t_i = max\{1,\, t_0 \cdot \alpha^i\}$, $i$ is the iteration number of the Gibbs sampler.

### C. Partial Empirical Bayes

We initialized the parameter $\alpha_0$ of the hyperprior used in BayesianSVD++ with the value obtained during a previous run after reaching convergence of the Gibbs sampler to the stationary distribution. This process, since it infers a prior distribution from data, can be regarded as an Empirical Bayes method [19, Section 5.6] with an underlying parametric distribution.

### D. Scaled weights in Monte Carlo average

During inference, the Gibbs sampler implemented in LibFM uses a Monte Carlo average of every sample as the maximum a posteriori of the Gaussian posterior distribution of each parameter. To prevent the burn-in period of the Gibbs sampler from producing low-quality samples for the maximum a posteriori, we tried to discard the first $k$ samples when computing the Monte Carlo average during inference. However, the validation results obtained by discarding 5, 10, or 50 samples were consistently worse than the ones obtained while considering the full Monte Carlo average. This behavior could be explained by a regularizing effect of the first samples. Following this intuition, we tried to give more weight to the first samples $s_i$ when computing the Monte Carlo average to obtain the maximum a posteriori of some parameter $p$:

$$p = \frac{\sum_{i=0}^{n} w_i s_i}{\sum_{i=0}^{n} w_i} \qquad w_i = 1 + \omega_0 \gamma^i \tag{4}$$

In equation 4 the simple average is obtained when every $w_i = 1$, the "Scaled Monte Carlo" results reported in table II are obtained using $\omega_0 = 2$, $\gamma = 0.95$, and obtain a small improvement over the baseline.

### E. Ensemble

For the final submission, we implemented an ensemble of different models. Firstly, we generate predictions for the validation split using models trained on the training split. These level 1 models consist of 5 BayesianSVD++ with Empirical Bayes and Scaled Monte Carlo and 1 DeepRec model, all trained with different seeds. Secondly, we train various level 2 models (1 linear regressor, 3 ridge regressors, 1 XGBoost Regressor, and a simple average of the level 1 predictions) on features consisting of the predictions of the level 1 models, user ID, item ID, number of ratings of the user, number of ratings given to the item. Lastly, we make a simple average over the predictions of the level 2 models on the submission set.

## IV. Discussion

The main advantages induced by a fully Bayesian treatment of the model are related to regularization. Firstly, it lets us integrate the regularization parameters into the model, removing the need for complex hyperparameter tuning. Secondly,

TABLE II
BAYESIANSVD++ ABLATION STUDY

| Method | Validation | Public Test score | std |
|---|---|---|---|
| Baseline | 0.9688 | 0.9678 | $8.4 \cdot 10^{-5}$ |
| kNN features | 0.9695 | 0.9688 | $1.6 \cdot 10^{-4}$ |
| 3rd Order | 0.9687 | 0.9681 | $1.0 \cdot 10^{-4}$ |
| Annealing | 0.9689 | 0.9681 | $8.4 \cdot 10^{-5}$ |
| Empirical Bayes | 0.9686 | 0.9678 | $1.9 \cdot 10^{-4}$ |
| Scaled Monte Carlo | 0.9685 | 0.9676 | $1.5 \cdot 10^{-4}$ |
| EmpBayes + ScaledMC | 0.9684 | 0.9677 | $1.1 \cdot 10^{-4}$ |
| Ensemble | **0.9674** | **0.9668** | - |

it allows us to include the MCMC method that, following a Bayesian model selection criterion [19, Section 5.3], acts as an implicit regularizer. Moreover, MCMC improved by the scaled Monte Carlo average, leads to better validation performance by giving more weight to the samples from the prior distributions. Still, the primary drawback of BayesianSVD++ is that it needs to be entirely re-trained if a new user or item is added to the dataset, which would be significant in most real-world applications. Nonetheless, it does not have any impact on the performance in our setting, since the users and items are fixed. It's worth noting, however, how mathematical insights and powerful approximate inference techniques applied on simpler models can lead to better performance than that of more recent neural-based methods.

## V. EXPERIMENTS

Aside from testing additions to BayesianSVD++, we also tried to improve some of the approaches that looked promising after our initial run of baselines. We tried to increase the modeling power of the autoencoder-based DeepRec by adding an MLP component on top of its encoders and attempted to find a GNN-based solution more suited to our task in IGMC.

### A. DeepRec++

We experimented with combining different neural approaches, autoencoders, and MLPs, to exploit the advantages of both. More specifically, we trained two different DeepRec [6] autoencoders: one for the items and one for the users, in order to get reliable latent vector representations. As shown in Figure 2, to generate the final prediction, instead of relying on dense reconstructions like DeepRec or on a standard dot product, for each rating we feed to an MLP the embeddings of the corresponding user and item and extract the predicted 1-5 rating after 3 layers. This approach was also inspired by one of our baselines, NNMF [5] and by Neural Collaborative Filtering [20]. Both these models learn latent vector representations for users and items that are then used as input for an MLP, even though they initialize their embedding and only train them by backpropagating the MLP loss through the embedding layer. Our idea consists in training the latent vectors not only using the MLP loss on the ratings, but also the reconstruction loss of DeepRec. The primary motivation behind this joint training scheme was to start with more meaningful representations as inputs to the MLP. The results of

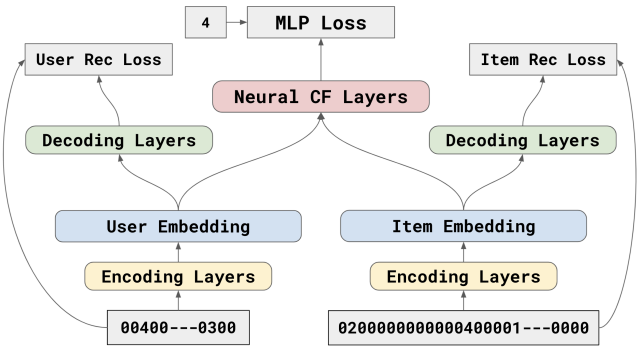


Fig. 2. DeepRec++ Network diagram

the new model, however, couldn't match the submission scores of DeepRec alone, despite the different hyperparameters and training schedules that we tested.

### B. IGMC

Matrix factorization (MF) is intrinsically transductive, meaning that the latent representations for user-item pairs are not generalizable to pairs unseen during training. Inductive Graph-based Matrix Completion (IGMC) [9] turns MF into an inductive model. For each observed rating, IGMC adds an edge between the corresponding user and item, eventually obtaining a bipartite graph from the rating matrix. As a consequence, predicting unknown ratings converts to predicting new edges in this graph. IGMC solves this problem by building for each user-item pair $(u, v)$ a subgraph with nodes $u, v$ and their neighbors within $h$ hops in the bipartite graph. Such local subgraphs contain graph pattern information about the rating that user $u$ may give to the item $v$. Eventually, a GNN [7] is trained through a graph regression that maps each subgraph to the rating given by its center user to its center item. A trained GNN can be applied to unseen users or items without retraining, thus making the model inductive. Even though the model seems promising, since it solves the CF problem by taking into account graph information, it shows a major drawback: the space complexity of the algorithm suffers from very high memory constants due to the subgraphs generation routine. Hundreds of GBs of RAM are needed to run the model on the CIL dataset. This major drawback makes IGMC hard to compare with the methods discussed above.

## VI. CONCLUSION

In this work, we have outlined the main contributions that allowed us to improve the performance of BayesianSVD++, one of the state-of-the-art architectures for CF. We have also shown that older MF-based methods still vastly outperform more recent and often more complex models on a dataset devoid of additional rating features. Our final submission obtains an RMSE of 0.96679 as its public test score on the provided dataset, placing our team at the second position on the public leaderboard for the Collaborative Filtering competition of the CIL course.

## REFERENCES


[1] S. Funk, "Netflix update: Try this at home," December 11th, 2006. [Online]. Available: https://sifter.org/~simon/journal/20061211.html

[2] 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*, ser. KDD '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 426–434. [Online]. Available: https://doi.org/10.1145/1401890.1401944

[3] S. Rendle, "Factorization machines with libfm," vol. 3, no. 3, May 2012. [Online]. Available: https://doi.org/10.1145/2168752.2168771

[4] S. Rendle, L. Zhang, and Y. Koren, "On the difficulty of evaluating baselines: A study on recommender systems," *CoRR*, vol. abs/1905.01395, 2019. [Online]. Available: http://arxiv.org/abs/1905.01395

[5] G. K. Dziugaite and D. M. Roy, "Neural network matrix factorization," *CoRR*, vol. abs/1511.06443, 2015. [Online]. Available: http://arxiv.org/abs/1511.06443

[6] O. Kuchaiev and B. Ginsburg, "Training deep autoencoders for collaborative filtering," 2017.

[7] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, and M. Sun, "Graph neural networks: A review of methods and applications," *CoRR*, vol. abs/1812.08434, 2018. [Online]. Available: http://arxiv.org/abs/1812.08434

[8] X. He, K. Deng, X. Wang, Y. Li, Y. Zhang, and M. Wang, "Lightgcn: Simplifying and powering graph convolution network for recommendation," *CoRR*, vol. abs/2002.02126, 2020. [Online]. Available: https://arxiv.org/abs/2002.02126

[9] M. Zhang and Y. Chen, "Inductive graph pattern learning for recommender systems based on a graph neural network," *CoRR*, vol. abs/1904.12058, 2019. [Online]. Available: http://arxiv.org/abs/1904.12058

[10] X. Wang, X. He, M. Wang, F. Feng, and T. Chua, "Neural graph collaborative filtering," *CoRR*, vol. abs/1905.08108, 2019. [Online]. Available: http://arxiv.org/abs/1905.08108

[11] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme, "BPR: bayesian personalized ranking from implicit feedback," *CoRR*, vol. abs/1205.2618, 2012. [Online]. Available: http://arxiv.org/abs/1205.2618

[12] S. Sedhain, A. Menon, S. Sanner, and L. Xie, "Autorec: Autoencoders meet collaborative filtering," 05 2015, pp. 111–112.

[13] F. Strub and J. Mary, "Collaborative Filtering with Stacked Denoising AutoEncoders and Sparse Inputs," in *NIPS Workshop on Machine Learning for eCommerce*, Montreal, Canada, Dec. 2015. [Online]. Available: https://hal.inria.fr/hal-01256422

[14] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," 2009.

[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*, ser. WWW '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 285–295. [Online]. Available: https://doi.org/10.1145/371920.372071

[16] S. Rendle, "Factorization machines," in *Proceedings of the 2010 IEEE International Conference on Data Mining*, ser. ICDM '10. USA: IEEE Computer Society, 2010, p. 995–1000. [Online]. Available: https://doi.org/10.1109/ICDM.2010.127

[17] M. Blondel, A. Fujino, N. Ueda, and M. Ishihata, "Higher-order factorization machines," 2016.

[18] A. N. Mikhail Trofimov, "tffm: Tensorflow implementation of an arbitrary order factorization machine," https://github.com/geffy/tffm, 2016.

[19] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[20] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua, "Neural collaborative filtering," *CoRR*, vol. abs/1708.05031, 2017. [Online]. Available: http://arxiv.org/abs/1708.05031