A Collaborative Effort in Analyzing Collaborative Filtering Methods

Stuart Heeb (stheeb), Samyak Jain (samyjain),
Johannes Kurz (kurzj), Yuval Nissan (ynissan)
Group: Runtime Terror
Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—Collaborative filtering is a technique widely used in recommender systems that simultaneously uses similarities between users and items to predict user preferences and produce suggestions. In this paper, three different types of methodologies, namely Matrix Factorization, Neural Networks, and Bayesian Factorization Machines are discussed. Experiments were conducted on user-movie rating data with model-specific structural/feature-related changes and a combination of different approaches. Results suggest that Bayesian Factorization Machines are the best performing method for this particular dataset.**

I. Introduction

Collaborative filtering is an application of the matrix completion task and exploits collective data from many users to predict a user's interaction with an item.

The CIL dataset provides an incomplete 10000×1000 user-movie matrix with $1'176'952$ movie integer ratings ranging from 1 to 5, so roughly 11% of the rating matrix is filled. The goal is to predict $1'176'952$ ratings for movies users have not yet provided a rating for as accurately as possible. As this problem is very relevant in the industry, a large variety of literature and models target it on which one can build to achieve good performance. Therefore, this paper presents various models from three categories (Matrix Factorization, Neural Networks, and Factorization Machines). It analyzes and compares their base performance on the provided dataset and experiments with changes to specific models, including structural and feature-related changes and the combination of different approaches, to improve their performance on the provided dataset.

We show that although the presented approaches provide very high or even state-of-the-art performance on MovieLens-1M/10M [1][2][3], they did not achieve similar scores on the CIL dataset displaying a visible performance gap. Furthermore, despite extensive hyper-parameter tuning, neural-network approaches could not even outperform the baseline. The best performance was shown by a Bayesian Factorization Machine using probit ranking, user-implicit, and item-implicit features, as well as SVD-generated user and movie embeddings as additional features, ultimately resulting in a public score of 0.96634.

II. Models and Methods

1. Matrix Factorization

First, we explored traditional matrix factorization methods presented in [4] and evaluated their performance on the CIL dataset.

a) SVD combined with ALS

This approach, which is presented as the baseline method in [5], begins with a pre-processing step, where the observed ratings are normalized per-item. The unobserved values are set to 0. The corresponding user-item matrix is then decomposed using Singular-Value Decomposition (SVD). The output is then passed to Alternating Least Squares (ALS) and the predicted complete user-item matrix is obtained via "denormalization".

b) Improved SVD

A regularized version of SVD along with biases can be used to enhance the prediction. Here, a single predicted value can be written as $\hat{r}_{ui} = \vec{p}_u^T \vec{q}_i + b_u + b_i$ [6] where $\vec{p}_u$ is the user factor, $\vec{q}_i$ is the item factor and b_u, b_i are the respective biases. The factors and biases are learned by minimizing the objective

$$\sum_{r_{ui} \in R_{\text{train}}} (r_{ui} - \hat{r}_{ui})^2 + \lambda(b_i^2 + b_u^2 + \|\vec{q}_i\|_2^2 + \|\vec{p}_u\|_2^2), \quad (1)$$

where R_{train} is the observed ratings and λ the regularization parameter. To apply this concept the Surprise library [7] was used.

c) Extensions of SVD & ALS

To enhance the performance of SVD, instead of setting the unobserved values to zero in the preprocessing step, they were initialised using a k-Nearest Neighbours approach as well as Gaussian processes.

2. Neural Networks

a) Neural Net for Collaborative Filtering

The idea of our approach is based on the work in [8], and includes additional inspiration from [9][10]. The code for our neural net uses some auxiliary functions directly or with minor adjustments from [8]. After comparing and experimenting with different architectures for the neural net, we settled on the following: The neural net has a two-dimensional input node, whose inputs are the user and movie ID respectively. After

this, each ID is separately fed into an embedding layer, with embedding dimensions `user_embedding_size` and `movie_embedding_size` respectively. These embeddings are then individually flattened and concatenated. The concatenated vector is fed through three dense layers with `activation` functions being ReLU or the sigmoid function, each followed by a dropout layer with probability 0.1. The first layer has `max_units` units, the second layer has half as many units and the third layer has a fourth as many units. After this, there is another dense layer with 100 units and a ReLU activation function. The final layer is a dense layer with a single output which provides the predicted rating.

b) Sparse-FC (formulas obtained from [11])

Sparse-FC [11] is an item-based autoencoder where the weight matrices $\mathbf{V}$ and $\mathbf{W}$ of the encoder and decoder are re-parameterized using a sparsifying kernel function K.

Re-parameterized by a kernel means that

$$\mathbf{V}_{ij} = \alpha_{ij} K(\vec{v}_i, \vec{u}_j) \text{ and } \mathbf{W}_{ij} = \beta_{ij} K(\vec{s}_i, \vec{t}_j) \tag{2}$$

where the d-dimensional vectors $\vec{v}_i$, $\vec{u}_j$, $\vec{s}_i$ and $\vec{t}_j$ as well as the scalars α_{ij} and β_{ij} are the trainable parameters of the model. The kernel function used is $K(\vec{x}, \vec{y}) = max(0, 1 - \|\vec{x} - \vec{y}\|_2^2)$ and is considered sparsifying because it only has final support and therefore the connection between a pair of neurons might go to zero. The model takes as input an item-vector $\vec{r_i} \in \mathbb{R}^n$ i.e. a vector for the i-th movie containing the rating of all n users and outputs an item-vector $\vec{r_i}' \in \mathbb{R}^n$ including predictions for the missing ratings.

The loss function L is

$$L = \text{MSE} + \lambda_2 \left(\sum_{ij} \alpha_{ij} + \sum_{ij} \beta_{ij} \right) + \lambda_0 \left(\sum_{ij} K(\vec{v}_i, \vec{u}_j) + \sum_{ij} K(\vec{s}_i, \vec{t}_j) \right), \tag{3}$$

where MSE is the mean squared error between the predicted and the true ratings. λ_2 can be seen as an L_2 regularization parameter and λ_0 as an L_0 regularization parameter.

c) GLocal-K (formulas obtained from [12])

GLocal-K [12] (Global-Local Kernel) currently provides state of the art performance on MovieLens-1M [2] and can be seen as an extension of Sparse-FC [11]. Being a kernelized sparsified autoencoder using the same kernel function K and same kernel layer as in Sparse-FC the architecture is nearly identical to Sparse-FC. But it includes an additional fine-tuning step in the training process. First the kernelized autoencoder is trained like in Sparse-FC, then in the fine-tuning step a global-kernel matrix is calculated in two steps:

- First average-pooling is applied to summarize each reconstructed item-vector $\vec{r_i}'$ computed by the trained autoencoder, i.e. $\mu_i = avgpool(\vec{r_i}')$.
- Then, given m convolution kernels $k_1, \ldots, k_m$ (one for each of the m movies), the global kernel is calculated as $GK = \sum_{i=1}^{m} (\mu_i \cdot k_i)$.

The global kernel is then applied to the initial user-item matrix R via convolution, resulting in $\hat{R} = R \otimes GK$. The autoencoder is then trained again with partially different hyper-parameters (compared to the first training step) using the global kernel-based rating matrix $\hat{R}$ as input, resulting in the final trained model.

d) Knowledge Graphs

Knowledge Graphs (KG) are a collection of fact triples (subject, relation, object). They are primarily used for link prediction, i.e., to infer new relations from the existing ones [13]. Here, we tried to transform the matrix completion problem by encoding users as subjects, ratings as relations, and movies as objects; the idea is to learn KG representations to infer unknown relations, aka ratings. We experimented with three different types of KG models, namely additive (TransE), multiplicative (DistMult), and additive+multiplicative (MuRE) [13]. For each user-movie combination, these models output each rating with a specific score based on their respective score functions. We applied a softmax activation on these scores and then took a weighted average of the ratings to get the final rating for the user-movie combination. The PyKEEN [14] library was used to implement Knowledge Graphs.

3. Factorization Machines

Traditional matrix factorization methods discussed in section II-A use only the user-item interaction matrix and it is difficult to include other auxiliary features to improve prediction. In contrast, factorization machines (FM) [15] allow to include more information as they use tuples of real-valued feature vectors and a numeric target variable to represent each user-item interaction [16], making them a more general framework. A 2nd order FM equation [15] is

$$\hat{y} = w_0 + \sum_{i=1}^{P} w_i x_i + \sum_{i=1}^{P-1} \sum_{j=i+1}^{P} \langle v_i, v_j \rangle x_i x_j \tag{4}$$

where w_0 is common bias, w_i is the weight associated with i^{th} feature, x_i is the i^{th} feature, v_i is the latent space embedding of the i^{th} feature and P is the number of features. Observe that if instead of the inner product of factorized embeddings, a specific weight w_{ij} would have been used, then the equation would be an instance of quadratic linear regression [16]. The usage of shared parameters not only reduces the number of learnable parameters but also helps model dependencies between different features and works well on sparse data [15].

[3] and [17] suggests Bayesian variant of factorization machines are state-of-the-art for Movielens 10M dataset. It is also known that Bayesian Inference of FM leads to better accuracy and automated hyper-parameter tuning[18]. Hence,

we explored Bayesian Factorization Machines (BFM) to solve the task. Given a prior distribution, the goal is to maximize the posterior probability of the model parameters [16]. For implementation purposes, we use the myFM package [19] that uses the Gibbs sampling approach to approximate the posterior distribution, as otherwise it is often intractable to do so. The following ideas were used to generate feature vectors:

- **User-item interaction**: binary one-hot encoding vectors for user and movie indicators.
- **User-implicit features**: bag-of-words predictor variable that comprises all movies rated by the user. This version of BFM is called Bayesian SVD++ [17][20].
- **Movie-implicit feature**: bag-of-words predictor variable that comprises all users that rated a movie. The version of Bayesian FM that uses user-implicit features and movie-implicit features is called Bayesian SVD++ flipped [17][20].
- **Weighted user/movie implicit-features**: in the original implementation of Bayesian SVD++ (flipped), equal normalized weights are given to both the user & movie implicit features. Here, weighted features were generated such that the weights were assigned proportionally to the rating. The idea was that weighted signals might improve the feature vector and perform better.
- **Ratings as feature**: bag-of-words predictor variable that comprises all ratings given by the user and all ratings received by a movie. Here, the idea was to learn biases, for example a movie receiving 4s/5s but no 1s/2s.
- **User/movie embeddings**: In the e-Commerce industry, usually, the user/product embeddings are used as feature vectors for different machine learning tasks [21]. Along the same lines, we generated embeddings using a non-biased version of SVD and used them as feature vectors.

Next, since the ratings by their very nature are ordinal, we explored ordered probit regression. The model assumes the existence of a latent variable determined by the factorization machine. It also assumes thresholds of the latent variable at which the rating changes [22][23]. These thresholds are called cutpoints and are also learned by the model. From these assumptions, the conditional distribution of the ratings given a feature vector can be derived. The weights and cutpoints are learned by utilizing MyFM's [19] ordered probit regression method that uses a Gibbs Sampler to estimate them. Thus, for one sample probability of every rating is obtained. The final prediction is obtained by doing a weighted average using these probabilities. Rank, the number of iterations, and the number of samples are the hyperparameters tuned using a GridSearch-based approach.

III. Results

1. Matrix Factorization

Method	*SVD + ALS*	*KNN + SVD + ALS*	*GP + SVD + ALS*	*Improved SVD*	*Improved SVD + ALS*
Public score	0.98759	0.98758	0.98759	0.98860	0.98950

Table I
THE SCORES FOR THE MATRIX FACTORIZATION METHODS.

Table I shows the relative comparison of various matrix factorization approaches. The experiments demonstrate that SVD + ALS, SVD Improved both can predict ratings up to a root mean squared error of around 0.9875. Having better initialisation does not significantly improve the performance. The hyper-parameters used for the different initializations can be found in VI-A1.

2. Neural Networks

Method	**Validation score**	**Public score**
NN for Collab. Filt.	0.99502	0.99341
Sparse FC	0.98648	1.00040
GLocal-K	0.99102	1.00440
Knowledge Graphs	1.29922	1.31130

Table II
THE SCORES FOR THE NEURAL NETWORK APPROACHES.

The scores for the different neural networks approaches are shown in Table II.

a) Neural Net for Collaborative Filtering

The neural network architecture we developed to predict ratings based on the dataset provided to us underperformed with respect to our expectations. From our research we know that neural nets of this kind are expected to perform better on datasets like MovieLens [1] than what we see for our dataset, which is a RMSE score of 0.99341. Keras-hypertune[24] was used to tune hyperparameters as well as internal parameters of the model. The parameters that achieved the best score can be found in Section VI-A2.

b) Sparse-FC

The experiments have shown that although Sparse-FC [11] is one of the highest performing models on MovieLens-10M [3] it completely underperformed on the CIL dataset and was not even close to beating the baseline. The best performing hyper-parameters can be found in the Section VI-A3 and were computed by a grid search using 90% of the data for training and 10% of the data for validation.

c) GLocal-K

GLocal-K was not able to show good results on the CIL dataset which due to its similarity to Sparse-FC is not very surprising. The additional fine-tuning step actually downgraded the performance compared to Sparse-FC. The best performing hyper-parameters can be found in VI-A4.

Hyper-parameter optimization was performed in the same ways as for Sparse-FC.

d) Knowledge Graphs

Here, we report the performance of the best model, i.e., TransE. The experiments suggest that Knowledge Graphs are not suitable for matrix completion tasks. However, they can be used to provide an efficient ranking of the products for a particular user [25] and also can be used in the context of the creation of Knowledge Base (metadata information about users/items) [26][27] to improve the recommendations.

3. Bayesian Factorization Machines

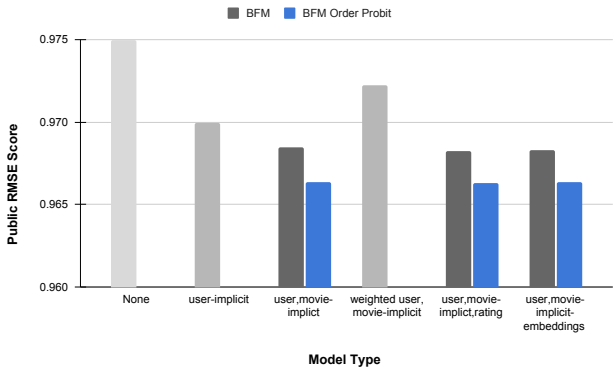


Figure 1. The scores for factorization machine approaches.

Figure 1 demonstrates the performance of Bayesian Factorization Machines using different features. On the x-axis incremental addition of feature/model change is shown. None denotes that no additional features except the user-item interaction is used. As expected, the BFMs do much better than traditional matrix factorization methods owing to the strengths of Bayesian Inference and the ability to add other auxiliary features. Adding user-implicit features significantly improves the performance and corroborates the findings of [17]. Adding movie implicit feature has a slight performance improvement. Contrary to the expectation, Weighted user/movie implicit features did not show performance improvement but became worse than uniform weights. However, it was still better than the original BFM, which contained no additional features. Adding ratings as a feature does not seem to significantly impact the performance. Adding the embeddings as feature vectors also seem to have little effect on performance. Usage of ordered probit regression appears to have a significant impact on performance. As seen in the previous trends, adding the rating/embeddings as features has little performance impact. A more detailed plot for validation RMSE can be found in Figure 2 in the appendix.

IV. Discussion

1. Dataset Analysis

The MovieLens [1] dataset is one of the most famous datasets for collaborative filtering, and it has the same setting as the CIL dataset, except it also contains a timestamp for each entry. The MovieLens-1M datasets is most comparable to the CIL dataset, as it contains 1'000'209 ratings from 6040 users and 3952 movies, which gives the matrix a sparsity of approximately 4.2%. The CIL dataset contains 1'176'952 ratings from 10'000 users and 1000 movies, which results in a sparsity of approximately 11.77%. The ratings 1-5 are distributed as follows:

$$\texttt{cil}: \quad 3.70\%, 8.43\%, 23.30\%, 27.59\%, 36.98\%$$
$$\texttt{ml-1m}: \quad 5.62\%, 10.75\%, 26.11\%, 34.89\%, 22.63\%$$

2. Implications Possibly Caused by Dataset Differences

We noticed that the state-of-the-art methods [2] for MovieLens-1M [1], in particular neural networks, did not perform well on the CIL dataset compared to what is achievable on the MovieLens-1M dataset.

This seems even more paradoxical when we consider the fact that the MovieLens-1M dataset is more than two times as sparse as the CIL dataset. Comparing the two datasets, we see that the distribution of ratings are more realistic for the MovieLens-1M dataset, after all it is based on real-world data. For the CIL dataset, the ratio of ratings is monotonically increasing as the rating increases, which might be considered unrealistic, as almost 37% of the given entries are rated with 5, and only around 12% have a rating of 1 or 2.

3. Overall Performance Analysis

Matrix factorization methods performed reasonably well on the provided data. As stated in the above sections, neural network-based approaches did not meet expectations. Bayesian Factorization Machines, which are the current state-of-the-art on Movielens-10M dataset [3] performed the best and ordered probit mechanism further improved the performance. Based on the performance we choose Bayesian SVD++ flipped with embeddings and order probit as the best performing model.

V. Summary

This paper discusses a comparative study of various collaborative filtering methods. Three approaches are explored: matrix factorization, neural networks, and Bayesian Factorization Machines. Several structural, feature-based, and combination techniques have been tried to improve the performance. In the end, a Bayesian Factorization Machine together with an ordered probit mechanism seem to be the best performing solution for the given task. Additional user/item metadata can easily be added to the factorization machine framework, which can lead to further performance improvement and be tried as future work.

References

[1] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, dec 2015. [Online]. Available: https://doi.org/10.1145/2827872

[2] "Recommendation systems on movielens 1m." [Online]. Available: https://paperswithcode.com/sota/collaborative-filtering-on-movielens-1m

[3] "Recommendation systems on movielens 10m." [Online]. Available: https://paperswithcode.com/sota/collaborative-filtering-on-movielens-10m

[4] G. Rätsch, "Lectures 4-6 slides in computational intelligence lab, spring semester 2022," March-April 2022.

[5] X. Li, "Tutorial 5 slides in computational intelligence lab, spring semester 2022," April 2022.

[6] A. Paterek, "Improving regularized singular value decomposition for collaborative filtering," in *Proceedings of KDD cup and workshop*, vol. 2007, 2007, pp. 5–8.

[7] N. Hug, "Surprise: A python library for recommender systems," *Journal of Open Source Software*, vol. 5, no. 52, p. 2174, 2020. [Online]. Available: https://doi.org/10.21105/joss.02174

[8] "Data Liftoff". (2018) How to recommend anything / deep recommender. [Online]. Available: https://www.kaggle.com/code/morrisb/how-to-recommend-anything-deep-recommender

[9] 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

[10] H. Xue, X. Dai, J. Zhang, S. Huang, and J. Chen, "Deep matrix factorization models for recommender systems," in *IJCAI*, 2017.

[11] L. Muller, J. Martel, and G. Indiveri, "Kernelized synaptic weight matrices," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 3654–3663. [Online]. Available: https://proceedings.mlr.press/v80/muller18a.html

[12] S. C. Han, T. Lim, S. Long, B. Burgstaller, and J. Poon, "Glocal-k: Global and local kernels for recommender systems," in *Proceedings of the 30th ACM International Conference on Information amp; Knowledge Management*, ser. CIKM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 3063–3067. [Online]. Available: https://doi.org/10.1145/3459637.3482112

[13] C. Allen, I. Balazevic, and T. M. Hospedales, "On understanding knowledge graph representation," 2019.

[14] "Pykeen." [Online]. Available: https://github.com/pykeen/pykeen

[15] S. Rendle, "Factorization machines," in *2010 IEEE International conference on data mining*. IEEE, 2010, pp. 995–1000.

[16] E. Lundquist. (2020) Factorization machines for item recommendation with implicit feedback data. [Online]. Available: tinyurl.com/2u7nwykw

[17] S. Rendle, L. Zhang, and Y. Koren, "On the difficulty of evaluating baselines: A study on recommender systems," *arXiv preprint arXiv:1905.01395*, 2019.

[18] S. Rendle, "Bayesian factorization machines," in *Proceedings of the NIPS Workshop on Sparse Representation and Low-rank Approximation*, 2011.

[19] T. Ohtsuki. myfm. [Online]. Available: https://github.com/tohtsky/myFM

[20] S. Rendle, "Scaling factorization machines to relational data," *Proceedings of the VLDB Endowment*, vol. 6, no. 5, pp. 337–348, 2013.

[21] L. Singh, S. Singh, S. Arora, and S. Borar, "One embedding to do them all," *arXiv preprint arXiv:1906.12120*, 2019.

[22] Ordinal regression. [Online]. Available: https://en.wikipedia.org/wiki/Ordinal_regression

[23] J. H. Albert and S. Chib, "Bayesian analysis of binary and polychotomous response data," *Journal of the American statistical Association*, vol. 88, no. 422, pp. 669–679, 1993.

[24] M. Cerliani. (2022) keras-hypetune. [Online]. Available: https://github.com/cerlymarco/keras-hypetune

[25] Y. Zhang, J. Wang, and J. Luo, "Knowledge graph embedding based collaborative filtering," *IEEE Access*, vol. 8, pp. 134 553–134 562, 2020.

[26] D. Zhang, L. Liu, Q. Wei, Y. Yang, P. Yang, and Q. Liu, "Neighborhood aggregation collaborative filtering based on knowledge graph," *Applied Sciences*, vol. 10, no. 11, p. 3818, 2020.

[27] R. Mu and X. Zeng, "Collaborative filtering recommendation algorithm based on knowledge graph," *Mathematical Problems in Engineering*, vol. 2018, 2018.

VI. Appendix

1. Hyperparameters

a) Baseline Initializations

The best parameters found for the baseline method (SVD + ALS), were the parameters given in [5]. i.e. the number of iterations was set to 20, the number of latent factors was set to 3 and the regularization coefficient was set to 0.1. We used these parameters also in the different baseline initializations, alongside the parameters needed separately for each one. For the k-NN iputer initialization, the number of nearest neighbours was set to 5, and the Euclidean metric was used to determine distance between observed user ratings. For the Gaussian processes initialization, the `noise` parameter was set to 1.0 and the `bandwidth` parameter was set to 0.1. For the implementation of the k-NN Imputer, the `sklearn` python package (class `sklearn.impute.KNNImputer`) was used.

b) Neural Net for Collaborative Filtering

The best parameters found by keras-hypetune[24] for the neural net are `user_embedding_size` $= 40$, `movie_embedding_size` $= 40$, `max_units` $= 800$, `lr` $= 0.001$, `batch_size` $= 128$, `activation` $=$ ReLU.

c) Sparse-FC

Parentheses contain the name of the hyper-parameters in the code and their value in the best performing configuration:

- dimension d of $\vec{v}_i$, $\vec{u}_j$, $\vec{s}_i$ and $\vec{t}_j$ (`n_dim` $= 10$)
- number of hidden layers of the encoder and decoder (both have the same number) (`n_hid` $= 200$)
- λ_2 (`lambda_2` $= 60.0$)
- λ_0 (`lambda_s` $= 0.013$)
- number of epochs (`n_epoch` $= 1$)
- maximum number of iterations performed by the L-BFGS-B optimizer (`output_every` $= 300$)

d) GLocalK

- see Sparse-FC (`n_dim` $= 5$)
- see Sparse-FC (`n_hid` $= 500$)
- see Sparse-FC (`lambda_2` $= 20.0$)
- see Sparse-FC (`lambda_s` $= 0.006$)
- see Sparse-FC (`n_layers` $= 2$)
- size of global kernel (`gk_size` $= 3$)
- maximum number of iterations performed by the L-BFGS-B optimizer for first training (`iter_p` $= 5$)
- maximum number of iterations performed by the L-BFGS-B optimizer for fine-tuning (`iter_f` $= 5$)
- number of epochs of first training step (`epoch_p` $= 30$)
- number of epochs of fine-tuning step (`epoch_f` $= 60$)
- scaling during calculation of global kernel (`dot_scale` $= 1.0$)

e) Knowledge Graphs

The optimal parameters are `model` $=$ TransE, `embedding_dim` $= 50$.

f) Factorization Machines

Different models had different optimal values, but majorly for BFM regression `rank` $= 12$ and ordered probit regression `rank` $= 16$ were found to be optimal with the number of samples as 495 and iterations as 500 for both types.

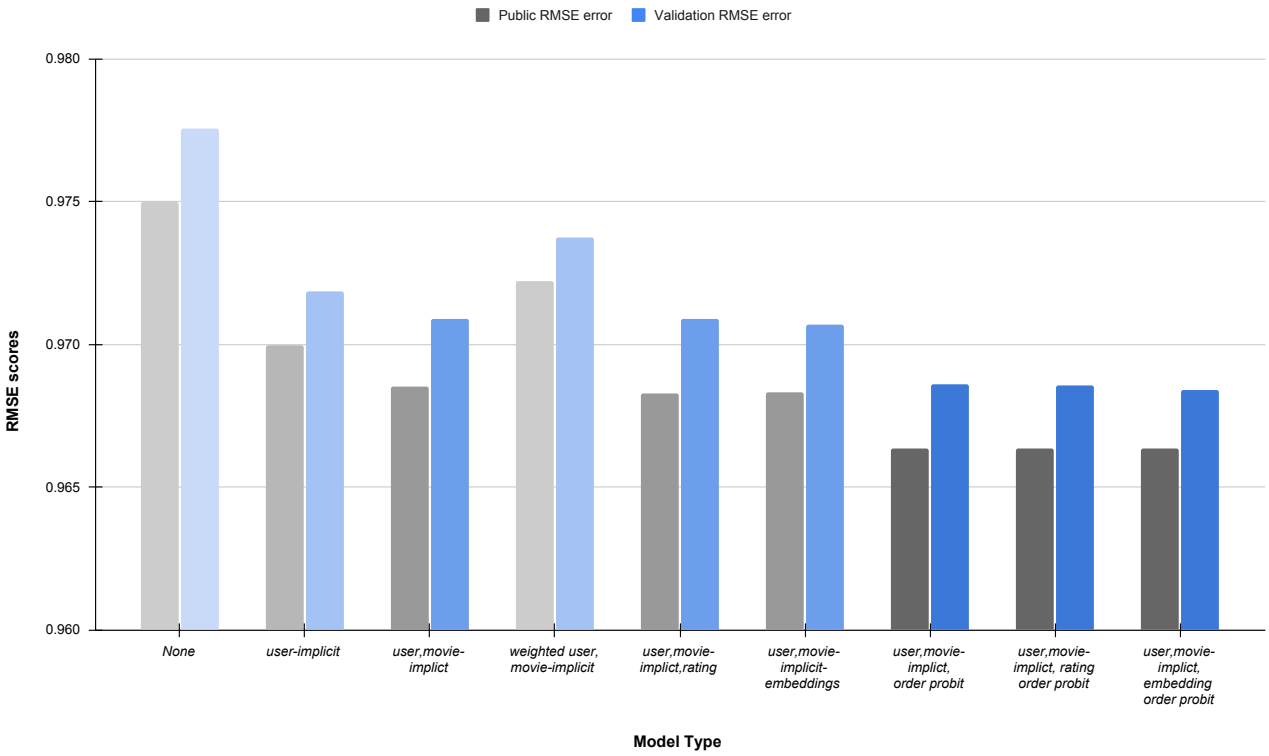


Figure 2. Public and Validation Scores for Factorization Machines

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):

A Collaborative Effort in Analyzing Collaborative Filtering Methods

Authored by (in block letters):

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

Name(s):	**First name(s):**
Heeb	Stuart
Jain	Samyak
Kurz	Johannnes
Nissan	Yuval

With my signature I confirm that

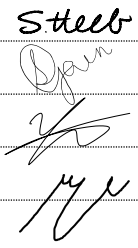
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

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

Place, date

Zurich, July 31st 2022

Signature(s)



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