An Ensemble of Matrix Factorization and Neural Models for Improved Recommendations

Group: CILly Geese

Carmel Baharav[1], Alessandro Girardi[1], Patryk Morawski[1], and Jakob Nogler[1]

[1]*Department of Computer Science, ETH Zurich, Switzerland*

***Abstract*—In the context of Machine Learning, collaborative filtering is the use of sparse user-item rating data to infer missing ratings. It has several applications, most notably within the realm of recommender systems. In this project report for Computational Intelligence Lab FS23, we describe various models that we utilize, including matrix factorization methods and neural network-based techniques. Besides implementing pre-existing and studied methods, we put forth two new models: Collaborative Filtering with Dual Autoencoder (CFDA++) and Graph-Enhanced Recommender with Neural Outputs and Transformations (GERNOT). We ultimately leverage an ensemble that takes a weighted average across model choices, scoring 0.9633 on the Kaggle competition.**

I. Introduction

Recommender systems have a rich history, dating back to the early 1990s when the concept of collaborative filtering was first introduced [1], [2]. Collaborative filtering relies on the idea that users who have agreed in the past will agree in the future, and it has since become a cornerstone of many successful recommender systems. Over the years, researchers and practitioners have developed and improved upon collaborative filtering techniques, leading to the emergence of classical matrix factorization methods and more recent advancements in deep learning-based models such as Neural Collaborative Filtering and Autoencoders.

This project explores the aforementioned established collaborative filtering techniques, and proposes two novel models: an improved version of Collaborative Filtering with Dual Autoencoder (CFDA++) and Graph-Enhanced Recommender with Neural Outputs and Transformations (GERNOT). Our final predictions are generated using an ensemble algorithm, which leverages the diverse predictions obtained from the different models at our disposal. By combining the outputs of various models, we exploit their differing strengths to improve overall performance. In this report, we will first introduce each of the models utilized as baselines and as members of the ensemble, as well as our design choices and customizations. Then, we will discuss our ensembling methodology and our testing procedure. Finally, we conclude with a discussion of the ensemble performance in contrast to various baselines, and possible future directions of inquiry.

In our setting we are provided with a dataset containing ratings given by 10,000 users for 1,000 different items. All ratings are represented as integer values, ranging from 1 to 5 stars. The task is to build a collaborative filtering algorithm that can predict missing ratings. Formally, we can describe the dataset and notation as follows:

- The set of users is denoted as $I = [n]$, where n is the total number of users;
- The set of movies (items) is denoted as $J = [m]$, where m is the total number of movies (items);
- The set $\Omega \subseteq I \times J$ represents the existing user-item pairs for which we have collected ratings;
- The ratings are stored in a matrix $R \in \{0, \dots, 5\}^{I \times J}$, where $R_{i,j}$ denotes the rating given by user i to movie j if $(i, j) \in \Omega$. If the rating is missing, $R_{i,j} = 0$;
- We use the notation $R_i := (R_{i,j})_{j \in J} \in \mathbb{R}^J$ to represent the user-vector for user i, and $R_j := (R_{i,j})_{i \in I} \in \mathbb{R}^I$ to represent the movie-vector for movie j;
- Given a user item pair (i, j) (which might also not be contained in Ω), our models predict a rating $\hat{R}_{i,j} \in \mathbb{R}$.

The objective will be to minimize the root-mean-squared error (RMSE) between the predicted ratings and the ground truth ratings for a given set of missing ratings.

II. Models and Methods

In this section we introduce the models that we implemented throughout our experiments and describe how we sought to improve their performance on our dataset. All of the code, as well as links to packages we used, can be found in our repository.

A. Alternating Least Squares, funkSVD, and SVD++

These first three approaches are matrix factorization methods, which means that they model the ratings matrix R as the product of a matrix of latent user factors, $U \in \mathbb{R}^{n \times k}$, and latent item factors, $V \in \mathbb{R}^{k \times m}$, for some $k \leq \min(n, m)$. The estimate of a user-item rating $\hat{R}_{i,j}$ is then given by the inner product of the user factor with the item factor:

$$\hat{R}_{i,j} = u_i^\top v_j.$$

In order to prevent overfitting, both Alternating Least Squares (ALS) and funkSVD add a regularization term to the loss with hyperparameter $\lambda > 0$:

$$\ell(U, V) = \frac{1}{2} \|\Pi_\Omega(Y - UV)\|_F^2 + \lambda(\|U\|_F^2 + \|V\|_f^2).$$

ALS minimizes this loss function by alternating between solving a closed form optimization problem for U in terms of V and then V in terms of U. In contrast, funkSVD, as introduced by Simon Funk [3] minimizes the above loss function using stochastic gradient descent and introduces user and item bias terms.

Finally, SVD++ [4] accounts for implicit ratings with the assumption that just by rating an item, a user exhibits more preference for it than if they had not rated that item. The estimate of a rating is updated to include $N(i)$, the set of items that the user rated, and y_j, learned factors for each item:

$$\hat{R}_{i,j} = \overline{R} + bu_i + bi_j + v_j^\top \left(u_i + |N(i)|^{-1/2} \sum_{j' \in N(i)} y_{j'} \right).$$

B. BayesianSVD++

Probabilistic matrix factorization is an alternative approach to estimating the latent factors for users and items that assumes the presence of Gaussian noise. Given factors U and V, as before, and Gaussian noise with precision α, the conditional distribution over observed rating matrices is:

$$p(R \mid U, V, \alpha) = \prod_{(i,j) \in \Omega} \mathcal{N}(R_{i,j} \mid (UV)_{ij}, \alpha^{-1}).$$

Additionally, the prior distributions over user and movie feature vectors are modeled as Guassian, e.g.:

$$p(U \mid \mu_U, \lambda_U) = \prod_{i=1}^{n} \mathcal{N}(U_i \mid \mu_U, \lambda_U^{-1}).$$

In [5], the authors propose a method to introduce additional priors on hyperparameters, and perform approximate inference using the Gibbs sampling algorithm (a Monte Carlo Markov Chain method). These methods can be adapted to factorization machines [6], a more general framework that can be used for matrix factorization but allows for additional feature engineering and inclusion of implicit feedback as is used in SVD++ [7]. We note here that we also included the test user-item pairs as part of the implicit ratings set, a design choice that led to improved performance. See Table I in Section III for a comparison of scores with and without this additional implicit feedback.

Despite being introduced in 2013 by a combination of [5], [4], [8] experiments from as recently as 2019 [9] indicate that variants of BayesianSVD++ outperform nearly all of the new collaborative filtering methods proposed in the intervening years. Corroborating this result, we found that BayesianSVD++ did extremely well on our provided dataset and single-handedly outperformed an ensemble of the other methods described in this paper.

C. Neural Collaborative Filtering

The previously described methods use the dot product as a similarity function: they merge a user embedding with an item embedding to generate a single score that represents the user's preference for the item. Another viable approach is leveraging neural networks to learn the similarity function, since they have the capability to approximate any function. This generalization has a strong mathematical basis, as demonstrated by multiple works [10], [11], showing how certain matrix factorization methods can be interpreted as a special case of neural collaborative filtering.

Given a user-movie pair, Neural Collaborative Filtering first maps the user and movie to their respective embeddings and then feeds the concatenation of the embeddings into a neural network. In the optimization process, we optimize the values which parameterize the embeddings and the layers of the neural network.

In practice, we achieved optimal results by employing the `torch.nn.Embedding` module to learn embeddings of size 32 for users and 16 for movies. Our neural network architecture consisted of two additional hidden layers, each with sizes 64 and 16, respectively. Throughout the optimization process, we applied commonly employed techniques for training neural networks to fine-tune the above-described objective and improve the overall performance of our model.

D. Autoencoders

Autoencoders are utilized to learn compact representations of user vectors $R_i \in \mathbb{R}^I$ in a lower-dimensional space $\mathbb{R}^k$ ($k \ll |I| = n$) and have been successfully applied in collaborative filtering tasks [12], [13], [14]. The autoencoder architecture is comprised of an encoder $E : \mathbb{R}^I \to \mathbb{R}^k$, generating the latent space representation, and a decoder $D : \mathbb{R}^k \to \mathbb{R}^I$, aiming to reconstruct the original vector.

In our case, both E and D are Multilayer Perceptrons, and the network is trained using gradient descent to minimize the mean squared loss:

$$\mathsf{MSE}(E, D) = \frac{1}{|\Omega|} \sum_{(i,j) \in \Omega} (R_{i,j} - D(E(R_i))_j)^2.$$

The predictions for previously unseen reviews can be obtained from the reconstructed user vectors as $\tilde{R}_{i,j} = D(E(R_i))_j$.

We endeavored to improve the autoencoder architecture through various strategies, such as employing one-hot encodings of reviews in the input vector, training the network to predict user-movie interactions $(i, j) \in \Omega$, and implementing an autoencoder to learn representations of movie-vectors instead of user-vectors. Our observations revealed that the first two modifications demonstrated potential for improvement and thus were incorporated into our baseline model. This model involved a 3-layer encoder, a 4-layer decoder, and a latent space dimension

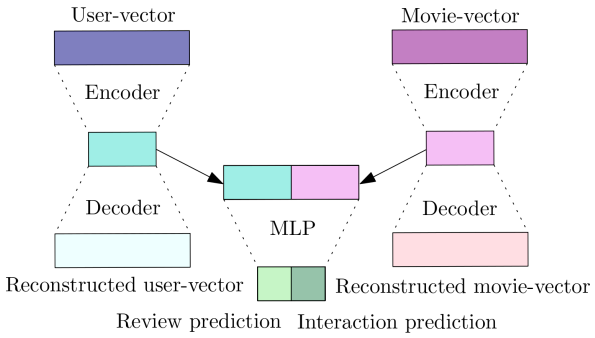

Figure 1. An overview of the CFDA++ architecture.

of $k = 64$. However, the improvements were inconsistent across different hyperparameter settings.

Additionally, we explored dropout and dense refeeding methods as described in [14], but these techniques did not yield favorable outcomes. We conjecture that the smaller size of our networks might have reduced the risk of overfitting, limiting the effectiveness of these regularization methods in our setting.

E. CFDA and CFDA++

In their recent paper, Liu and Wang [15] introduced the Collaborative Filtering with Dual Autoencoder (CFDA) architecture, which integrates the concepts of autoencoders and Neural Collaborative Filtering. The CFDA architecture is comprised of two autoencoder components, (E_I, D_I) and (E_J, D_J), responsible for handling movie and user vectors, respectively. Additionally, it includes a Multilayer Perceptron F, which takes latent representations obtained from E_I and E_J and predicts the review for the given user-movie pair. Joint training of all three parts aims to minimize the combined mean squared losses of the autoencoders and F, resulting in the final predictions as $\hat{R}_{i,j} = F(E_I(R_i), E_J(R_j))$.

To enhance the CFDA architecture, we extended it by incorporating implicit feedback, inspired by the idea underlying the development of SVD++. Consequently, F was modified to output two values approximating $R_{i,j}$ and $\mathbb{1}_{(i,j)\in\Omega}$ respectively for each input user-movie pair (i, j). The revised architecture is illustrated in Figure 1. Moreover, in contrast to the original paper's use of Ω only, we trained our model on the set $\Omega \cup \{(i_l, j_l) \overset{iid}{\sim} Unif(I \times J) \mid l \in [n']\}$ to introduce training instances where the user has not interacted with the given movie. In our experiments, the optimal performance was achieved with $n' := 2|\Omega|$.

The conducted experiments revealed that the modified architecture, named CFDA++, outperformed CFDA significantly and scored better than the autoencoder and NCF approaches. A comparison of the best scores achieved by both models is provided in Table I. The findings show the effectiveness of CFDA++ in collaborative filtering tasks and underscore the value of incorporating implicit feedback to enhance recommendation accuracy.

F. LightGCN

One of the approaches that was considered is LightGCN, a relatively recent model model based on graph convolutions. The latest work conducted by Xiangnan He et al. [16] mainly focused on understanding which factors helped the model work and which were unhelpful or even detrimental, proposing changes over NGCF [17] that simplified the learning procedure.

A graph is constructed from the positive user-item interactions with a rating of at least 3, with each user and each item being a vertex in the graph. When providing a recommendation, the model takes advantage of the information of related vertices one layer after the next, first the items related to the user, then the users related to these items, a further layer of items related to these users and so on. The authors of the original paper argue that by doing so the model is able to learn a better embedding that explicitly utilizes the collaborative signal.

Graph Neural Networks are based around two functions, an aggregation function which is a permutation-invariant operation performed on all the neighbouring vertices, and an update function which is used to update a vertex internal state when all the messages from neighbouring vertices have been aggregated. LightGCN uses simple rules for aggregation and update, summing up the state of the neighbours and normalizing using the graph Laplacian, which can be interpreted as a discount factor which reduces the importance of farther away vertices.

To form the final embeddings of the users and the items a simple average between the states is used. The authors show that this captures the effect of self-connections which are typically used in the update function. The final predictions are computed from the inner product.

The original paper does not suggest the addition normalization parameters since it is used for recommendation, while here we are trying to obtain a numerical value from 1 to 5. Normalizing over the user scores proved successful, with a much faster convergence rate and a lower error when keeping otherwise the same hyperparameters. Normalizing over the items had an even bigger effect.

G. GERNOT

As part of our experimental endeavors, we sought to integrate three deep neural network-based architectures, namely Autoencoders, Neural Collaborative Filtering, and LightGCN. The result of this combination is our novel architecture named Graph-Enhanced Recommender with Neural Outputs and Transformations (GERNOT), which bears resemblance to LightGCN while introducing significant differentiating aspects.

The first notable distinction is related to the initial embeddings $\mathbf{e_i}^{(0)}$. Unlike conventional approaches employing a lookup table, our model generates these embeddings through a Multilayer Perceptron, using both movie-vectors and user-vectors. This involves employing two separate MLP Encoders, one for processing user-vectors and the

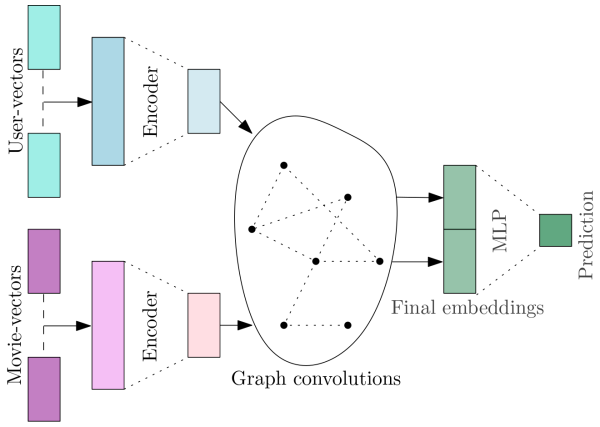

Figure 2. An overview of the GERNOT architecture.

other for movie-vectors. Subsequently, the standard Graph convolutions are performed, leading to the final embeddings for each user and movie.

The second divergence lies in the method used to compute predictions for user-movie pairs. Instead of employing a simple dot-product between the corresponding embeddings, we opt to feed the obtained embeddings into a Multilayer Perceptron, which then calculates the prediction. For a comprehensive illustration of the entire GERNOT architecture, refer to the diagram presented in Figure 2.

Despite the theoretical justification and the combination of three high-performing models, GERNOT did not achieve the same level of performance as any of its individual predecessors. This suggests that certain design choices in the model may have introduced an unnecessarily complicated structure, hindering its effectiveness.

H. Ensemble

For the final submission, we adopted an ensemble approach, incorporating multiple implemented models with diverse hyperparameter configurations. To synthesize the predictions from each model effectively, we employed a weighted average of the respective prediction vectors. These ensemble weights were iteratively optimized through the use of gradient descent, aiming to minimize the mean squared error on a withheld validation set. For comprehensive insights into the composition and characteristics of the ensemble, we invite interested readers to consult our repository.

III. Results

Table I presents the best scores achieved on both the validation set and Kaggle by each individual model, as well as the scores achieved by our final ensemble. The validation set used to compute the validation score is the same set used to train the ensemble weights.

Most of our implementations individually surpassed the public baseline on Kaggle, with the exception of SVD++ and ALS. The relatively bad performance of SVD++ might be due to the particular implementation that was used.

[1]Using only user-movie interactions from the training set.
[2]Using all the user-movie interactions given.

Model	Validation Score	Public Test Score
ALS	0.9890	0.9887
funkSVD	0.9882	0.9872
SVD++	0.9954	0.9941
BayesianSVD++[1]	0.9713	0.9692
BayesianSVD++[2]	0.9671	0.9648
NCF	0.9864	0.9855
Autoencoder	0.9802	0.9789
CFDA	0.9827	0.9817
CFDA++[1]	0.9812	0.9796
CFDA++[2]	0.9801	0.9778
LightGCN	0.9840	0.9794
GERNOT	0.9882	0.9873
Ensemble	**0.9652**	**0.9633**

Table I
INDIVIDUAL RUNS OF EACH OF THE MODELS.

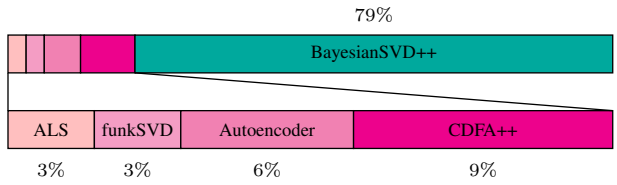

Figure 3. Overview of the contribution of each model to the ensemble.

Refer to Figure 3 for a visualization of the weights learned by the ensemble, as described in Section II-H. BayesianSVD++ contributes significantly to this ensemble due to its outstanding individual performance compared to other models. However, we found that incorporating the other models in the ensemble still resulted in an improvement over a purely BayesianSVD++ ensemble.

IV. Discussion

In this report, we gave an overview of all of the models used as baselines and final ensemble components. Additionally, we discussed numerous design choices and their impact on model performance. Inspired by the pre-existing techniques, we also proposed and implemented two novel models.

The first, CFDA++, incorporates implicit feedback into the CFDA model with impressive results. This evokes two further lines of questioning: could other implicit feedback be utilized for CFDA (for example, other datasets include timestamps)? Also, what other new models could benefit from this treatment of implicit feedback? The second model, GERNOT, combines Autoencoders, Neural Collaborative Filtering, and LightGCN to middling success. We would be excited to see further development of GERNOT in a way that realizes its full potential.

Ultimately, the ensemble outperformed all individual models, showcasing the benefits of leveraging diverse models to improve overall performance. The BayesianSVD++ model achieved the highest validation and public test scores, followed by LightGCN and CFDA++.

REFERENCES

[1] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "Grouplens: An open architecture for collaborative filtering of netnews," in *CSCW '94, Proceedings of the Conference on Computer Supported Cooperative Work, Chapel Hill, NC, USA, October 22-26, 1994*, J. B. Smith, F. D. Smith, and T. W. Malone, Eds. ACM, 1994, pp. 175–186. [Online]. Available: https://doi.org/10.1145/192844.192905

[2] B. M. Sarwar, G. Karypis, J. A. Konstan, and J. Riedl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the Tenth International World Wide Web Conference, WWW 10, Hong Kong, China, May 1-5, 2001*, V. Y. Shen, N. Saito, M. R. Lyu, and M. E. Zurko, Eds. ACM, 2001, pp. 285–295. [Online]. Available: https://doi.org/10.1145/371920.372071

[3] S. Funk. Netflix update: Try this at home. [Online]. Available: https://sifter.org/simon/journal/20061211.html

[4] 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, Las Vegas, Nevada, USA, August 24-27, 2008*, Y. Li, B. Liu, and S. Sarawagi, Eds. ACM, 2008, pp. 426–434. [Online]. Available: https://doi.org/10.1145/1401890.1401944

[5] R. Salakhutdinov and A. Mnih, "Bayesian probabilistic matrix factorization using markov chain monte carlo," in *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*, ser. ACM International Conference Proceeding Series, W. W. Cohen, A. McCallum, and S. T. Roweis, Eds., vol. 307. ACM, 2008, pp. 880–887. [Online]. Available: https://doi.org/10.1145/1390156.1390267

[6] R. Freudenthaler, Schmidt-Thieme, "Bayesian factorization machines," in *Proceedings of the NIPS Workshop on Sparse Representation and Low-rank Approximation*, 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:18707748

[7] S. Rendle, "Factorization machines," in *ICDM 2010, The 10th IEEE International Conference on Data Mining, Sydney, Australia, 14-17 December 2010*, G. I. Webb, B. Liu, C. Zhang, D. Gunopulos, and X. Wu, Eds. IEEE Computer Society, 2010, pp. 995–1000. [Online]. Available: https://doi.org/10.1109/ICDM.2010.127

[8] ——, "Scaling factorization machines to relational data," *Proc. VLDB Endow.*, vol. 6, no. 5, pp. 337–348, 2013. [Online]. Available: http://www.vldb.org/pvldb/vol6/p337-rendle.pdf

[9] 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

[10] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua, "Neural collaborative filtering," in *Proceedings of the 26th International Conference on World Wide Web, WWW 2017, Perth, Australia, April 3-7, 2017*, R. Barrett, R. Cummings, E. Agichtein, and E. Gabrilovich, Eds. ACM, 2017, pp. 173–182. [Online]. Available: https://doi.org/10.1145/3038912.3052569

[11] S. Rendle, W. Krichene, L. Zhang, and J. R. Anderson, "Neural collaborative filtering vs. matrix factorization revisited," in *RecSys 2020: Fourteenth ACM Conference on Recommender Systems, Virtual Event, Brazil, September 22-26, 2020*, R. L. T. Santos, L. B. Marinho, E. M. Daly, L. Chen, K. Falk, N. Koenigstein, and E. S. de Moura, Eds. ACM, 2020, pp. 240–248. [Online]. Available: https://doi.org/10.1145/3383313.3412488

[12] S. Sedhain, A. K. Menon, S. Sanner, and L. Xie, "Autorec: Autoencoders meet collaborative filtering," in *Proceedings of the 24th International Conference on World Wide Web Companion, WWW 2015, Florence, Italy, May 18-22, 2015 - Companion Volume*, A. Gangemi, S. Leonardi, and A. Panconesi, Eds. ACM, 2015, pp. 111–112. [Online]. Available: https://doi.org/10.1145/2740908.2742726

[13] F. Strub, J. Mary, and P. Philippe, "Collaborative filtering with stacked denoising autoencoders and sparse inputs," in *NIPS workshop on machine learning for eCommerce*, 2015.

[14] O. Kuchaiev and B. Ginsburg, "Training deep autoencoders for collaborative filtering," *CoRR*, vol. abs/1708.01715, 2017. [Online]. Available: http://arxiv.org/abs/1708.01715

[15] X. Liu and Z. Wang, "CFDA: collaborative filtering with dual autoencoder for recommender system," in *International Joint Conference on Neural Networks, IJCNN 2022, Padua, Italy, July 18-23, 2022*. IEEE, 2022, pp. 1–7. [Online]. Available: https://doi.org/10.1109/IJCNN55064.2022.9892705

[16] 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

[17] 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