Enhancing Movie Recommendations with Hybrid Ensemble Techniques

Piero Neri, Robin Oester, Lukas Nüesch
Group: Winterhold, Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—With the use of collaborative filtering (CF), the behavioral patterns of user groups are analyzed to infer the interests of individuals. In this paper, we propose an ensemble-based approach that combines several different methods to deduce unobserved movie ratings for given users. Our contributions include optimizing well-established baseline models such as Alternating Least Squares (ALS) and creating a comprehensive CF experimental framework. Additionally, we introduce Kolmogorov-Arnold Networks (KANs) as a replacement for traditional Multi-Layer Perceptrons (MLPs) and explore advanced feature engineering techniques in Bayesian Factorization Machines. Our ensembling approach achieves a public score of 0.9649 in a Kaggle competition held as part of the ETH Zurich CIL 2024 course. Furthermore, we gain valuable insights into the effectiveness of various methods for collaborative filtering.**

I. Introduction

Recommender systems are software tools designed to suggest items users might find useful or interesting. They are widely used across various industries, with companies like Netflix heavily relying on them to enhance user experience and drive business growth. Netflix notably hosted the Netflix Prize [1], offering a million-dollar reward to the team that significantly improved the company's recommender system's performance. One prominent category of recommender systems is collaborative filtering, which makes recommendations based on users preferences on a set of items and other users' preferences [18].

Our project involves participating in a Kaggle competition focused on collaborative filtering methods. The dataset, reminiscent of the Netflix Prize dataset, contains $1,176,952$ ratings from $n = 10,000$ users across $m = 1,000$ movies, with ratings ranging from 1 to 5. It can hence be seen as a sparse matrix $A \in \mathbb{R}^{n \times m}$. Our task is to predict missing user ratings, and our model's performance is assessed using the root mean square error (RMSE) metric. The evaluation is based on a public test set, which constitutes 50% of the total test data, with the remaining 50% reserved for the final assessment.

We implement three classical baseline methods and several more advanced methods. The latter are enhanced by incorporating techniques from deep learning, K-Means clustering features, or even novel Kolmogorov-Arnold Networks. All approaches are used in a final ensembling step to generate our predictions for the Kaggle competition. Throughout the implementation process, we build an experimental infrastructure that can be used as a starting point for inventing further methods in a collaborative-filtering setting. The code is publicly accessible.[1]

II. Models and Methods

A. Baseline Models

1) Alternating Least Squares (ALS): ALS [9] aims to learn the matrices $U \in \mathbb{R}^{n \times k}$ and $V \in \mathbb{R}^{k \times m}$ in order to approximate the ratings matrix A as $A \approx UV$ inducing the rank-k constraint on A, by minimizing the following non-convex objective:

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

where $\lambda > 0$ and Π_Ω masks the unobserved entries in A. As parameter dependencies in the objective form a bipartite graph between the rows of U and the columns of V, the part of the objective that depends on the columns v_j of V can be isolated. If U is fixed, each v_j results as the solution to a least squares problem of that isolated objective:

$$v_j^* = \left(\sum_i \omega_{ij} u_i u_i^T + \lambda I\right)^{-1} \left(\sum_i \omega_{ij} a_{ij} u_i\right) \quad (2)$$

The same procedure applies for the rows u_i of U when V is fixed. ALS alternates between these steps until convergence. In our implementation, we initialize U and V using a standard uniform distribution.

2) Singular Value Projection (SVP): SVP [13], the second baseline method, is a projected gradient method that approximates the ratings matrix A by iteratively applying $[\cdot]_k$, a projection onto the set of rank k matrices using SVD. The update steps are as follows:

$$\begin{aligned} A_0 &= 0 \\ A_{t+1} &= [\, A_t + \eta \Pi_\Omega(A - A_t)\,]_k, \quad \eta > 0 \end{aligned} \quad (3)$$

3) Singular Value Thresholding (SVT): The goal of SVT [2], the third baseline method, is to minimize the following surrogate relaxed objective:

$$\tau\|X\|_* + \frac{1}{2}\|X\|_F^2 \quad \text{s.t.} \quad \Pi_\Omega(A - X) = 0 \quad (4)$$

where $\|.\|_*$ denotes the nuclear norm and is used as the convex envelope of the rank function on matrices. If this objective is minimized, we aim to obtain a good low-rank

[1] https://github.com/robin-oester/cil-project

approximation of the matrix A. The steps of SVT can be summarized the following:

$$\begin{aligned} X_t &= \text{shrink}_\tau(Y_{t-1}), \\ Y_t &= Y_{t-1} + \eta_t \Pi_\Omega(A - X_t), \quad \eta_t > 0 \end{aligned} \tag{5}$$

where $Y_0 = 0$, $\text{shrink}_\tau(Y) = U\text{diag}(\sigma_i - \tau)_+ V^T$ and $Y = U\text{diag}(\sigma_i)V^T$. X_t represents the approximated matrix after t iterations.

For all baseline methods, we perform a preprocessing step where we normalize the ratings matrix by item, although we also tested user-wise normalization. We found the hyperparameters using a grid-search approach. These values were further validated using 10-fold cross-validation.

B. SVD++

SVD++, as introduced in Koren [7], is a latent factor model that aims to transform users and items into the same latent factor space, enabling comparisons between these latent representations. Latent factor models based on SVD attempt to predict the user-item rating r_{ui} by incorporating the overall average rating μ, along with b_u and b_i, the observed deviations of user u and item i from μ. This prediction is further refined by the inner product between the item and user factor vectors, q_i and p_u, where $q_i, p_u \in \mathbb{R}^k$. SVD++ extends this model by introducing additional item factor vectors $y_i \in \mathbb{R}^k$ and by complementing p_u with $|N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j$, which captures the implicit feedback of user u on $N(u)$, the set of items user u has rated. Thus, SVD++ can be written as:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T (p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j) \tag{6}$$

where b_u, b_i and $\sum_{j \in N(u)} y_j$ are precomputed following methods from the literature [8][21]. To estimate q_i and p_u, SGD is applied to the regularized squared loss between r_{ui} and $\hat{r}_{ui}$. Our contribution consists of applying regularization and optimization techniques from deep learning in the context of SVD++. Specifically, we replace SGD with adaptive moment estimation (Adam) [6], perform backpropagation using a batch size of 4096 samples, and extend the loss function with a regularized mean squared error over the batch. These changes enhance convergence speed by providing more accurate gradient estimates, making it possible to stop training after 5 epochs. Additionally, we employ an exponential learning rate schedule and weight decay, which further stabilize training and prevent overfitting. The hyperparameters were carefully chosen by experimenting with various values on a validation set and measuring generalization error using 10-fold cross-validation, leading to the choice of $k = 180$ factors for the latent factor vectors.

C. Factorization Machines (FM)

Factorization Machines, introduced by Rendle [15], are a versatile class of supervised learning models. They are utilized in regression, classification, and ranking tasks. By generalizing linear regression and matrix factorization to incorporate higher-order feature interactions, similar to Support Vector Machines (SVM), they provide robust performance across various applications. Moreover, their ability to be optimized in linear time makes them particularly effective for handling sparse data. Formally, we can define the most common Factorization Machine of order 2 as:

$$\hat{y}(x) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d} \sum_{j=i+1}^{d} x_i x_j \langle v_i, v_j \rangle \tag{7}$$

where $x \in \mathbb{R}^d$ is the input vector, $w_0 \in \mathbb{R}$ the bias term, $w \in \mathbb{R}^d$ the weights, and $V \in \mathbb{R}^{d \times k} = [v_1, ..., v_k]$ the latent factors of the input features. $\langle \cdot, \cdot \rangle$ denotes the vector product.

As proposed by Freudenthaler [3], the model can be enhanced by incorporating structured Bayesian inference. Apart from better accuracy, Bayesian Factorization Machines (BFM) also allow for automatic hyperparameter tuning. They use Gaussian hyperpriors on the weights and the latent factors, with efficient Gibbs sampling for posterior inference.

Our implementation largely follows the tutorial of the Python library myFM [20] and the ideas presented by Rendle [16] as well as Rendle, Zhang, and Koren [17].

1) Base Model: Using myFM, we implement the base model of BFM. To encode the users and movies, we use one-hot encodings of the user and movie IDs. Since the feature matrices can become very large and exhibit many repeating patterns, Rendle [16] proposes the use of Relational Blocks to condense the feature matrix. This technique allows us to efficiently train the model even if more features are added.

2) Implicit Features: To improve upon the basic BFM, Rendle, Zhang, and Koren [17] proposed extending the model with implicit features. They borrow the idea from SVD++, where along with user embeddings, they also add a term that includes a combination of user-item interactions for a given user. Similarly, for BFMs, we add two multi-hot encoded feature vectors: one that includes all movies a user has rated and another that includes all users that have rated a given movie. To account for the different numbers of interactions, we normalize each feature vector with the factor $\frac{1}{\sqrt{\max(\#\text{interactions}, 1)}}$

3) Ordinal Regression: We also evaluate a variant of the BFM model that employs an ordinal probit model [12] for rating prediction. The primary distinction from the base model is that it treats ratings as ordinal rather than continuous data, which is beneficial since the difference between ratings may not be consistent.

4) K-Means and Statistical Features: Expanding on the idea of implicit features, we integrate K-Means cluster features. These are calculated by clustering the implicit features of user-movie and movie-user interactions. We then add the cluster ID as a one-hot encoded feature to the model.

We tested cluster sizes ranging from 2^1 to 2^8 using 10-fold cross-validation and found a cluster size of 4 to be best. We also tried the addition of statistical one-hot encoded features. These features include the mean and standard deviation of the ratings for each user and movie, where we use a binning technique to represent value intervals. We experimented with bin sizes ranging from 2^1 to 2^8 but could not find any improvements over the base model.

Due to the automatic hyperparameter tuning, we only need to specify the rank and the number of iterations. Since a higher rank and more iterations generally lead to better results during testing, we settled on a rank of $k = 15$ and 1000 iterations for all experiments as a trade-off between runtime and accuracy.

D. Autoencoder

Autoencoders can be used to learn low dimensional representation of high dimensional data. It has been shown by Kuchaiev and Ginsburg [10] that they are also applicable in a CF setting. Inspired by their approach, we have decided on a symmetrically structured network that tries to reconstruct the ratings of a given user. Hence, the encoder maps the input $x \in \mathbb{R}^m$ to the latent $z \in \mathbb{R}^k$, where we measure the best performance for $k = 32$. It consists of an MLP with a single hidden layer of dimension 256, uses LeakyReLU as activation and makes heavy use of dropout. Symmetrically, the decoder maps back to $\hat{x} \in \mathbb{R}^m$. The use of LeakyReLU, which has non-zero activation on negative values and dropout improved accuracy, which is on-par with state-of-the-art research [10]. The loss function is MSE restricted on the observed ratings and we impute the missing values with 0 to not induce any biased information.

E. Neural Collaborative Filtering

Neural collaborative filtering (NCF) consists of all approaches that use neural networks to solve collaborative filtering problems. We implement three models that are based on He et al. [5]. Each model comes with its own approach to NCF allowing us to gain insights into the effectiveness of different neural architectures and training strategies in improving recommendation accuracy. Particularly, our last model uses the novel Kolmogorov-Arnold Networks as its main building block. These networks were introduced by Liu et al. [11] and are based on the Kolmogorov-Arnold Representation Theorem [19]. It makes them universal function approximators sharing the same representational power as MLPs. As opposed to MLPs, however, KANs learn the activation function itself, not the layer weights. Each layer consists of a number of trainable activation functions $\phi(x)$ where the input is given as the sum of the previous layer outputs. Formally, we have:

$$\phi(x) = w_b b(x) + w_s \text{spline}(x)$$

where $b(x) = \text{silu}(x) = \frac{x}{1+e^{-x}}$. We use a linear combination of order-3 B-splines to optimize the activation function: $\text{spline}(x) = \sum_j c_j B_j(x)$. All c_j's are trainable together with w_b and w_s. In our work, we specifically want to assess the performance of KAN-based models in the NCF setting. We list our models with their most important features:

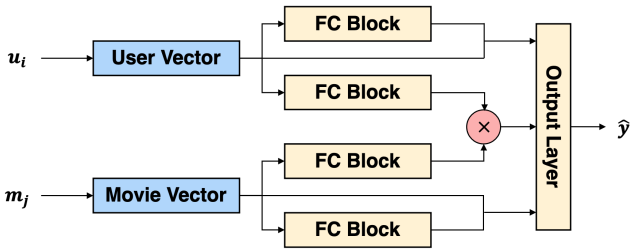

Figure 1: *Architecture of our NCF model. The user and movie are embedded into 128-dimensional vectors, before features are extracted using fully-connected blocks. In the end all features are concatenated and the final prediction is made using a designated output layer.*

1) NCFCombined: This model consists of the two sub-models GMF and MLP. The General Matrix Factorization model performs an element-wise product between user and movie embeddings. These features are linearly combined to predict the ratings. The MLP part learns separate embeddings that are then concatenated and run through three MLP layers for the rating prediction. As proposed, we train both sub-models independently using the Adam optimizer, before initializing the NCFCombined model with these pretrained weights. This large model performs a weighted average (with parameter $\alpha \in [0, 1]$) of the sub-model predictions using SGD as optimizing procedure. We empirically decided on $\alpha = 0.5$, i.e., both sub-models contribute equally to the final prediction.

2) MLP-based NCF: This model is inspired by He et al. [5], however, it is trained as a whole. The architecture can be found in Figure 1. Each fully-connected block is an MLP with a single hidden layer and uses ReLU as activation function. Our experiments show that using residual connections as introduced in He et al. [4] on the FC blocks are beneficial. The width of the hidden layer is 384 and the blocks capture features for the user, movie as well as for the interaction between them. The output layer consists of a two-layer hierarchical MLP. All of our MLP layers make use of dropout as the model is prone to overfit.

3) KAN-based NCF: Contrary to model (ii), the KAN-based model uses fully-connected KANs instead of MLPs for the FC blocks. Since the elements of KANs are more expressive, the width of the hidden layer can be reduced to 32. Our measurements show that using a hierarchical MLP is superior to using KANs for combining the intermediate features. All KANs use a grid of size 5 where the grid ranges from -1 to 1. The splines are of order 3.

Since finding the hyperparameters for our models is expensive, we followed a greedy approach. By splitting our training dataset into training and validation dataset, we trained our models until a minimal validation loss is reached. Then, we iterate by selecting the next promising set of hyperparameters. Specifically, we tuned the learning rates and weight decay of the optimizer (we mostly use Adam) as well as model-specific parameters such as embedding dimension, layer width, etc. Additionally, we implement several methods for normalizing the ratings. They can either be normalized by user, by movie or by the statistics of all ratings.

F. Ensembling

Ensembling aims at reducing the generalization error through combining individual outputs of different models. Two common approaches, namely blending and stacking, involve a meta-learner, which is a model specifically trained for finding a suitable function to combine different models. Our implementation of blending involves the separation of the whole dataset into training and validation set. Then, we train a set of models on the training set before predicting the validation set and the test set. The predictions from the validation set are used for training the meta-model, which in turn combines the test set results for the final prediction. Stacking is implemented through applying k-fold cross-validation on our models. The predictions on the hold-out validation sets are gathered and used to train the meta-model. All models are then trained on the entire dataset to make the predictions on the test set, which are used by the meta-model for the final predictions. For blending, we used a $90\%/10\%$ training/validation split. For stacking, we used $k = 10$ folds. As meta-models, we compared basic LinearRegression and GradientBoostingRegressor [14].

III. Experiments & Results

To compare our methods, we performed 10-fold cross-validation and reported the average RMSE on the hold-out set as the local score. We obtained the public RMSE by predicting the competition test set. The results are summarized in Table I. Regarding the NCF approaches, we observe that combining the smaller sub-models GMF and MLP into NCFCombined results in a higher score than using them separately. This aligns with findings in the literature [5]. Interestingly, the performance difference between MLP-based NCF and KAN-based NCF is negligible in terms of accuracy. However, the model architecture and training process is optimized for MLP-based NCF, which puts the KAN-based approach at a disadvantage. Additionally, KANs require approximately five times more computation, significantly increasing runtime. The BFM models demonstrate the best overall performance among non-ensemble models. Notably, supporting the findings of Rendle, Zhang, and Koren [17], incorporating implicit features leads to the largest improvement. Further enhancements using ordinal regression are also explored, showing promising results. Although adding K-Means features yields only marginally better results, it produces our best-performing non-ensemble model, highlighting the model's reliance on feature engineering. Applying blending and stacking to various model combinations, we find that increasing the number of models consistently reduces RMSE, regardless of the performance of individual methods. Unsurprisingly, stacking outperforms blending due to the availability of more data for training the meta-regressor. However, the performance difference is minimal, making blending a preferable choice in time-constrained settings. Additionally, stacking requires careful attention to prevent data leakage, which can otherwise undermine its effectiveness.

Method	Local CV Score	Public Score
ALS	0.9899	0.9876
SVP	0.9885	0.9860
SVT	0.9870	0.9844
SVD++	0.9784	0.9759
Autoencoder	0.9819	0.9799
NCFCombined	0.9826	0.9820
- GMF	0.9848	0.9906
- MLP	0.9878	0.9844
MLP-based NCF	0.9843	0.9806
KAN-based NCF	0.9834	0.9807
BFM (Base)	0.9776	0.9750
BFM (Implicit)	0.9701	0.9676
BFM (Implicit + op)	0.9682	0.9661
BFM (Implicit + op + K-Means)	0.9680	0.9660
Blending (Linear Regression)	-	0.9651
Stacking (Linear Regression)	-	**0.9649**

Table I: *Results of our methods: local score is the average RMSE from 10-fold cross-validation; public score is the RMSE on the competition test set.*

IV. Conclusion & Future Work

In this work, we optimized well-known baseline models such as ALS, SVP, and SVT and introduced novel approaches to SVD++, NCF, and BFM. Utilizing our open-source framework, users can easily integrate new methods and conduct collaborative filtering experiments. Our results demonstrate that applying deep learning techniques to SVD++ facilitates faster and more stable convergence, enabling more efficient hyperparameter validation and exploration. Additionally, we found that Kolmogorov-Arnold Networks serve as a viable alternative to MLP-based NCF models, offering comparable accuracy despite longer training times. Furthermore, incorporating advanced features like K-Means clustering improves the accuracy of Bayesian Factorization Machines. Our ensemble-based approach, which integrates multiple models, achieves substantial performance gains over standard baseline methods.

References

[1] James Bennett and Stan Lanning. "The Netflix Prize". In: 2007. URL: https://api.semanticscholar.org/CorpusID:9528522.

[2] Jian-Feng Cai, Emmanuel J. Candes, and Zuowei Shen. *A Singular Value Thresholding Algorithm for Matrix Completion*. 2008. arXiv: 0810.3286 [math.OC]. URL: https://arxiv.org/abs/0810.3286.

[3] Christoph Freudenthaler. "Bayesian Factorization Machines". In: 2011. URL: https://api.semanticscholar.org/CorpusID:18707748.

[4] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: https://arxiv.org/abs/1512.03385.

[5] Xiangnan He et al. *Neural Collaborative Filtering*. 2017. arXiv: 1708.05031 [cs.IR]. URL: https://arxiv.org/abs/1708.05031.

[6] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[7] Yehuda Koren. "Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model". In: KDD'08. AT&T Labs – Research 180 Park Ave, Florham Park, NJ 07932, Aug. 2008.

[8] Yehuda Koren. "The BellKor Solution to the Netflix Grand Prize". In: 2009. URL: https://api.semanticscholar.org/CorpusID:6114578.

[9] Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix Factorization Techniques for Recommender Systems". In: *Computer* 42.8 (2009), pp. 30–37. DOI: 10.1109/MC.2009.263.

[10] Oleksii Kuchaiev and Boris Ginsburg. *Training Deep AutoEncoders for Collaborative Filtering*. 2017. arXiv: 1708.01715 [stat.ML]. URL: https://arxiv.org/abs/1708.01715.

[11] Ziming Liu et al. "KAN: Kolmogorov-Arnold Networks". In: *arXiv preprint arXiv:2404.19756* (2024).

[12] Peter McCullagh. "Regression Models for Ordinal Data". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 42.2 (Dec. 2018), pp. 109–127. ISSN: 0035-9246. DOI: 10.1111/j.2517-6161.1980.tb01109.x. eprint: https://academic.oup.com/jrsssb/article-pdf/42/2/109/49097431/jrsssb_42_2_109.pdf. URL: https://doi.org/10.1111/j.2517-6161.1980.tb01109.x.

[13] Raghu Meka, Prateek Jain, and Inderjit S. Dhillon. *Guaranteed Rank Minimization via Singular Value Projection*. 2009. arXiv: 0909.5457 [cs.LG]. URL: https://arxiv.org/abs/0909.5457.

[14] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[15] Steffen Rendle. "Factorization Machines". In: *2010 IEEE International Conference on Data Mining*. 2010, pp. 995–1000. DOI: 10.1109/ICDM.2010.127.

[16] Steffen Rendle. "Scaling factorization machines to relational data". In: *Proc. VLDB Endow.* 6.5 (Mar. 2013), pp. 337–348. ISSN: 2150-8097. DOI: 10.14778/2535573.2488340. URL: https://doi.org/10.14778/2535573.2488340.

[17] Steffen Rendle, Li Zhang, and Yehuda Koren. *On the Difficulty of Evaluating Baselines: A Study on Recommender Systems*. 2019. arXiv: 1905.01395 [cs.IR]. URL: https://arxiv.org/abs/1905.01395.

[18] Markus Schedl et al. "Music Recommendation Systems: Techniques, Use Cases, and Challenges". In: *Recommender Systems Handbook (3rd edition)*. Ed. by Francesco Ricci, Lior Rokach, and Bracha Shapira. New York, NY: Springer, 2022, pp. 927–971. ISBN: 978-1-0716-2197-4. DOI: 10.1007/978-1-0716-2197-4_24. URL: https://doi.org/10.1007/978-1-0716-2197-4_24.

[19] Johannes Schmidt-Hieber. *The Kolmogorov-Arnold representation theorem revisited*. 2021. arXiv: 2007.15884 [cs.LG]. URL: https://arxiv.org/abs/2007.15884.

[20] Takahiro Shinya. *myFM: A library for Bayesian Factorization Machines*. Commit b9ba70ea38d9370d3ad50a9d25b2ff825eaa30ef. 2024. URL: https://github.com/tohtsky/myFM/tree/b9ba70ea38d9370d3ad50a9d25b2ff825eaa30ef.

[21] Zhengzheng Xian et al. "New Collaborative Filtering Algorithms Based on SVD + + and Differential Privacy". In: 2017. URL: https://api.semanticscholar.org/CorpusID:54032117.