The Impact of Feature Quantity on Recommendation Algorithm Performance

Lukas Wegmeth[1]

[1]Intelligent Systems Group, University of Siegen, Germany

Abstract

Recent model-based Recommender Systems (RecSys) algorithms emphasize using features, also called side information, in their design, similar to algorithms in Machine Learning (ML). In contrast, some of the most popular and traditional algorithms for RecSys solely focus on a given user-item-rating relation without including side information. An essential category of these is matrix factorization-based algorithms, e.g., Singular Value Decomposition and Alternating Least Squares, which are known to have high performance on RecSys datasets. This paper aims to provide a performance comparison and assessment of RecSys and ML algorithms when side information is included. We chose the Movielens-100K dataset for a case study since it is a standard for comparing RecSys algorithms. We compared six different feature sets with varying quantities of features, which were generated from the baseline data and evaluated on 19 RecSys algorithms, baseline ML algorithms, Automated Machine Learning (AutoML) pipelines, and state-of-the-art RecSys algorithms that incorporate side information. The results show that additional features benefit all algorithms we evaluated. However, the correlation between feature quantity and performance is not monotonic for AutoML and RecSys. In these categories, an analysis of feature importance revealed that the quality of features matters more than quantity. Throughout our experiments, the average performance on the feature set with the lowest number of features is ∼6% worse compared to that with the highest in terms of the Root Mean Squared Error. An interesting observation is that AutoML outperforms matrix factorization-based RecSys algorithms when additional features are used. Almost all algorithms that can include side information perform better when using the highest quantity of features. In the other cases, the performance difference is negligible (<1%). The results show a clear positive trend for the effect of feature quantity and the critical effects of feature quality on the evaluated algorithms.

Keywords

Feature Engineering, Recommender Systems, Automated Machine Learning

1. Introduction

Matrix factorization-based Recommender System (RecSys) algorithms are specialized for predicting missing entries, e.g., ratings, in sparsely filled user-item matrices. Many often-used benchmark datasets exist [1, 2], representing such a RecSys task. Some of these datasets include side information, also called features, which are not used by the RecSys algorithms mentioned above. Instead, the data is directly reduced to a sparse user-item matrix, ignoring side information. In contrast, Machine Learning (ML) algorithms are broad in their applications and usually profit from the availability of additional, meaningful features [3, 4, 5]. As an extension, Automated Machine Learning (AutoML) techniques further increase ML performance through automated algorithm selection and hyperparameter optimization. Furthermore, the same tasks RecSys algorithms intend to solve can generally be solved by (Auto)ML algorithms. Recent advances in RecSys have led to more sophisticated model-based algorithms, with the likes of Factorization Machines [6] and especially the latest Deep Neural Networks (DNN), that can incorporate side information and are more similar to their ML relatives [7, 8, 9, 10, 11]. However, the performance gap between (Auto)ML and RecSys has not been explicitly researched.

Feature engineering is a broad topic that is well documented and researched due to its positive influences on ML [12, 13, 5, 14, 15]. It summarizes many feature-processing techniques that mainly intend to increase prediction performance. Today, such feature engineering techniques are standard for many ML pipelines. Among those techniques is the curation of features by selection and extraction. Additional

AICS'24: 32nd Irish Conference on Artificial Intelligence and Cognitive Science, December 09–10, 2024, Dublin, Ireland
lukas.wegmeth@uni-siegen.de (L. Wegmeth)
0000-0001-8848-9434 (L. Wegmeth)

real-world features and features extracted from existing data often benefit a model's performance. In this context, comparing the impact of feature quantity on the performance of RecSys and ML algorithms provides a meaningful indicator for the effects of feature engineering. However, we could not find previous comparative studies on the effect of feature quantity on RecSys algorithms.

Due to the aforementioned positive effects of feature engineering techniques in ML algorithms, we hypothesized that the same effects could also be shown for RecSys algorithms. The following two research questions arise since a performance comparison between (Auto)ML and RecSys regarding feature quantity is absent in the literature. In a RecSys problem setting, how does feature quantity impact the performance:

- *of ML, AutoML, and RecSys algorithms in general?*
- *of RecSys algorithms compared to (Auto)ML algorithms?*

We explore these questions through a case study on the Movielens-100K [16] dataset. The code that produced the results reported in this paper is available on our GitHub repository[1].

2. Method

We evaluated 19 algorithms from nine libraries (Table 1) on six feature sets generated from the Movielens-100K [16] dataset (Table 2).

Table 1
An overview of the evaluated algorithms and their category. We evaluated the algorithms as implemented in their respective library. Importantly, it is shown whether the algorithm can incorporate side information and how their hyperparameters were tuned.

Category	Library	Algorithm	Uses side information	Hyperparameter tuning
Baseline	Scikit-Learn [17]	Constant Predictor	-	-
Machine Learning		Linear Regressor	X	-
		K Nearest Neighbors	X	SMAC3 [18] (200 runs)
		Random Forest Regressor	X	Random search: 20 iterations per fold
		Histogram Gradient Boosting Regressor	X	Random search: 20 iterations per fold
	XGBoost [19]	Extreme Gradient Boosting Regressor	X	-
AutoML	Auto-Sklearn [20]	Best algorithm varies by fold	X	One hour search per fold
	H2O AutoML [21]	Best algorithm varies by fold	X	One hour search per fold
	FLAML [22]	Best algorithm varies by fold	X	One hour search per fold
RecSys Matrix Factorization	Surprise [23]	SVD	-	-
		SVDpp	-	-
		KNNBaseline	-	-
	Lenskit [24]	User-User kNN collaborative filtering	-	-
		Item-Item kNN collaborative filtering	-	-
		Biased Alternating Least Squares	-	-
	LibRecommender [25]	SVDpp	-	Manually
RecSys Models		Wide & Deep	X	SMAC3 [18] (300 runs)
		Deep Interest Network	X	SMAC3 [18] (100 runs)
	MyFM [26]	Bayesian Factorization Machine	X	-

Movielens-100K [16] is one of the regularly used explicit feedback algorithm performance evaluation datasets in the RecSys community [27, 28, 29, 30]. The full dataset consists of a table with user IDs and item IDs and their observed ratings, where each rating has a timestamp. Additionally, each user ID and item ID contains a set of features specific to them. The user features are their age, gender, occupation, and (North American) ZIP code. The item features are their movie genre, title, release date, and IMDb URL. We solve the prediction of ratings as a regression task and measure and compare the performance of a given algorithm through the Root Mean Squared Error.

[1] https://code.isg.beel.org/recsys-feature-quantity

To analyze the impact of the number of features on algorithms, we cut and/or enriched the features of the original dataset and finally grouped them, resulting in six separate feature sets (Table 2). The default feature set contains most of the basic features of the original dataset. The idea is to use as many original features as possible that require little to no further processing. For this reason, the item's title, IMDb URL, and the user's ZIP codes were removed.

From the observed user-item-ratings relation, many statistical features can be engineered. They can also be calculated separately in terms of the users and items. We calculated the following nine statistical features regarding the users and items: mean, median, mode, minimum, maximum, standard deviation, kurtosis, and skew. This provides a total of 18 additional features. Notably, these features were only calculated on the training set after splitting the data into a separate training and test set to avoid leaking information about the training set into the test set. These engineered features should provide helpful additional information to the algorithms at hand.

Finally, additional real-world data points can be added to the list of available features. For this, we chose the median and mean household income and population of a period as close as possible to the release date of the original dataset. A set of these data points grouped by ZIP codes from the US from 2006 to 2010 [31] is the earliest publicly available data directly related to the user features.

From various combinations of the feature sets mentioned above, we created six combinations to perform the experiments on. An overview of the sets and their names that we refer to from here on is listed in Table 2.

Table 2
An overview of the evaluated feature sets that we generated from the base Movielens-100K [16] dataset either by cutting or enriching features. The columns denote the contained features in the named feature sets shown in the rows and provide the total number of features.

features → set name ↓	user ID item ID	user stat. feat. item stat. feat.	rating timestamp user occupation user age user gender movie genre movie release date	user ZIP code ZIP code income ZIP code population	number of features categorical features count as one
stripped-no-stats	X	-	-	-	2
stripped-with-stats	X	X	-	-	20
basic-no-stats	X	-	X	-	8
basic-with-stats	X	X	X	-	26
feature-expansion-no-stats	X	-	X	X	11
feature-expansion-with-stats	X	X	X	X	29

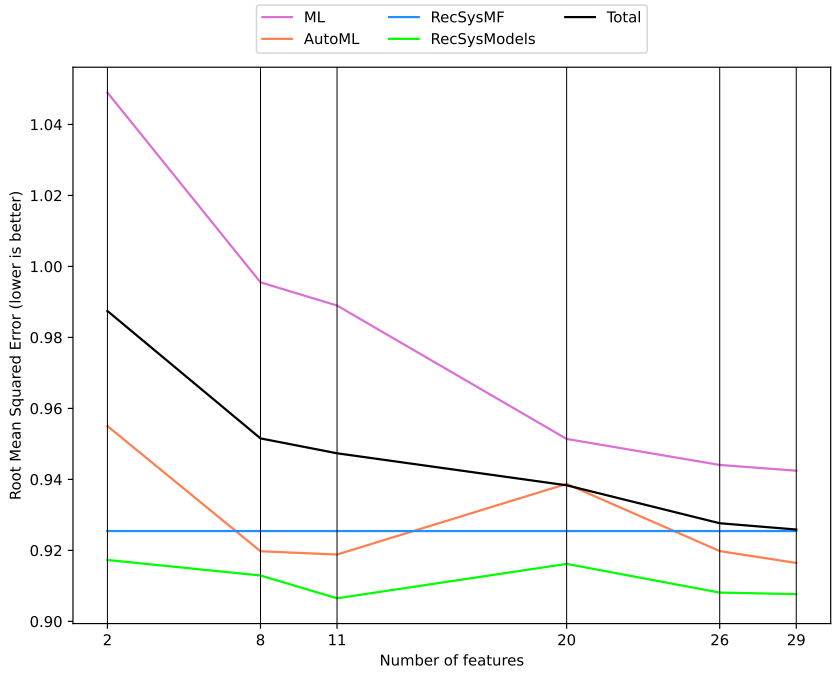
We applied additional processing steps to some of the features to make them suitable for the evaluation, sometimes depending on the dataset or automatically as an algorithm requires. Generally, we tried to stay as close to the original features as possible, making changes only where sensible or necessary. As a result, we did not treat the user ID and item ID as categorical features where applicable. The movie genre is provided as a categorical feature by default, which we did not change. However, the user's occupation is not provided as a categorical feature, so we transformed it into one. Movie release dates are provided in a date format, and we converted them to a signed UNIX timestamp representation. The user's age and zip code are special cases. As provided in the original set, we treated the age as an integer for the 'basic' sets. We removed the ZIP code because it contains some non-numerical entries and because we did not intend to filter ratings in these sets. For the 'feature-expansion' sets, we divided the age by 18 to create five age categories and then treated the age as a categorical feature. We had to keep the ZIP code to add the mean and median household income and population features. Finally, we removed entries with ZIP codes that were not contained in the additional feature sets, which incurs a loss in observed ratings of 7.05%. The remaining ZIP codes range from '00000' to '99999'. To use them as a feature, we selected only the first digit of each ZIP code and then transformed it into a categorical feature. We chose to apply this processing step because the first digit in the ZIP code has a geographical meaning and, therefore, serves as an estimation for the residential area of each user.

To have an equal ground for algorithm comparisons, we applied some constraints. We performed all experiments on implementations in publicly available libraries to increase accessibility and reproducibility. The Movielens-100K [16] dataset contains explicit ratings as integers ranging from one to five. Therefore, we only chose algorithms that can take explicit ratings as input and predict ratings in that same format. We evaluated the algorithms using five-fold cross-validation. Since one of the research questions is about a comparison of the performance of algorithms against each other, we performed hyperparameter tuning on algorithms that do not default to a tuned parameter setup for the Movielens-100K [16] dataset. Depending on the algorithm, we manually tuned the hyperparameters with a random search or using SMAC3 [18], an all-purpose hyperparameter optimization tool. We set the time budget of AutoML tools to one hour for each fold. Table 1 lists all libraries and algorithms and their categories, whether they use side information, and how their hyperparameters were tuned.

3. Results

Figure 1 aggregates the results gathered during the experiments and clearly shows that a higher feature quantity generally results in a lower Root Mean Square Error (RMSE). In particular, when comparing evaluations of the highest quantity of features with the lowest, the RMSE is *10% lower for ML, 4% lower for AutoML, 1% lower for model-based RecSys, and 6% in total.*

Figure 1: This plot shows the RMSE performance of the algorithm categories as denoted in Table 1 on the feature sets denoted in Table 2. Each line plots the average RMSE of an algorithm category evaluated on each feature set represented by their number of features. The vertical lines and labels on the x-axis denote the collected data points. The first three data points do not include statistical features, while the final three do. The plot shows that more features result in higher performance in most cases.



The evaluation shows that AutoML outperforms the traditionally strong matrix factorization-based RecSys contenders in most of the evaluated feature sets. When only the 'basic' feature sets are included, the RMSE is 1% lower. Notably, however, there is an increase in RMSE between the feature sets with eleven features which is 'feature-expansion-no-stats' and 20 features which is 'stripped-with-stats'. In these special cases, feature quantity alone can not significantly improve the algorithm's performance. A likely reason is the feature importance seen in Figure 2, which shows that the feature set with the higher feature quantity is missing essential features like the rating timestamp or item genre.

Figure 2: The ordered Gini feature importance in percent. We evaluated these with a Random Forest Regressor that we fitted on the training data of the largest feature set. The chart shows the impact of each feature on the trained model. These important values are not necessarily true for the evaluated algorithms but provide a good estimation nonetheless. The feature names denote if they are an item or user feature. Statistical features are prefixed with 'i' and 'u' to denote this.

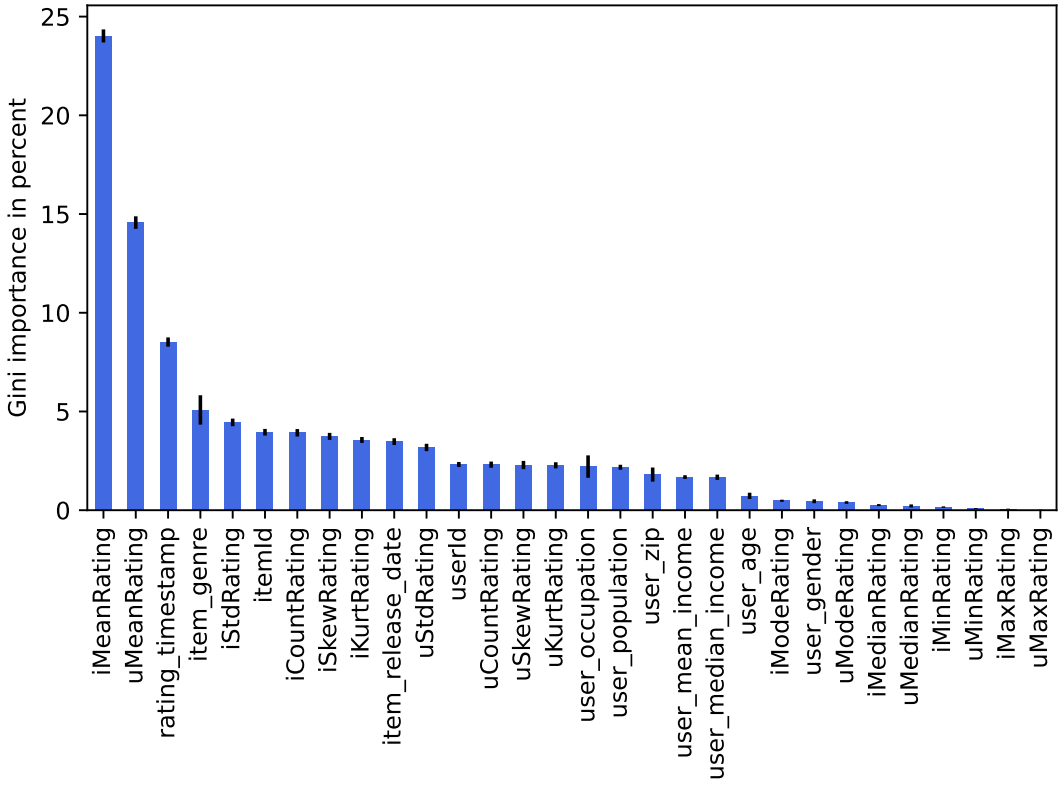
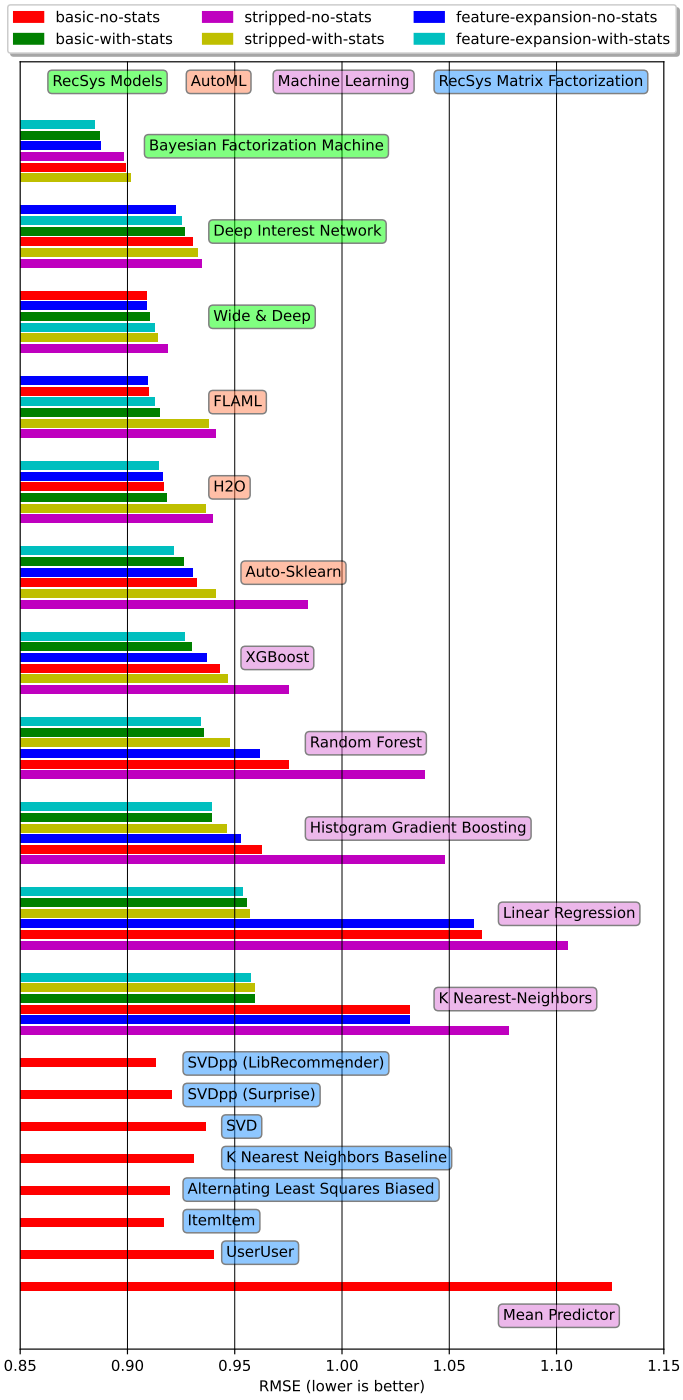


Figure 3 provides a detailed overview of all experiments. It shows that the performance difference and ranking in terms of the feature sets for RecSys and AutoML algorithms enormously varies by the algorithm, which is in contrast to the ML algorithms, where the ranking is mostly the same. Notably, the cross-validation procedure averages the results of multiple evaluations on an algorithm, and the figures show these aggregations. However, the evaluations of an algorithm had only marginal performance differences (<2%). Therefore, the reported averaged results shown here are relatively stable. Furthermore, we observe that the performance of the different feature sets is divided into two groups for AutoML. One of the groups contains the 'stripped' feature sets for which the search could not find a proper result even when provided with statistical features. In the other group, all other feature sets are tightly packed together, with a slight lead for the most extensive feature set, indicating the preference for a good combination and a higher quantity of real-world and statistical features.

Regarding the model-based RecSys algorithms, the DNN approaches that are Wide & Deep and Deep Interest Network do not show a clear trend toward more favorable features. However, the Bayesian Factorization Machine is one of the most exciting results. Its performance increases clearly with feature quantity. Though a RecSys algorithm by nature, it could also be used to solve more general ML problems by design. It exhibits a mix of the behavior of ML and AutoML algorithms in its results, which can be seen in Figure 3. The most significant difference is that its performance far exceeds that of any other algorithm compared to any feature set, making it a prime example of the capability of additional features for ML and RecSys.

As expected, ML algorithms consistently demonstrated improved performance with an increased number of features. This shows that the provided features are of good enough quality and distribution. The feature sets have an almost equal ranking order across all ML algorithms.

Figure 3: This figure shows the detailed evaluation results that lead to the main conclusions presented in this paper. It shows the RMSE of the algorithms listed in Table 1 on the feature sets listed in Table 2. The bar chart is grouped by algorithm, and each group's bars are ordered by RMSE ascending. A lower RMSE is better. Algorithms that can not use additional features perform the same on all feature sets. For these only, the 'basic' feature set is plotted. These results are aggregated over algorithm categories in Figure 1.



Overall, the results of our study indicate a clear preference for using more features. The important caveat to this is the feature type and quality. As reported, the relation between feature quantity and algorithm performance is not monotonic for AutoML and RecSys due to feature importance. The evaluated model-based RecSys algorithms performed best in this study, but our evaluation shows how even this gap can be closed by introducing additional features.

4. Discussion

We conclude that including statistical features is the most straightforward way to increase any algorithm's performance. Figure 2 shows that the mean rating per user and item is incredibly effective. Additionally, the feature importance analysis shows that some statistical features are significant while others are comparatively insignificant. This indicates the advantages of feature selection techniques that were out of this research's scope.

We obtained only simple additional features in this work. However, they positively impacted the performance of most of the tested algorithms. The results may be significantly better if more care and time are given to feature engineering. In addition to introducing new features, there are numerous considerations regarding refining existing features, including those introduced in this work. One such is that user IDs and item IDs, for example, are technically categorical features and should be treated as such, which was not always the case within the experiments due to constraints in the algorithms. There are also different ways to represent such categorical features, which should be explored in this context. For example, we consciously decided to represent age as a categorical feature for one of the feature sets. These decisions have a potentially enormous impact on the performance of any algorithm and should be dealt with carefully.

The biggest challenge for implementing the discoveries in this paper is likely to find suitable new features. Finding good data that supplies an existing dataset is a challenging task. However, collecting such feature data from the start should be reasonably easy when recording a new dataset. Given our results, we encourage future dataset collection tasks to include as many features as possible. Our work has shown that, for our scenario, if the goal is prediction performance, it is highly likely that a good selection of features combined with dedicated tuning will lead to better results. For future work on algorithm evaluations, we propose that pipelines include feature engineering in terms of feature quantity because it may significantly improve results.

References

[1] Z. Zaier, R. Godin, L. Faucher, Evaluating recommender systems, in: 2008 International Conference on Automated Solutions for Cross Media Content and Multi-Channel Distribution, 2008, pp. 211–217. doi:10.1109/AXMEDIS.2008.21.

[2] J. L. Herlocker, J. A. Konstan, L. G. Terveen, J. T. Riedl, Evaluating collaborative filtering recommender systems, ACM Trans. Inf. Syst. 22 (2004) 5–53. URL: https://doi.org/10.1145/963770.963772. doi:10.1145/963770.963772.

[3] V. Sugumaran, K. Ramachandran, Effect of number of features on classification of roller bearing faults using svm and psvm, Expert Systems with Applications 38 (2011) 4088–4096. URL: https://www.sciencedirect.com/science/article/pii/S0957417410010298. doi:https://doi.org/10.1016/j.eswa.2010.09.072.

[4] A. L. Blum, P. Langley, Selection of relevant features and examples in machine learning, Artificial Intelligence 97 (1997) 245–271. URL: https://www.sciencedirect.com/science/article/pii/S0004370297000635. doi:https://doi.org/10.1016/S0004-3702(97)00063-5, relevance.

[5] S. Khalid, T. Khalil, S. Nasreen, A survey of feature selection and feature extraction techniques in machine learning, in: 2014 science and information conference, IEEE, 2014, pp. 372–378.

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

[7] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, et al., Wide & deep learning for recommender systems, in: Proceedings of the 1st workshop on deep learning for recommender systems, 2016, pp. 7–10.

[8] G. Zhou, X. Zhu, C. Song, Y. Fan, H. Zhu, X. Ma, Y. Yan, J. Jin, H. Li, K. Gai, Deep interest network for click-through rate prediction, in: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining, 2018, pp. 1059–1068.

[9] H.-T. Cheng, L. Koc, J. Harmsen, T. Shaked, T. Chandra, H. Aradhye, G. Anderson, G. Corrado, W. Chai, M. Ispir, R. Anil, Z. Haque, L. Hong, V. Jain, X. Liu, H. Shah, Wide and deep learning for recommender systems, in: Proceedings of the 1st Workshop on Deep Learning for Recommender Systems, DLRS 2016, Association for Computing Machinery, New York, NY, USA, 2016, p. 7–10. URL: https://doi.org/10.1145/2988450.2988454. doi:10.1145/2988450.2988454.

[10] M. Gridach, Hybrid deep neural networks for recommender systems, Neurocomputing 413 (2020) 23–30. URL: https://www.sciencedirect.com/science/article/pii/S0925231220309966. doi:https://doi.org/10.1016/j.neucom.2020.06.025.

[11] P. Covington, J. Adams, E. Sargin, Deep neural networks for youtube recommendations, in: Proceedings of the 10th ACM Conference on Recommender Systems, RecSys '16, Association for Computing Machinery, New York, NY, USA, 2016, p. 191–198. URL: https://doi.org/10.1145/2959100.2959190. doi:10.1145/2959100.2959190.

[12] A. Zheng, A. Casari, Feature engineering for machine learning: principles and techniques for data scientists, " O'Reilly Media, Inc.", 2018.

[13] G. Dong, H. Liu, Feature engineering for machine learning and data analytics, CRC Press, 2018.

[14] G. Chandrashekar, F. Sahin, A survey on feature selection methods, Computers & Electrical Engineering 40 (2014) 16–28.

[15] C. Seger, An investigation of categorical variable encoding techniques in machine learning: binary versus one-hot and feature hashing, 2018.

[16] F. M. Harper, J. A. Konstan, The movielens datasets: History and context, ACM Trans. Interact. Intell. Syst. 5 (2015). URL: https://doi.org/10.1145/2827872. doi:10.1145/2827872.

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

[18] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, F. Hutter, Smac3: A versatile bayesian optimization package for hyperparameter optimization, 2021. arXiv:2109.09831.

[19] T. Chen, C. Guestrin, XGBoost: A scalable tree boosting system, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, ACM, New York, NY, USA, 2016, pp. 785–794. URL: http://doi.acm.org/10.1145/2939672.2939785. doi:10.1145/2939672.2939785.

[20] M. Feurer, A. Klein, J. Eggensperger, Katharina Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, in: Advances in Neural Information Processing Systems 28 (2015), 2015, pp. 2962–2970.

[21] E. LeDell, S. Poirier, H2O AutoML: Scalable automatic machine learning, 7th ICML Workshop on Automated Machine Learning (AutoML) (2020). URL: https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_61.pdf.

[22] C. Wang, Q. Wu, FLO: fast and lightweight hyperparameter optimization for automl, CoRR abs/1911.04706 (2019). URL: http://arxiv.org/abs/1911.04706. arXiv:1911.04706.

[23] N. Hug, Surprise: A python library for recommender systems, Journal of Open Source Software 5 (2020) 2174. URL: https://doi.org/10.21105/joss.02174. doi:10.21105/joss.02174.

[24] M. D. Ekstrand, Lenskit for python: Next-generation software for recommender systems experiments, in: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, CIKM '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 2999–3006. URL: https://doi.org/10.1145/3340531.3412778. doi:10.1145/3340531.3412778.

[25] massquantity, Librecommender, 2022. URL: https://github.com/massquantity/LibRecommender.

[26] T. Ohtsuki, myfm, 2022. URL: https://github.com/tohtsky/myFM.

[27] S. Forouzandeh, K. Berahmand, M. Rostami, Presentation of a recommender system with ensemble learning and graph embedding: a case on movielens, Multimedia Tools and Applications 80 (2021) 7805–7832.

[28] U. Kuzelewska, Clustering algorithms in hybrid recommender system on movielens data, Studies

in logic, grammar and rhetoric 37 (2014) 125–139.
[29] Z. Yang, L. Xu, Z. Cai, Z. Xu, Re-scale adaboost for attack detection in collaborative filtering recommender systems, Knowledge-Based Systems 100 (2016) 74–88.
[30] M. F. Aljunid, M. Dh, An efficient deep learning approach for collaborative filtering recommender system, Procedia Computer Science 171 (2020) 829–836.
[31] U. o. M. Institute for Social Research, Median household income [2006-2010], 2020. URL: https://www.psc.isr.umich.edu/dis/census/Features/tract2zip/.