

CaMeLS: Cooperative Meta-Learning Service for Recommender Systems

Lukas Wegmeth¹, Joeran Beel¹

¹Intelligent Systems Group, University of Siegen, Adolf-Reichwein-Straße 2, 57076 Siegen, Germany

Abstract

We present CaMeLS, a proof of concept of a cooperative meta-learning service for recommender systems. CaMeLS leverages the computing power of recommender systems users by uploading their metadata and algorithm evaluation scores to a centralized environment. Through the resulting database, CaMeLS then offers meta-learning services for everyone. Additionally, users may access evaluations of common data sets immediately to know the best-performing algorithms for those data sets. The metadata table may also be used for other purposes, e.g., to perform benchmarks. In the initial version discussed in this paper, CaMeLS implements automatic algorithm selection through meta-learning over two recommender systems libraries. Automatic algorithm selection saves users time and computing power and does not require expertise, as the best algorithm is automatically found over multiple libraries. The CaMeLS database contains 20 metadata sets by default. We show that the automatic algorithm selection service is already on par with the single best algorithm in this default scenario. CaMeLS only requires a few seconds to predict a suitable algorithm, rather than potentially hours or days if performed manually, depending on the data set. The code is publicly available on our GitHub <https://camels.recommender-systems.com>.

Keywords

recommender systems, benchmark, model selection, algorithm selection, meta-learning, automated machine learning

1. Introduction


Model selection is an essential technique with many advantages for machine learning and, by extension, recommender systems (RecSys). It can reduce the required time to build a meaningful predictor, reduce user expertise requirements and increase prediction performance. Standard methods for model selection, like random search, require expertise from the user to set up the search space and require additional time and processing power as many rounds of validation need to be completed. Additionally, depending on the task, these techniques may be inefficient or yield sub-optimal results [1]. With the rise of automated machine learning (AutoML), new methods like Bayesian hyperparameter optimization [2] were adopted, improving results but still having similar requirements [3]. When there are multiple algorithms to choose, these requirements get even more complicated.


It is not yet possible to quickly and easily achieve results similar to state-of-the-art hyperparameter optimization techniques by other means. However, it is possible to warm-start this

Perspectives on the Evaluation of Recommender Systems Workshop (PERSPECTIVES 2022), September 22nd, 2022, co-located with the 16th ACM Conference on Recommender Systems, Seattle, WA, USA.

✉ lukas.wegmeth@uni-siegen.de (L. Wegmeth); joeran.beel@uni-siegen.de (J. Beel)

ORCID [0000-0001-8848-9434](https://orcid.org/0000-0001-8848-9434) (L. Wegmeth); [0000-0002-4537-5573](https://orcid.org/0000-0002-4537-5573) (J. Beel)

 © 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

optimization through meta-learning [4]. Some state-of-the-art AutoML tools employ meta-learning for this and many other purposes due to its power to transfer knowledge from one task to another [5, 6]. Meta-learning and AutoML only recently surfaced in RecSys, but have already proved to be valuable in different model selection tasks [7, 8, 9, 10, 11].

Meta-learning is an application of machine learning. It is bound to the same constraints, e.g., the requirement for a sufficient amount of data. AutoML tools like *Auto-sklearn* [4] manually gather such data and craft a diverse meta-learner specific to the task at hand. However, these tools are not directly applicable to RecSys because the data and tasks differ drastically from regular machine learning, e.g., in sparsity.

Due to the success of meta-learning for general machine learning, we hypothesize that, if sufficient data were present in a RecSys context, these techniques would also excel in the RecSys domain. Of course, acquiring vast amounts of diverse data is challenging. There is no abundance of public data sets for RecSys, and the list only grows slowly¹. Additionally, the quality of available data sets may not be sufficient. Furthermore, private data sets may differ from public ones, making generalizing a meta-learner to these even harder.

To circumvent the problems mentioned above, we present the concept of a cooperative meta-learning service for RecSys and a proof of concept that we call CaMeLS. Our group introduced the idea for use with general machine learning and implemented a proof of concept in a previous publication. It shows that the system provides an immense advantage in terms of time and computing power [12]. With CaMeLS, we extend on this, specifically for RecSys, and provide the first meta-learning performance evaluation. CaMeLS is an automatic algorithm selection service, which solves a sub-task of model selection. Our evaluation of CaMeLS shows that it is on par with the single best algorithm directly out of the box. Additionally, getting the correct or suitable algorithm with CaMeLS merely takes a few seconds, compared to potentially multiple hours if done manually.

Conceptually, we envision an environment where users share relevant information about RecSys data sets without the need to share sensitive, personal, and private data. Therefore, CaMeLS collects only the metadata of input data sets and their performance metrics on a set of algorithms. Users perform the training and evaluation on their machine, and CaMeLS collects the results in a centralized environment open to anyone to read and write. This procedure leverages the computing power of contributing users by making results available to everyone. This cooperative effort converges into a collection of metadata sets and performance values from which a powerful meta-learner can be built and continuously extended, fully accessible by the community. Such a collection has many additional benefits. It saves time for everyone since users can retrieve stored results instead of repeatedly computing evaluations for common tasks. It also serves as a tabular metadata set that users can retrieve to develop and perform thorough benchmarks on.

2. CaMeLS

While there are multiple ways to realize the introduced concept, we present a prototypical implementation of CaMeLS as proof of concept. We implemented CaMeLS as a traditional

¹<https://cseweb.ucsd.edu/~jmcauley/datasets.html>

client-server model with an open API. CaMeLS stores the metadata of RecSys data sets and their evaluation scores on RecSys algorithms. It immediately returns the best algorithm if evaluations for the input data set are already stored. Otherwise, predicting the best algorithm for unseen data only takes a few seconds. Using CaMeLS is easy: uploading evaluations and predicting algorithm performance need a single function call.

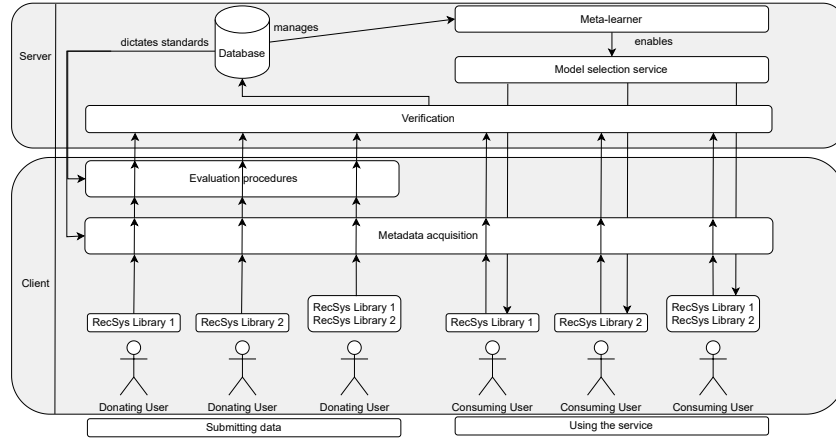


Figure 1: A diagram of the workflow for the proof of concept. Fundamentally, there are two groups of users which are donors and consumers. They share a standardized pipeline whose settings are dictated by the database setup.

The workflow in CaMeLS depends on whether the user is submitting evaluation data or using the service. It is shown in Figure 1. We call these users donors and consumers, respectively. If users volunteer as donors, their input data set passes through a server-dictated preprocessing to standardize the metadata extraction process. Should the data hash already be present on the server, the user decides if they repeat the evaluation. Next, the metadata is calculated and stored on the server. The user then trains the selected algorithms on their input data and evaluates them. The splitting of the data set into a train and test set and the evaluation of predictions from the trained models are standardized through shared, configurable functions. Standardized metric computations ensure that the performance of each algorithm, no matter from which library, is computed equally, which makes performance scores of different libraries comparable. Finally, the user uploads the evaluation scores to the server. The server finally verifies the upload and stores the values.

If a user wishes to consume the service, they must pass their input data set through the same standardized preprocessing steps that the donors originally went through. So, the metadata is calculated and uploaded to the CaMeLS server. If the server knows the data by its hash, it returns a recommended algorithm for the input data. Otherwise, a trained meta-learner for the targeted user setup has to exist on the server to continue. If one does not exist, the management policy decides if it is trained on-demand or scheduled by a controlling administrator. The user chooses the meta-learner that then predicts the algorithm performance for the previously unknown input metadata and returns the predicted performance of all algorithms to the client. The client may then, for example, automatically construct a model with default parameters based on the algorithm with the highest predicted performance.

The meta-learner is a multi-label regressor that predicts the performance score of the metadata set on each algorithm. By default, CaMeLS uses random forest regression by *scikit-learn* [13] for the meta-learner. Alternatively, CaMeLS also allows the integration of other meta-learners. The meta-learner learns the relationship between the complexity of data sets and algorithm performances. The metadata set contains one training instance for each available data set. The features of the metadata set correspond to 17 complexity measures listed in Appendix A. The ground truth of the metadata set corresponds to the evaluation scores of one metric for each algorithm. As a result, there are separate meta-learners and metadata sets for every available metric. Consequently, a meta-learner predicts the evaluation scores of the associated metric for each algorithm on unseen data.

For now, CaMeLS supports algorithms from the RecSys libraries *Lenskit* [14] and *Surprise* [15]. Appendix C lists all algorithms with their official descriptions. In addition, it uses the generalized *Movielens*² data loading routines provided by *Lenskit* and extends these with more routines for some common data sets and data set families like *Amazon*³. Appendix B contains an organized list of the data sets. As a result, CaMeLS supports more than 30 data sets right away. The implemented evaluation metrics are the runtime, normalized mean absolute error (NMAE), mean absolute error (MAE), normalized root mean squared error (NRMSE), and root means squared error (RMSE). Currently, CaMeLS supports the task of predicting explicit ratings with the option to extend to implicit ranking prediction and any other user-defined tasks. The CaMeLS database has a simple and extensible structure to store more complex relations in the future, e.g., a more complex metadata system presented by Amazon [16].

3. Evaluation

To evaluate CaMeLS, we simulated a client donating metadata and evaluation scores of 20 of the supported data sets. The evaluated data sets and additional information about the metadata and algorithms is listed in the Appendix. We collected the performance metrics for the 16 supported algorithms. Due to resource constraints, we collected the data with holdout validation and configured data pruning where each user in each data set has to have at least five and at most 1000 ratings. With five performance metrics, this yields a total of 1600 evaluations. Since each meta-learner trains on one metric at a time and because we treat the model selection problem as a multi-label regression problem, there is only one instance per metadata set for the meta-learner training. Hence, the meta-learner for each metric only learns from 20 instances with 16 labels each in this procedure.

We evaluate the meta-learners' predictive performance by performing leave-one-out cross-validation. Because randomness affects the evaluation due to the small size of our metadata set, we average the results over 50 evaluation repetitions. We show the evaluation results for the random forest meta-learner on the MAE and RMSE metrics in Table 1. The evaluation shows that the RMSE meta-learner outperforms the single best algorithm in selection accuracy by 4.2% and is on par with its average error. The MAE meta-learner is worse than the single best algorithm with a 5.1% lower selection accuracy and 0.01 higher average error. However, the

²<https://grouplens.org/datasets/movielens/>

³<https://nijianmo.github.io/amazon/index.html>

difference between the meta-learner and single best algorithm is marginal for both, especially considering the average error. In algorithm selection, beating the single best algorithm is a standard minimum requirement. And we can surpass it with CaMeLS for the RMSE meta-learner even with scarce data and basic complexity measures.

Table 1

This table shows the performance of the random forest meta-learner on the MAE and RMSE metrics. We compare it to the oracle and single best algorithm. The oracle knows the ground truth and always picks the best algorithm. The single best algorithm is the algorithm that most often performed best on the training metadata set. It shows the selection accuracy and average error of the selection method cross-validated using the leave-one-out method and averaged over 50 validation repetitions.

| Mean Absolute Error | | | Root Mean Squared Error | | |
|---------------------|--------------------|------------------|-------------------------|--------------------|------------------|
| Average Error | Selection Accuracy | Selection Method | Average Error | Selection Accuracy | Selection Method |
| 0.91 | 100% | Oracle | 1.26 | 100% | Oracle |
| 0.92 | 25% | Single Best | 1.28 | 30% | Single Best |
| 0.93 | 19.9% | Meta-Learner | 1.28 | 34.2% | Meta-Learner |

In addition to its predictive capabilities, CaMeLS saves time and computing power. Manually reading the data sets into two different libraries and performing evaluations on each algorithm may take multiple hours or days, depending on the size of the input data set. Contrarily, CaMeLS can predict a suitable algorithm in a few seconds from reading the data.

4. Discussion

While meta-learning provides an opportunity for model selection in RecSys, there are still many challenges to overcome with our presented concept. Model selection through meta-learning in RecSys is not an easy task and requires further research into, e.g., metadata acquisition. Sufficient data must be collected to start up and improve the service. However, the evaluation has shown that even a small amount of data may already provide an immediate benefit. Of course, the initial data collection task can be performed automatically on popular data sets, similar to what we did for CaMeLS.

The idea is that users also contribute data voluntarily. A simple incentive can be the implicitly assumed benevolence of users to improve the service for everyone. But more tangible incentives may be found. Depending on the use case, there must be considerations for whether the upload should be an opt-in or opt-out procedure. At the same time, the upload routine must be easily accessible, so users will not feel a burden when uploading their data. Additionally, if there are no restrictions on usage, the server host should consider the free-rider problem.

There is a range of other research questions to be answered, e.g., what are the best complexity measures considering their computational effort, and what is the ideal setup for the meta-learner? Right now, metadata acquisition and meta-learner training are both relatively high-speed due to their simplicity. When more complex metadata is involved, the metadata calculation will take longer and possibly discourage users from donating data. The benefit-cost ratio of any task performed with this system is especially significant for the clients.

References

- [1] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization., *Journal of machine learning research* 13 (2012).
- [2] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, S.-H. Deng, Hyperparameter optimization for machine learning models based on bayesian optimizationb, *Journal of Electronic Science and Technology* 17 (2019) 26–40. URL: <https://www.sciencedirect.com/science/article/pii/S1674862X19300047>. doi:<https://doi.org/10.11989/JEST.1674-862X.80904120>.
- [3] P. Matuszyk, R. T. Castillo, D. Kottke, M. Spiliopoulou, A comparative study on hyperparameter optimization for recommender systems, in: *Workshop on Recommender Systems and Big Data Analytics (RS-BDA'16)*, volume 13, 2016.
- [4] M. Feurer, K. Eggenesperger, S. Falkner, M. Lindauer, F. Hutter, Auto-sklearn 2.0: The next generation, *CoRR abs/2007.04074* (2020). URL: <https://arxiv.org/abs/2007.04074>. arXiv:2007.04074.
- [5] M. Grobelnik, J. Vanschoren, Warm-starting darts using meta-learning, 2022. URL: <https://arxiv.org/abs/2205.06355>. doi:10.48550/ARXIV.2205.06355.
- [6] L. Zimmer, M. Lindauer, F. Hutter, Auto-pytorch: Multi-fidelity metalearning for efficient and robust autodl, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43 (2021) 3079–3090.
- [7] M. Luo, F. Chen, P. Cheng, Z. Dong, X. He, J. Feng, Z. Li, Metaselector: Meta-learning for recommendation with user-level adaptive model selection, *CoRR abs/2001.10378* (2020). URL: <https://arxiv.org/abs/2001.10378>. arXiv:2001.10378.
- [8] A. Nechaev, V. Meltsov, N. Zhukova, Utilizing metadata to select a recommendation algorithm for a user or an item, in: *CEUR Workshop Proceedings*, 2020.
- [9] H. Bharadhwaj, Meta-learning for user cold-start recommendation, in: *2019 International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–8. doi:10.1109/IJCNN.2019.8852100.
- [10] A. Collins, D. Tkaczyk, J. Beel, One-at-a-time: A meta-learning recommender-system for recommendation-algorithm selection on micro level, *arXiv preprint arXiv:1805.12118* (2018).
- [11] R. Anand, J. Beel, Auto-surprise: An automated recommender-system (autorecsys) library with tree of parzens estimator (tpe) optimization, in: *Fourteenth ACM Conference on Recommender Systems*, 2020, pp. 585–587.
- [12] M. Arambakam, J. Beel, Federated meta-learning: Democratizing algorithm selection across disciplines and software libraries, in: *7th ICML Workshop on Automated Machine Learning (AutoML)*, 2020.
- [13] 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.
- [14] 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*, 2020, pp. 2999–3006.
- [15] 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.
- [16] S. Schelter, J.-H. Boese, J. Kirschnick, T. Klein, S. Seufert, Automatically tracking metadata and provenance of machine learning experiments, in: Machine Learning Systems Workshop at NIPS, 2017, pp. 27–29.

A. Complexity Measures

The list of the complexity measure that CaMeLS calculates and uses as metadata.

1. Number of users
2. Number of items
3. Minimum rating
4. Maximum rating
5. Mean rating
6. Normalized mean rating
7. Number of instances
8. Highest number of rating by a single user
9. Lowest number of ratings by a single user
10. Highest number of ratings on a single item
11. Lowest number of ratings on a single item
12. Mean number of ratings by a single user
13. Mean number of ratings on a single item
14. Rating skew
15. Rating kurtosis
16. Rating standard deviation
17. Rating variance

B. Data Sets

The list of data sets supported by CaMeLS. Bold text indicates that the data set was used in the evaluation.

MovieLens Source:<https://grouplens.org/datasets/movielens/>

1. **MovieLens 100K**
2. **MovieLens 1M**
3. MovieLens 10M
4. MovieLens 20M
5. **MovieLens Latest Small**

Amazon Source:<https://nijianmo.github.io/amazon/index.html>

1. **amazon-all-beauty**
2. **amazon-appliances**
3. **amazon-arts-crafts-and-sewing**
4. amazon-automotive
5. amazon-books
6. amazon-cds-and-vinyl
7. amazon-cell-phones-and-accessories
8. amazon-clothing-shoes-and-jewelry
9. **amazon-digital-music**
10. amazon-electronics
11. **amazon-fashion**
12. **amazon-gift-cards**
13. amazon-grocery-and-gourmet-food
14. **amazon-industrial-and-scientific**
15. amazon-home-and-kitchen
16. amazon-kindle-store
17. **amazon-luxury-beauty**
18. **amazon-magazine-subscriptions**
19. amazon-movies-and-tv
20. **amazon-musical-instruments**
21. amazon-office-products
22. amazon-patio-lawn-and-garden
23. amazon-pet-supplies
24. **amazon-prime-pantry**
25. **amazon-software**
26. amazon-sports-and-outdoors
27. amazon-tools-and-home-improvement
28. amazon-toys-and-games
29. **amazon-video-games**

BookCrossing Source:<https://grouplens.org/datasets/book-crossing/>

EachMovie Source:<http://www.gatsby.ucl.ac.uk/~chuwei/data/EachMovie/eachmovie.html>

Jester Source:<http://eigentaste.berkeley.edu/dataset/>

1. **Jester3**
2. **Jester4**

C. Algorithms

The list of algorithms supported by CaMeLS.

Lenskit [14] algorithms with descriptions from their official documentation:

1. UserUser: User-user nearest-neighbor collaborative filtering.
2. ItemItem: Item-item nearest-neighbor collaborative filtering.
3. BiasedMF: Biased matrix factorization trained with alternating least squares.
4. BiasedSVD: Biased matrix factorization for implicit feedback using SciKit-Learn's SVD solver.
5. FunkSVD: Algorithm class implementing FunkSVD matrix factorization.
6. Bias: A user-item bias rating prediction algorithm.

Surprise [15] algorithms with descriptions from their official documentation:

1. NormalPredictor: Algorithm predicting a random rating based on the distribution of the training set, which is assumed to be normal.
2. Baseline: Algorithm predicting the baseline estimate for given user and item.
3. KNNBasic: A basic collaborative filtering algorithm.
4. KNNWithMeans: A basic collaborative filtering algorithm, taking into account the mean ratings of each user.
5. KNNWithZScore: A basic collaborative filtering algorithm, taking into account the z-score normalization of each user.
6. KNNBaseline: A basic collaborative filtering algorithm taking into account a baseline rating.
7. SVD: The famous SVD algorithm, as popularized by Simon Funk during the Netflix Prize.
8. NMF: A collaborative filtering algorithm based on Non-negative Matrix Factorization.
9. SlopeOne: A simple yet accurate collaborative filtering algorithm.
10. CoClustering: A collaborative filtering algorithm based on co-clustering.