RecPack: An(other) Experimentation Toolkit for Top-N Recommendation using Implicit Feedback Data

Lien Michiels*
Robin Verachtert*
Froomle
Antwerp, Belgium
University of Antwerp
Antwerp, Belgium

Bart Goethals
University of Antwerp
Antwerp, Belgium
Monash University
Australia
Froomle
Antwerp, Belgium

ABSTRACT

RecPack is an easy-to-use, flexible and extensible toolkit for top-N recommendation with implicit feedback data. Its goal is to support researchers with the development of their recommendation algorithms, from similarity-based to deep learning algorithms, and allow for correct, reproducible and reusable experimentation. In this demo, we give an overview of the package and show how researchers can use it to their advantage when developing recommendation algorithms.

CCS CONCEPTS

• **Information systems → Recommender systems**; *Collaborative filtering*; **Open source software**; • **Computing methodologies → Learning from implicit feedback**.

KEYWORDS

Python, open-source framework, evaluation, top-N recommendation, implicit feedback data

ACM Reference Format:
Lien Michiels, Robin Verachtert, and Bart Goethals. 2022. RecPack: An(other) Experimentation Toolkit for Top-N Recommendation using Implicit Feedback Data. In *Sixteenth ACM Conference on Recommender Systems (RecSys '22), September 18–23, 2022, Seattle, WA, USA*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3523227.3551472

1 INTRODUCTION

Over the past decade, recommender systems have become a staple of online user experiences across industries, from media to tourism or e-commerce. At the same time, the domain of recommender systems' research has evolved from a focus on rating prediction tasks, e.g. the Netflix 1 Million Dollars Prize [5], to top-N recommendation [26]. Today, the state-of-the-art in top-N recommendation advances at a very fast pace. More and more, researchers are encouraged to share their code to enable reproducibility and increase the transparency of their research process [30]. However, this code is often of low quality and/or is shared without instructions on how to set up your environment to allow reproduction of the study [1, 30]. Even if this code allows the experiments to be reproduced, which Trisovic et al. [30] found to be about 25% of the time, this does not mean that the code can also be reused. Code is considered *reusable* when it allows other researchers to easily conduct similar experiments, e.g., in new contexts or on new data [1]. Reusable code allows science to progress at an even faster pace. Unfortunately writing reusable code requires significant effort on the part of the researcher [1]. With RecPack, our aim is to assist researchers in this important effort.

RecPack is an experimentation toolkit for top-N recommendation using implicit feedback data written in Python, with a familiar interface and clear documentation. Its goal is to support researchers who advance the state-of-the-art in top-N recommendation to write reproducible and reusable experiments. RecPack comes with a range of different datasets, recommendation scenarios, state-of-the-art baselines and metrics. Wherever possible, RecPack sets sensible defaults. For example, the hyperparameters of all recommendation algorithms included in RecPack are initialized to the best performing settings found in the original experiments. The design of RecPack is heavily inspired by the interface of `scikit-learn`, a popular Python package for classification, regression, and clustering tasks. Data scientists who are familiar with `scikit-learn` will already have an intuitive understanding of how to work with RecPack. On top of this, RecPack was developed with a production mindset: All contributions are rigorously reviewed and tested. The RecPack maintainers strive to maintain a test coverage of more than ninety percent at all times. Using RecPack, researchers can:

- **Quickly develop algorithms** by using one of RecPack's abstract algorithm base classes, which represent popular recommendation paradigms such as factorization and item-to-item similarity.
- **Compare their algorithms against state-of-the-art baselines** in several different recommendation scenarios using a variety of performance metrics.
- **Tune hyperparameters** of both baselines and their own implementations with minimal data leakage.

In recent years, many other Python packages for top-N recommendation have been released [e.g. 3, 29, 35]. However, these focus more on the purpose of 'benchmarking', i.e., quick and accurate comparisons of state-of-the-art recommendation algorithms. Consequently, they provide access through the use of a configuration language or command-line interface. RecPack on the other hand

*Both authors contributed equally to this research.

RecSys '22, September 18–23, 2022, Seattle, WA

ACM ISBN 978-1-4503-9278-5/22/09.
https://doi.org/10.1145/3523227.3551472

wishes to support researchers with the development of their own algorithms through repeated refinement, experimentation and bug-fixing in e.g., a Notebook environment.

In this demo, we will give you an overview of the different components of RecPack and how to use them to run your experiments and speed up the development of your own recommendation algorithms.

2 RECPACK LIBRARY

A typical experimentation pipeline for top-N recommendation is shown in Figure 1[1]. RecPack provides a dedicated module to support you with each step. In this Section, we will discuss the functionality of each of these modules.

Datasets. RecPack's datasets provide easy access to many of the most popular collaborative filtering datasets [4, 13, 14, 18, 21, 24, 33]. To use one, simply instantiate a Dataset object, e.g., `d = MovieLens25M()`. Using RecPack's datasets saves you time in two ways. First, if the dataset is publicly available RecPack will download it for you. If you already have a copy of the dataset or access to the dataset is restricted, you can specify a path and filename to tell RecPack where to find it. Second, for each dataset a set of default preprocessing filters is defined. Using these default preprocessing filters increases the comparability between your experiments and those of other researchers. Of course, this default filtering can be turned off when creating the Dataset, after which you can add your own preprocessing filters, as detailed in the next paragraph. If your dataset of choice is not included in RecPack we recommend you load it as a pandas DataFrame [34].

Preprocessing. In collaborative filtering, it is customary to first apply preprocessing filters to the raw dataset and then transform it into a user-item interaction matrix [2, 8, 29]. If you use one of RecPack's built-in datasets with the default preprocessing filters, this is done for you. If you are using your own dataset or wish to override the default preprocessing filters, the first step is to create the filters of your choice.

RecPack currently supports a choice of seven popular preprocessing filters. MinUsersPerItem, MinItemsPerUser and MaxItemsPerUser are used to remove outliers that can negatively impact recommendation performance. NMostPopular an NMostRecent can be used to limit the size of the item catalogue. Deduplicate eliminates repeat interactions by retaining only the first occurrence of every user-item pair. Finally, MinRating selects ratings equal to or higher than a minimum rating value. It is used to transform rating data into implicit feedback data.

To apply these filters to your own DataFrame, create a DataFramePreprocessor. This DataFramePreprocessor will apply the filters and transform the user and item identifiers into consecutive matrix indices. To add filters to either a Dataset or a DataFramePreprocessor, use their add_filter method.

Matrix. In RecPack, a user-item interaction matrix, optionally with timestamps, is represented by an InteractionMatrix object. To create one, call the load method of your Dataset or pass your pandas DataFrame to the DataFramePreprocessor's process method. You can also create an InteractionMatrix directly from your own preprocessed pandas DataFrame or a SciPy [32] Sparse csr_matrix. The InteractionMatrix provides different views of your data. For example, you can extract your data as a csr_matrix with user-item interaction counts, the same user-item interactions binarized, or a list of items interacted with for every user, sorted from first to last interaction. The best thing about the InteractionMatrix is that for most experiments you do not have to worry about it. The Scenarios, Algorithms and Pipeline, discussed in the following paragraphs, know exactly what to do with it.

Scenarios. Avoiding data leakage, yet evaluating a recommendation algorithm on a representative task is a difficult challenge. Recommendations are used in many different contexts. In one, it may be most important to make reasonable recommendations for previously unseen users. In another, it is important to get the user's next move exactly right. RecPack comes with an elaborate set of recommendation scenarios, covering virtually all train-validation-test splits encountered in the scientific literature on recommender systems [2, 6, 8, 25, 29]. When you do not need to tune any hyperparameters, pass validation=False to the constructor of your Scenario to split your InteractionMatrix into a training and test set. When you do wish to tune your hyperparameters, pass validation=True instead to obtain a training, validation, and test set. For a complete overview of all supported scenarios, see the *documentation*.

Algorithms. RecPack currently includes over twenty state-of-the-art recommendation algorithms that cover some of the most popular recommendation paradigms: Item-to-item similarity [7, 12, 22, 31], factorization [2, 17, 23], auto-encoders [19, 27, 28], session-based [10, 16] and time-aware recommendation algorithms [20]. Additionally, it provides three abstract base classes that you can use to accelerate the development of your own algorithm: ItemSimilarityMatrixAlgorithm, FactorizationAlgorithm, and TorchMLAlgorithm. These base classes implement a significant portion of the shared recommendation logic so that you can focus your efforts on the things that make your recommendation algorithm unique. All of RecPack's algorithms implement scikit-learn's BaseEstimator interface: They have fit and predict methods and expect hyperparameters to be passed when the object is created.

Postprocessing. Real world recommender systems often apply postprocessing, in the form of business rules, to the predictions made by the recommendation algorithm. In an e-commerce context, for example, it is customary to exclude sensitive or age-restricted items. In a news context, the recommendations are limited to articles published in the last two days. RecPack currently allows you to either select a subset of items through the use of SelectItems or exclude unwanted items via the ExcludeItems PostFilter.

Metrics. Of course, the ultimate goal of a recommendation experiment is to evaluate the performance of a recommendation algorithm. RecPack comes with a selection of the most commonly used metrics in Top-N ranking [2, 25]. For example, to obtain the *NDCG@20* of your algorithm, first create the metric `ndcg = NDCGK(20)`. Next, pass the targets and predictions to the metric's calculate method. To obtain the average *NDCG@20* over users, use `ndcg.value`. However, RecPack's metrics also allow for a more

[1]Icons made by Freepik, Gregor Cresnar, Uniconlabs, alexdndz and orvipixel from www.flaticon.com

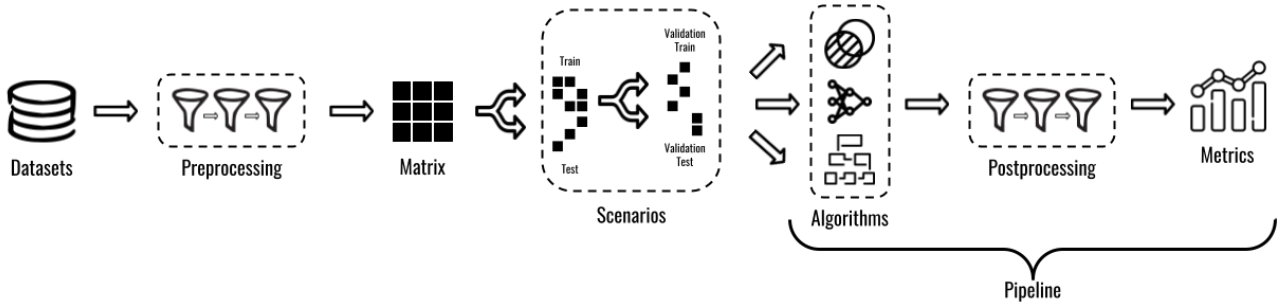

Figure 1: Top-N Recommendation Pipeline: First, a dataset is preprocessed and transformed into a user-item interaction matrix. Next, this matrix is split into a training, validation and test dataset. These datasets are then used to first train algorithms and later make recommendations. Finally recommendations are postprocessed, after which performance metrics are computed.

fine-grained analysis of the performance of your algorithms. To inspect detailed performance results, e.g., the *NDCG@20* of individual users, use `ndcg.results`. For a complete overview of all metrics, see the *documentation*.

Pipelines. RecPack's `Pipeline` helps you to determine the optimal hyperparameters for a given algorithm and dataset, apply postprocessing, and evaluate the performance of your algorithm (and baselines) on a selection of performance metrics. To use it, first create a `pb = PipelineBuilder()`. Then, pass your `Scenario` to its `set_data_from_scenario` method to initialize the training, validation and test set. Use the `add_algorithm` method to add any algorithms you want to evaluate and `add_metric` to add performance metrics. To obtain a `Pipeline`, call `p = pb.build()` and then use `p.run()` to run your pipeline. For more advanced use, see the *documentation*.

3 SETUP

RecPack is available for download from *PyPI*. In the *documentation*, you will find *Getting Started Guides* as well as detailed documentation of each of RecPack's modules. The code is open-source and available on *GitLab*. RecPack is licensed under AGPL [9]. To raise issues or ask questions about the use of RecPack, check out the source code and contribution guidelines on *GitLab*.

4 DEMO

In the demo, we first implement `MLP`, a neural matrix factorization algorithm proposed in He et al. [15], to showcase how RecPack's `TorchMLAlgorithm`'s base class assists you in the development of your own recommendation algorithms. We then carry out a recommendation experiment. First, we select the `MovieLens25M Dataset` and transform it into an `InteractionMatrix`. Next, we split this `InteractionMatrix` into a training, validation and test dataset using the `WeakGeneralization Scenario`. Subsequently, we create a `PipelineBuilder` and start building a `Pipeline` in which we compare the performance of `MLP` to `WMF` and `BPRMF`, two baselines included in RecPack. We define a parameter grid for hyperparameter tuning, an optimization metric, and evaluation metrics. Finally, we build and run our pipeline and evaluate the performance of each of the algorithms.

5 FUTURE WORK

RecPack is being actively maintained and developed. In the near future, its maintainers plan to add beyond-accuracy metrics [11] and hybrid and content-based recommendation algorithms [2].

REFERENCES

[1] 2021. But is the code (re)usable? *Nature Computational Science* 1, 7 (01 Jul 2021), 449–449. https://doi.org/10.1038/s43588-021-00109-9

[2] Charu C. Aggarwal. 2016. *Recommender Systems: The Textbook*. Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-29659-3

[3] Vito Walter Anelli, Alejandro Bellogin, Antonio Ferrara, Daniele Malitesta, Felice Antonio Merra, Claudio Pomo, Francesco Maria Donini, and Tommaso Di Noia. 2021. Elliot: A Comprehensive and Rigorous Framework for Reproducible Recommender Systems Evaluation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, Canada) *(SIGIR '21)*. Association for Computing Machinery, New York, NY, USA, 2405–2414. https://doi.org/10.1145/3404835.3463245

[4] David Ben-Shimon, Alexander Tsikinovsky, Michael Friedmann, Bracha Shapira, Lior Rokach, and Johannes Hoerle. 2015. RecSys Challenge 2015 and the YOOCHOOSE Dataset. In *Proceedings of the 9th ACM Conference on Recommender Systems* (Vienna, Austria) *(RecSys '15)*. Association for Computing Machinery, New York, NY, USA, 357–358. https://doi.org/10.1145/2792838.2798723

[5] James Bennett and Stan Lanning. 2007. The Netflix Prize. In *Proceedings of KDD Cup and Workshop 2007* (San Jose, California, USA) *(KDDCup '07)*. Association for Computing Machinery, New York, NY, USA, 4 pages. https://www.cs.uic.edu/~liub/KDD-cup-2007/NetflixPrize-description.pdf

[6] Pedro G. Campos, Fernando Díez, and Iván Cantador. 2014. Time-Aware Recommender Systems: A Comprehensive Survey and Analysis of Existing Evaluation Protocols. *User Modeling and User-Adapted Interaction* 24, 1–2 (feb 2014), 67–119. https://doi.org/10.1007/s11257-012-9136-x

[7] Mukund Deshpande and George Karypis. 2004. Item-Based Top-N Recommendation Algorithms. *ACM Trans. Inf. Syst.* 22, 1 (jan 2004), 143–177. https://doi.org/10.1145/963770.963776

[8] Kim Falk. 2019. *Practical Recommender Systems*. Manning. https://www.manning.com/books/practical-recommender-systems

[9] Free Software Foundation. 2016. GNU Affero General Public License Version 3 (AGPL-3.0). Accessed 26 July 2022.

[10] Diksha Garg, Priyanka Gupta, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. 2019. Sequence and Time Aware Neighborhood for Session-Based Recommendations: STAN *(SIGIR'19)*. Association for Computing Machinery, New York, NY, USA, 1069–1072. https://doi.org/10.1145/3331184.3331322

[11] Mouzhi Ge, Carla Delgado-Battenfeld, and Dietmar Jannach. 2010. Beyond Accuracy: Evaluating Recommender Systems by Coverage and Serendipity. In *Proceedings of the Fourth ACM Conference on Recommender Systems* (Barcelona, Spain) *(RecSys '10)*. Association for Computing Machinery, New York, NY, USA, 257–260. https://doi.org/10.1145/1864708.1864761

[12] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. 2015. E-Commerce in Your Inbox: Product Recommendations at Scale. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Sydney, NSW, Australia) *(KDD '15)*. Association for Computing Machinery, New York, NY, USA, 1809–1818. https://doi.org/10.1145/2783258.2788627

[13] Jon Atle Gulla, Lemei Zhang, Peng Liu, Özlem Özgöbek, and Xiaomeng Su. 2017. The Adressa Dataset for News Recommendation. In *Proceedings of the*

International Conference on Web Intelligence (Leipzig, Germany) *(WI '17)*. Association for Computing Machinery, New York, NY, USA, 1042–1048. https://doi.org/10.1145/3106426.3109436

[14] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Trans. Interact. Intell. Syst.* 5, 4, Article 19 (dec 2015), 19 pages. https://doi.org/10.1145/2827872

[15] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In *Proceedings of the 26th International Conference on World Wide Web* (Perth, Australia) *(WWW '17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 173–182. https://doi.org/10.1145/3038912.3052569

[16] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-based Recommendations with Recurrent Neural Networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (San Juan, Puerto Rico), Yoshua Bengio and Yann LeCun (Eds.). https://doi.org/10.48550/ARXIV.1511.06939

[17] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *2008 Eighth IEEE International Conference on Data Mining*. 263–272. https://doi.org/10.1109/ICDM.2008.22

[18] Michael Kechinov. 2020. CosmeticsShop E-commerce Dataset. https://www.kaggle.com/mkechinov/ecommerce-events-history-in-cosmetics-shop Accessed: 2022-07-26.

[19] Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, and Tony Jebara. 2018. Variational Autoencoders for Collaborative Filtering. In *Proceedings of the 2018 World Wide Web Conference* (Lyon, France) *(WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 689–698. https://doi.org/10.1145/3178876.3186150

[20] Nathan N. Liu, Min Zhao, Evan Xiang, and Qiang Yang. 2010. Online Evolutionary Collaborative Filtering. In *Proceedings of the Fourth ACM Conference on Recommender Systems* (Barcelona, Spain) *(RecSys '10)*. Association for Computing Machinery, New York, NY, USA, 95–102. https://doi.org/10.1145/1864708.1864729

[21] Netflix. 2019. Netflix Prize Data. https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data Accessed: 2022-07-26.

[22] Xia Ning and George Karypis. 2011. SLIM: Sparse Linear Methods for Top-N Recommender Systems. In *2011 IEEE 11th International Conference on Data Mining*. 497–506. https://doi.org/10.1109/ICDM.2011.134

[23] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence* (Montreal, Quebec, Canada) *(UAI '09)*. AUAI Press, Arlington, Virginia, USA, 452–461.

[24] Retailrocket. 2017. Retailrocket Recommender System Dataset. https://www.kaggle.com/datasets/retailrocket/ecommerce-dataset Accessed: 2022-07-26.

[25] Francesco Ricci, Lior Rokach, Bracha Shapira, and Paul B. Kantor. 2011. *Recommender Systems Handbook*. Springer US, Boston, MA. https://doi.org/10.1007/978-0-387-85820-3

[26] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2000. Analysis of Recommendation Algorithms for E-Commerce. In *Proceedings of the 2nd ACM Conference on Electronic Commerce* (Minneapolis, Minnesota, USA) *(EC '00)*. Association for Computing Machinery, New York, NY, USA, 158–167. https://doi.org/10.1145/352871.352887

[27] Ilya Shenbin, Anton Alekseev, Elena Tutubalina, Valentin Malykh, and Sergey I. Nikolenko. 2020. *RecVAE: A New Variational Autoencoder for Top-N Recommendations with Implicit Feedback*. Association for Computing Machinery, New York, NY, USA, 528–536. https://doi.org/10.1145/3336191.3371831

[28] Harald Steck. 2019. Embarrassingly Shallow Autoencoders for Sparse Data. In *The World Wide Web Conference* (San Francisco, CA, USA) *(WWW '19)*. Association for Computing Machinery, New York, NY, USA, 3251–3257. https://doi.org/10.1145/3308558.3313710

[29] Zhu Sun, Di Yu, Hui Fang, Jie Yang, Xinghua Qu, Jie Zhang, and Cong Geng. 2020. *Are We Evaluating Rigorously? Benchmarking Recommendation for Reproducible Evaluation and Fair Comparison*. Association for Computing Machinery, New York, NY, USA, 23–32. https://doi.org/10.1145/3383313.3412489

[30] Ana Trisovic, Matthew K. Lau, Thomas Pasquier, and Mercè Crosas. 2022. A large-scale study on research code quality and execution. *Scientific Data* 9, 1 (21 Feb 2022), 60. https://doi.org/10.1038/s41597-022-01143-6

[31] Koen Verstrepen and Bart Goethals. 2014. Unifying Nearest Neighbors Collaborative Filtering. In *Proceedings of the 8th ACM Conference on Recommender Systems* (Foster City, Silicon Valley, California, USA) *(RecSys '14)*. Association for Computing Machinery, New York, NY, USA, 177–184. https://doi.org/10.1145/2645710.2645731

[32] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[33] Hao Wang, Binyi Chen, and Wu-Jun Li. 2013. Collaborative Topic Regression with Social Regularization for Tag Recommendation. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence* (Beijing, China) *(IJCAI '13)*. AAAI Press, 2719–2725.

[34] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman (Eds.). 56 – 61. https://doi.org/10.25080/Majora-92bf1922-00a

[35] Wayne Xin Zhao, Junhua Chen, Pengfei Wang, Qi Gu, and Ji-Rong Wen. 2020. Revisiting Alternative Experimental Settings for Evaluating Top-N Item Recommendation Algorithms. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (Virtual Event, Ireland) *(CIKM '20)*. Association for Computing Machinery, New York, NY, USA, 2329–2332. https://doi.org/10.1145/3340531.3412095