United We Stand, Divided We Fall: Leveraging Ensembles of Recommenders to Compete with Budget Constrained Resources

Pietro Maldini*
Independent
Italy, IT, Alserio
pietro.maldini@mail.polimi.it

Alessandro Sanvito
Independent
Germany, DE, Stuttgart
alessandro1.sanvito@mail.polimi.it

Mattia Surricchio
Independent
Italy, IT, Milano
mattiasu96@live.it

ABSTRACT

In this paper we provide an overview of the approach we used as team Surricchi1 for the ACM RecSys Challenge 2022[1]. The competition, sponsored and organized by Dressipi, involves a typical session-based recommendation task in the fashion industry domain. Our proposed method[2] leverages an ensemble of multiple recommenders selected to capture diverse facets of the input data. Such a modular approach allowed our team to achieve competitive results with a score of 0.1994 Mean Reciprocal Rank at 100 (~7.6% less than the first qualified team). We obtained this result by leveraging only publicly and freely available computational resources [3] and our own laptops. Part of the merit also lies in the size of this year's dataset (~5 million data points), which democratized the challenge to a larger public and allowed us to join the challenge as independent researchers.

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → **Classification and regression trees**; **Neural networks**.

KEYWORDS

collaborative filtering, content based filtering, graph recommenders, neural networks, re-ranking

ACM Reference Format:
Pietro Maldini, Alessandro Sanvito, and Mattia Surricchio. 2022. United We Stand, Divided We Fall: Leveraging Ensembles of Recommenders to Compete with Budget Constrained Resources. In *RecSys Challenge 2022 (RecSysChallenge22), September 18–23, 2022, Seattle, WA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3556702.3556845

1 INTRODUCTION

The ever-growing importance of recommender systems in the industry extends to the fashion domain, where a personalized buying experience can prove beneficial both from the consumer point of view and from the industrial perspective. ACM RecSys Challenge 2022, organized by Dressipi, challenges researchers in the field to predict, from a session of observed items, the product eventually bought by the user. The task, which would be hard in itself, is made even more complex by:

- the anonymization of the sessions, which limits the scope of the available information to the current session. This prevents creation of user based features.
- the anonymization of items and features, which prevents feature engineering driven by domain knowledge.
- the random truncation of the testing sessions.

We propose a two steps solution to solve such challenges. The first step consists of a committee of recommenders that generates candidates items for each session. In particular, similarity-based recommenders address short sessions, while neural-based recommenders tackle longer, more nuanced sessions. The second step consists of an ensemble of rankers to re-rank these candidates.

The paper is organized as follows. In section 2, we introduce the problem, the dataset and the evaluation metrics of the Challenge. In section 3, we present the feature engineering process and some important features in detail. In section 4, we describe the components of the proposed solution. In section 5, we describe our training procedure. In section 6, we report the experimental results of the most promising models. Lastly, in section 7 we draw the conclusions.

2 PROBLEM FORMULATION

The ACM RecSys Challenge 2022 requires participants to predict which fashion item would be bought by each user, only knowing data about his current navigation session. Each session contains the list of seen items by the user, and an additional dataset is available with a list of features associated with each product. This competition poses a significant challenge to predicting the bought item: many purchased products in the test set are not available during the training phase of the models. A clever approach must be employed to provide meaningful recommendations to tackle the cold start problem for those items.

Dataset. The full dataset comprises 1.1 million online retail sessions in the fashion domain, sampled from 18 months. The dataset only contains the sessions that ended with at least one product purchased, and the considered articles are clothing and footwear. Additionally to the sessions, content data for each purchased and viewed item is available: this data represents descriptive labels assigned to a product, such as colour. In the case of a session with multiple purchases, one of them is selected at random. Finally, the

*All authors contributed equally to this research.
[1]Challenge description at http://www.recsyschallenge.com/2022/
[2]Code repository https://github.com/pm390/recsys2022
[3]Such as Google Colab and Kaggle

RecSysChallenge22, September 18–23, 2022, Seattle, WA, USA

ACM ISBN 978-1-4503-9856-5/22/09…$15.00
https://doi.org/10.1145/3556702.3556845

session is truncated one item before the purchased item's first occurrence. This truncation leads to a 30.76% of session with a single seen item; this percentage increases to 34.24% if we consider only the unique items seen in a session. Such a result shows how a relevant percentage of sessions contains a single item reloaded multiple times; this phenomenon must be considered to correctly model predictions.

Objective evaluation. Recommendations are evaluated by computing the Mean Reciprocal Rank (MRR) on a list of 100 predicted items per session. Reciprocal Rank (RR) calculates the reciprocal of the rank at which the first relevant document was retrieved. When averaged across sessions, the measure is called MRR [5]. Out of 18 months of available data, the last month has been removed from the training set and used to test the participants' submissions. Sessions belonging to the last month are split randomly in equal percentages between validation and test sets, and each session is truncated randomly between 50% and 100% of its length. Due to this truncation the percentage of single items sessions increases for the test set. In particular, 33.9% of the test sessions have only an item, this increases to 37.3% if we consider only unique items. This split enforces the development of models capable of predicting the purchased item as soon as possible; at worst, half of the whole session will be missing.

Data leakage. In this new edition of the ACM RecSys Challenge, great importance has been given to data leakage. The usage of data from the validation sessions to compute features or to train the models is forbidden, this constraint must be considered carefully while developing the recommenders. In the previous edition of the ACM RecSys Challenge, many of the top teams used data from the validation dataset to train the models [2, 6, 7, 12]. The usage of validation data resulted in significant performance improvements concerning models trained without the validation data. However, obtained results are misleading and do not faithfully represent the real-world application of recommenders.

3 FEATURES

Feature engineering proved to be a decisive step to improving the performance of our models, in particular the neural ones. We computed two classes of features for our neural recommenders:

- **Session features**: Related to characteristics of the whole session;
- **Navigation features**: Related to every single step in the navigation.

Furthermore, we processed the input sessions to build and extract similarity matrices used in standard recommenders and neural ones.

3.1 Data Cleaning: Removal of reloaded pages

Before computing any feature, we cleaned our dataset by removing all the duplicated consecutive items that appeared with a time difference lower than 30 seconds in each session. We associated this phenomenon with page reloading; we decided to remove the duplicates to clean the dataset and remove a possible bias in the training of our neural models. The network could give more importance to items that appear multiple times, while in reality, they are just the result of reloading a page (maybe due to connection problems).

3.2 User Rating Matrix

We built a User Rating Matrix (URM) from the sessions provided in the dataset. We made such a matrix by placing the session id on the rows and the item id on the columns. We set every value of the matrix to the number of times a specific item is present in a session. We also created two alternative versions of URM. The first version with a temporal *Weight decay* (URM_W), we multiply each row of the URM by an exponentially decaying term with the temporal distance (computed in days) with the test sessions. The second version exploits a temporal *Weight decay Within* each session (URM_WW), we multiply the weight of each item with an exponentially decaying weight that depends on the distance from the last-seen item in the session. The *URM_W* allows us to incorporate the temporal distance from the test set, while the *URM_WW* allows us to incorporate the information of precedence of an item in a session.

3.3 Item Content Matrix

We built an Item Content Matrix (ICM) from the sessions, the categories and the features related to each item provided in the dataset. We created such a matrix by placing the item id in the rows of the matrix and encoding each possible couple of categories and feature values as columns. Furthermore, we added columns that also encoded single categories and not couples of *(category, feature_value)*. Before such encoding, we removed the couples of features and categories seen in less than 30 items from the dataset. Furthermore, we create also a dense representation of the ICM training an autoencoder over the rows of the ICM. The resulting representation is employed to train our neural models and to create additional features.

3.4 Session features

We crafted a specific set of features that represented properties related to the session. In particular, these features fall into the following categories:

- **Date related features**: we extracted information about each session's starting DateTime and ending DateTime, and we transformed them using both sine and cosine transformations to help our models understand the cyclic nature of time.
- **Session properties features**: we extracted features to represent the properties of each session, for example, the number of items seen in the session, the session length in seconds and the average time spent looking at an item. A peculiar feature we implemented is *user_went_afk*; this feature is a boolean value that is true when there was an item on which the user spent longer than 30 minutes. This threshold model a reasonable time frame where the user was probably not actively interacting with the website.
- **Special dates**: we encoded boolean values to represent particular dates that might be relevant to model shopping patterns. For example, we introduced features to represent the Christmas period or the Black Friday week. In particular, we introduced a feature *hot_hour* to describe the time frame between the 18 and 20 of each day. Our data analysis showed how the majority of purchases happened during these hours.

- **Session similarity**: we used Singular Value Decomposition (SVD) to measure the internal similarity of a session, more precisely, how similar all the items were overall. To do so, we collected all the items seen in a session, extracted the feature vector for each product, stacked the features vector in columns and finally computed the SVD decomposition of the resulting matrix (after normalization). We took the first eigenvalue and used it as a similarity measure within the session. The intuition behind the method is that if the first eigenvalue of the SVD decomposition is close to 1, the first basis of the decomposition matrixes is enough to reconstruct the original matrix. Hence, the column vectors were similar. We compute this measure using the original sparse ICM and its dense representation.

3.5 Navigation features

We generated a set of features related to the single items seen during the session. We obtained additional time series with the same length of the user session, in particular:

- **Time Delta**: we introduced a time series modelling the elapsed time between the current item and the previous one. The first seen item has a time delta equal to 0.
- **Pairwise sequential similarity**: we introduced time series that modelled the similarity score of the current item with the previous one, using the cosine similarity. We created two versions: the first leveraging the ICM to represent the item features, the second with the compressed features obtained from the autoencoder.

4 MODELS

In this section, we describe the models we trained. In particular, the models can be distinguished into two main categories: Candidate Selectors and Rankers. Figure 1 shows a schematic representation of our solution.

4.1 Candidate Selectors

The first group of models are the Candidate Selectors, given a session these models provide a list of item suggestions. In particular, these models in the literature focus on having a high recall to reduce the likelihood of not having the correct item among the candidates.

4.1.1 Content Based and Collaborative Filtering Recommenders. [4] To generate candidates for sessions, we trained several recommenders. In particular we trained:

- **Item Collaborative Filtering** (ICF)[10]: Computes the similarities between the items using the sessions' information to compare them pairwise. Given a session, it suggests items similar to those seen in the session.
- **Item Content Based Filtering** (ICBF)[10]: Computes the similarities between the items using their features to compare them pairwise. Given a session, it suggests items similar to those seen in the session.
- **User Collaborative Filtering** (UCF)[10]: Computes the similarities between users using the sessions' information to compare them pairwise. We consider only around the last 25% of the sessions. Given a session, it suggests items seen in sessions similar to the current one.

We trained a version, of each of these algorithms, for each of the following distance metrics: tanimoto, tversky, jaccard, dice, pearson correlation coefficient, cosine, adjusted cosine and asymmetric cosine. These models have been trained two times, once with the URM and one time with the URM_WW. For ICF and UCF we also trained another version with URM_W. Overall, we trained 24 ICF models, 24 UCF models and 16 ICBF models.

4.1.2 Graph Based Recommenders. [4] We used P^3_α [4] and RP^3_β [9] algorithms to generate further candidates for our rankers. P^3_α leverages random walks between users and items to provide recommendations. RP^3_β builds on top of P^3_α and modifies its outcome by dividing the similarities by each item's popularity raised to the power of a coefficient β. Each one of the previously mentioned graph based recommenders has been trained once for each version of the URM matrix (see subsection 3.2) for a total of 6 graph-based recommenders.

4.1.3 Neural Recommenders. To generate additional candidate items, we also used three different Neural Networks to capture distinct aspects of the sequential data of the sessions. We used a multi-task [13] objective to optimize Neural Recommenders. They predicted not only the bought item given a session but also the dense representation of its features. In particular, we considered:

- **Gated Recurrent Unit**(GRU)-based network [3]: Sees the items in the order of visit of the user to predict the bought item.
- **Bidirectional Long Short Term Memory**(LSTM)-based network [8]: Sees the items in both the order of visit and in reverse order of visit to predict the bought item.
- **Transformer**-based network[11]: Sees the session as a whole and learns to focus on the most relevant parts of it to predict the bought item.

4.2 Rankers

The second group of models we used are Ranker models. Given a list of candidate items, they re-rank the candidates to get a better ordering. In particular, we used *Session features* and the scores of the candidates provided by the *Candidate Selectors* as features. To obtain stable predictions, we used a group of ten LightGBMRankers with a LambdaRank [1] objective. The ranking problem is transformed into a pairwise regression problem. The algorithm takes a pair of items at a time and computes an ordering of them, in this way, a ranking function is optimized so that relevant items get a higher score while non-relevant items get a lower score.

5 TRAINING

5.1 Candidate Selection

We first trained our Candidate Selectors over the sessions in the first 16 months. To get a general idea of their performance but not to fine-tune them, we checked their performance on the 17^{th} month. These models were then used to generate candidates for the sessions in the 17^{th} month. In particular, in this phase, we

[4] We customized the implementations from https://github.com/MaurizioFD/RecSys_Course_AT_PoliMi

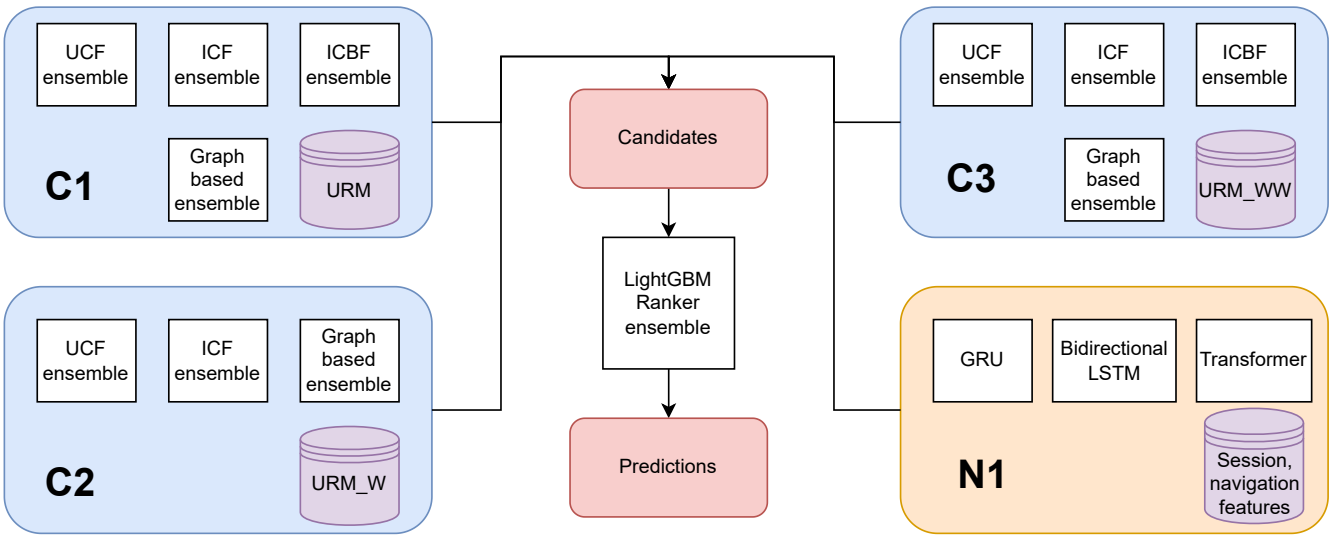

Figure 1: Ensemble and candidates generation. C1 groups all the similarity based recommenders with standard *URM*. C2 groups all the similarity based candidates with *URM_W*. C3 groups all the similarity based recommenders with *URM_WW*. N1 groups all the neural recommenders that use *session features* and *navigation features*.

Table 1: Performance of single recommenders on public leaderboard. Training over 1M sessions, inference over 50k sessions.

	ICF	UCF	ICBF	RP^3_β	GRU	Bidirectional LSTM	Transformer
MRR	0.1554	0.1719	0.067	0.1582	0.1803	0.1753	0.1795
Training	30s	75s	23s	78s	45m(GPU)	50m(GPU)	60m(GPU)
Inference	48s	46s	78s	52s	15s(GPU)	15s(GPU)	22s(GPU)

Table 2: Performance of LightGBM ranker ensembles on public leaderboard. Training on 14M candidates, inference on 50k sessions.

	Ranker C1	Ranker C1,N1	10 Rankers C1,N1	10 Rankers C1,C2,C3,N1
MRR	0.1771	0.1951	0.1970	**0.1984**
Training	4m	5m	33m	40m
Inference	48s	53s	7m	11m

focused on having a good recall for similarity-based recommenders, whereas we focused on the accuracy of the neural recommenders.

5.2 Rankers Training

Generated candidates are combined with session features into a dataset and given as input to the rankers; as a target, we added a column that indicates whether the candidate is the item bought in the session. We drop any session where the bought item is not within the list of candidates. Including the correct item, when the Candidate Selectors failed to include it, would significantly boost the accuracy of the model while not representing its strength. After obtaining this dataset, we perform a 10-fold split to obtain ten couples of <train, validation> datasets. We use these couples to train 10 LightGBMRankers to optimize the *MRR*@100.

5.3 Candidate Selection and inference for test sessions

To perform predictions on the test sessions, we train the Candidates Selectors again with the same hyperparameters using data from all the 17 training months. After that, we generate candidates for the sessions in the 18^{th} month using these new models. We create a dataset analogous to the one described in subsection 5.2. We use the 10 LightGBMRankers trained before to predict a score for each candidate, and for each session, we take the top 100 candidates with the highest average score.

6 RESULTS

Table 1 and Table 2 provide a quantitative measure of the quality of each recommendation model and the gains obtained by merging the

recommendations of single models and re-ranking them. Training and inference times have been computed leveraging an Apple Silicon M1 Pro CPU. Neural models have been trained using a NVIDIA Tesla T4 GPU. [5]

Recommendation models. Table 1 shows the performance of the recommendation models on the public leaderboard. Neural-based recommender systems reach higher MRR values than similarity-based models at the expense of longer training times. The GRU-based model provided the best MRR performance in the group. Interestingly enough, however, the UCF achieved performance comparable to the neural recommender systems while training for a significant shorter time using a CPU.

Ensembles. Table 2 provides an overview of the performance of re-ranking the candidates of different ensembles with LightGBM-Rankers. The re-ranking of candidates originated only by the C1 recommenders (see Figure 1) does not offer any improvement over a single UCF recommender. However, introducing neural-based candidates leads to an overall boost in MRR. To further improve the objective metric and have better generalization capabilities, we average the results provided by ten different LightGBMRankers. Finally, we improved the diversity of the available candidates by training more models using *URM_W* and *URM_WW* and adding them to the ensemble, thus providing a further boost in performance. Our final model scored *0.1994 MRR* in the private leaderboard, proving how our solution is not overfitting and provides good generalization on unseen data.

Computational considerations. The solution we proposed leverages a noticeable amount of recommenders: 73 candidate selectors and 10 rankers. However, such a solution has been carefully crafted to achieve competitive performance, while leveraging the limited amount of computational resources we had access to. Each Candidate Selector specifically models a different facet of the data and with limited memory consumption: in particular, ICF, UCF, ICBF and RP^3_β have a memory consumption below 2GB; the neural models do not exceed 12GB of memory consumption. The rankers exceeded this limit and required approximately 70GB to be trained. However, results show how training times are more than reasonable if we consider that the training has been performed on a laptop CPU.

[5] Free resource available on Google Colab

7 CONCLUSIONS

The ACM RecSys Challenge 2022 tested the ability to predict the most likely item bought by a user given the list of previously observed items in the context of a session. We crafted several features that summarized the session properties and the navigation patterns of the user. With scarce resources, we showed how the re-ranking of candidates generated by the combination of apparently simple models reaches results close ($\sim 7.6\%$) to the top-scoring solutions.

REFERENCES

[1] Christopher Burges, Robert Ragno, and Quoc Le. 2006. Learning to rank with nonsmooth cost functions. *Advances in neural information processing systems* 19 (2006).

[2] Luca Carminati, Giacomo Lodigiani, Pietro Maldini, Samuele Meta, Stiven Metaj, Arcangelo Pisa, Alessandro Sanvito, Mattia Surricchio, Fernando Benjamín Pérez Maurera, Cesare Bernardis, and Maurizio Ferrari Dacrema. 2021. *Lightweight and Scalable Model for Tweet Engagements Predictions in a Resource-Constrained Environment.* Association for Computing Machinery, New York, NY, USA, 28–33. https://doi.org/10.1145/3487572.3487597

[3] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259* (2014).

[4] Colin Cooper, Sang Hyuk Lee, Tomasz Radzik, and Yiannis Siantos. 2014. Random Walks in Recommender Systems: Exact Computation and Simulations. In *Proceedings of the 23rd International Conference on World Wide Web* (Seoul, Korea) *(WWW '14 Companion).* Association for Computing Machinery, New York, NY, USA, 811–816. https://doi.org/10.1145/2567948.2579244

[5] Nick Craswell. 2009. *Mean Reciprocal Rank.* Springer US, Boston, MA, 1703–1703. https://doi.org/10.1007/978-0-387-39940-9_488

[6] Michal Daniluk, Jacek Dabrowski, Barbara Rychalska, and Konrad Goluchowski. 2021. Synerise at RecSys 2021: Twitter User Engagement Prediction with a Fast Neural Model. In *RecSysChallenge '21: Proceedings of the Recommender Systems Challenge 2021* (Amsterdam, Netherlands) *(RecSysChallenge 2021).* Association for Computing Machinery, New York, NY, USA, 15–21. https://doi.org/10.1145/3487572.3487599

[7] Chris Deotte, Bo Liu, Benedikt Schifferer, and Gilberto Titericz. 2021. GPU Accelerated Boosted Trees and Deep Neural Networks for Better Recommender Systems. In *RecSysChallenge '21: Proceedings of the Recommender Systems Challenge 2021* (Amsterdam, Netherlands) *(RecSysChallenge 2021).* Association for Computing Machinery, New York, NY, USA, 7–14. https://doi.org/10.1145/3487572.3487605

[8] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[9] Bibek Paudel, Fabian Christoffel, Chris Newell, and Abraham Bernstein. 2016. Updatable, Accurate, Diverse, and Scalable Recommendations for Interactive Applications. *ACM Trans. Interact. Intell. Syst.* 7, 1, Article 1 (dec 2016), 34 pages. https://doi.org/10.1145/2955101

[10] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to recommender systems handbook. In *Recommender systems handbook.* Springer, 1–35.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[12] Maksims Volkovs, Felipe Perez, Zhaoyue Cheng, Jianing Sun, Sajad Norouzi, Anson Wong, Pawel Jankiewicz, and Barum Rho. 2021. User Engagement Modeling with Deep Learning and Language Models. In *RecSysChallenge'21: Proceedings of the Recommender Systems Challenge 2021.* 22–27.

[13] Yu Zhang and Qiang Yang. 2021. A Survey on Multi-Task Learning. *IEEE Transactions on Knowledge and Data Engineering* (2021), 1–1. https://doi.org/10.1109/TKDE.2021.3070203