Generator-Ranker model on MeLi Challenge 2020

Ramos-Gomez, Matías
mframos3@uc.cl

Pérez-Facuse, Jorge
jiperez11@uc.cl

ABSTRACT

The MeLI Challenge 2020 was a recommendation system challenge aimed at people from Latin America. Given a week of a user's navigation records, the challenge consists in recommending the ten most likely items of the next purchase. In this paper, we present our solution to this challenge, which consist on a generator-ranker model that predict the item that the user will buy on its next purchase based on the previous interactions. The model works by generating candidates that are likely to be purchased, and then this candidates are ranked by the ranker, which is a sequential recommendation system. We compare our model with different approaches to solve this challenge, and discuss the development process of the solution. We also highlight several insights that we found while participating on the challenge.

Keywords

Sequential recommendation, Candidate ranker architecure

1. INTRODUCTION

Mercado Libre is an e-commerce and fintech company in Latin America, based on a platform where users buy, sell, advertise, deliver, fund and pay goods and services through the Internet. Mercado Libre is building an entrepreneurial ecosystem that is democratizing trade, money and payments, empowering millions of people in Latin America.

In order to promote development on Machine Learning in the continent, since 2019 Mercado Libre is hosting a yearly challenge, with a validated data set and an actual business problem to solve, so ML practitioners can use real life examples to learn and test their results.

Mercado Libre (MeLi) Challenge 2020 is about building a ML model to predict next purchase based on the user's navigation history. Mercado Libre's platform hosts millions of products and service listings. Users navigate through them on countless occasions, looking for what they want to buy or just to get ideas on a new purchase. Understanding better what they might be interested in it's critical to the business, so Mercado Libre's platform can help them find what they're looking for by providing customized recommendations based on the listings that are most suitable for each user's needs and preferences.

Given a week of a user's navigation records, the challenge consists in recommending the ten most likely items of the next purchase.

2. MELI DATASET

MeLi provides three datasets for the challenge. The first dataset is the train dataset, which contains the user's history of navigation one week prior up to two hours before a purchase. It also provides the item that the user finally bought. The user's navigation records consists of 2 types of interactions:

- **View**: This interaction registers the ID of an item whenever the user visits the website of the item.
- **Search**: This interaction registers the text query of the search bar whenever the user uses it.

Additionally, each interaction includes a timestamp that indicates when the interaction was made. An example of a session can be seen on Figure 1.

The second dataset is the test dataset. This dataset is similar to the train dataset, but sessions do not include information about the item bought. This dataset is used to generate the recommendation submissions to the MeLi challenge platform.

The third dataset contains additional data about the items. It includes the title of the item, the price, the domain, the category, and other less relevant data. Among this features, the domain is specifically important because the metric used to rank submissions in the challenge assigns score to items that match the domain of the target item. The domain groups items that are similar to each other, but it has no explicit relationship with the category of items.

3. RELATED WORK

3.1 General Recommendation

General Recommendation aims to deliver users a subset of items from the system that are most likely to be preferred by them. In order to execute this task, several approaches have been developed. Collaborative Filtering (CF) is an approach that models users' preferences based solely on historical interactions and it has a wide use in the field. Early on,

```
{'user_history': [
        {'event_info': 2443411,
         'event_timestamp': '2019-09-27T10:43:47.778-0400',
         'event_type': 'view'},

        {'event_info': 2443411,
         'event_timestamp': '2019-09-27T10:47:21.195-0400',
         'event_type': 'view'},

        {'event_info': 'NUMEROS RESIDENCIAIS INOX',
         'event_timestamp': '2019-09-27T10:47:43.019-0400',
         'event_type': 'search'},

        {'event_info': 'NUMEROS RESIDENCIAIS INOX 2',
         'event_timestamp': '2019-09-27T10:48:45.530-0400',
         'event_type': 'search'},

        {'event_info': 522000,
         'event_timestamp': '2019-09-27T10:49:19.537-0400',
         'event_type': 'view'}
    ],
'item_bought': 2659106}
```


Figure 1: Example of a user's session of the train dataset

two methods of CF were used. This methods are memory based and model based. Memory based CF uses interaction data to compute similarity between users (user-user) or items (item-item). Model based methods such as Matrix Factorization (MF) use latent factor models to learn user and item representations. Recently, with the popularization of deep learning, Neural Collaborative Filtering (NCF) has been developed, which estimates user preferences with the usage of Multi Layer Perceptrons (MLP).

3.2 Sequential Recommendation

Sequential Recommendation fulfills the same task as before but it takes in consideration the order of the user's interaction. Generally this order is chronological. RNN models are the popular choice for recommendations as it suits the sequential structure of the user's interactions. Essentially, these models encode the user's interactions into a vector, with recurrent architectures. For instance, GRU4Rec utilizes Gated Recurrent Units with ranking loss to predict the next item in the sequence. With the rise of attention mechanisms in NLP for sequential text data, recommender models such as SASRec have been developed. SASRec is a left-to-right unidirectional sequential recommendation model architecture based on Transformers. Models that are based on Transformers have outperformed models based on GRU and LSTM such as GRU4Rec.

Another model based on transformers used for recommendation is BERT4Rec, which stands for Bidirectional Encoder Representations from Transformer. This model is similar to SASRec but it employs bidirectional sel

4. SOLUTION

First, it is necessary to define the problem to be solved.

4.1 Problem Statement

Let $U = \{u_1, u_2, ..., u_{|U|}\}$ denote a set of user sessions, $V = \{v_1, .., v_{|V|}\}$ denote the set of items and the set $S_u = \{s_1^u, ..., s_t^u, ..., s_{n_u}^u\}$ denotes the interactions of the user session u sorted in chronological order, where s_t^u denotes the interaction t of the user session u, n_u denotes the length of the interaction sequence for user u, and $j_u \in V$ denotes the item that the user u bought. Given a certain S_u, we aim to predict the item bought j_u. This can be formalized as modeling the probability over all items v being the item bought j_u for a certain user u:

$$p(j_u = v|S_u)$$

This problem is specific to MeLi Challenge, but it can be generalized as a sequential recommendation problem where we aim to predict the next item that the user interacts, based on its previous interactions as j_u is the last interaction of the sequence.

4.2 Solution architecture

The solution that we propose is a model which consists of a candidate generator and a candidate ranker. For each user sequence S_u, k items are retrieved from the item corpus V using a generator G based on an ensemble of baselines. Afterwards, the k items are ranked by a ranker R that consists of a sequential based recommendation system, using as input the sequence S_u.

$$PossibleItems = G(S_u, k) \tag{1}$$

where $|PossibleItems| = k$. Then, recommendations are extracted with:

$$Predictions = R(S_u, PossibleItems) \tag{2}$$

where $Predictions$ is the set of generated items sorted by the probability of being the target item j_u.

4.3 Candidate Generator

The candidate generation strategy uses three baselines sequentially to select up to k items. If one baseline does not provide enough items, the next baseline is used.

$$b_1 \longrightarrow b_2 \longrightarrow b_3$$

- b_1: The first method in the series selects the items visited by the user with priority on the latest views.
- b_2 : For the second method, the frequencies of the pairs item view - item bought are stored. Afterwards, the method selects the items bought that are paired with items that the user has viewed in its session. The baseline prioritizes the selection by the highest frequency of a pair.
- b_3 : The last method selects items from the domains of the items viewed by the user. It prioritizes the domain most visited by the user. Then it selects the items of the selected domain prioritizing the top selling ones.

The item selection stops at any given method if the k items are successfully retrieved.

4.4 Candidate Ranker

SASRec [1] is used as the candidate ranker. SASRec stands for Self-Attention Sequential Recommender Model, and as its name implies, uses the attention mechanism to generate recommendations of the next interaction based on the previous ones of a given user. The architecture of the

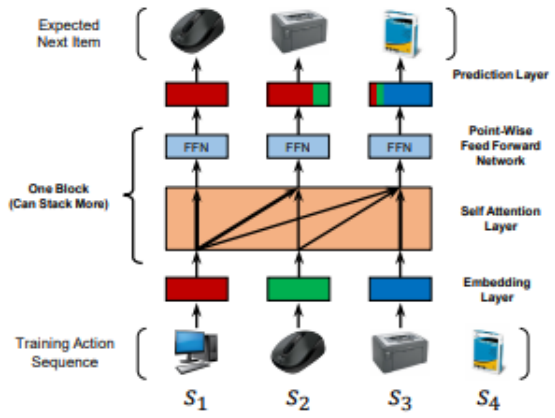

Figure 2: Architecture of the SASRec model. Extracted from [1]

model consists of an embedding layer, L transformer [4] layers, and a prediction layer. The input of the embedding layer is a sequence s of length n, where n is the maximum sequence length that the model can handle. The embedding layer also adds a positional embedding to keep in track the temporal sequence of the interactions. After the embedding layer, L transformer layers are stacked and then the final output is given to the prediction layer, that uses a MF layer to predict the relevance of an item v. The ranker architecture is illustrated on Figure 2

Transformer layers only pay attention to the items that are before in the sequence. Other attention based sequential recommendation models such as [3] also consider the posterior items, but we did not use this model because there where not good implementations available.

5. METHODOLOGY

5.1 Data Used

All three provided datasets by MeLi were used to train and test the models. In order to train SASRec with the train dataset, sequences were created by selecting only the "view" interactions, sorting them by chronological order, and then adding the ID of the item bought at the end of the sequence. Finally, as SASRec requires a fixed length for the sequences, they were either truncated or extended to set their length to n, where n is the maximum length that the model supports. To truncate them, the latest interactions were kept, and to extend them, items were added as padding. In some experiments, the sequences of the test dataset were also used to train thee model. The same pre-processing as the train dataset was applied but without adding the item bought at the end of the sequence because there is no information of the item bought on the test dataset. After all the pre-processing, some statistics of the datasets are presented on Table 1. Do note that the average sequence length on the table is without truncating or adding padding.

As for the items dataset, we only used the information of the domain of each item to build the candidate generator, but it was not used on the training of the candidate ranker.

5.2 Development Process

Table 1: Dataset Statistics

Train set sessions	413163
Test set sessions	177070
Number of items	2102277
Number of domains	7894
Average sequence length	16.62

Due to the nature of the challenge, we had limited time and limited number of submissions per day to try different models and create a possible solution to the challenge. Because of that, the development of our solution was not a linear process but a result from an iterative process through a whole week of work. In this subsection, we detail the three iterations on our work to create the final solution presented in the previous section.

5.2.1 *First Iteration*

On the first iteration of work, we had to get familiar with the challenge, the data, and the possible methods to solve the problem. This iteration was focused on researching both papers and implementations of recommendation systems that could help us solve the problem, with special emphasis on sequential recommendation systems because of the sequential nature of the data. In this iteration we tried using RecBole [6] to test SASRec and other sequential models on the MeLI dataset, but we found that the library was limited in extracting new recommendations, and also the training time was significant, so we failed to properly test any models using that library. We also used an open source implementation [1] of chainRec[5] to get recommendations and make our first submission on the MeLI platform, but the score that we obtained was not competitive enough.

5.2.2 *Second Iteration*

In this iteration, we researched alternative methods to solve the problem. We explored and tested the baselines presented on section 4.3, that yield recommendations based on fixed rules. We found ou We also tried using association rules between domains using the library mlxtend [2]. The baselines performed very well, but the association rules were really bad so we discarded that approach. We also tried to use association rules between items, but due to memory problems we weren't able to make them work at all.

We also tried another implementation[2] of SASRec created using PyTorch, and this time we were able to train the model properly and extract recommendations to make submissions. We trained the model using only the train dataset and also using both the train and test dataset. Our results were better when using both datasets, so we maintained that on our future experiments.

5.2.3 *Final Iteration*

On our final iteration we focused on developing our final model using the generator-ranker architecture that we proposed on section 4. We fine-tuned the parameters of the generator and the ranker to achieve the highest score possible on the challenge. We will detail more the parameters tuning on the next subsection.

[1]https://github.com/MengtingWan/chainRec
[2]https://github.com/pmixer/SASRec.pytorch

5.3 Parameters tuning

To achieve the highest rank on the challenge leaderboard, we fine-tuned the hyper-parameters of both the candidate generator and the candidate ranker. On the candidate ranker, we tried changing the maximum length of the sequence n, the number of transformer layers L and the number of attention heads on the transformer layers h. Each set of hyper-parameters was trained for 50 epochs, and them we tested them locally using a split of the train dataset (this split was created leaving the two last interactions as validation and testing sets). When we found models with high values on the challenge metric, we trained them again using both the train and test datasets, and using all the sequence without leaving a part for validation or testing. On our experiments we found that the max length of the sequence parameter performed better when keeping it at 20, so in the evaluation we only used that value.

To train the all the SASRec models (both the stand-alone versions and the models used as rankers in our solution), we used the same loss functions and training method that the original authors of the SASRec papers used.

As for the candidate generator, different combinations of baselines were tested. In order to test the combinations a custom metric was defined. The metric calculates the iDCG@10 obtainable with the selected candidates. This conducts the search for a candidate generator that focuses on retrieving relevant items without taking in account the order of the selection. The metric is an upper bound for the obtainable nDCG@10 of the whole candidate ranker model (maximum nDCG@10 after ranking). All combinations of sequences of baselines were tested. As for the number of items to select k, the greater this parameter, the higher the score on the custom metric but it will be harder for the ranker model to output the final 10 recommendations in order. Taking this in consideration, values k=20, k=50 and k=100 were tested.

6. EVALUATION AND RESULTS

To evaluate our models, we tested them both locally using a split of the train dataset and using a submission to the MeLI platform, that ranked 30% of the prediction made on the test dataset. The submission format was 10 recommendations of each of the sessions on the test dataset, so only the test dataset was evaluated when making submissions. On table 2 are detailed the results of all the submissions that we made to the MeLI platform. Do note that the NDCG@10 on the table is the modified NDCG@10 that MeLI uses on the challenge, and the generator and ranker on the table are the ones detailed on section 4.

Table 2: Submission Results

Model	NDCG@10
chainRec	0,079
SASRec 1 epoch (only train data)	0,080
SASRec 30 epochs (only train data)	0,090
SASRec 50 epochs (train and test data)	0,147
Baseline: Last viewed items	0,237
Generator only	0,238
Generator k=20 + Ranker L=4 h=4	0,252
Generator k=20 + Ranker L=3 h=1	**0,267**
Generator k=100 + Ranker L=3 h=1	0,239

It is also worth noting that on the submissions of the normal SASRec model, we predicted the probability for all the items present on the dataset, and that not only affected the metrics value, but also the time that it took on generating the recommendations. Generating the recommendations for all the sessions on the test dataset took 9 hours aprox on all the SASRec models, while when using SASRec as a ranker the model only had to predict the probability for 20-100 items for each session, which took considerably less time, being only 30 minutes at most. As for the training time of SASRec, each epoch took around 5 minutes to be completed.

7. DISCUSSION

In this section we will discuss the results obtained, and write some important insights that we found while participating on the challenge.

First, as we can see on the results presented on Table 2, our generator-ranker solution performed consistently better that all the others methods that we tested. That is to be expected, because the generator was based on the best combination of the baselines, and the baselines already outperformed the other methods like SASRec and chainRec. The best set of hyper-parameters of our model was with 20 generated candidates, 3 transformer layers and 1 attention head, which placed us on rank 15 of the leaderboard of the challenge. If we compare the three submissions of our model, first we can see there is a trade-off in the number k of candidates. This is because although a fewer number of candidates means that the probability of the target item j_u being one of the candidates is lower, this makes the task of the ranker easier, as it has less items to rank. We can also note that the best configuration of the SASRec model was when using 3 transformer layers and one attention head. This configuration was better than 4 layers and 4 attention heads, and the reason of that is mentioned on the original SASRec paper: for short sequences, models with fewer layers and attention head perform better. The MeLI dataset has a short mean sequence length (compared to other the other datasets tested on the original SASRec paper), so it is expected that our best model was the one with less layers. Finally, we have to mention that we were not able to test more configurations of the ranker because we had limited submissions per day (only 3 submissions per day per account) and we developed this solution on the final days of the challenge, so we couldn't evaluate more configurations.

We also noted importance of training SASRec on both train and test datasets. If we check the versions of SASRec trained with only the train dataset, both of them performed very bad compared to the other models. In comparison, the SASRec model trained on both datasets performer much better. Although that model was also trained for more epochs, the main factor of the improvement was without a doubt the extra data, because around the epoch 30 the training loss no longer diminished. The main reason of this is because the actual sequences used to generate the submissions were the ones on the test dataset, so it makes sense that the model will be able to predict better for those sequences if they were seen before in the training. That is the main reason of why we used both datasets when training all the rankers used in our final model.

Another thing worthy of discussion is why SASRec performed better after selecting some candidates than ranking all the possible items on the dataset. We believe that the

main reason of this is because of the way SASRec is trained: on each iteration, the model is asked to predict the next item from a set composed of the next item and some random items used as negative sampling. Although this makes the training much faster than just predicting for all the other items, this makes the model unprepared to actually discriminate the correct item from the whole item corpus, because in the whole training the model only had to select the next item from a small set, and not from all the items present in the dataset. That is why if we make predictions for a smaller set (like the generated candidate set), the model will be able to perform better as it will be a situation similar to the training

Although we explained why the generator-ranker architecture works better than just using SASRec on all items, we could have further improved our models if we changed a little the negative sampling strategy on the training. Because the negative sampling just selected random items, it was easier for the model to select the next item on the training phase because, apart from the correct item, all the other items had almost no relationship with the previous sequence because they were simply randomly sampled. But when we use the items from the generator, all these candidates are closely related to the input sequence, simply because they come from the different baselines detailed previously. As such, it is expected that is a harder task to identify the next item from a set of closely related items than from a set of just random items and a correct item. To solve this problem, one alternative would be changing the negative sampling strategy and instead of just sampling random items, it could sample items from the generator. That way, the ranker would be prepared to select the correct item from a closely related. Unfortunately, we were not able to test this negative sampling method because the challenge ended before we could implement this.

Finally, we want to mention some of our early mistakes that we made while facing the challenge. Learning from this mistakes could be useful when facing challenges similar to the MeLI challenge.

- One of our biggest mistakes was trying to use RecBole to generate the recommendations. This is because although the possibility of testing various models with the same data interface seems attractive, the library was still very new and because of that was very limited when extracting new recommendations, had few examples and resources on how to use it and trained to slow to be usable. We lost a lot of time because of this, and finally we had to abandon the idea of using the library because we couldn't make it generate the submissions that we needed.

- Trying to use chainRec was another of our mistakes. The problem with chain rec was that it was not suited for the MeLI challenge dataset. This was because chainRec is used for data that has a monotonic behavior, like interaction sequences, but not actually on sequential data. For example, the authors use it on a dataset that has a monotonic interaction chain of "click", "purchase", "review", "recommend". That means that the interaction of a user with an item follows in a monotonic way the described sequence. It does not take in account the sequence between different items, and because of that it was not a model useful for us. Furthermore, the MeLI dataset was not monotonic, because there wasn't a clear order of the interactions (sometimes, the item boughts were never seen on the user history, and because of that we can't say that the "view" interactions always come before buying the item). All these reasons make chainRec unsuited for this challenge, and we realized this to late as we had already wasted time training the model and generating predictions.

- Another mistake was trying to use association rules on this challenge. This is because the number of items was huge, and the association rules didn't fit on the RAM. In fact, all memory based methods where unsuited for this challenge, because the amount of items was too big to fit on the memory. As for the association rules between different domains, they were to weak to be of any use on the challenge.

8. CONCLUSION

In this paper, we presented our solution to the MeLI challenge 2020, and we analyzed both our process while developing this solution and the results that we obtained. Although we didn´t won the challenge, it was a nice experience where we learned how it was to compete on recommendation systems challenges, and all the difficulties that comes with trying to create a solution for a problem on a very limited time. We also had the opportunity to experiment with a state-of-art sequential recommendation system like SASRec, and learn how these models are trained and then used on real problems.

9. REFERENCES

[1] W. Kang and J. J. McAuley. Self-attentive sequential recommendation. *CoRR*, abs/1808.09781, 2018.

[2] S. Raschka. Mlxtend: Providing machine learning and data science utilities and extensions to python's scientific computing stack. *The Journal of Open Source Software*, 3(24), Apr. 2018.

[3] F. Sun, J. Liu, J. Wu, C. Pei, X. Lin, W. Ou, and P. Jiang. Bert4rec: Sequential recommendation with bidirectional encoder representations from transformer. *CoRR*, abs/1904.06690, 2019.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008. Curran Associates, Inc., 2017.

[5] M. Wan and J. McAuley. Item recommendation on monotonic behavior chains. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 86–94, 2018.

[6] W. X. Zhao, S. Mu, Y. Hou, Z. Lin, K. Li, Y. Chen, Y. Lu, H. Wang, C. Tian, X. Pan, Y. Min, Z. Feng, X. Fan, X. Chen, P. Wang, W. Ji, Y. Li, X. Wang, and J.-R. Wen. Recbole: Towards a unified, comprehensive and efficient framework for recommendation algorithms. *arXiv preprint arXiv:2011.01731*, 2020.