# Generator-Ranker model on MeLi Challenge 2020

Ramos-Gomez, Matías
mframos3@uc.cl

Pérez-Facuse, Jorge
jiperez11@uc.cl

## ABSTRACT

MeLi Challenge 2020 is a recommendation challenge for Latin American participants. The goal is to build a model that yields the best ten item recommendations given one week of user's historical interactions in the Mercado Libre's platform. In this paper, we present our solution to this challenge, which consists of a generator-ranker model that predicts the item that the user will buy on its next purchase based on its previous interactions. The model works by generating candidates that are likely to be purchased, which are later ranked by a sequential recommendation model. 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.

## 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 [1].

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

## 2. DATASET AND SCORING

### 2.1 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 two 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.

### 2.2 Submission scoring

Submissions are scored using average NDCG with custom relevance function:

$$DCG = \sum_{pos=1}^{n} \frac{relevance_{pos}}{\log(1+pos)} \tag{1}$$

$$relevance(y,\hat{y}) = \begin{cases} 12 & if \quad y = \hat{y} \\ 1 & if \quad domain_y = domain_{\hat{y}} \\ 0 & otherwise \end{cases} \tag{2}$$

If the recommendation hits the correct item ID of the user session's target purchase, 12 points of relevance are assigned. Instead, if the item does not match the correct item ID but it does correspond to the same domain, 1 point is assigned. In

```
{'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**

any other case the relevance score is 0. All relevance scores are discounted by position as presented in equation 1.

The ideal recommendation set will be that comprised of the same item ID as the target in the 1st position, followed by 9 item IDs belonging to the same domain ID as the target item in the remaining recommendations.

## 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 [2]. Early on, 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) [4].

### 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 [3]. 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 [5].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 self-attention [7].

## 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 $S_u = \{s^u_1, ..., s^u_t, ..., s^u_{n_u}\}$ denotes the interactions of the user session $u$ sorted in chronological order, where $s^u_t$ 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 $u$:

$$p(j_u = v|S_u) \tag{3}$$

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 $S_u$.

$$PossibleItems = G(S_u, k) \tag{4}$$

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

$$Predictions = R(S_u, PossibleItems) \tag{5}$$

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

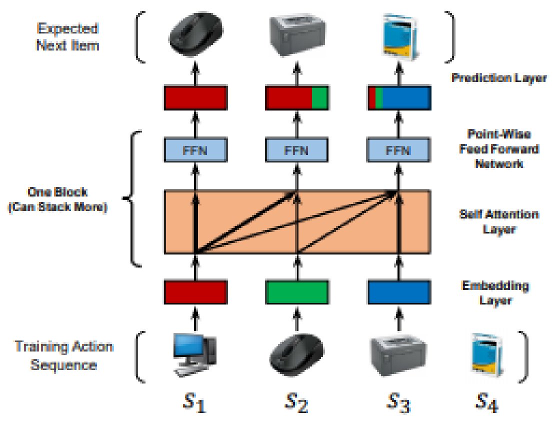

**Figure 2: Architecture of the SASRec model. Extracted from [5]**

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 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 model consists of an embedding layer, $L$ transformer [8] 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 BERT4Rec 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.

**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 |

### 5.2 Development Process

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 Python library [10] to test SASRec and other sequential models. The library is new (first released in Nov 3, 2020) and does not have a flexible interface yet. Extracting recommendations from trained models was limited and training time was too long, this did not allow us to properly test any model with a ranked sumbission on the MeLi platform.

We also used an open source implementation [1] of chainRec[9] 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*

[1]https://github.com/MengtingWan/chainRec

In this iteration, we researched alternative methods to solve the problem. We explored and tested the baselines presented on section 4.3, which yield recommendations based on fixed rules. We also tried using association rules between domains using the library mlxtend [6]. The baselines performed very well, but the association rules did no so we discarded them as a possible approach. Association rules between items could not be obtained due to memory limitations.

We tried a PyTorch implementation[2] of SASRec, 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 in section 4.
We fine-tuned the parameters of the generator and the ranker in order to get the most out of the architecture. The generator was modified and the ranker's parameters tuned to maximize the NDCG score for the submission. More about the settings and parameter tuning of model will be detailed in the next subsection.

### 5.3 Configuration and parameters tuning

To achieve the highest rank on the challenge leaderboard, we explored different candidate generator configurations and fine-tuned the hyper-parameters of the candidate ranker.

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 ten recommendations in order.

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, 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.

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

To train all SASRec models (both the standalone 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.

## 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, which ranked 30% of the prediction made on the test dataset. The submission consists of ten recommendations for each of the sessions on the test dataset, so only the test dataset was evaluated when making submissions. On table 2 scores of all the submissions that we made to the MeLi platform are detailed. 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 $b_3$ | 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 worth noting that on the submissions of the normal SASRec model, we predicted the probability over all the items present on the dataset. This not only affected the metrics value, but also it increased the time that it took on generating the recommendations. Generating the recommendations for all the sessions on the test dataset took approximately 9 hours 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 present 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 other methods like SASRec and chainRec. The best settings of our model consists of a candidate generator with $b_1 \longrightarrow b_2 \longrightarrow b_3$ configuration that yields $k = 20$ candidates and the SASRec candidate ranker with 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. This is explained on the original SASRec paper: for short sequences, models with fewer layers and attention head perform better. MeLi dataset has a short mean sequence length (compared to other the other datasets tested on the original SASRec paper), so it is expected for a configuration with less layers to perform better.

It is important to mention that we were not able to test more configurations for the ranker due to the limitation of daily submissions (only 3 submissions per day per account) as we developed this solution on the final days of the challenge.

We also noted the importance of training SASRec on both train and test datasets. The two SASRec submissions trained with only the train dataset, performed poorly compared to the other models. In comparison, the SASRec model trained on both datasets performed much better. Although that model was also trained for more epochs, the main factor of the improvement was the extra data, because around epoch 30, training loss decrement stabilized. The importance of the incorporation of the sequences of the test dataset on training is that those sequences are the ones the model will recommend for. The model will be able to predict better for those sequences if they were during training. This the main reason of why we used both datasets when training all the rankers used in our final model.

Another aspect worth noting is that SASRec performed better after selecting some candidates rather 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 withdrawn by 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. This is because in the whole training stage, 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 similar to its training process.

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 modify the negative sampling strategy. The negative sampling used just selected random items. The model had to discriminate the best item among items that mostly had no relationship with the previous sequence due to its sampling method. On the other hand, on the ranking of candidates given by the generator, all items are strongly related to the input sequence, making it hard for the model to distinguish the correct item as it only learned to perform the task in an easier situation (one correct item and randomly selected items). It is necessary to make the ranking in the training stage as similar as possible to the real scenario. 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 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 advantageous, the library was still very new, thus its interface was not developed enough which made extracting new recommendations a hard task. It also had few examples and resources on how to use it and it was too slow to be used in a challenge context were the need of fast testing of models is a priority. We lost a significant amount of time trying to use the library, and moved on search of other tools.

- Trying to use chainRec was another of our mistakes because it does not suit the challenge data. chainRec is a model that unifies the spectrum of implicit and explicit user feedback on a monotonic behavior where any signal necessarily implies the presence of a weaker (or more implicit) signal. Example of an interaction monotonic chain is ”click” < ”purchase” < ”review” <”recommend” were ”review” action implies a ”purchase” action, which implies a ”click” action. The model does not consider the chronological order of interactions and MeLi dataset’s interactions ”view” - ”purchase” are not monotonic. In many sessions the item bought was never seen on the user history (there is a two hour window between the last interaction record and the purchase of an item on every session). We realized how unsuited was chainRec for the challenge after working with it. The time wasted with this model could have been used to further improve the candidate ranker model.

- Another mistake was trying to use association rules on this challenge. This is because the size of the item corpus was large, and the association rules didn’t fit on the RAM. In fact, all memory based methods where unsuited for this challenge as they would not fit in memory. As for the association rules between different domains, they were too weak in terms of support 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 an experience which allowed us to learn more about recommendation on a competitive setting with real data. Every strategy that we chose to develop meant a significant investment on time which was limited. 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 used on real data.

## 9. REFERENCES

[1] Meli challenge 2020. https://ml-challenge.mercadolibre.com.

[2] R. Chen, Q. Hua, Y. Chang, B. Wang, L. Zhang, and X. Kong. A survey of collaborative filtering-based recommender systems: From traditional methods to hybrid methods based on social networks. *IEEE Access*, 6:64301–64320, 2018.

[3] H. Fang, D. Zhang, Y. Shu, and G. Guo. Deep learning for sequential recommendation: Algorithms, influential factors, and evaluations, 2020.

[4] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T.-S. Chua. Neural collaborative filtering, 2017.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.