Team

Members of the team: Lyalikov Nikita (@kridr), Aliev Mishan (@MishanAliev) and Iakovleva Tatiana (@bogsolntca).

Problem setup

We took kaggle competition [OTTO – Multi-Objective Recommender System]
The goal is to train a model and predict items for users for different targets (clicks, carts, orders).

Our goal is to:

1) Try out GNN model for recommendations and compare it with strong baselines
2) Try out different hypothesis to use information about some targets to predict other targets so that we will see gain in metrics

Methodology

Scenario

Warm-start (we only make predictions for users with some interactions)

Preprocessing data

Initial data is stored in json format and is of size 11 Gb. Every user has a list of interactions of three possible types and timestamps. Our goal is

1) Take a chunk of data. Chunk is some users + all their interactions
2) Explode data, so that we have pairs of users and items
3) Filter data based on precalculated timestamps
    a) 0.4 to 0.6 quantile for train
    b) 0.6 to 0.7 quantiles for test
4) Filter data based on type of interactions
5) Save three prepared chunks into partitioned parquet files
6) Repeat 1-5 long enough. We were not able to perform this process for all chunks of users, because of long calculations, but enough users were taken.

Holdout construction

We used a test for taking holdout and did not use test for anything else.

We took the first 5 interactions of a user for holdout. If a user did not have at least 5 interactions, then we padded it.

Metrics

We considered three metrics

- HR@200
- MRR@200
- Recall@200

We predicted 200 candidates for each user, because we do not consider provided models as final, but as a first-stage model in a two-stage recommender system. So we expect, that in reality some reranker is applied after initial candidate retrieval (that is provided with proposed models)

Metrics are calculated for users that are both in train and test

Models

LightGCN

LightGCN is SOTA in GNN-based recommendations at the moment of publication.

LightGCN, includes only the most essential component in GCN — neighborhood aggregation — for collaborative filtering.

Specifically, LightGCN learns user and item embeddings by linearly propagating them on the user-item interaction graph, and uses the weighted sum of the embeddings learned at all layers as the final embedding

The whole process can be expressed via such equations

$$\mathbf{e}_u^{(k+1)} = \sum_{i \in \mathcal{N}_u} \frac{1}{\sqrt{|\mathcal{N}_u|}\sqrt{|\mathcal{N}_i|}} \mathbf{e}_i^{(k)},$$

$$\mathbf{e}_i^{(k+1)} = \sum_{u \in \mathcal{N}_i} \frac{1}{\sqrt{|\mathcal{N}_i|}\sqrt{|\mathcal{N}_u|}} \mathbf{e}_u^{(k)}.$$

Where N_i, N_u are neighborhoods of items or users
Model is trained using BPR loss, which is a pairwise loss that encourages the prediction of an observed entry to be higher than its unobserved counterparts.

Only trainable parameters are embeddings at the 0-th level

Implementation from PyTorch Geometric was used

Baselines

Alternating Least Squares

At the planning stage it was assumed that the PureSVD method would be used, but during work it turned out that this was very computationally expensive: to get recommendations we need to multiply matrix 1 with shape (304164, 200) and 60832800 non-zero elements and matrix 2 with shape (200, 680757) and 136151400 non-zero elements, which requires O(60832800*136151400*680757)=O(10^21) elementary operations that will take approximately 10^13 seconds or 316 887 years on an average laptop. So, I decided to switch this method to ALS.
Alternating Least Squares (ALS) is an efficient matrix factorization technique for collaborative filtering in recommender systems. Its decomposes a user-item interaction matrix (e.g., ratings matrix) into two lower-dimensional factor matrices: 1) user matrix (U) that represents users in a latent factor space, capturing their latent preferences and tendencies; 2) item matrix (V) that represents items in the same latent factor space, capturing their latent attributes and features. Now, if we know some user representation u_i, we can get a prediction with just simple multiplication V @ u_i.
3 different models were tested:

1) Standart ALS without changing default parameters (ALS_1) - to make sure that the default model is better than the tuned graph model;
2) Standart ALS without changing default parameters and with modified data taking into account timestep (ALS_2) - to try to improve model 1).
3) Standart ALS without changing default parameters and with modified data taking into account timestep but with clamped items count (ALS_3) - to try to recommend items from “long tail” (clip interactions up to 20).

Implementation from the library “implicit” was used.

KNN

In the context of recommendation systems, KNN is often used for user-based collaborative filtering. The idea is to find users who are similar to the target user based on their preferences and recommend items that these similar users have liked. It can be a simple and effective approach for recommendation systems, especially in scenarios where the data is not too sparse, and the user or item similarities play a significant role in determining preferences.
Also tf-idf normalization was considered.

Comparison

	clicks			carts			orders		
	HR @200	MRR @200	Recall @200	HR @200	MRR @200	Recall @200	HR @200	MRR @200	Recall @200
LightGCN	0.0403	0.004	0.0002	0.0592	0.0086	0.0152	0.0577	0.0059	0.0132
Random	0.0008	0.0002	0.0002	0.0022	0.0004	0.0004	0.0024	0.0005	0.0005
ALS_1	0.2355	0.0202	0.0018	0.1049	0.0141	0.0309	0.0616	0.0098	0.0155
ALS_2	**0.302**	0.0251	**0.0024**	**0.1537**	**0.0203**	**0.0469**	**0.1017**	**0.014**	0.026
ALS_3	0.2875	**0.0236**	0.0023	0.1501	0.0198	0.0455	**0.1017**	**0.014**	**0.0261**
KNN	0.0411	0.0049	0.0003	0.0223	0.0032	0.0001	0.0097	0.0014	0.0001
KNN_norm	0.043	0.0056	0.0003	0.022	0.0032	0.0001	0.0095	0.0013	**0.0001**

Additional hypothesis

It is natural to assume that targets are sequential. The sequence goes as follows:

1. Click
2. Add to cart
3. Order

This is what data analysis showed us

	fraction
orders_with_prev_cart / orders	0.776207
orders_with_prev_click / orders	0.434138
orders_with_prev_cart_and_prev_click / orders	0.260690
carts_with_prev_clicks / carts	0.382738

Merging recommendations

The idea is to use other recommendations for enhancing the quality. For example, It is natural to assume that if a user is recommended something to order, then we can use information about recommendations of carts. In other words, if an item is both in order and cart recommendation

lists, that we need to upvote it. That process should be done on initially bigger recommendations lists, because intersections are not likely to happen frequently.

	HR @200	MRR @200	Recall @200
Orders	**0.0577**	**0.0059**	**0.0132**
Orders| Carts	0.0512	0.0056	0.0117

Not confirmed

Transfer of information

During the process of training our first models, all predictable actions were treated separately. In order to fix it, the transfer of information between these 3 datasets was included. It was important here, because we consider only a small period of time, not a whole timeline, and just because of this limitation, we have already missed some patterns. This transfer was organized in a particular way: we included some extra interactions. For example, if a particular user X ordered an item Y, but there was not such a pair in the dataset with carts or a pair in the dataset with clicks, we added this pair into both dataframes.

KNN model	HR @200	MRR @200	Recall @200
Carts	**0.0223**	**0.0032**	**0.0001**
Carts + Orders	0.011	0.0006	0.0001

Not confirmed

Conclusion

- LightGCN is worse than provided baseline, so we should not use it
- Additional hypotheses were not confirmed