Course project

Основное

- Дедлайн - 26 июля 23:59
- Целевая метрика precision@5
- Бейзлайн решения - MainRecommender (https://github.com/geangohn/recsys-tutorial/blob/master/src/recommenders.py)
- Сдаем ссылку на github с решением. В решении должны быть отчетливо видна метрика на новом тестовом сете из файла retail_test1.csv, то есть вам нужно для всех юзеров из этого файла выдать выши рекомендации, и посчитать на actual покупках precision@5.

!! Мы не рассматриваем холодный старт для пользователя, все наши пользователя одинаковы во всех сетах, поэтому нужно позаботиться об их исключении из теста.

Hints:

Сначала просто попробуйте разные параметры MainRecommender:

- N в топ-N товарах при формировании user-item матирцы (сейчас топ-5000)
- Различные веса в user-item матрице (0/1, кол-во покупок, log(кол-во покупок + 1), сумма покупки, ...)
- Разные взвешивания матрицы (TF-IDF, BM25 - у него есть параметры)
- Разные смешивания рекомендаций (обратите внимание на бейзлайн - прошлые покупки юзера)

Сделайте MVP - минимально рабочий продукт - (пусть даже top-popular), а потом его улучшайте

Если вы делаете двухуровневую модель - следите за валидацией

Import libs

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

# Для работы с матрицами
from scipy.sparse import csr_matrix

# Матричная факторизация
from implicit import als

# Модель второго уровня
from lightgbm import LGBMClassifier

import os, sys
module_path = os.path.abspath(os.path.join(os.pardir))
if module_path not in sys.path:
    sys.path.append(module_path)

# Написанные нами функции
from metrics import precision_at_k, recall_at_k
from utils import prefilter_items
from recommenders import MainRecommender
```

Read data

In [2]:

```
PATH_DATA = "../data"
```

In [3]:

```
data = pd.read_csv(os.path.join(PATH_DATA,'retail_train.csv'))
item_features = pd.read_csv(os.path.join(PATH_DATA,'product.csv'))
user_features = pd.read_csv(os.path.join(PATH_DATA,'hh_demographic.csv'))
```

In [4]:

```
data.head(3)
```

Out[4]:

	user_id	basket_id	day	item_id	quantity	sales_value	store_id	retail_disc	trans_time
0	2375	26984851472	1	1004906	1	1.39	364	-0.6	1631
1	2375	26984851472	1	1033142	1	0.82	364	0.0	1631
2	2375	26984851472	1	1036325	1	0.99	364	-0.3	1631

In [5]:

```
item_features[:3]
```

Out[5]:

	PRODUCT_ID	MANUFACTURER	DEPARTMENT	BRAND	COMMODITY_DESC	SUB_COMMOD
0	25671	2	GROCERY	National	FRZN ICE	ICE - CRUSHE
1	26081	2	MISC. TRANS.	National	NO COMMODITY DESCRIPTION	NO SUBCO DES
2	26093	69	PASTRY	Private	BREAD	BREAD:ITALIAN

In [6]:

```
user_features.head(3)
```

Out[6]:

	AGE_DESC	MARITAL_STATUS_CODE	INCOME_DESC	HOMEOWNER_DESC	HH_COMP_DESC
0	65+	A	35-49K	Homeowner	2 Adults No Kids
1	45-54	A	50-74K	Homeowner	2 Adults No Kids
2	25-34	U	25-34K	Unknown	2 Adults Kids

Set global const

In [7]:

```
ITEM_COL = 'item_id'
USER_COL = 'user_id'
ACTUAL_COL = 'actual'

# N = Neighbors
N_PREDICT = 50
```

Process features dataset

In [8]:

```
# column processing
item_features.columns = [col.lower() for col in item_features.columns]
user_features.columns = [col.lower() for col in user_features.columns]

item_features.rename(columns={'product_id': ITEM_COL}, inplace=True)
user_features.rename(columns={'household_key': USER_COL }, inplace=True)
```

Split dataset for train, eval, test

In [9]:

```
# Важна схема обучения и валидации!
# -- давние покупки -- | -- 6 недель -- | -- 3 недель --
# подобрать размер 2-ого датасета (6 недель) --> learning curve (зависимость метрики recall


VAL_MATCHER_WEEKS = 6
VAL_RANKER_WEEKS = 3
```

In [10]:

```
# берем данные для тренировки matching модели
data_train_matcher = data[data['week_no'] < data['week_no'].max() - (VAL_MATCHER_WEEKS + VA

# берем данные для валидации matching модели
data_val_matcher = data[(data['week_no'] >= data['week_no'].max() - (VAL_MATCHER_WEEKS + VA
                        (data['week_no'] < data['week_no'].max() - (VAL_RANKER_WEEKS))]

# берем данные для тренировки ranking модели
data_train_ranker = data_val_matcher.copy()  # Для наглядности. Далее мы добавим изменения,

# берем данные для теста ranking, matching модели
data_val_ranker = data[data['week_no'] >= data['week_no'].max() - VAL_RANKER_WEEKS]
```

In [11]:

```
# сделаем объединенный сет данных для первого уровня (матчинга)
df_join_train_matcher = pd.concat([data_train_matcher, data_val_matcher])
```

In [12]:

```
def print_stats_data(df_data, name_df):
    print(name_df)
    print(f"Shape: {df_data.shape} Users: {df_data[USER_COL].nunique()} Items: {df_data[ITE
```

In [13]:

```
print_stats_data(data_train_matcher,'train_matcher')
print_stats_data(data_val_matcher,'val_matcher')
print_stats_data(data_train_ranker,'train_ranker')
print_stats_data(data_val_ranker,'val_ranker')
```

```
train_matcher
Shape: (2108779, 12) Users: 2498 Items: 83685
val_matcher
Shape: (169711, 12) Users: 2154 Items: 27649
train_ranker
Shape: (169711, 12) Users: 2154 Items: 27649
val_ranker
Shape: (118314, 12) Users: 2042 Items: 24329
```

In [14]:

```
# выше видим разброс по пользователям и товарам и дальше мы перейдем к warm-start (только и
```

In [15]:

```
data_val_matcher.head(2)
```

Out[15]:

	user_id	basket_id	day	item_id	quantity	sales_value	store_id	retail_disc	trans
2104867	2070	40618492260	594	1019940	1	1.00	311	-0.29	
2107468	2021	40618753059	594	840361	1	0.99	443	0.00	

Prefilter items

In [16]:

```
n_items_before = data_train_matcher['item_id'].nunique()

data_train_matcher = prefilter_items(data_train_matcher, item_features=item_features, take_

n_items_after = data_train_matcher['item_id'].nunique()
print('Decreased # items from {} to {}'.format(n_items_before, n_items_after))
```

```
C:\Users\voron\а учеба\рекомендательные системы\cource_project\utils.py:20:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  data['price'] = data['sales_value'] / (np.maximum(data['quantity'], 1))

Decreased # items from 83685 to 5001
```

Make cold-start to warm-start

In [17]:

```
# ищем общих пользователей
common_users = data_train_matcher.user_id.values

data_val_matcher = data_val_matcher[data_val_matcher.user_id.isin(common_users)]
data_train_ranker = data_train_ranker[data_train_ranker.user_id.isin(common_users)]
data_val_ranker = data_val_ranker[data_val_ranker.user_id.isin(common_users)]

print_stats_data(data_train_matcher,'train_matcher')
print_stats_data(data_val_matcher,'val_matcher')
print_stats_data(data_train_ranker,'train_ranker')
print_stats_data(data_val_ranker,'val_ranker')
```

```
train_matcher
Shape: (861404, 13) Users: 2495 Items: 5001
val_matcher
Shape: (169615, 12) Users: 2151 Items: 27644
train_ranker
Shape: (169615, 12) Users: 2151 Items: 27644
val_ranker
Shape: (118282, 12) Users: 2040 Items: 24325
```

Init/train recommender

In [18]:

```
recommender = MainRecommender(data_train_matcher)
```

```
WARNING:root:OpenBLAS detected. Its highly recommend to set the environment
variable 'export OPENBLAS_NUM_THREADS=1' to disable its internal multithread
ing

  0%|          | 0/15 [00:00<?, ?it/s]

  0%|          | 0/5001 [00:00<?, ?it/s]
```

Варианты, как получить кандидатов

Можно потом все эти варианты соединить в один

(!) Если модель рекомендует < N товаров, то рекомендации дополняются топ-популярными товарами до N

In [19]:

```
# Берем тестового юзера 2375
```

In [20]:

```
recommender.get_als_recommendations(2375, N=5)
```

Out[20]:

```
[899624, 1044078, 844179, 832678, 871756]
```

In [21]:

```
recommender.get_own_recommendations(2375, N=5)
```

Out[21]:

```
[948640, 918046, 847962, 907099, 873980]
```

In [22]:

```
recommender.get_similar_items_recommendation(2375, N=5)
```

Out[22]:

```
[1046545, 1044078, 937292, 907099, 15778319]
```

In [23]:

```
recommender.get_similar_users_recommendation(2375, N=5)
```

Out[23]:

```
[1096573, 1133654, 918638, 935578, 1097398]
```

Eval recall of matching

Измеряем recall@k

Это будет в ДЗ:

A) Попробуйте различные варианты генерации кандидатов. Какие из них дают наибольший recall@k ?

- Пока пробуем отобрать 50 кандидатов (k=50)
- Качество измеряем на data_val_lvl_1: следующие 6 недель после трейна

Дают ли own recommendtions + top-popular лучший recall?

B)* Как зависит recall@k от k? Постройте для одной схемы генерации кандидатов эту зависимость для k = {20, 50, 100, 200, 500}
C)* Исходя из прошлого вопроса, как вы думаете, какое значение k является наиболее разумным?

In [24]:

```
result_eval_matcher = data_val_matcher.groupby(USER_COL)[ITEM_COL].unique().reset_index()
result_eval_matcher.columns=[USER_COL, ACTUAL_COL]
result_eval_matcher.head(2)
```

Out[24]:

	user_id	actual
0	1	[853529, 865456, 867607, 872137, 874905, 87524...
1	2	[15830248, 838136, 839656, 861272, 866211, 870...

In [39]:

```
# N = Neighbors
N_PREDICT = 1000
recommend_model= [recommender.get_own_recommendations, recommender.get_similar_items_recomm
                  recommender.get_als_recommendations]
```

In [40]:

```
%%time
def res_eval(df_result, target_col_name, recommend_model, N_PREDICT= 50):
    result_col_name = str(recommend_model).split('_')[1]

    df_result[result_col_name] = df_result[target_col_name].apply(lambda x: recommend_model
    return df_result
```

```
Wall time: 0 ns
```

In [41]:

```
%%time
for i in recommend_model:
    res_eval(result_eval_matcher, USER_COL, i, N_PREDICT)
```

```
Wall time: 1min 39s
```

In [42]:

```
result_eval_matcher[:3]
```

Out[42]:

	user_id	actual	own	similar	als
0	1	[853529, 865456, 867607, 872137, 874905, 87524...	[856942, 9297615, 5577022, 877391, 9655212, 88...	[1135983, 843450, 9420048, 5577022, 939323, 98...	[1097909, 8090541, 1073120, 856942, 962615, 82...
1	2	[15830248, 838136, 839656, 861272, 866211, 870...	[911974, 1076580, 1103898, 5567582, 1056620, 9...	[8090537, 5569845, 1044078, 985999, 880888, 81...	[5569230, 916122, 919534, 866211, 844179, 1029...
2	4	[883932, 970760, 1035676, 1055863, 1097610, 67...	[6391541, 1052294, 891423, 936470, 1137010, 11...	[1038214, 846550, 990762, 999714, 6514160, 847...	[891423, 982790, 6391541, 987044, 5569327, 901...

In [43]:

```
# %%time
# # для понятности расписано все в строчку, без функций, ваша задача уметь оборачивать все
# result_eval_matcher['own_rec'] = result_eval_matcher[USER_COL].apply(lambda x: recommende
# result_eval_matcher['sim_item_rec'] = result_eval_matcher[USER_COL].apply(lambda x: recom
# result_eval_matcher['als_rec'] = result_eval_matcher[USER_COL].apply(lambda x: recommende
```

In [44]:

```
%%time
# result_eval_matcher['sim_user_rec'] = result_eval_matcher[USER_COL].apply(lambda x: recom
```

```
Wall time: 0 ns
```

Recall и Precision

In [45]:

```
# # сырой и простой пример как можно обернуть в функцию
def evalRecall(df_result, target_col_name, recommend_model):
    result_col_name = 'result'
    df_result[result_col_name] = df_result[target_col_name].apply(lambda x: recommend_model
    return df_result.apply(lambda row: recall_at_k(row[result_col_name], row[ACTUAL_COL], k
```

In [46]:

```
# evalRecall(result_eval_matcher, USER_COL, recommender.get_own_recommendations)
```

In [47]:

```
def calc_recall(df_data, top_k):
    for col_name in df_data.columns[2:]:
        yield col_name, df_data.apply(lambda row: recall_at_k(row[col_name], row[ACTUAL_COL
```

In [48]:

```
def calc_precision(df_data, top_k):
    for col_name in df_data.columns[2:]:
        yield col_name, df_data.apply(lambda row: precision_at_k(row[col_name], row[ACTUAL_
```

Recall@50 of matching

In [63]:

```
res_sort = pd.DataFrame()
TOPk = [5, 20, 50, 100, 200, 500]
TOPK_RECALL = 50
```

In [64]:

```
for i in TOPk:
    print(i, sorted(calc_recall(result_eval_matcher, i), key=lambda x: x[1],reverse=True))
```

```
5 [('own', 0.018201887674891018), ('als', 0.013298971088361897), ('similar',
0.006312770546948524)]
20 [('own', 0.039284276793729055), ('als', 0.0293901627902611), ('similar',
0.0181059498144665)]
50 [('own', 0.0652565703814516 5), ('als', 0.048026757227193566), ('similar',
0.03412443794456494)]
100 [('own', 0.09604492955885016), ('als', 0.06913132890602255), ('similar',
0.05389108372213493)]
200 [('own', 0.13537278412833254), ('als', 0.09787044213556072), ('similar',
0.08550390298636432)]
500 [('own', 0.18205324555508703), ('als', 0.146042583729739), ('similar',
0.13586596417817104)]
```

Precision@5 of matching

In [65]:

```
TOPK_PRECISION = 5
TOP_PRECISION = [5, 20, 50, 100, 200, 500]
```

In [66]:

```
for i in TOP_PRECISION:
    print(sorted(calc_precision(result_eval_matcher, i), key=lambda x: x[1],reverse=True))
```

```
[('own', 0.17712691771269176), ('als', 0.12059507205950719), ('similar', 0.0
6620176662017666)]
[('own', 0.10485820548582056), ('als', 0.07812645281264528), ('similar', 0.0
4888423988842399)]
[('own', 0.07247791724779172), ('als', 0.05580660158066016), ('similar', 0.0
3814969781496978)]
[('own', 0.05525801952580195), ('als', 0.04230125523012552), ('similar', 0.0
30660158066015807)]
[('own', 0.04180381218038123), ('als', 0.031011157601115766), ('similar', 0.
0246094839609484)]
[('own', 0.024346815434681545), ('als', 0.019328684332868433), ('similar',
0.017038586703858674)]
```

Ranking part

Обучаем модель 2-ого уровня на выбранных кандидатах

- Обучаем на data_train_ranking
- Обучаем *только* на выбранных кандидатах
- *Я для примера* сгенерирую топ-50 кадидиатов через get_own_recommendations
- (!) Если юзер купил < 50 товаров, то get_own_recommendations дополнит рекоммендации топ-популярными

In [67]:

```
# -- давние покупки -- | -- 6 недель -- | -- 3 недель --
```

Подготовка данных для трейна

In [68]:

```
# взяли пользователей из трейна для ранжирования
df_match_candidates = pd.DataFrame(data_train_ranker[USER_COL].unique())
df_match_candidates.columns = [USER_COL]
```

In [69]:

```
# собираем кандитатов с первого этапа (matcher)
df_match_candidates['candidates'] = df_match_candidates[USER_COL].apply(lambda x: recommend
```

In [70]:

```
df_match_candidates.head(2)
```

Out[70]:

	user_id	candidates
0	2070	[1105426, 1097350, 879194, 948640, 928263, 944...
1	2021	[950935, 1119454, 835578, 863762, 1019142, 102...

In [71]:

```
# разворачиваем товары
df_items = df_match_candidates.apply(lambda x: pd.Series(x['candidates']), axis=1).stack().
df_items.name = 'item_id'
```

In [72]:

```
df_match_candidates = df_match_candidates.drop('candidates', axis=1).join(df_items)
```

In [73]:

```
df_match_candidates.head(4)
```

Out[73]:

	user_id	item_id
0	2070	1105426
0	2070	1097350
0	2070	879194
0	2070	948640

Check warm start

In [74]:

```
print_stats_data(df_match_candidates, 'match_candidates')
```

```
match_candidates
Shape: (2151000, 2) Users: 2151 Items: 4659
```

Создаем трейн сет для ранжирования с учетом кандидатов с этапа 1

In [75]:

```
df_ranker_train = data_train_ranker[[USER_COL, ITEM_COL]].copy()
df_ranker_train['target'] = 1  # тут только покупки

df_ranker_train = df_match_candidates.merge(df_ranker_train, on=[USER_COL, ITEM_COL], how='

# чистим дубликаты
df_ranker_train = df_ranker_train.drop_duplicates(subset=[USER_COL, ITEM_COL])

df_ranker_train['target'].fillna(0, inplace= True)
```

In [76]:

```
df_ranker_train.target.value_counts()
```

Out[76]:

```
0.0    2007992
1.0      31019
Name: target, dtype: int64
```

In [77]:

```
df_ranker_train.head(2)
```

Out[77]:

	user_id	item_id	target
0	2070	1105426	0.0
1	2070	1097350	0.0

(!) На каждого юзера 50 item_id-кандидатов

In [78]:

```
df_ranker_train['target'].mean()
```

Out[78]:

```
0.015212767366139761
```

Ранжирование

Градиентный бустинг

1. binary
2. lambdarank
3. rank_xendcg

1. binary:logistic
2. rank:pairwise
3. rank:ndcg
4. rank:map

1. RMSE
2. QueryRMSE
3. PairLogit
4. PairLogitPairwise
5. YetiRank
6. YetiRankPairwise

Слайд из [презентации (https://github.com/aprotopopov/retailhero_recommender/blob/master/slides/retailhero_recommender.pdf)](https://github.com/aprotopopov/retailhero_recommender/blob/master/slides/retailhero_recommender.pdf) решения 2-ого места X5 Retail Hero

- Пока для простоты обучения выберем LightGBM с loss = binary. Это классическая бинарная классификация
- Это пример *без* генерации фич

Подготавливаем фичи для обучения модели

Описательные фичи

In [79]:

```
item_features.head(2)
```

Out[79]:

	item_id	manufacturer	department	brand	commodity_desc	sub_commodity_desc	curr_siz
0	25671	2	GROCERY	National	FRZN ICE	ICE - CRUSHED/CUBED	
1	26081	2	MISC. TRANS.	National	NO COMMODITY DESCRIPTION	NO SUBCOMMODITY DESCRIPTION	

In [80]:

```
user_features.head(2)
```

Out[80]:

	age_desc	marital_status_code	income_desc	homeowner_desc	hh_comp_desc	household_s
0	65+	A	35-49K	Homeowner	2 Adults No Kids	
1	45-54	A	50-74K	Homeowner	2 Adults No Kids	

In [81]:

```
df_ranker_train = df_ranker_train.merge(item_features, on='item_id', how='left')
df_ranker_train = df_ranker_train.merge(user_features, on='user_id', how='left')

df_ranker_train.head(2)
```

Out[81]:

	user_id	item_id	target	manufacturer	department	brand	commodity_desc	sub_commod
0	2070	1105426	0.0	69	DELI	Private	SANDWICHES	SANDV
1	2070	1097350	0.0	2468	GROCERY	National	DOMESTIC WINE	VALUE GLA

Фичи user_id: - Средний чек - Средняя сумма покупки 1 товара в каждой категории - Кол-во покупок в каждой категории - Частотность покупок раз/месяц - Долю покупок в выходные - Долю покупок утром/днем/вечером

Фичи item_id: - Кол-во покупок в неделю - Среднее ол-во покупок 1 товара в категории в неделю - (Кол-во покупок в неделю) / (Среднее ол-во покупок 1 товара в категории в неделю) - Цена (Можно посчитать из retil_train.csv) - Цена / Средняя цена товара в категории

Фичи пары user_id - item_id - (Средняя сумма покупки 1 товара в каждой категории (берем категорию item_id)) - (Цена item_id) - (Кол-во покупок юзером конкретной категории в неделю) - (Среднее кол-во покупок всеми юзерами конкретной категории в неделю) - (Кол-во покупок юзером конкретной категории в неделю) / (Среднее кол-во покупок всеми юзерами конкретной категории в неделю)

Поведенческие фичи

Чтобы считать поведенческие фичи, нужно учесть все данные что были до data_val_ranker

In [101]:

```
df_join_train_matcher.head()
```

Out[101]:

	user_id	basket_id	day	item_id	quantity	sales_value	store_id	retail_disc	trans_time
0	2375	26984851472	1	1004906	1	1.39	364	-0.60	1631
1	2375	26984851472	1	1033142	1	0.82	364	0.00	1631
2	2375	26984851472	1	1036325	1	0.99	364	-0.30	1631
3	2375	26984851472	1	1082185	1	1.21	364	0.00	1631
4	2375	26984851472	1	8160430	1	1.50	364	-0.39	1631

!!! Пока выполните нотбук без этих строк, потом вернитесь и запустите их, обучите ранкер и посмотрите на метрики с ранжированием

In [102]:

```
df_ranker_train = df_ranker_train.merge(df_join_train_matcher.groupby(by=ITEM_COL).agg('sal

df_ranker_train = df_ranker_train.merge(df_join_train_matcher.groupby(by=ITEM_COL).agg('qua

df_ranker_train = df_ranker_train.merge(df_join_train_matcher.groupby(by=ITEM_COL).agg(USER

df_ranker_train = df_ranker_train.merge(df_join_train_matcher.groupby(by=USER_COL).agg(USER

df_ranker_train = df_ranker_train.merge(df_join_train_matcher.groupby(by=USER_COL).agg('sal

df_ranker_train = df_ranker_train.merge(df_join_train_matcher.groupby(by=ITEM_COL).agg('qua

df_ranker_train = df_ranker_train.merge(df_join_train_matcher.groupby(by=USER_COL).agg('qua


df_ranker_train = df_ranker_train.merge(df_join_train_matcher.groupby(by=ITEM_COL).agg('qua

df_ranker_train = df_ranker_train.merge(df_join_train_matcher.groupby(by=USER_COL).agg('qua


df_ranker_train = df_ranker_train.merge(df_join_train_matcher.groupby(by=ITEM_COL).agg(USER

df_ranker_train = df_ranker_train.merge(df_join_train_matcher.groupby(by=USER_COL).agg(USER
```

In [103]:

```
df_ranker_train.head()
```

Out[103]:

	user_id	item_id	target	manufacturer	department	brand	commodity_desc	sub_commod
0	2070	1105426	0.0	69	DELI	Private	SANDWICHES	SANDV
1	2070	1097350	0.0	2468	GROCERY	National	DOMESTIC WINE	VALUE GLA
2	2070	879194	0.0	69	DRUG GM	Private	DIAPERS & DISPOSABLES	BABY D
3	2070	948640	0.0	1213	DRUG GM	National	ORAL HYGIENE PRODUCTS	WH S
4	2070	928263	0.0	69	DRUG GM	Private	DIAPERS & DISPOSABLES	BABY D

5 rows × 27 columns

In [104]:

```
X_train = df_ranker_train.drop('target', axis=1)
y_train = df_ranker_train[['target']]
```

In [105]:

```
cat_feats = X_train.columns[2:].tolist()
X_train[cat_feats] = X_train[cat_feats].astype('category')

cat_feats
```

Out[105]:

```
['manufacturer',
 'department',
 'brand',
 'commodity_desc',
 'sub_commodity_desc',
 'curr_size_of_product',
 'age_desc',
 'marital_status_code',
 'income_desc',
 'homeowner_desc',
 'hh_comp_desc',
 'household_size_desc',
 'kid_category_desc',
 'total_item_sales_value',
 'total_quantity_value',
 'item_freq',
 'user_freq',
 'total user sales value'.
```

Обучение модели ранжирования

In [106]:

```
lgb = LGBMClassifier(objective='binary',
                     max_depth=10,
                     n_estimators=500,
                     learning_rate=0.1,
                     categorical_column=cat_feats)

lgb.fit(X_train, y_train)

train_preds = lgb.predict_proba(X_train)
```

```
C:\Users\voron\AppData\Roaming\Python\Python37\site-packages\sklearn\utils\v
alidation.py:63: DataConversionWarning: A column-vector y was passed when a
1d array was expected. Please change the shape of y to (n_samples, ), for ex
ample using ravel().
  return f(*args, **kwargs)
C:\Users\voron\AppData\Roaming\Python\Python37\site-packages\lightgbm\basic.
py:1245: UserWarning: categorical_column in param dict is overridden.
  _log_warning('{} in param dict is overridden.'.format(cat_alias))
```

In [107]:

```
df_ranker_predict = df_ranker_train.copy()
```

In [108]:

```
df_ranker_predict['proba_item_purchase'] = train_preds[:,1]
```

Подведем итоги

Мы обучили модель ранжирования на покупках из сета data_train_ranker и на кандитатах от own_recommendations, что является тренировочным сетом, и теперь наша задача предсказать и оценить именно на тестовом сете.

Evaluation on test dataset

In [109]:

```
result_eval_ranker = data_val_ranker.groupby(USER_COL)[ITEM_COL].unique().reset_index()
result_eval_ranker.columns=[USER_COL, ACTUAL_COL]
result_eval_ranker.head(2)
```

Out[109]:

	user_id	actual
0	1	[821867, 834484, 856942, 865456, 889248, 90795...
1	3	[835476, 851057, 872021, 878302, 879948, 90963...

Eval matching on test dataset

In [110]:

```
%%time
result_eval_ranker['own_rec'] = result_eval_ranker[USER_COL].apply(lambda x: recommender.ge
```

```
Wall time: 12.5 s
```

In [111]:

```
# померяем precision только модели матчинга, чтобы понимать влияние ранжирования на метрик

sorted(calc_precision(result_eval_ranker, TOPK_PRECISION), key=lambda x: x[1], reverse=True
```

Out[111]:

```
[('own_rec', 0.14441176470588235)]
```

Eval re-ranked matched result on test dataset

Вспомним df_match_candidates сет, который был получен own_recommendations на юзера x, набор пользователей мы фиксировали и он одинаков, значи и прогноз одинаков, поэ тому мы можем использовать этот датафрейм для переранжирования.

In [112]:

```
def rerank(user_id):
    return df_ranker_predict[df_ranker_predict[USER_COL]==user_id].sort_values('proba_item_
```

In [113]:

```
result_eval_ranker['reranked_own_rec'] = result_eval_ranker[USER_COL].apply(lambda user_id:
```

Проверьте данные метрики с фичами и без (PS должен быть прирост)

In [114]:

```
# смотрим на метрики выше и сравниваем что с ранжированием и без, добавляем фичи и то же см
# в первом приближении метрики должны расти с использованием второго этапа

print(*sorted(calc_precision(result_eval_ranker, TOPK_PRECISION), key=lambda x: x[1], rever
```

```
('reranked_own_rec', 0.21859007832898172)
('own_rec', 0.14441176470588235)

C:\Users\voron\а учеба\рекомендательные системы\cource_project\metrics.py:2
0: RuntimeWarning: invalid value encountered in long_scalars
  return flags.sum() / len(recommended_list)
```

Оценка на тесте для выполнения курсового проекта

In [115]:

```
# df_transactions = pd.read_csv('../data/transaction_data.csv')
```

In [120]:

```
df_test = pd.read_csv('../data/retail_test1.csv')
```

In [121]:

```
df_test.head()
```

Out[121]:

	user_id	basket_id	day	item_id	quantity	sales_value	store_id	retail_disc	trans_time
0	1340	41652823310	664	912987	1	8.49	446	0.0	52
1	588	41652838477	664	1024426	1	6.29	388	0.0	8
2	2070	41652857291	664	995242	5	9.10	311	-0.6	46
3	1602	41665647035	664	827939	1	7.99	334	0.0	1741
4	1602	41665647035	664	927712	1	0.59	334	-0.4	1741

In [122]:

```
result_test = df_test.groupby(USER_COL)[ITEM_COL].unique().reset_index()
result_test.columns=[USER_COL, ACTUAL_COL]
result_test.head(2)
```

Out[122]:

	user_id	actual
0	1	[880007, 883616, 931136, 938004, 940947, 94726...
1	2	[820165, 820291, 826784, 826835, 829009, 85784...

In [126]:

```
print(*sorted(calc_precision(result_test, TOPK_PRECISION), key=lambda x: x[1], reverse=True
```

In [123]:

```
TOPK_PRECISION = 5
print(sorted(calc_precision(result_test, TOPK_PRECISION), key=lambda x: x[1],reverse=True))
```

```
[]
```

Берем топ-k предсказаний, ранжированных по вероятности, для каждого юзера

In [ ]:

```
```

In [ ]:

```
```