Вебинар 6. Двухуровневые модели рекомендаций

Зачем 2 уровня?

- Классические модели классификации (lightgbm) зачастую работают лучше, чем рекоммендательные модели (als, lightfm)
- Данных много, предсказаний много (# items * # users) --> с таким объемом lightgbm не справляется
- Но рекомендательные модели справляются!

Отбираем top-N (200) *кандидатов* с помощью простой модели (als) --> переранжируем их сложной моделью (lightgbm) и выберем top-k (10).

Как отбирать кандидатов?

Вариантов множество. Тут нам поможет *MainRecommender*. Пока в нем реализованы далеко не все возможные способы генерации кандидатов

- Генерируем топ-k кандидатов
- Качество кандидатов измеряем через **recall@k**
- recall@k показывает какую долю из купленных товаров мы смогли выявить (рекомендовать) нашей моделью

Практическая часть

Import libs

In [78]:

```
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 [79]:

```
data = pd.read_csv('../data/retail_train.csv')
item_features = pd.read_csv('../data/product.csv')
user_features = pd.read_csv('../data/hh_demographic.csv')
```

Process features dataset

In [80]:

```
ITEM_COL = 'item_id'
USER_COL = 'user_id'
```

In [81]:

```
# 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 [82]:

```
# 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 [83]:

```
# 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 [84]:

```
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 [85]:

```
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 [86]:

```
# выше видим разброс по пользователям и товарам
```

In [87]:

```
data_train_matcher.head(2)
```

Out[87]:

	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

Prefilter items

In [88]:

```
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\les 6\utils.py:20: SettingWi
thCopyWarning:
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 [89]:

```
# 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
```

In [90]:

```
# Теперь warm-start по пользователям
```

Init/train recommender

In [91]:

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

```
  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 [92]:

```
# Берем тестового юзера 2375
```

In [93]:

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

Out[93]:

```
[899624, 1106523, 1044078, 871756, 8090521]
```

In [94]:

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

Out[94]:

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

In [95]:

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

Out[95]:

```
[1046545, 1044078, 999270, 1012584, 1133312]
```

In [96]:

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

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_matcher: 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 [97]:

```
ACTUAL_COL = 'actual'
```

In [99]:

```
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[99]:

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

In [152]:

```
# N = Neighbors
N_PREDICT = 500
recommend_model= [recommender.get_own_recommendations, recommender.get_similar_items_recomm
                  recommender.get_als_recommendations]
```

In [153]:

```
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
```

In [154]:

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

In [155]:

```
result_eval_matcher[:3]
```

Out[155]:

	user_id	actual	re	sim_item_rec	als_rec	res	own	similar	
0	1	[853529, 865456, 867607, 872137, 874905, 87524...	[856942, 9297615, 5577022, 877391, 9655212, 88...	[842762, 1007512, 9297615, 5577022, 9803207, 9...	[962615, 9858819, 856942, 883616, 858001, 5577...	[856942, 9297615, 5577022, 877391, 9655212, 88...	[856942, 9297615, 5577022, 877391, 9655212, 88...	[842762, 1007512, 9297615, 5577022, 9803207, 9...	[962 985 85 88 85 5
1	2	[15830248, 838136, 839656, 861272, 866211, 870...	[911974, 1076580, 1103898, 5567582, 1056620, 9...	[1137346, 5569845, 1044078, 985999, 880888, 81...	[5569230, 916122, 1040807, 5569845, 1054567, 8...	[911974, 1076580, 1103898, 5567582, 1056620, 9...	[911974, 1076580, 1103898, 5567582, 1056620, 9...	[1137346, 5569845, 1044078, 985999, 880888, 81...	[556 91 104 556 105
2	4	[883932, 970760, 1035676, 1055863, 1097610, 67...	[6391541, 1052294, 891423, 936470, 1137010, 11...	[1038214, 846550, 990762, 999714, 6514160, 854...	[891423, 6391541, 1052294, 982790, 1075368, 92...	[6391541, 1052294, 891423, 936470, 1137010, 11...	[6391541, 1052294, 891423, 936470, 1137010, 11...	[1038214, 846550, 990762, 999714, 6514160, 854...	[89 639 105 98 107

In [ ]:

```
# result_eval_matcher['sim_item_rec'] = result_eval_matcher[USER_COL].apply(lambda x: recom
```

In [145]:

```
# res_eval(result_eval_matcher, USER_COL, recommender.get_own_recommendations, N_PREDICT)
```

...

In [142]:

```
a = str('recommender.get_als_recommendations').split('_')[1]
# a = a.split('_')
a
```

Out[142]:

```
'als'
```

In [130]:

```
# %%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
```

```
Wall time: 1min 11s
```

In [131]:

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

```
Wall time: 0 ns
```

Пример оборачивания

In [156]:

```
# # сырой и простой пример как можно обернуть в функцию
def evalRecall(df_result, target_col_name, recommend_model,result_col_name):
    result_col_name = result_col_name
    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 [133]:

```
evalRecall(result_eval_matcher, USER_COL, recommender.get_own_recommendations, 'res')
```

Out[133]:

```
0.0441195473958354
```

In [134]:

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

In [135]:

```
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 [136]:

```
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 [157]:

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

In [158]:

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

```
20 [('re', 0.039284276793729055), ('res', 0.039284276793729055), ('own', 0.0
39284276793729055), ('als_rec', 0.029571335711236067), ('als', 0.02957133571
1236067), ('sim_item_rec', 0.017892325490142767), ('similar', 0.017892325490
142767)]
50 [('re', 0.06525657038145165), ('own', 0.06525657038145165), ('als_rec',
0.048397986462560875), ('als', 0.048397986462560875), ('res', 0.044119547395
8354), ('sim_item_rec', 0.03342448465786803), ('similar', 0.0334244846578680
3)]
100 [('re', 0.09604492955885016), ('own', 0.09604492955885016), ('als_rec',
0.07027401151302852), ('als', 0.07027401151302852), ('sim_item_rec', 0.05311
7575905850485), ('similar', 0.053117575905850485), ('res', 0.044119547395835
4)]
200 [('re', 0.13537278412833254), ('own', 0.13537278412833254), ('als_rec',
0.09816568167820107), ('als', 0.09816568167820107), ('sim_item_rec', 0.08621
946929988802), ('similar', 0.08621946929988802), ('res', 0.044119547395835
4)]
500 [('re', 0.18205324555508703), ('own', 0.18205324555508703), ('als_rec',
0.14663209707606234), ('als', 0.14663209707606234), ('sim_item_rec', 0.13628
08770545324), ('similar', 0.1362808770545324), ('res', 0.0441195473958354)]
```

Precision@5 of matching

In [161]:

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

In [162]:

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

```
[('re', 0.10485820548582056), ('res', 0.10485820548582056), ('own', 0.104858
20548582056), ('als_rec', 0.07819618781961879), ('als', 0.0781961878196187
9), ('sim_item_rec', 0.04686192468619247), ('similar', 0.04686192468619247)]
[('res', 0.09614132961413296), ('re', 0.07247791724779172), ('own', 0.072477
91724779172), ('als_rec', 0.05534170153417016), ('als', 0.0553417015341701
6), ('sim_item_rec', 0.036634123663412364), ('similar', 0.03663412366341236
4)]
[('res', 0.09614132961413296), ('re', 0.05525801952580195), ('own', 0.055258
01952580195), ('als_rec', 0.042278010227801026), ('als', 0.04227801022780102
6), ('sim_item_rec', 0.0296931659693166), ('similar', 0.0296931659693166)]
[('res', 0.09614132961413296), ('re', 0.04180381218038123), ('own', 0.041803
81218038123), ('als_rec', 0.03090190609019061), ('als', 0.0309019060901906
1), ('sim_item_rec', 0.024423523942352393), ('similar', 0.02442352394235239
3)]
[('res', 0.09614132961413296), ('re', 0.024346815434681545), ('own', 0.02434
6815434681545), ('als_rec', 0.019295211529521156), ('als', 0.019295211529521
156), ('sim_item_rec', 0.01697071129707113), ('similar', 0.0169707112970711
3)]
```

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 [56]:

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

Подготовка данных для трейна

In [57]:

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

In [58]:

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

In [59]:

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

Out[59]:

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

In [39]:

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

In [41]:

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

In [42]:

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

Out[42]:

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

Check warm start

In [43]:

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

```
match_candidates
Shape: (107550, 2) Users: 2151 Items: 4574
```

Создаем трейн сет для ранжирования с учетом кандидатов с этапа 1

In [44]:

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

In [45]:

```
df_ranker_train.head()
```

Out[45]:

	user_id	item_id	target
2104867	2070	1019940	1
2107468	2021	840361	1
2107469	2021	856060	1
2107470	2021	869344	1
2107471	2021	896862	1

Не хватает нулей в датасете, поэтому добавляем наших кандитатов в качество нулей

In [46]:

```
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 [47]:

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

Out[47]:

```
0.0    99177
1.0     7795
Name: target, dtype: int64
```

In [48]:

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

Out[48]:

	user_id	item_id	target
0	2070	1105426	0.0
1	2070	1097350	0.0

(!) На каждого юзера 50 item_id-кандидатов

In [49]:

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

Out[49]:

```
0.07286953595333358
```

Ранжирование

Градиентный бустинг

Microsoft LightGBM

1. binary
2. lambdarank
3. rank_xendcg

dmlc XGBoost

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

CatBoost

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

In [50]:

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

Out[50]:

	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 [51]:

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

Out[51]:

	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 [52]:

```
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[52]:

	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 GLAS

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 кол-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)

In [53]:

```
df_ranker_train.head()
```

Out[53]:

	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 [
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 [

In [54]:

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

In [55]:

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

cat_feats
```

Out[55]:

```
['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']
```

Обучение модели ранжирования

In [75]:

```
lgb = LGBMClassifier(objective='binary',
                     max_depth=8,
                     n_estimators=300,
                     learning_rate=0.05,
                     categorical_column=cat_feats)

lgb.fit(X_train, y_train)

train_preds = lgb.predict_proba(X_train)
```

```
/home/quasar/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.
py:63: DataConversionWarning: A column-vector y was passed when a 1d array w
as expected. Please change the shape of y to (n_samples, ), for example usin
g ravel().
  return f(*args, **kwargs)
/home/quasar/anaconda3/lib/python3.8/site-packages/lightgbm/basic.py:1245: U
serWarning: categorical_column in param dict is overridden.
  _log_warning('{} in param dict is overridden.'.format(cat_alias))
```

In [76]:

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

In [77]:

```
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 [78]:

```
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[78]:

	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 [79]:

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

```
CPU times: user 4.31 s, sys: 4.02 ms, total: 4.31 s
Wall time: 4.31 s
```

In [80]:

```
# 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[80]:

```
[('own_rec', 0.1444117647058813)]
```

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 [81]:

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

In [82]:

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

In [83]:

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

```
('reranked_own_rec', 0.15331592689294912)
('own_rec', 0.1444117647058813)

/data/home/quasar/projects_personal/GeekBrainsRecommendations/lessons/webina
r_6/metrics.py:20: RuntimeWarning: invalid value encountered in long_scalars
  return flags.sum() / len(recommended_list)
```

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

Домашнее задание

Задание 1.

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_matcher: 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?

Задание 2.

Обучите модель 2-ого уровня, при этом:

- Добавьте минимум по 2 фичи для юзера, товара и пары юзер-товар
- Измерьте отдельно precision@5 модели 1-ого уровня и двухуровневой модели на data_val_ranker
- Вырос ли precision@5 при использовании двухуровневой модели?