Рекомендательные системы

Урок 6. Двухуровневые модели рекомендаций

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?

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]:
```
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 [3]:
```
ITEM_COL = 'item_id'
USER_COL = 'user_id'
```

In [4]:
```
# 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 [5]:
```
# Важна схема обучения и валидации!
# -- давние покупки -- | -- 6 недель -- | -- 3 недель --
# подобрать размер 2-ого датасета (6 недель) --> learning curve (зависимость метрики recall@k
# от размера датасета)


VAL_MATCHER_WEEKS = 6
VAL_RANKER_WEEKS = 3
```

In [6]:
```
# берем данные для тренировки matching модели
data_train_matcher = data[data['week_no'] < data['week_no'].max() - \
                          (VAL_MATCHER_WEEKS + VAL_RANKER_WEEKS)]

# берем данные для валидации matching модели
data_val_matcher = data[(data['week_no'] >= data['week_no'].max() - \
                         (VAL_MATCHER_WEEKS + VAL_RANKER_WEEKS)) &  \
                         (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, и они будут отличаться

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

In [7]:
```
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[ITEM_COL].nunique()}")
```

In [8]:
```
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 [9]:
```
# выше видим разброс по пользователям и товарам
```

In [10]:
```
data_train_matcher.head(2)
```

Out[10]:

	user_id	basket_id	day	item_id	quantity	sales_value	store_id	retail_disc	trans_time	week_no	coupon_disc	coupon_match_disc
0	2375	26984851472	1	1004906	1	1.39	364	-0.6	1631	1	0.0	0.0
1	2375	26984851472	1	1033142	1	0.82	364	0.0	1631	1	0.0	0.0

Prefilter items

In [11]:
```
# Префильтрация. Оставляем 5000 товаров.
```

In [12]:
```
n_items_before = data_train_matcher['item_id'].nunique()

data_train_matcher = prefilter_items(data_train_matcher, \
                                     item_features=item_features, take_n_popular=5000)

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

```
/home/sil/ML/RS/Lesson_6/HW/utils.py:20: SettingWithCopyWarning:
Decreased # items from 83685 to 5001
```

Make cold-start to warm-start

In [13]:
```
# 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 [14]:
```
common_users
```

Out[14]:
```
array([2375, 1364, 1364, ...,  856,  856,  856])
```

In [15]:
```
data_val_matcher.head(3)
```

Out[15]:

	user_id	basket_id	day	item_id	quantity	sales_value	store_id	retail_disc	trans_time	week_no	coupon_disc	coupon_match_disc
2104867	2070	40618492260	594	1019940	1	1.00	311	-0.29	40	86	0.0	0.0
2107468	2021	40618753059	594	840361	1	0.99	443	0.00	101	86	0.0	0.0
2107469	2021	40618753059	594	856060	1	1.77	443	-0.09	101	86	0.0	0.0

In [16]:
```
data_train_ranker.head(3)
```

Out[16]:

	user_id	basket_id	day	item_id	quantity	sales_value	store_id	retail_disc	trans_time	week_no	coupon_disc	coupon_match_disc
2104867	2070	40618492260	594	1019940	1	1.00	311	-0.29	40	86	0.0	0.0
2107468	2021	40618753059	594	840361	1	0.99	443	0.00	101	86	0.0	0.0
2107469	2021	40618753059	594	856060	1	1.77	443	-0.09	101	86	0.0	0.0

In [17]:
```
data_val_ranker.head(3)
```

Out[17]:

	user_id	basket_id	day	item_id	quantity	sales_value	store_id	retail_disc	trans_time	week_no	coupon_disc	coupon_match_disc
2277416	338	41260573635	636	840173	1	1.99	369	0.0	112	92	0.0	0.0
2277417	338	41260573635	636	1037348	1	0.89	369	-0.3	112	92	0.0	0.0
2277418	338	41260573635	636	5592737	2	1.58	369	-0.2	112	92	0.0	0.0

In [18]:
```
# Теперь warm-start по пользователям
```

Init/train recommender

In [19]:
```
recommender = MainRecommender(data_train_matcher)  # from recommenders.py
```

```
WARNING:root:Intel MKL BLAS detected. Its highly recommend to set the environment variable 'export MKL_NUM
_THREADS=1' to disable its internal multithreading
```

```
100%                                          15/15 [00:01<00:00, 9.88it/s]
```

```
100%                                          5001/5001 [00:30<00:00, 162.72it/s]
```

In [20]:
```
recommender
```

Out[20]:
```
<recommenders.MainRecommender at 0x7f95189286d0>
```

In [21]:
```
recommender.top_purchases.head(3)  # Топ покупок каждого юзера
```

Out[21]:

	user_id	item_id	quantity
182094	1479	977374	116
42966	358	5569230	115
199231	1609	6632283	112

In [22]:
```
recommender.overall_top_purchases[:3]  # Топ покупок по всему датасету
```

Out[22]:
```
[1029743, 1106523, 5569230]
```

In [23]:
```
# user_item_matrix = recommender.user_item_matrix
# print(type(user_item_matrix))
# print(user_item_matrix)
```

Варианты, как получить кандидатов

In [24]:
```
# Берем тестового юзера 2375
```

In [25]:
```
recommender.get_als_recommendations(2375, N=5)
```

Out[25]:
```
[899624, 871756, 1106523, 1044078, 844179]
```

In [26]:
```
recommender.get_own_recommendations(2375, N=5)
```

Out[26]:
```
[948640, 918046, 847962, 907099, 873980]
```

In [27]:
```
recommender.get_similar_items_recommendation(2375, N=5)
```

Out[27]:
```
[1046545, 1044078, 1044078, 824758, 15778319]
```

In [28]:
```
recommender.get_similar_users_recommendation(2375, N=5)
```

Out[28]:
```
[1097398, 1087411, 1008288, 894360, 928749]
```

Eval recall of matching

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?

In [29]:
```
ACTUAL_COL = 'actual'
```

In [30]:
```
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[30]:

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

In [31]:
```
# Рекомендуем топ-N товаров
# N = Neighbors

N_PREDICT = 50
```

In [32]:
```
%%time
# для понятности расписано все в строчку, без функций, ваша задача уметь оборачивать все это в функции
result_eval_matcher['own_rec'] = result_eval_matcher[USER_COL]. \
        apply(lambda x: recommender.get_own_recommendations(x, N=N_PREDICT))

result_eval_matcher['sim_item_rec'] = result_eval_matcher[USER_COL]. \
        apply(lambda x: recommender.get_similar_items_recommendation(x, N=50))

result_eval_matcher['als_rec'] = result_eval_matcher[USER_COL]. \
        apply(lambda x: recommender.get_als_recommendations(x, N=50))
```

```
CPU times: user 57.6 s, sys: 3.04 s, total: 1min
Wall time: 33.3 s
```

In [33]:
```
%%time
result_eval_matcher['sim_user_rec'] = result_eval_matcher[USER_COL]. \
        apply(lambda x: recommender.get_similar_users_recommendation(x, N=50))
```

```
CPU times: user 7min 41s, sys: 519 ms, total: 7min 42s
Wall time: 7min 43s
```

Пример оборачивания

In [34]:
```
# # 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(x, N=25))
    return df_result.apply(lambda row: recall_at_k(row[result_col_name], \
                                                   row[ACTUAL_COL], k=N_PREDICT), axis=1).mean()
```

In [35]:
```
evalRecall(result_eval_matcher, USER_COL, recommender.get_own_recommendations)
```

Out[35]:
```
0.044119547395835505
```

In [36]:
```
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], k=top_k), axis=1).mean()
```

In [37]:
```
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_COL], k=top_k), axis=1).mean()
```

Задание 1:
A) Попробуйте различные варианты генерации кандидатов. Какие из них дают наибольший recall@k ?

Recall@50 of matching

In [38]:
```
TOPK_RECALL = 50
```

In [39]:
```
sorted(calc_recall(result_eval_matcher, TOPK_RECALL), key=lambda x: x[1],reverse=True)
```

Out[39]:
```
[('own_rec', 0.06525657038145175),
 ('als_rec', 0.04844843757605246),
 ('result', 0.044119547395835505),
 ('sim_item_rec', 0.033859054213855266),
 ('sim_user_rec', 0.006968610473199277)]
```

Precision@5 of matching

In [40]:
```
TOPK_PRECISION = 5
```

In [41]:
```
sorted(calc_precision(result_eval_matcher, TOPK_PRECISION), key=lambda x: x[1],reverse=True)
```

Out[41]:
```
[('own_rec', 0.17712691771268974),
 ('result', 0.17712691771268974),
 ('als_rec', 0.11864249186424834),
 ('sim_item_rec', 0.06183170618317097),
 ('sim_user_rec', 0.012273361227336096)]
```

In [42]:
```
result_eval_matcher.head(2)
```

Out[42]:

	user_id	actual	own_rec	sim_item_rec	als_rec	sim_user_rec	result
0	1	[853529, 865456, 867607, 872137, 874905, 87524...	[856942, 9297615, 5577022, 877391, 9655212, 88...	[884686, 1007512, 9297615, 5577022, 1117824, 9...	[8090541, 1116050, 1108844, 1037332, 1061747, ...	[8293439, 1062572, 1019247, 1016785, 1096228, ...	[856942, 9297615, 5577022, 877391, 9655212, 88...
1	2	[15830248, 838136, 839656, 861272, 866211, 870...	[911974, 1076580, 1103898, 5567582, 1056620, 9...	[1137346, 5569845, 1044078, 985999, 880888, 81...	[5569230, 1029743, 1017369, 1106523, 916122, 1...	[911974, 830202, 9419422, 875392, 903529, 1052...	[911974, 1076580, 1103898, 5567582, 1056620, 9...

In [43]:
```
ACTUAL_COL
```

Out[43]:
```
'actual'
```

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

Вывод:

Own recommendtions + top-popular дают лучший ***recall***:
Recall@50_of_matching ('own_rec') = 0.17712691771268974 для TOPK_RECALL = 50

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 [44]:
```
TOPK_RECALLS =[20, 50, 100, 200, 500]
```

In [45]:
```
for TOPK_RECALL in TOPK_RECALLS:
    print(TOPK_RECALL, ':', sorted(calc_recall(result_eval_matcher, TOPK_RECALL), \
                                   key=lambda x: x[1],reverse=True), '\n')
```

```
20 : [('own_rec', 0.03928427679372909), ('result', 0.03928427679372909), ('als_rec', 0.02913130958993557
7), ('sim_item_rec', 0.017684277994578466), ('sim_user_rec', 0.0037972321814665894)]

50 : [('own_rec', 0.06525657038145175), ('als_rec', 0.04844843757605246), ('result', 0.04411954739583550
5), ('sim_item_rec', 0.033859054213855266), ('sim_user_rec', 0.006968610473199277)]

100 : [('own_rec', 0.06525657038145175), ('als_rec', 0.04844843757605246), ('result', 0.04411954739583550
5), ('sim_item_rec', 0.033859054213855266), ('sim_user_rec', 0.006968610473199277)]

200 : [('own_rec', 0.06525657038145175), ('als_rec', 0.04844843757605246), ('result', 0.04411954739583550
5), ('sim_item_rec', 0.033859054213855266), ('sim_user_rec', 0.006968610473199277)]

500 : [('own_rec', 0.06525657038145175), ('als_rec', 0.04844843757605246), ('result', 0.04411954739583550
5), ('sim_item_rec', 0.033859054213855266), ('sim_user_rec', 0.006968610473199277)]
```

```
20: 'own_rec'   0.03928427679372909,
    'result'    0.03928427679372909,
    'als_rec'   0.02977277631156393,
    'sim_item_rec'  0.018713048812415596,
    'sim_user_rec'  0.003993622951283449


50: 'own_rec'   0.06525657038145175,
    'als_rec'   0.047859963674155404,
    'result'    0.044119547395835505,
    'sim_item_rec'  0.03445942417894158,
    'sim_user_rec'  0.007531080937345071


100:'own_rec'   0.06525657038145175,
    'als_rec'   0.047859963674155404,
    'result'    0.044119547395835505,
    'sim_item_rec'  0.03445942417894158
    'sim_user_rec'  0.007531080937345071


200:'own_rec'   0.06525657038145175,
    'als_rec'   0.047859963674155404,
    'result'    0.044119547395835505,
    'sim_item_rec'  0.03445942417894158,
    'sim_user_rec'  0.007531080937345071
```

```
500:'own_rec'   0.06525657038145175,
    'als_rec'   0.047859963674155404,
    'result'    0.044119547395835505,
    'sim_item_rec'  0.03445942417894158,
    'sim_user_rec'  0.007531080937345071
```

Задание 1:

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?

Вывод:

Видно, что при **k=20**, проседает значение **recall@k=0.03928427679372909**.
При **k=50, recall@k (Own recommendtions + top-popular) =0.06525657038145175** достигает максимального значения и при дальнейшем увеличении значения **k recall@k** не изменяется.

Таким образом из выбранного ряда, наиболее разумным является значение **k=50**.

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

3 временных интервала

-- давние покупки -- | -- 6 недель -- | -- 3 недель --

Подготовка данных для трейна

In [46]:
```
# взяли пользователей из трейна для ранжирования
df_match_candidates = pd.DataFrame(data_train_ranker[USER_COL].unique())
df_match_candidates.columns = [USER_COL]
```

In [47]:
```
# собираем кандитатов с первого этапа (matcher)
df_match_candidates['candidates'] = df_match_candidates[USER_COL]. \
            apply(lambda x: recommender.get_own_recommendations(x, N=N_PREDICT))
```

In [48]:
```
df_match_candidates.head(2)
```

Out[48]:

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

In [49]:
```
df_items = df_match_candidates.apply(lambda x: pd.Series(x['candidates']), axis=1). \
                stack().reset_index(level=1, drop=True)
df_items.name = 'item_id'
```

In [50]:
```
df_match_candidates = df_match_candidates.drop('candidates', axis=1).join(df_items)
```

In [51]:
```
df_match_candidates.head(4)
```

Out[51]:

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

Check warm start

In [52]:
```
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 [53]:
```
df_ranker_train = data_train_ranker[[USER_COL, ITEM_COL]].copy()
df_ranker_train['target'] = 1  # тут только покупки
```

In [54]:
```
df_ranker_train.head()
```

Out[54]:

	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 [55]:
```
df_ranker_train = df_match_candidates.merge(df_ranker_train, on=[USER_COL, ITEM_COL], how='left')

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

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

In [56]:
```
df_ranker_train.target.value_counts()
```

Out[56]:
```
0.0    99177
1.0     7795
Name: target, dtype: int64
```

In [57]:
```
df_ranker_train.head(2)
```

Out[57]:

	user_id	item_id	target
0	2070	1105426	0.0
1	2070	1097350	0.0

(!) На каждого юзера 50 item_id-кандидатов

In [58]:
```
df_ranker_train['target'].mean()
```

Out[58]:
```
0.07286953595333358
```

In [ ]:
```

```

Подготавливаем фичи для обучения модели

In [59]:
```
item_features.head(2)
```

Out[59]:

	item_id	manufacturer	department	brand	commodity_desc	sub_commodity_desc	curr_size_of_product
0	25671	2	GROCERY	National	FRZN ICE	ICE - CRUSHED/CUBED	22 LB
1	26081	2	MISC. TRANS.	National	NO COMMODITY DESCRIPTION	NO SUBCOMMODITY DESCRIPTION	

In [60]:
```
user_features.head(2)
```

Out[60]:

	age_desc	marital_status_code	income_desc	homeowner_desc	hh_comp_desc	household_size_desc	kid_category_desc	user_id
0	65+	A	35-49K	Homeowner	2 Adults No Kids	2	None/Unknown	1
1	45-54	A	50-74K	Homeowner	2 Adults No Kids	2	None/Unknown	7

In [61]:
```
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[61]:

	user_id	item_id	target	manufacturer	department	brand	commodity_desc	sub_commodity_desc	curr_size_of_product	age_desc	marital_s
0	2070	1105426	0.0	69	DELI	Private	SANDWICHES	SANDWICHES - (COLD)		45-54	
1	2070	1097350	0.0	2468	GROCERY	National	DOMESTIC WINE	VALUE GLASS WINE	4 LTR	45-54	

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)

retail_train.csv

- user_id - идентификатор пользователя
- basket_id - идентификатор корзины
- day - день
- item_id - идентификатор товара
- quantity - количество
- sales_value - продаж значение
- store_id - идентификатор магазина
- retail_disc - розничные продажи описание
- trans_time - транзакция время
- week_no - № недели
- coupon_disc - купон описание
- coupon_match_disc - совпадение купонов

transaction_data.csv

- household_key - домохозяйство ключ
- BASKET_ID - идентификатор корзины
- DAY - день
- PRODUCT_ID - идентификатор продукта
- QUANTITY - количество
- SALES_VALUE - значение продаж
- STORE_ID - идентификатор хранилища
- RETAIL_DISC - розничные продажи описание
- TRANS_TIME - транзакция время
- WEEK_NO - № недели
- COUPON_DISC - купон описание
- COUPON_MATCH_DISC - совпадение купонов

product.csv

- item_id - идентификатор продукта
- manufacturer - производитель
- department - отдел

- brand - бренд
- commodity_desc - товар описание
- sub_commodity_desc - описание подраздела товара
- curr_size_of_product

hh_demographic.csv

- age_desc - возраст
- marital_status_code - код семейного положения
- income_desc - доход
- homeowner_desc - домовладелец
- hh_comp_desc - домохозяйство комплексное описание
- household_size_desc - размер домохозяйства
- kid_category_desc - категория для детей
- user_id - идентификатор пользователя

retail_train.csv

- **user_id**: [2375 1364 1130 ... 1077 1581 1984]
- **basket_id**: [26984851472 26984851516 26984896261 ... 41655820646 41655829421 41656790510]
- **day**: [1 2 3 4 ... 659 660 661 662 663]
- **item_id**: [1004906 1033142 1036325 ... 13217063 13217800 6430664]
- **quantity**: [1 2 3 ... 8497 14592 22451]
- **sales_value**: [1.39 0.82 0.99 ... 49.52 34.24 111.92]
- **store_id**: [364 31742 31642 412 337 ... 3179 670 246 3047 3385]
- **retail_disc**: [-0.6 0. -0.3 ... -26.24 -17.78 -19.22]
- **trans_time**: [1631 1642 1520 ... 307 313 433]
- **week_no**: [1 2 3 4 ... 90 91 92 93 94 95]
- **coupon_disc**: [0. -1. -0.4 -0.75 -0.59 -0.55 ... -4.37 -8.95 -0.43]
- **coupon_match_disc**: [0. -0.4 -0.25 -0.45 ... -3. -1.75 -5.8 -1.4 -1.6 -1.15]

transaction_data.csv

- **household_key**: [2375 1364 1130 ... 1581 1984 2325]
- **BASKET_ID**: [26984851472 26984851516 26984896261 ... 42302712298 42305362497 42305362535]
- **DAY**: [1 2 3 4 ... 707 708 709 710 711]
- **PRODUCT_ID**: [1004906 1033142 1036325 ... 133449 6923644 14055192]
- **QUANTITY**: [1 2 3 ... 8859 21207 3989]
- **SALES_VALUE**: [1.39 0.82 0.99 ... 44.14 41.09 33.04]
- **STORE_ID**: [364 31742 31642 412 ... 180 213 1128 411 576 1084]
- **RETAIL_DISC**: [-0.6 0. -0.3 ... -21.49 -22.54 -27.02]
- **TRANS_TIME**: [1631 1642 1520 ... 307 313 433]
- **WEEK_NO**: [1 2 3 4 ... 99 100 101 102]
- **COUPON_DISC**: [0. -1. -0.4 -0.75 -0.59 -0.55 -2. ... -4.38 -7.75 -0.83 -17.24 -9.49]
- **COUPON_MATCH_DISC**: [0. -0.4 -0.25 -0.45 -0.75 -0.5 ... -1.4 -1.6 -1.15 -2.4]

hh_demographic.csv

- **AGE_DESC**: ['65+' '45-54' '25-34' '35-44' '19-24' '55-64']
- **MARITAL_STATUS_CODE**: ['A' 'U' 'B']
- **INCOME_DESC**: ['35-49K' '50-74K' '25-34K' '75-99K' 'Under 15K' '100-124K' '15-24K' '125-149K' '150-174K' '250K+' '175-199K' '200-249K']
- **HOMEOWNER_DESC**: ['Homeowner' 'Unknown' 'Renter' 'Probable Renter' 'Probable Owner']
- **HH_COMP_DESC**: ['2 Adults No Kids' '2 Adults Kids' 'Single Female' 'Unknown' 'Single Male' '1 Adult Kids']
- **HOUSEHOLD_SIZE_DESC**: ['2' '3' '4' '1' '5+']
- **KID_CATEGORY_DESC**: ['None/Unknown' '1' '2' '3+']
- **household_key**: [1 7 8 13 16 17 18 19 20 22 25 27 31 39 ...]

product.csv

- **PRODUCT_ID**: [25671 26081 26093 ... 18293696 18294080 18316298]
- **MANUFACTURER**: [2 69 16 ... 2748 4868 2227]
- **DEPARTMENT**: ['GROCERY' 'MISC. TRANS.' 'PASTRY' 'DRUG GM' 'MEAT-PCKGD' ... 'PHOTO' 'VIDEO' 'PHARMACY SUPPLY']
- **BRAND**: ['National' 'Private']
- **COMMODITY_DESC**: ['FRZN ICE' 'NO COMMODITY DESCRIPTION' 'BREAD' ...]
- **SUB_COMMODITY_DESC**: ['ICE - CRUSHED/CUBED' 'NO SUBCOMMODITY DESCRIPTION' ... 'ROSES OTHER']
- **CURR_SIZE_OF_PRODUCT**: ['22 LB' ' ' '50 OZ' ... '6.3 IN' '35 LD' '2 LTR PET']

In [62]:
```
df_ranker_train.head()
```

Out[62]:

	user_id	item_id	target	manufacturer	department	brand	commodity_desc	sub_commodity_desc	curr_size_of_product	age_desc	marital_s
0	2070	1105426	0.0	69	DELI	Private	SANDWICHES	SANDWICHES - (COLD)		45-54	
1	2070	1097350	0.0	2468	GROCERY	National	DOMESTIC WINE	VALUE GLASS WINE	4 LTR	45-54	
2	2070	879194	0.0	69	DRUG GM	Private	DIAPERS & DISPOSABLES	BABY DIAPERS	14 CT	45-54	
3	2070	948640	0.0	1213	DRUG GM	National	ORAL HYGIENE PRODUCTS	WHITENING SYSTEMS	3 OZ	45-54	
4	2070	928263	0.0	69	DRUG GM	Private	DIAPERS & DISPOSABLES	BABY DIAPERS	13 CT	45-54	

Фичи user_id:

In [63]:
```
# 1. Находим средний чек по корзине для пользователя
mean_basket_checks = \
    data.fillna(value=0).groupby(['basket_id'])['sales_value'].mean().reset_index()

mean_basket_checks.rename(columns={'sales_value': 'mean_basket_check'}, inplace=True)
# mean_basket_checks.set_index(['basket_id'], append=True)
mean_basket_checks.head()
```

Out[63]:

	basket_id	mean_basket_check
0	26984851472	1.182000
1	26984851516	2.071667
2	26984896261	2.274000
3	26984905972	0.510000
4	26984945254	1.176667

In [64]:
```
# 2. Находим средний чек для пользователя
mean_user_checks = data.fillna(value=0).groupby(['user_id'])['sales_value'].mean().reset_index()

mean_user_checks.rename(columns={'sales_value': 'mean_user_check'}, inplace=True)
mean_user_checks.head()
```

Out[64]:

	user_id	mean_user_check
0	1	2.492077
1	2	2.783893
2	3	2.918223
3	4	3.987076
4	5	3.420502

In [65]:
```
# 3. Находим средний чек для пользователя за каждую неделю
mean_basket_week_checks = \
    data.fillna(value=0).groupby(['user_id', 'week_no'])['sales_value'].mean().reset_index()

mean_basket_week_checks.rename(columns={'sales_value': 'mean_basket_week_check'}, inplace=True)
mean_basket_week_checks.head()
```

Out[65]:

	user_id	week_no	mean_basket_week_check
0	1	8	2.622000
1	1	10	3.425000
2	1	13	2.241667
3	1	14	2.757826
4	1	15	3.144118

In [66]:
```
# 3. Находим средний чек для пользователя в неделю
mean_user_week_checks = \
    (data.fillna(value=0).groupby(['user_id'])['sales_value'].sum()
    /len(data['week_no'].unique())).reset_index()

mean_user_week_checks.rename(columns={'sales_value': 'mean_user_week_check'}, inplace=True)
mean_user_week_checks.head()
```

Out[66]:

	user_id	mean_user_week_check

	user_id	mean_user_week_check
0	1	41.683263
1	2	19.194211
2	3	27.308421
3	4	12.632737

In [67]:
```
df_ranker_train = df_ranker_train.merge(mean_user_checks, on='user_id', how='left')
df_ranker_train = df_ranker_train.merge(mean_user_week_checks, on=['user_id'], how='left')

df_ranker_train.head(2)
```

Out[67]:

	user_id	item_id	target	manufacturer	department	brand	commodity_desc	sub_commodity_desc	curr_size_of_product	age_desc	marital_s
0	2070	1105426	0.0	69	DELI	Private	SANDWICHES	SANDWICHES - (COLD)		45-54	
1	2070	1097350	0.0	2468	GROCERY	National	DOMESTIC WINE	VALUE GLASS WINE	4 LTR	45-54	

Фичи пары item_id

In [68]:
```
# Средний чек товара в корзине, среднее количество товара в корзине
mean_basket_item_checks = data.fillna(value=0).groupby(['item_id']) \
        [['sales_value', 'quantity']].mean().reset_index()


mean_basket_item_checks.rename(columns={'sales_value': 'item_check', 'quantity': 'item_quantity'}, inplace=
mean_basket_item_checks.head()
```

Out[68]:

	item_id	item_check	item_quantity
0	25671	6.98	2.0
1	26081	0.99	1.0
2	26093	1.59	1.0
3	26190	1.54	1.0
4	26355	1.98	2.0

In [69]:
```
# Цена товара
item_prices = (data.fillna(value=0).groupby(['item_id']) \
        ['sales_value'].mean()/data.fillna(value=0).groupby(['item_id']) \
        ['quantity'].mean()).reset_index()


item_prices.rename(columns={0: 'item_price'}, inplace=True)

item_prices.head(10)
```

Out[69]:

	item_id	item_price
0	25671	3.49
1	26081	0.99
2	26093	1.59
3	26190	1.54
4	26355	0.99
5	26426	2.29
6	26540	0.93
7	26601	7.59
8	26636	2.50
9	26691	2.79

In [70]:
```
df_ranker_train = df_ranker_train.merge(mean_basket_item_checks, on='item_id', how='left')
df_ranker_train = df_ranker_train.merge(item_prices, on=['item_id'], how='left')

df_ranker_train.head(2)
```

Out[70]:

	user_id	item_id	target	manufacturer	department	brand	commodity_desc	sub_commodity_desc	curr_size_of_product	age_desc	...	incor
0	2070	1105426	0.0	69	DELI	Private	SANDWICHES	SANDWICHES - (COLD)		45-54	...	
1	2070	1097350	0.0	2468	GROCERY	National	DOMESTIC WINE	VALUE GLASS WINE	4 LTR	45-54	...	

2 rows × 21 columns

Фичи пары user_id - item_id

In [71]:
```
# Средняя сумма покупки 1 товара, среднее количество товара в корзине пользователя
sum_sale_quantities = data.fillna(value=0).groupby(['user_id', 'item_id']) \
        [['sales_value', 'quantity']].mean().reset_index()

sum_sale_quantities.rename(columns={'sales_value': 'sum_sale', 'quantity': 'item_quantity'}, inplace=True)
sum_sale_quantities.head(10)
```

Out[71]:

	user_id	item_id	sum_sale	item_quantity
0	1	819312	5.670000	1.000000
1	1	820165	1.750000	3.500000
2	1	821815	3.380000	2.000000
3	1	821867	0.690000	1.000000
4	1	823721	2.990000	1.000000
5	1	823990	6.700000	1.000000
6	1	825123	3.993333	1.333333
7	1	826695	2.690000	1.000000
8	1	827656	3.670000	1.000000
9	1	827671	1.490000	1.000000

In [72]:
```
# Средняя сумма покупки каждой категории
mean_user_item_checks = data.fillna(value=0).groupby(['item_id', 'user_id']) \
        [['sales_value']].mean().reset_index()

mean_user_item_checks.rename(columns={'sales_value': 'user_item_sales_value'}, inplace=True)
mean_user_item_checks.head()
```

Out[72]:

	item_id	user_id	user_item_sales_value
0	25671	325	13.96
1	25671	358	3.49
2	25671	1228	3.49
3	26081	1675	0.99
4	26093	1032	1.59

In [73]:
```
# (Средний чек на конкретной категории купленный юзером в неделю) /
# (Средний чек купленный всеми юзерами конкретной категории в неделю)

user_item_week_ratios = (
      ( data.fillna(value=0).groupby(['item_id', 'user_id', 'week_no'])['sales_value'].mean() /
        len(data['week_no'].unique()) ) /
      ( data.fillna(value=1).groupby(['item_id', 'week_no'])['sales_value'].mean() /
        len(data['week_no'].unique()) )
).reset_index()

user_item_week_ratios.rename(columns={'sales_value': 'user_item_week_ratio'}, inplace=True)
# mean_basket_check.loc[mean_basket_check['sales_value'] > 1]
user_item_week_ratios.tail(10)
```

Out[73]:

	item_id	week_no	user_id	user_item_week_ratio
2325493	17974317	95	2094	0.825743
2325494	17991689	95	2317	1.000000
2325495	17991691	95	2317	1.000000
2325496	18000012	95	1680	0.750000

	item_id	week_no	user_id	user_item_week_ratio
2325497	18000012	95	1769	1.500000
2325498	18000012	95	1906	0.750000
2325499	18024155	95	1186	1.000000
2325500	18024556	95	1797	1.000000
2325501	18024556	95	2070	1.000000

In [74]:
```
#  (Средний чек на конкретной категории купленный юзером в неделю) /
# (Среднее кол-во покупок конкретной категории всеми юзерами в неделю)

user_item_week_check = (data.fillna(value=0).groupby(['item_id', 'user_id', 'week_no']) \
        ['sales_value'].mean()/data.fillna(value=1).groupby(['item_id', 'week_no']) \
        ['quantity'].mean()).reset_index()


user_item_week_check.rename(columns={0: 'user_item_week_checks'}, inplace=True)

user_item_week_check.head()
```

Out[74]:

	item_id	week_no	user_id	user_item_week_checks
0	25671	23	1228	3.49
1	25671	36	358	3.49
2	25671	59	325	3.49
3	26081	36	1675	0.99
4	26093	66	1032	1.59

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?

Подготавливаем фичи для обучения модели

In [75]:
```
df_ranker_train = df_ranker_train.merge(mean_user_checks, on='user_id', how='left')
df_ranker_train = df_ranker_train.merge(mean_user_week_checks, on=['user_id'], how='left')

df_ranker_train = df_ranker_train.merge(mean_basket_item_checks, on='item_id', how='left')
df_ranker_train = df_ranker_train.merge(item_prices, on=['item_id'], how='left')

df_ranker_train = df_ranker_train.merge(user_item_week_ratios, on=['item_id', 'user_id'], how='left')
df_ranker_train = df_ranker_train.merge(user_item_week_check, on=['user_id', 'item_id'], how='left')

df_ranker_train.head(2)
```

Out[75]:

	user_id	item_id	target	manufacturer	department	brand	commodity_desc	sub_commodity_desc	curr_size_of_product	age_desc	...	item_p
0	2070	1105426	0.0	69	DELI	Private	SANDWICHES	SANDWICHES - (COLD)		45-54	...	3.9
1	2070	1105426	0.0	69	DELI	Private	SANDWICHES	SANDWICHES - (COLD)		45-54	...	3.9

2 rows × 30 columns

In [76]:
```
# df_ranker_train.to_csv('../data/df_ranker_train.csv', chunksize=100000)
```

In [77]:
```
# df_ranker_train = pd.read_csv('../data/df_ranker_train.csv')
# df_ranker_train.head(2)
```

In [78]:
```
X_train = df_ranker_train.drop('target', axis=1)
y_train = df_ranker_train[['target']]
```

In [79]:
```
cat_feats = X_train.columns[2:].tolist()
X_train[cat_feats] = X_train[cat_feats].astype('category')

cat_feats
```

Out[79]:

```
['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',
 'mean_user_check_x',
 'mean_user_week_check_x',
 'item_check_x',
 'item_quantity_x',
 'item_price_x',
 'mean_user_check_y',
 'mean_user_week_check_y',
 'item_check_y',
 'item_quantity_y',
 'item_price_y',
 'week_no_x',
 'user_item_week_ratio']
```

Обучение модели ранжирования

In [80]:
```
# Модель второго уровня
from lightgbm import LGBMClassifier
```

In [81]:
```
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/sil/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/_label.py:235: DataConversionWarnin
g: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
  y = column_or_1d(y, warn=True)
/home/sil/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/_label.py:268: DataConversionWarnin
g: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,
), for example using ravel().
  y = column_or_1d(y, warn=True)
/home/sil/anaconda3/lib/python3.7/site-packages/lightgbm/basic.py:1513: UserWarning: categorical_column in
param dict is overridden.
  _log_warning(f'{cat_alias} in param dict is overridden.')
```

In [82]:
```
df_ranker_predict = df_ranker_train.copy()
```

In [83]:
```
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 [84]:
```
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[84]:

	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 [85]:
```
%%time
result_eval_ranker['own_rec'] = result_eval_ranker[USER_COL].apply(lambda x: recommender.get_own_recommendat
```

```
CPU times: user 11.6 s, sys: 20 ms, total: 11.6 s
Wall time: 11.9 s
```

In [86]:
```
# померяем precision только модели матчинга, чтобы понимать влияение ранжирования на метрики

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

Out[86]:
```
[('own_rec', 0.1444117647058813)]
```

Eval re-ranked matched result on test dataset

Вспомним df_match_candidates сет, который был получен own_recommendations на юзерах, набор пользователей мы фиксировали и он одинаков, значи и прогноз одинаков, поэтому мы можем использовать этот дафрейм для переранжирования.

In [87]:
```
def rerank(user_id):
    return df_ranker_predict[df_ranker_predict[USER_COL]==user_id].sort_values('proba_item_purchase', ascend
```

In [88]:
```
result_eval_ranker['reranked_own_rec'] = result_eval_ranker[USER_COL].apply(lambda user_id: rerank(user_id)
```

In [89]:
```
print(*sorted(calc_precision(result_eval_ranker, TOPK_PRECISION), key=lambda x: x[1], reverse=True), sep='\n
```

```
('own_rec', 0.1444117647058813)
('reranked_own_rec', 0.1377545691905989)

/home/sil/ML/RS/Lesson_6/HW/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

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?

Вывод:

После добавления по 2 фичи для юзера, товара и пары юзер-товар значение precision@5 двухуровневой модели на data_val_ranker = 0.1444117647058813, не выросло по сравнению с precision@5 модели 1-ого уровня = 0.17712691771268974.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: