

Рекомендательные системы

Курсовой проект

1. На основе данного notebook-а построить модели 1-го и модели 2-го уровня (градиентный бустинг).
2. Нужно для всех юзеров из файла **retail_test1.csv** выдать рекомендации, и посчитать на **actual** покупках **precision@5**.
3. Сдать ссылку на github с решением. В решении должны быть отчетливо видна метрика на новом тестовом сете из файла **retail_test1.csv** (см. п.3.).
4. Бейзлайн решения - [MainRecommender \(https://github.com/geangohn/recsys-tutorial/blob/master/src/recommenders.py\)](https://github.com/geangohn/recsys-tutorial/blob/master/src/recommenders.py)
5. Не рассматриваем холодный старт для пользователя, все наши пользователи одинаковы во всех сетях, поэтому нужно позаботиться об их исключении из теста.
6. **Цель:** Добиться для **precision@5** максимально возможного значения.

Import libs

```
B [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

```
B [2]: PATH_DATA = ".././data"

B [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'))
```

Set global const

```
B [4]: ITEM_COL = 'item_id'
USER_COL = 'user_id'
ACTUAL_COL = 'actual'

# N = Neighbors
N_PREDICT = 50
```

Process features dataset

```
B [5]: # 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)
```

B [6]: item_features.head(2)

Out[6]:

	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	

B [7]: user_features.head(2)

Out[7]:

	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

B [8]: data.head(2)

Out[8]:

	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

Split dataset for train, eval, test

B [9]: *# 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

B [10]: *# берем данные для тренировки matching модели (1-й уровень)*
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]

B [11]: *# сделаем объединенный сет данных для первого уровня (матчинга)*
df_join_train_matcher = pd.concat([data_train_matcher, data_val_matcher])

B [12]: df_join_train_matcher.tail(2)

Out[12]:

	user_id	basket_id	day	item_id	quantity	sales_value	store_id	retail_disc	trans_time	week_no	coupon_disc	coupon_match_disc
2282323	462	41297773713	635	10180324	1	3.00	304	-0.29	2040	91	0.0	0.0
2282324	462	41297773713	635	12731714	1	4.08	304	0.00	2040	91	0.0	0.0

B [13]: *# data_train_matcher.head(2)*

B [14]: *# data_val_matcher.head(2)*

B [15]: **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()}")

B [16]: 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

Выше видим разброс по пользователям и товарам (?) и дальше мы перейдем к warm-start (только известные пользователи)

```
B [17]: data_val_matcher.head(2)
```

```
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
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

```
B [ ]:
```

Prefilter items

```
B [18]: # from utils import prefilter_items
```

```
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_8/course_project_rs/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://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
data['price'] = data['sales_value'] / (np.maximum(data['quantity'], 1))
```

```
Decreased # items from 83685 to 5001
```

Make cold-start to warm-start

```
B [19]: # 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

```
B [20]: # from recommenders import MainRecommender
```

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

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

```
HBox(children=(FloatProgress(value=0.0, max=15.0), HTML(value='')))
```

```
HBox(children=(FloatProgress(value=0.0, max=5001.0), HTML(value='')))
```

```
B [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

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

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

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

```
B [23]: # Берем тестового юзера 2375
```

```
B [24]: recommender.get_als_recommendations(2375, N=5)
```

```
Out[24]: [899624, 1044078, 871756, 844179, 12301839]
```

```
B [25]: recommender.get_own_recommendations(2375, N=5)
```

```
Out[25]: [948640, 918046, 847962, 907099, 873980]
```

```
B [26]: recommender.get_similar_items_recommendation(2375, N=5)
```

```
Out[26]: [1046545, 1044078, 999270, 934399, 15778319]
```

```
B [27]: recommender.get_similar_users_recommendation(2375, N=5)
```

```
Out[27]: [1133654, 835618, 861494, 835351, 1096573]
```

Eval recall of matching

Измеряем recall@k

Это будет в ДЗ:

А) 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 recommendations + top-popular лучший recall?

В)* Как зависит recall@k от k? Постройте для одной схемы генерации кандидатов эту зависимость для k = {20, 50, 100, 200, 500}

С)* Исходя из прошлого вопроса, как вы думаете, какое значение k является наиболее разумным?

```
B [28]: 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[28]:
```

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

```
B [29]: %%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=N_PREDICT))

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

CPU times: user 1min 35s, sys: 7.82 s, total: 1min 43s
Wall time: 32.3 s

```
B [30]: N_PREDICT
```

Out[30]: 50

```
B [31]: %%time
# result_eval_matcher['sim_user_rec'] = result_eval_matcher[USER_COL]. \
#     apply(lambda x: recommender.get_similar_users_recommendation(x, N=N_PREDICT))

CPU times: user 8 µs, sys: 0 ns, total: 8 µs
Wall time: 6.2 µs
```

Пример оборачивания

```
B [32]: # # 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()
```

```
B [33]: evalRecall(result_eval_matcher, USER_COL, recommender.get_own_recommendations)
```

Out[33]: 0.044119547395835505

```
B [34]: 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()
```

```
B [35]: 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()
```

Recall@50 of matching

```
B [36]: TOPK_RECALL = 50
```

```
B [37]: sorted(calc_recall(result_eval_matcher, TOPK_RECALL), key=lambda x: x[1],reverse=True)
```

Out[37]: [('own_rec', 0.06525657038145175),
('als_rec', 0.04746155729859833),
('result', 0.044119547395835505),
('sim_item_rec', 0.03464680379050876)]

Precision@5 of matching

```
B [38]: TOPK_PRECISION = 5
```

```
B [39]: sorted(calc_precision(result_eval_matcher, TOPK_PRECISION), key=lambda x: x[1],reverse=True)
```

Out[39]: [('own_rec', 0.17712691771268974),
('result', 0.17712691771268974),
('als_rec', 0.11585309158530817),
('sim_item_rec', 0.06062296606229701)]

```
B [40]: result_eval_matcher.head(2)
```

Out[40]:

	user_id	actual	own_rec	sim_item_rec	als_rec	result
0	1	[853529, 865456, 867607, 872137, 874905, 87524...]	[856942, 9297615, 5577022, 877391, 9655212, 88...]	[824758, 1007512, 9297615, 5577022, 9803545, 9...]	[976214, 5577022, 962615, 1119942, 5569374, 11...]	[856942, 9297615, 5577022, 877391, 9655212, 88...]
1	2	[15830248, 838136, 839656, 861272, 866211, 870...]	[911974, 1076580, 1103898, 5567582, 1056620, 9...]	[8090537, 5569845, 1044078, 985999, 880888, 81...]	[5569230, 866211, 916122, 1029743, 1127831, 97...]	[911974, 1076580, 1103898, 5567582, 1056620, 9...]

Наиболее разумным является значение k=50 (по результатам дз 6-го урока).

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 каидиатов через get_own_recommendations
- (!) Если юзер купил < 50 товаров, то get_own_recommendations дополнит рекомендации топ-популярными

```
B [41]: # 3 временных интервала
# -- давние покупки -- | -- 6 недель -- | -- 3 недель --
```

Подготовка данных для трейна

```
B [42]: # взяли пользователей из трейна для ранжирования
df_match_candidates = pd.DataFrame(data_train_ranker[USER_COL].unique())
df_match_candidates.columns = [USER_COL]
```

```
B [43]: # собираем кандидатов с первого этапа (matcher)
df_match_candidates['candidates'] = df_match_candidates[USER_COL]. \
    apply(lambda x: recommender.get_own_recommendations(x, N=N_PREDICT))
```

```
B [44]: df_match_candidates.head(2)
```

Out[44]:

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

```
B [45]: # разворачиваем товары
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'
```

```
B [46]: df_match_candidates = df_match_candidates.drop('candidates', axis=1).join(df_items)
```

```
B [47]: df_match_candidates.head(4)
```

Out[47]:

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

Check warm start

```
B [48]: 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

```
B [49]: df_ranker_train = data_train_ranker[[USER_COL, ITEM_COL]].copy()
df_ranker_train['target'] = 1 # тут только покупки
```

```
B [50]: df_ranker_train.head(3)
```

Out[50]:

	user_id	item_id	target
2104867	2070	1019940	1
2107468	2021	840361	1
2107469	2021	856060	1

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


```
B [51]: 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)
```

```
B [52]: df_ranker_train.target.value_counts()
```

```
Out[52]: 0.0    99177
         1.0     7795
         Name: target, dtype: int64
```

```
B [53]: df_ranker_train.head(2)
```

```
Out[53]:
```

	user_id	item_id	target
0	2070	1105426	0.0
1	2070	1097350	0.0

(!) На каждого юзера 50 item_id-кандидатов

```
B [54]: df_ranker_train['target'].mean()
```

```
Out[54]: 0.07286953595333358
```

```
B [ ]:
```

Подготавливаем фичи для обучения модели

```
B [55]: item_features.head(2)
```

```
Out[55]:
```

	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	

```
B [56]: user_features.head(2)
```

```
Out[56]:
```

	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

```
B [57]: 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[57]:
```

	user_id	item_id	target	manufacturer	department	brand	commodity_desc	sub_commodity_desc	curr_size_of_product	age_desc	marital_sta
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)

Поведенческие фичи

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

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

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]

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']
- **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']

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 ...]


```
B [58]: df_ranker_train.head(2)
```

Out[58]:

	user_id	item_id	target	manufacturer	department	brand	commodity_desc	sub_commodity_desc	curr_size_of_product	age_desc	marital_sta
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:

```
B [59]: # 2. Находим средний чек для пользователя
mean_user_checks = df_join_train_matcher.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[59]:

	user_id	mean_user_check
0	1	2.494884
1	2	2.783893
2	3	2.936759
3	4	3.987076
4	5	3.420502

```
B [60]: # 4. Находим средний чек для пользователя в неделю
mean_user_week_checks = \
    (df_join_train_matcher.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[60]:

	user_id	mean_user_week_check
0	1	39.576737
1	2	19.194211
2	3	26.894526
3	4	12.632737
4	5	7.885158

Фичи пары item_id

```
B [61]: # Средний чек товара в корзине, среднее количество товара в корзине
mean_basket_item_checks = df_join_train_matcher.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=True)
mean_basket_item_checks.head()
```

Out[61]:

	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

```

В [62]: # Цена товара
item_prices = (df_join_train_matcher.fillna(value=0).groupby(['item_id']) \
               ['sales_value'].mean()/df_join_train_matcher.fillna(value=0).groupby(['item_id']) \
               ['quantity'].mean()).reset_index()

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

```

```

Out[62]:
   item_id  item_price
0    25671         3.49
1    26081         0.99
2    26093         1.59
3    26190         1.54
4    26355         0.99

```

Фичи пары user_id - item_id

```

В [63]: # (Средний чек на конкретной категории купленный юзером в неделю) /
# (Средний чек купленный всеми юзерами конкретной категории в неделю)

user_item_week_ratios = (
    (df_join_train_matcher.fillna(value=0).groupby(['item_id', 'user_id', 'week_no'])['sales_value'].mean() /
      len(df_join_train_matcher['week_no'].unique())) /
    (df_join_train_matcher.fillna(value=1).groupby(['item_id', 'week_no'])['sales_value'].mean() /
      len(df_join_train_matcher['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(2)

```

```

Out[63]:
   item_id  week_no  user_id  user_item_week_ratio
2210411  17829232      91    1163                1.0
2210412  17829232      91    1596                1.0

```

```

В [64]: # 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[64]:
   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

```

```

В [65]: # 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[65]:
   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

```

```
B [66]: # Даная фича резко снижает метрику. В чём дело?
# (Средний чек на конкретной категории купленный юзером в неделю) /
# (Среднее кол-во покупок конкретной категории всеми юзерами в неделю)
user_item_week_check = (df_join_train_matcher.fillna(value=0).groupby(['item_id', 'user_id', 'week_no']) \
                        ['sales_value'].mean()/df_join_train_matcher.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(2)
```

Out[66]:

	item_id	week_no	user_id	user_item_week_checks
0	25671	23	1228	3.49
1	25671	36	358	3.49

Подготавливаем фичи для обучения модели

```
B [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 = 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(sum_sale_quantities, on=['item_id', 'user_id'], how='left')
df_ranker_train = df_ranker_train.merge(mean_user_item_checks, on=['user_id', 'item_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[67]:

	user_id	item_id	target	manufacturer	department	brand	commodity_desc	sub_commodity_desc	curr_size_of_product	age_desc	...	househ
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 × 24 columns

```
B [ ]:
```

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

```

B [68]: # df_ranker_train = df_ranker_train.merge(df_join_train_matcher. \
#                                               groupby(by=ITEM_COL).agg('sales_value').sum(). \
#                                               rename('total_item_sales_value'), how='left',on=ITEM_COL)

# df_ranker_train = df_ranker_train.merge(df_join_train_matcher. \
#                                         groupby(by=ITEM_COL).agg('quantity').sum(). \
#                                         rename('total_quantity_value'), how='left',on=ITEM_COL)

# df_ranker_train = df_ranker_train.merge(df_join_train_matcher. \
#                                         groupby(by=ITEM_COL).agg(USER_COL).count(). \
#                                         rename('item_freq'), how='left',on=ITEM_COL)

# df_ranker_train = df_ranker_train.merge(df_join_train_matcher. \
#                                         groupby(by=USER_COL).agg(USER_COL).count(). \
#                                         rename('user_freq'), how='left',on=USER_COL)

# df_ranker_train = df_ranker_train.merge(df_join_train_matcher. \
#                                         groupby(by=USER_COL).agg('sales_value').sum(). \
#                                         rename('total_user_sales_value'), how='left',on=USER_COL)

# df_ranker_train = df_ranker_train.merge(df_join_train_matcher. \
#                                         groupby(by=ITEM_COL).agg('quantity').sum(). \
#                                         rename('item_quantity_per_week')/df_join_train_matcher.week_no.nunique(), how=

# df_ranker_train = df_ranker_train.merge(df_join_train_matcher. \
#                                         groupby(by=USER_COL).agg('quantity').sum(). \
#                                         rename('user_quantity_per_week') / df_join_train_matcher.week_no.nunique(), \
#                                         how='left',on=USER_COL)

# df_ranker_train = df_ranker_train.merge(df_join_train_matcher. \
#                                         groupby(by=ITEM_COL).agg('quantity').sum(). \
#                                         rename('item_quantity_per_basket')/df_join_train_matcher.basket_id.nunique(),
#                                         how='left',on=ITEM_COL)

# df_ranker_train = df_ranker_train.merge(df_join_train_matcher. \
#                                         groupby(by=USER_COL).agg('quantity').sum(). \
#                                         rename('user_quantity_per_baskter')/df_join_train_matcher.basket_id.nunique(),
#                                         how='left',on=USER_COL)

# df_ranker_train = df_ranker_train.merge(df_join_train_matcher. \
#                                         groupby(by=ITEM_COL).agg(USER_COL).count(). \
#                                         rename('item_freq_per_basket')/df_join_train_matcher.basket_id.nunique(), \
#                                         how='left',on=ITEM_COL)

# df_ranker_train = df_ranker_train.merge(df_join_train_matcher. \
#                                         groupby(by=USER_COL).agg(USER_COL).count(). \
#                                         rename('user_freq_per_basket')/df_join_train_matcher.basket_id.nunique(), \
#                                         how='left',on=USER_COL)

```

```

B [69]: df_ranker_train.head(2)

```

```

Out[69]:

```

	user_id	item_id	target	manufacturer	department	brand	commodity_desc	sub_commodity_desc	curr_size_of_product	age_desc	...	househ
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 × 24 columns

```

B [70]: X_train = df_ranker_train.drop('target', axis=1)
y_train = df_ranker_train[['target']]

```

```

B [71]: cat_feats = X_train.columns[2:].tolist() # ?

```

```

B [72]: X_train[cat_feats] = X_train[cat_feats].astype('category')

# cat_feats

```

Обучение модели ранжирования

```

B [73]: # Модель второго уровня
from lightgbm import LGBMClassifier

```

```
B [74]: lgb = LGBMClassifier(objective='binary',
                           max_depth=10, # Глубина дерева
                           n_estimators=500, # Количество деревьев
                           # boosting='dart',
                           # num_iterations=200,
                           learning_rate=0.5,
                           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: DataConversionWarning: A column-ve
ctor 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: DataConversionWarning: A column-ve
ctor 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 i
s overridden.
  _log_warning(f'{cat_alias} in param dict is overridden.')
```

```
B [75]: df_ranker_predict = df_ranker_train.copy()
```

```
B [76]: df_ranker_predict['proba_item_purchase'] = train_preds[:,1]
```

Подведем итоги

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

Evaluation on test dataset

```
B [77]: 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[77]:
```

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

Eval matching on test dataset

```
B [78]: %%time
result_eval_ranker['own_rec'] = result_eval_ranker[USER_COL]. \
    apply(lambda x: recommender.get_own_recommendations(x, N=N_PREDICT))
```

```
CPU times: user 7.47 s, sys: 19.3 ms, total: 7.49 s
Wall time: 7.57 s
```

```
B [79]: # померяем precision только модели матчинга, чтобы понимать влияние ранжирования на метрики

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

```
Out[79]: [('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, значит и прогноз одинаков, поэтому мы можем использовать этот датафрейм для переранжирования.

```
B [80]: def rerank(user_id):
        return df_ranker_predict[df_ranker_predict[USER_COL]== \
                                user_id].sort_values('proba_item_purchase', ascending=False).head(5).item_id.tolist()
```

```
B [81]: result_eval_ranker['reranked_own_rec'] = result_eval_ranker[USER_COL].apply(lambda user_id: rerank(user_id))
```

```
B [82]: result_eval_ranker.head(3)
```

Out[82]:

	user_id	actual	own_rec	reranked_own_rec
0	1	[821867, 834484, 856942, 865456, 889248, 90795...	[856942, 9297615, 5577022, 877391, 9655212, 88...	[9655212, 9297615, 8091601, 8293439, 979792]
1	3	[835476, 851057, 872021, 878302, 879948, 90963...	[1092937, 1008714, 12132312, 1075979, 998206, ...	[]
2	6	[920308, 926804, 946489, 1006718, 1017061, 107...	[13003092, 972416, 995598, 923600, 1138596, 10...	[1098844, 12757544, 900802, 13115971, 897088]

```
B [83]: print(*sorted(calc_precision(result_eval_ranker, TOPK_PRECISION), key=lambda x: x[1], reverse=True), sep='\n')
```

/home/sil/ML/RS/lesson_8/course_project_rs/metrics.py:20: RuntimeWarning: invalid value encountered in long_scalars
return flags.sum() / len(recommended_list)

('reranked_own_rec', 0.23007832898172093)
('own_rec', 0.1444117647058813)

max_depth=8,
n_estimators=300,
learning_rate=0.05

Без фиць:
('reranked_own_rec', 0.15331592689294912)
('own_rec', 0.1444117647058813)

С фичаями из методички
('reranked_own_rec', 0.17221932114882363)
('own_rec', 0.1444117647058813)

С моими фицами
('reranked_own_rec', 0.22036553524803945)
('own_rec', 0.1444117647058813)

Всё вместе (из методички + мои)
('reranked_own_rec', 0.21900783289816986)
('own_rec', 0.1444117647058813)

max_depth=10,
n_estimators=500,
learning_rate=0.1

С моими фицами
('reranked_own_rec', 0.22684073107049363) +
('own_rec', 0.1444117647058813)

(max_depth=4)
('reranked_own_rec', 0.2212010443864205)
('own_rec', 0.1444117647058813)

max_depth=10
learning_rate=0.5
('reranked_own_rec', 0.23007832898172093) +++
('own_rec', 0.1444117647058813)

(learning_rate=0.01)
('reranked_own_rec', 0.21900783289816994)
('own_rec', 0.1444117647058813)

(learning_rate=0.08)
('reranked_own_rec', 0.2253785900783266) +-
('own_rec', 0.1444117647058813)

Оценка на тесте для выполнения курсового проекта

```
B [84]: PATH_DATA
```

Out[84]: '../../data'

```
B [85]: # df_transactions = pd.read_csv('../data/transaction_data.csv')
# df_test = pd.read_csv('retail_test1.csv')
```

```
B [86]: df_test = pd.read_csv(os.path.join(PATH_DATA, 'retail_test1.csv'))
```

```
B [87]: # df_test.head(3)
```

```
B [88]: df_test = df_test[df_test.user_id.isin(common_users)]
df_test.head(3)
```

```
Out[88]:
```

	user_id	basket_id	day	item_id	quantity	sales_value	store_id	retail_disc	trans_time	week_no	coupon_disc	coupon_match_disc
0	1340	41652823310	664	912987	1	8.49	446	0.0	52	96	0.0	0.0
1	588	41652838477	664	1024426	1	6.29	388	0.0	8	96	0.0	0.0
2	2070	41652857291	664	995242	5	9.10	311	-0.6	46	96	0.0	0.0

```
B [89]: result_test = df_test.groupby(USER_COL)[ITEM_COL].unique().reset_index()
result_test.columns=[USER_COL, ACTUAL_COL]
result_test.head(2)
```

```
Out[89]:
```

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

```
B [91]: print_stats_data(df_test, 'test')
```

```
test
Shape: (88665, 12) Users: 1883 Items: 20492
```

Eval matching on test dataset

```
B [92]: %%time
result_test['own_rec'] = result_test[USER_COL]. \
    apply(lambda x: recommender.get_own_recommendations(x, N=N_PREDICT))
```

```
CPU times: user 12.3 s, sys: 104 ms, total: 12.4 s
Wall time: 12.6 s
```

```
B [93]: # померяем precision только модели матчинга, чтобы понимать влияние ранжирования на метрики

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

```
Out[93]: [('own_rec', 0.1227827934147629)]
```

Eval re-ranked matched result on test dataset

```
B [94]: result_test['reranked_own_rec'] = result_test[USER_COL].apply(lambda user_id: rerank(user_id))
```

```
B [95]: print(*sorted(calc_precision(result_test, TOPK_PRECISION), key=lambda x: x[1], reverse=True), sep='\n')
```

```
/home/sil/ML/RS/lesson_8/course_project_rs/metrics.py:20: RuntimeWarning: invalid value encountered in long_scalars
    return flags.sum() / len(recommended_list)
```

```
('reranked_own_rec', 0.17747747747747583)
('own_rec', 0.1227827934147629)
```

Результат:

```
('reranked_own_rec', 0.17747747747747583)
('own_rec', 0.1227827934147629)
```



```
B [96]: result_test.head()
```

Out[96]:

	user_id	actual	own_rec	reranked_own_rec
0	1	[880007, 883616, 931136, 938004, 940947, 94726...	[856942, 9297615, 5577022, 877391, 9655212, 88...	[9655212, 9297615, 8091601, 8293439, 979792]
1	2	[820165, 820291, 826784, 826835, 829009, 85784...	[911974, 1076580, 1103898, 5567582, 1056620, 9...	[1103898, 1076580, 911974, 898847, 9416729]
2	3	[827683, 908531, 989069, 1071377, 1080155, 109...	[1092937, 1008714, 12132312, 1075979, 998206, ...	[]
3	6	[956902, 960791, 1037863, 1119051, 1137688, 84...	[13003092, 972416, 995598, 923600, 1138596, 10...	[1098844, 12757544, 900802, 13115971, 897088]
4	7	[847270, 855557, 859987, 863407, 895454, 90663...	[998519, 894360, 7147142, 9338009, 896666, 939...	[1122358, 9338009, 870882, 993838, 7147142]

```
B [ ]:
```