In [2]:

```
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, coo_matrix

# Детерминированные алгоритмы
from implicit.nearest_neighbours import ItemItemRecommender, CosineRecommender, TFIDFRecomm

# Метрики
from implicit.evaluation import train_test_split
from implicit.evaluation import precision_at_k, mean_average_precision_at_k, AUC_at_k, ndcg
```

In [3]:

```
data = pd.read_csv('data/retail_train.csv')
data.head(2)
```

Out[3]:

	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

In [4]:

```
test_size_weeks = 3

data_train = data[data['week_no'] < data['week_no'].max() - test_size_weeks]
data_test = data[data['week_no'] >= data['week_no'].max() - test_size_weeks]
```

In [5]:

```
popularity = data.groupby('item_id')['sales_value'].sum().reset_index()
popularity[:3]
```

Out[5]:

	item_id	sales_value
0	25671	20.94
1	26081	0.99
2	26093	1.59

In [6]:

```
pb = data_train.groupby('item_id')['sales_value'].sum().reset_index()
pb.columns=['item_id', 'sales']
r_sum = np.sum(pb['sales'])
pb['sales'] = pb['sales']/r_sum
# pb.head(2)
pb['sales'].sum()
```

Out[6]:

```
0.9999999999999999
```

Задание 1. Weighted Random Recommendation

Напишите код для случайных рекоммендаций, в которых вероятность рекомендовать товар прямо пропорциональна логарифму продаж

- Можно сэмплировать товары случайно, но пропорционально какому-либо весу
- Например, прямопропорционально популярности. Вес = log(sales_sum товара)

Бейзлайн

Создадим датафрейм с покупками юзеров на тестовом датасете (последние 3 недели)

In [7]:

```
result = data_test.groupby('user_id')['item_id'].unique().reset_index()
result.columns=['user_id', 'actual']
result.head(2)
```

Out[7]:

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

In [8]:

```
test_users = result.shape[0]
new_test_users = len(set(data_test['user_id']) - set(data_train['user_id']))

print('В тестовом дата сете {} юзеров'.format(test_users))
print('В тестовом дата сете {} новых юзеров'.format(new_test_users))
```

```
В тестовом дата сете 2042 юзеров
В тестовом дата сете 0 новых юзеров
```

In [9]:

```
def weighted_random_recommendation(items,pb, n=5):
    """Случайные рекомендации"""
    pb = np.array(pb)
    items = np.array(items)
    recs = np.random.choice(items, size=n, replace=False, p = pb)

    return recs.tolist()
```

In [10]:

```
%%time

items = data_train.item_id.unique()

result['weighted_random_recommendation'] = result['user_id'].apply(lambda x: weighted_rando

result.head(2)
```

Wall time: 3.57 s

Out[10]:

	user_id	actual	weighted_random_recommendation
0	1	[821867, 834484, 856942, 865456, 889248, 90795...	[484031, 9677881, 1578452, 851146, 12128605]
1	3	[835476, 851057, 872021, 878302, 879948, 90963...	[820499, 990444, 1002739, 6443124, 889551]

Задание 2. Улучшение бейзлайнов и ItemItem

- Попробуйте улучшить бейзлайны, считая случаный на топ-5000 товаров
- Попробуйте улучшить разные варианты ItemItemRecommender, выбирая число соседей K.

Random top

In [11]:

```
popularity = data_train.groupby('item_id')['quantity'].sum().reset_index()
popularity.rename(columns={'quantity': 'n_sold'}, inplace=True)

popularity.head()
```

Out[11]:

	item_id	n_sold
0	25671	6
1	26081	1
2	26093	1
3	26190	1
4	26355	2

In [12]:

```
top_5000 = popularity.sort_values('n_sold', ascending=False).head(5000)#.item_id.tolist()
top_5000
```

Out[12]:

	item_id	n_sold
55470	6534178	190227964
55430	6533889	15978434
55465	6534166	12439291
55576	6544236	2501949
43620	1404121	1562004
...	...	...
51914	5574045	120
10610	864048	119
7777	838842	119
26071	1004001	119
23122	977033	119

5000 rows × 2 columns

In [23]:

```
data_train_top = top_5000.merge(data_train, on='item_id')
```

In [24]:

```
len(data_train_top['item_id'].unique())
```

Out[24]:

```
5000
```

In [15]:

```
len(data_train['item_id'].unique())
```

Out[15]:

```
86865
```

In [25]:

```
def random_recommendation_top(items, n=5):
    """Случайные рекомендации"""

    items = np.array(items)
    recs = np.random.choice(items, size=n, replace=False)

    return recs.tolist()
```

In [26]:

```
%%time

items = data_train_top.item_id.unique()

result['random_recommendation_top'] = result['user_id'].apply(lambda x: random_recommendati

result.head(2)
```

```
Wall time: 404 ms
```

Out[26]:

	user_id	actual	weighted_random_recommendation	random_recommendation_top
0	1	[821867, 834484, 856942, 865456, 889248, 90795...	[484031, 9677881, 1578452, 851146, 12128605]	[943169, 857554, 963686, 9527257, 905539]
1	3	[835476, 851057, 872021, 878302, 879948, 90963...	[820499, 990444, 1002739, 6443124, 889551]	[918733, 9524511, 1075827, 1049920, 904774]

itemitemRecommender

In [27]:

```
top_5000 = popularity.sort_values('n_sold', ascending=False).head(5000).item_id.tolist()
```

In [29]:

```
data_train.head()
```

Out[29]:

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

In [30]:

```
# Заведем фиктивный item_id (если юзер покупал товары из топ-5000, то он "купил" такой това
data_train.loc[ ~ data_train['item_id'].isin(top_5000), 'item_id'] = 6666
data_train.head(100)
```

```
C:\Users\voron\AppData\Roaming\Python\Python37\site-packages\pandas\core\ind
exing.py:1720: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  self._setitem_single_column(loc, value, pi)
```

Out[30]:

	user_id	basket_id	day	item_id	quantity	sales_value	store_id	retail_disc	trans_time
0	2375	26984851472	1	1004906	1	1.39	364	-0.60	1631
1	2375	26984851472	1	1033142	1	0.82	364	0.00	1631
2	2375	26984851472	1	1036325	1	0.99	364	-0.30	1631
3	2375	26984851472	1	1082185	1	1.21	364	0.00	1631
4	2375	26984851472	1	8160430	1	1.50	364	-0.39	1631
...	...	...	...	...	...	...	...	...	...
95	1060	26985040735	1	9553288	1	8.49	315	0.00	1251
96	1351	26985052379	1	903230	1	0.99	447	-0.30	1955
97	744	26985165432	1	6666	0	0.00	31582	0.00	1119
98	212	26985205886	1	822346	1	1.25	288	-0.34	1341
99	212	26985205886	1	830887	1	2.29	288	-0.70	1341

100 rows × 12 columns

In [31]:

```
user_item_matrix = pd.pivot_table(data_train,
                                  index='user_id', columns='item_id',
                                  values='quantity',
                                  aggfunc='count',
                                  fill_value=0
                                 )

user_item_matrix[user_item_matrix > 0] = 1 # так как в итоге хотим предсказать
user_item_matrix = user_item_matrix.astype(float) # необходимый тип матрицы для implicit

# переведем в формат sparse matrix
sparse_user_item = csr_matrix(user_item_matrix).tocsr()
```

In [34]:

```
user_item_matrix.shape
```

Out[34]:

```
(2499, 5001)
```

In [35]:

```
user_item_matrix.sum().sum() / (user_item_matrix.shape[0] * user_item_matrix.shape[1]) * 10
```

Out[35]:

```
5.33770796861036
```

In [36]:

```
userids = user_item_matrix.index.values
itemids = user_item_matrix.columns.values

matrix_userids = np.arange(len(userids))
matrix_itemids = np.arange(len(itemids))

id_to_itemid = dict(zip(matrix_itemids, itemids))
id_to_userid = dict(zip(matrix_userids, userids))

itemid_to_id = dict(zip(itemids, matrix_itemids))
userid_to_id = dict(zip(userids, matrix_userids))
```

In [86]:

```
%%time
K = 1
model = ItemItemRecommender(K=K, num_threads=4) # K - кол-во билжайших соседей


model.fit(csr_matrix(user_item_matrix).T.tocsr(),  # На вход item-user matrix
          show_progress=True)


recs = model.recommend(userid=userid_to_id[2],  # userid - id от 0 до N
                        user_items=csr_matrix(user_item_matrix).tocsr(),   # на вход user-i
                        N=5, # кол-во рекомендаций
                        filter_already_liked_items=False,
                        filter_items=None,
                        recalculate_user=True)
```

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

Wall time: 1.08 s
```

In [87]:

```
recs
```

Out[87]:

```
[(0, 2498.0), (3408, 2155.0), (301, 1284.0), (3587, 1278.0), (111, 1261.0)]
```

In [88]:

```
recs
```

Out[88]:

```
[(0, 2498.0), (3408, 2155.0), (301, 1284.0), (3587, 1278.0), (111, 1261.0)]
```

In [89]:

```
[id_to_itemid[rec[0]] for rec in recs]
```

Out[89]:

```
[6666, 1082185, 840361, 1098066, 826249]
```

In [91]:

```
%%time

result['itemitem'+ f'_{str(K)}'] = result['user_id'].apply(lambda user_id: [id_to_itemid[re
                                    model.recommend(userid=userid_to_id[user_id],
                                    user_items=sparse_user_item,   # на вход user-item matr
                                    N=5,
                                    filter_already_liked_items=False,
                                    filter_items=None,
                                    recalculate_user=True)])
result.head()
```

Wall time: 80 ms

Out[91]:

	user_id	actual	weighted_random_recommendation	random_recommendation_top	itemitem_
0	1	[821867, 834484, 856942, 865456, 889248, 90795...	[484031, 9677881, 1578452, 851146, 12128605]	[943169, 857554, 963686, 9527257, 905539]	[666 108218 98176 112783 99524
1	3	[835476, 851057, 872021, 878302, 879948, 90963...	[820499, 990444, 1002739, 6443124, 889551]	[918733, 9524511, 1075827, 1049920, 904774]	[666 108218 98176 109806 99524
2	6	[920308, 926804, 946489, 1006718, 1017061, 107...	[10204679, 1111343, 957330, 754002, 12262649]	[6632986, 938300, 857638, 951190, 910109]	[666 108218 98176 112783 99524
3	7	[840386, 889774, 898068, 909714, 929067, 95347...	[1315112, 1026340, 1952718, 1079594, 855116]	[1132789, 922636, 900586, 853235, 841220]	[666 108218 98176 112783 99524
4	8	[835098, 872137, 910439, 924610, 992977, 10412...	[826690, 12604177, 1073405, 8181106, 537818]	[908346, 1039126, 1004596, 951221, 1121059]	[666 108218 98176 112783 109806

In [ ]: