Рекомендательные системы

Урок 2. Бейзлайны и детерминированные алгоритмы item-item

Домашнее задание

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)

Задание 2. Улучшение бейзлайнов и ItemItem

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

Выполнил Соковнин ИЛ

В [1]:
```
!pip install implicit
```

```
Requirement already satisfied: implicit in /home/sil/anaconda3/lib/python3.7/site-packages (0.4.8)
Requirement already satisfied: scipy>=0.16 in /home/sil/anaconda3/lib/python3.7/site-packages (from implicit) (1.4.1)
Requirement already satisfied: tqdm>=4.27 in /home/sil/anaconda3/lib/python3.7/site-packages (from implicit) (4.42.1)
Requirement already satisfied: numpy in /home/sil/anaconda3/lib/python3.7/site-packages (from implicit) (1.18.1)
```

В [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, TFIDFRecommender, BM25Recommender

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

В [3]:
```
data = pd.read_csv('../webinar_2/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	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

Train-test split

В рекомендательных системах корректнее использовать train-test split по времени, а не случайно.
Возьмём последние 3 недели в качестве теста.

В [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]
```

1. Бейзлайны

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

In [5]:
```
result = data_test.groupby('user_id')['item_id'].unique().reset_index()
result.columns=['user_id', 'actual']
result.head(2)
```

Out[5]:

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

In [6]:
```
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
```

1.1 Random recommendation

In [7]:
```
def random_recommendation(items, n=5):
    """Случайные рекоммендации"""

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

    return recs.tolist()
```

In [8]:
```
%%time

items = data_train.item_id.unique()

result['random_recommendation'] = result['user_id'].apply(lambda x: random_recommendation(items, n=5))

result.head(2)
```

```
CPU times: user 4.96 s, sys: 260 ms, total: 5.22 s
Wall time: 6.13 s
```

Out[8]:

	user_id	actual	random_recommendation
0	1	[821867, 834484, 856942, 865456, 889248, 90795...	[897922, 1059763, 1090576, 5573397, 6919438]
1	3	[835476, 851057, 872021, 878302, 879948, 90963...	[2002088, 974029, 1079275, 9426963, 6396210]

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)

In [9]:
```
import random

def weighted_random_recommendation(items_weights, n=5):
    """Случайные рекоммендации

    Input
    -----
    items_weights: pd.DataFrame
        Датафрейм со столбцами item_id, weight. Сумма weight по всем товарам = 1
    """
    # Подсказка: необходимо модифицировать функцию random_recommendation()

    # recs = random.choices(list(items_weights['item_id']), weights=tuple(items_weights['weight']), k=n)
    items = np.array(items_weights['item_id'])
    weights = np.array(items_weights['weight'])
    recs = np.random.choice(items, size=n, p=weights)  # p>0

    return recs
```

В [10]:
```
%%time

# your_code
items_weights= data_train[['item_id','sales_value']]

# При x-> 0, log(x)-> -oo, поэтому 0 заменяем на не нулевое значение
items_weights.loc[items_weights['sales_value'] == 0, 'sales_value'] = 0.001

items_weights['weight'] = np.log(items_weights['sales_value'])

# Значения weight < 0, заменяем на 0
items_weights.loc[(items_weights['weight'] < 0)] = 0

# items_weights
min_value = items_weights['weight'].min()
max_value = items_weights['weight'].max()
summ = items_weights['weight'].sum()
items_weights = items_weights[['item_id','weight']].groupby('item_id').sum()

# Сумма weight по всем товарам = 1
items_weights['weight'] = items_weights['weight']/summ

items_weights['item_id'] = items_weights.index  # Преобразуем индек в поле 'item_id'
# items_weights.head(3)
```

```
/home/sil/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.py:965: 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)
  self.obj[item] = s
/home/sil/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:7: 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)
  import sys


CPU times: user 327 ms, sys: 94.6 ms, total: 421 ms
Wall time: 578 ms
```

В [11]:
```
result['weighted_random_recommendation'] = result['user_id'].apply(lambda x: weighted_random_recommendation(items_weight
result.head(2)
```

Out[11]:

	user_id	actual	random_recommendation	weighted_random_recommendation
0	1	[821867, 834484, 856942, 865456, 889248, 90795...	[897922, 1059763, 1090576, 5573397, 6919438]	[1088681, 960318, 1079997, 6534178, 953351]
1	3	[835476, 851057, 872021, 878302, 879948, 90963...	[2002088, 974029, 1079275, 9426963, 6396210]	[1112073, 1005675, 5585510, 1036159, 865178]

В [12]:
```
# # Вариант без использования логарифма

# %%time

# items_weights= data_train[['item_id','sales_value']]

# min_value = items_weights['sales_value'].min()
# max_value = items_weights['sales_value'].max()
# summ = items_weights['sales_value'].sum()

# items_weights = items_weights[['item_id','sales_value']].groupby('item_id').sum()
# items_weights['weight'] = items_weights['sales_value']/summ
# items_weights = items_weights.drop('sales_value', 1)
# items_weights['item_id'] = items_weights.index  # Преобразуем индек в поле

# result['weighted_random_recommendation'] = result['user_id'].apply(lambda x: weighted_random_recommendation(items_weigh
# result.head(2)
```

Задание 2. Улучшение бейзлайнов и ItemItem

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

2.1 Улучшить бейзлайны, считая случаный на топ-5000 товаров

B [13]:
```
popularity = data_train.groupby('item_id')['quantity', 'sales_value'].sum().reset_index()
popularity.rename(columns={'quantity': 'n_sold'}, inplace=True)
```

```
/home/sil/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:1: FutureWarning: Indexing with multiple keys (im
plicitly converted to a tuple of keys) will be deprecated, use a list instead.
  """Entry point for launching an IPython kernel.
```

B [14]:
```
top_5000 = popularity.sort_values('n_sold', ascending=False).head(5000).item_id.tolist()
```

B [15]:
```
# Заведем фиктивный item_id (если юзер покупал товары из топ-5000, то он "купил" такой товар)
# data_train.loc[ ~ data_train['item_id'].isin(top_5000), 'item_id'] = 6666
# data_train.head(100)
```

B [16]:
```
top_5000[:2]
```

Out[16]:
```
[6534178, 6533889]
```

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

Out[17]:

	item_id	n_sold	sales_value
0	25671	6	20.94
1	26081	1	0.99

B [18]:
```
weighted_top_5000 = popularity.sort_values('n_sold', ascending=False).head(5000)
```

B [19]:
```
weighted_top_5000.head(2)
```

Out[19]:

	item_id	n_sold	sales_value
55470	6534178	190227964	447799.94
55430	6533889	15978434	40483.34

1.1 Random recommendation

B [20]:
```
def random_recommendation_5000(items, n=5):
    """Случайные рекоммендации"""

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

    return recs.tolist()
```

B [21]:
```
%%time

items = weighted_top_5000.item_id.unique()

result['random_recommendation_5000'] = result['user_id'].apply(lambda x: random_recommendation_5000(top_5000, n=5))

result.head(2)
```

```
CPU times: user 1.12 s, sys: 49.2 ms, total: 1.17 s
Wall time: 1.27 s
```

Out[21]:

	user_id	actual	random_recommendation	weighted_random_recommendation	random_recommendation_5000
0	1	[821867, 834484, 856942, 865456, 889248, 90795...	[897922, 1059763, 1090576, 5573397, 6919438]	[1088681, 960318, 1079997, 6534178, 953351]	[892004, 893739, 1022105, 8020234, 852600]
1	3	[835476, 851057, 872021, 878302, 879948, 90963...	[2002088, 974029, 1079275, 9426963, 6396210]	[1112073, 1005675, 5585510, 1036159, 865178]	[1052335, 908988, 6632283, 871570, 987838]

1.2 Popularity-based recommendation

B [22]:
```
def popularity_recommendation_5000(data, n=5):
    """Топ-n популярных товаров"""

    popular = data.groupby('item_id')['sales_value'].sum().reset_index()
    popular.sort_values('sales_value', ascending=False, inplace=True)

    recs = popular.head(n).item_id

    return recs.tolist()
```

B [23]:
```
%%time

# Можно так делать, так как рекомендация не зависит от юзера
popular_recs = popularity_recommendation_5000(data_train, n=5)

result['popular_recommendation_5000'] = result['user_id'].apply(lambda x: popular_recs)
result.head(2)
```

```
CPU times: user 113 ms, sys: 12.1 ms, total: 125 ms
Wall time: 255 ms
```

Out[23]:

	user_id	actual	random_recommendation	weighted_random_recommendation	random_recommendation_5000	popular_recommendation_5000
0	1	[821867, 834484, 856942, 865456, 889248, 90795...	[897922, 1059763, 1090576, 5573397, 6919438]	[1088681, 960318, 1079997, 6534178, 953351]	[892004, 893739, 1022105, 8020234, 852600]	[6534178, 6533889, 1029743, 6534166, 1082185]
1	3	[835476, 851057, 872021, 878302, 879948, 90963...	[2002088, 974029, 1079275, 9426963, 6396210]	[1112073, 1005675, 5585510, 1036159, 865178]	[1052335, 908988, 6632283, 871570, 987838]	[6534178, 6533889, 1029743, 6534166, 1082185]

1.3 Weighted random recommender

B [24]:
```
import random

def weighted_random_recommendation_5000(items_weights, n=5):
    """Случайные рекоммендации

    Input
    -----
    items_weights: pd.DataFrame
        Датафрейм со столбцами item_id, weight. Сумма weight по всем товарам = 1
    """

    items = np.array(items_weights['item_id'])
    weights = np.array(items_weights['weight'])
    recs = np.random.choice(items, size=n, p=weights)  # p>0

    return recs
```

B [25]:
```
%%time

items_weights_5000 = weighted_top_5000[['item_id','sales_value']]

# При x-> 0, log(x)-> -oo, поэтому 0 заменяем на не нулевое значение
items_weights_5000.loc[items_weights_5000['sales_value'] == 0, 'sales_value'] = 0.001

items_weights_5000['weight'] = np.log(items_weights_5000['sales_value'])

# Значения weight < 0, заменяем на 0
items_weights_5000.loc[(items_weights_5000['weight'] < 0)] = 0

# items_weights
min_value = items_weights_5000['weight'].min()
max_value = items_weights_5000['weight'].max()
summ = items_weights_5000['weight'].sum()
items_weights_5000 = items_weights_5000[['item_id','weight']].groupby('item_id').sum()

# Сумма weight по всем товарам = 1
items_weights_5000['weight'] = items_weights_5000['weight']/summ

items_weights_5000['item_id'] = items_weights_5000.index  # Преобразуем индек в поле 'item_id'
# items_weights.head(2)
```

```
CPU times: user 9.35 ms, sys: 4.34 ms, total: 13.7 ms
Wall time: 62.9 ms

/home/sil/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.py:965: 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)
  self.obj[item] = s
```

B [26]:
```
result['weighted_random_recommendation_5000'] = result['user_id'].apply(lambda x: weighted_random_recommendation_5000(it
result.head(2)
```

Out[26]:

	user_id	actual	random_recommendation	weighted_random_recommendation	random_recommendation_5000	popular_recommendation_5000	wei
0	1	[821867, 834484, 856942, 865456, 889248, 90795...	[897922, 1059763, 1090576, 5573397, 6919438]	[1088681, 960318, 1079997, 6534178, 953351]	[892004, 893739, 1022105, 8020234, 852600]	[6534178, 6533889, 1029743, 6534166, 1082185]	[933
1	3	[835476, 851057, 872021, 878302, 879948, 90963...	[2002088, 974029, 1079275, 9426963, 6396210]	[1112073, 1005675, 5585510, 1036159, 865178]	[1052335, 908988, 6632283, 871570, 987838]	[6534178, 6533889, 1029743, 6534166, 1082185]	[82

2.2 Попробуйте улучшить разные варианты ItemItemRecommender, выбирая число соседей K.

B [27]:
```
popularity = data_train.groupby('item_id')['quantity'].sum().reset_index()
popularity.rename(columns={'quantity': 'n_sold'}, inplace=True)

popularity.head()
```

Out[27]:

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

B [28]:
```
top_5000 = popularity.sort_values('n_sold', ascending=False).head(5000).item_id.tolist()
```

B [29]:
```
data_train.head(2)
```

Out[29]:

	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

B [30]:
```
# Заведем фиктивный item_id (если юзер покупал товары из топ-5000, то он "купил" такой товар)
data_train.loc[ ~ data_train['item_id'].isin(top_5000), 'item_id'] = 6666
data_train.head(2)
```

```
/home/sil/anaconda3/lib/python3.7/site-packages/pandas/core/indexing.py:965: 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)
  self.obj[item] = s
```

Out[30]:

	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

B [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()
```

B [32]:
```
user_item_matrix.shape
```

Out[32]: (2499, 5001)

B [33]:
```
user_item_matrix.sum().sum() / (user_item_matrix.shape[0] * user_item_matrix.shape[1]) * 100
```

Out[33]:
```
5.33770796861036
```

B [34]:
```
# создаем словари мапинга между id бизнеса к строчному id матрицы

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))
```

B [ ]:
```

```

Fit

B [154]:
```
%%time

K=1
K=2
K=3
K=4
K=5
K=6
K=10
K=15
K=25
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-item matrix
                       N=5, # кол-во рекомендаций
                       filter_already_liked_items=False,
                       filter_items=None,
                       recalculate_user=True)
```

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


CPU times: user 1.63 s, sys: 329 ms, total: 1.95 s
Wall time: 1.78 s
```

B [155]:
```
[id_to_itemid[rec[0]] for rec in recs]
```

Out[155]:
```
[6666, 1082185, 981760, 1098066, 840361]
```

B [156]:
```
%%time

result['itemitem'] = result['user_id'].apply(lambda user_id: [
                                    id_to_itemid[rec[0]]  for rec in model.recommend(userid=userid_to_id[user_id],
                                              user_items=sparse_user_item,   # на вход user-item matrix
                                              N=5,
                                              filter_already_liked_items=False,
                                              filter_items=None,
                                              recalculate_user=True)
                                                  ])
```

```
CPU times: user 280 ms, sys: 6.94 ms, total: 287 ms
Wall time: 390 ms
```

B [157]:
```
# result.head(5)
```

B [158]:
```
# Функции из 1-ого вебинара
import os, sys

from metrics import precision_at_k, recall_at_k
```

В [159]:
```
for name_col in result.columns[1:]:
    print(f"{round(result.apply(lambda row: precision_at_k(row[name_col], row['actual']), axis=1).mean(),4)}:{name_col}"

#
# Исходные метрики
#
# 1.0:actual
# 0.0008:random_recommendation
# 0.1552:popular_recommendation
# 0.1368:itemitem (K=5)


# 1.0:actual
# 0.0005:random_recommendation
# 0.016:weighted_random_recommendation
# 0.0064:random_recommendation_5000
# 0.1552:popular_recommendation_5000
# 0.0164:weighted_random_recommendation_5000
# 0.1368:itemitem (K=5)

# K=1
# 0.1923:itemitem

# K=2
# 0.192:itemitem

# K=3
# 0.1861:itemitem

# K=4
# 0.1449:itemitem

# K=5
# 0.1368:itemitem

# K=6
# 0.1421:itemitem

# K=10
# 0.1509:itemitem

# K=15
# 0.1532:itemitem

# K=25
# 0.15:itemitem

# Максимальное значение метрики для itemitem получили при K=1 - 0.1923:itemitem
# Затем метрика начинает уменьшаться до K=5. Затем  метрика снова начинает увеличиваться до K=15.
# Затем значение метрики остаётся примерно одинаковым.
```

```
1.0:actual
0.0005:random_recommendation
0.0159:weighted_random_recommendation
0.006:random_recommendation_5000
0.1552:popular_recommendation_5000
0.0165:weighted_random_recommendation_5000
0.15:itemitem
```

В [ ]:

В [ ]:

В [ ]:

В [ ]:

В [ ]:

В [ ]: