Рекомендательные системы

Урок 3. Коллаборативная фильтрация

Домашнее задание

1) Попытаться ответить на вопросы/выдвинуть гипотезы

2) Доделать прошлые домашния задания

3) Прочитать статьи BM25/MatrixFactorization

Практика:

4) Поэкспериментировать с ALS (grid-search)

In [ ]:

In [ ]:

Задание 4. Поэкспериментировать с ALS (grid-search)

1. Базовое применение

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.als import AlternatingLeastSquares
from implicit.nearest_neighbours import bm25_weight, tfidf_weight

# Функции из 1-ого вебинара
import os, sys

module_path = os.path.abspath(os.path.join(os.pardir))
print(module_path)
if module_path not in sys.path:
    sys.path.append(module_path)
```

```
/home/sil/ML/RS/Lesson_3
```

In [2]:

```
# from metrics import precision_at_k, recall_at_k
```

In [3]:

```
def precision(recommended_list, bought_list):
    bought_list = np.array(bought_list)
    recommended_list = np.array(recommended_list)
    flags = np.isin(bought_list, recommended_list)
    return flags.sum() / len(recommended_list)

def precision_at_k(recommended_list, bought_list, k=5):
    return precision(recommended_list[:k], bought_list)
```

In [4]:

```
path = '../../Lesson_2/webinar_2/'
```

In [5]:

```
# data = pd.read_csv('/home/sil/ML/RS/Lesson_2/webinar_2/data/retail_train.csv')
data = pd.read_csv('../../Lesson_2/webinar_2/data/retail_train.csv')
data = pd.read_csv(path + '/data/retail_train.csv')
data.head(2)
```

Out[5]:

	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

In [6]:

```
data = pd.read_csv(path + '/data/transaction_data.csv')

data.columns = [col.lower() for col in data.columns]
data.rename(columns={'household_key': 'user_id',
                    'product_id': 'item_id'},
          inplace=True)


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]

data_train.head(10)
```

Out[6]:

	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	-0.60	1631	1	0.0	0.0
1	2375	26984851472	1	1033142	1	0.82	364	0.00	1631	1	0.0	0.0
2	2375	26984851472	1	1036325	1	0.99	364	-0.30	1631	1	0.0	0.0
3	2375	26984851472	1	1082185	1	1.21	364	0.00	1631	1	0.0	0.0
4	2375	26984851472	1	8160430	1	1.50	364	-0.39	1631	1	0.0	0.0
5	2375	26984851516	1	826249	2	1.98	364	-0.60	1642	1	0.0	0.0
6	2375	26984851516	1	1043142	1	1.57	364	-0.68	1642	1	0.0	0.0
7	2375	26984851516	1	1085983	1	2.99	364	-0.40	1642	1	0.0	0.0
8	2375	26984851516	1	1102651	1	1.89	364	0.00	1642	1	0.0	0.0
9	2375	26984851516	1	6423775	1	2.00	364	-0.79	1642	1	0.0	0.0

In [7]:

```
item_features = pd.read_csv(path + '/data/product.csv')
item_features.columns = [col.lower() for col in item_features.columns]
item_features.rename(columns={'product_id': 'item_id'}, inplace=True)

item_features.head(2)
```

Out[7]:

	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 [8]:

```
item_features.department.unique()
```

Out[8]:

```
array(['GROCERY', 'MISC. TRANS.', 'PASTRY', 'DRUG GM', 'MEAT-PCKGD',
       'SEAFOOD-PCKGD', 'PRODUCE', 'NUTRITION', 'DELI', 'COSMETICS',
       'MEAT', 'FLORAL', 'TRAVEL & LEISUR', 'SEAFOOD', 'MISC SALES TRAN',
       'SALAD BAR', 'KIOSK-GAS', 'ELECT &PLUMBING', 'GRO BAKERY',
       'GM MERCH EXP', 'FROZEN GROCERY', 'COUP/STR & MFG', 'SPIRITS',
       'GARDEN CENTER', 'TOYS', 'CHARITABLE CONT', 'RESTAURANT', 'RX',
       'PROD-WHS SALES', 'MEAT-WHSE', 'DAIRY DELI', 'CHEF SHOPPE', 'HBC',
       'DELI/SNACK BAR', 'PORK', 'AUTOMOTIVE', 'VIDEO RENTAL', ' ',
       'CNTRL/STORE SUP', 'HOUSEWARES', 'POSTAL CENTER', 'PHOTO', 'VIDEO',
       'PHARMACY SUPPLY'], dtype=object)
```

In [9]:

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

Out[9]:

	user_id	actual
0	1	[879517, 934369, 1115576, 1124029, 5572301, 65...
1	3	[823704, 834117, 840244, 913785, 917816, 93870...

In [10]:

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

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

In [11]:

```
data_train.head(5)
```

Out[11]:

	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	-0.60	1631	1	0.0	0.0
1	2375	26984851472	1	1033142	1	0.82	364	0.00	1631	1	0.0	0.0
2	2375	26984851472	1	1036325	1	0.99	364	-0.30	1631	1	0.0	0.0
3	2375	26984851472	1	1082185	1	1.21	364	0.00	1631	1	0.0	0.0
4	2375	26984851472	1	8160430	1	1.50	364	-0.39	1631	1	0.0	0.0

In [12]:

```
# Заведем фиктивный item_id

data_train.loc[~data_train['item_id'].isin(top_5000), 'item_id'] = 999_999

user_item_matrix = pd.pivot_table(data_train,
                                  index='user_id', columns='item_id',
                                  values='quantity', # Можно пробоват ьдругие варианты
                                  aggfunc='count',
                                  fill_value=0
                                 )

user_item_matrix = user_item_matrix.astype(float) # необходимый тип матрицы для implicit

# переведем в формат saprse matrix
sparse_user_item = csr_matrix(user_item_matrix)

user_item_matrix.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.htm
l#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#re
turning-a-view-versus-a-copy)
  self.obj[item] = s
```

Out[12]:

item_id	202291	397896	420647	480014	545926	707683	731106	818980	819063	819227	...	15926885	15926886	15926887	15926927	159270
user_id																
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	2.0	0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0

3 rows × 5001 columns

In [13]:

```
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))
```

Alternating Least Squares (ALS)

In [14]:

```
%%time

model = AlternatingLeastSquares(factors=64,
                                regularization=0.05,
                                iterations=15,
                                calculate_training_loss=True,
                                num_threads=10,
                                use_gpu=False)

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

```
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 [00:45<00:00, 3.04s/it, loss=0.0491]


CPU times: user 6.15 s, sys: 1.12 s, total: 7.27 s
Wall time: 4.13 s
```

In [15]:

```
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)
```

In [16]:

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

Out[16]:

```
[1106523, 5569230, 1133018, 999999, 1082185]
```

In [17]:

```
def get_recommendations(user, model, N=5):
    res = [id_to_itemid[rec[0]] for rec in
                model.recommend(userid=userid_to_id[user],
                                user_items=sparse_user_item,   # на вход user-item matrix
                                N=N,
                                filter_already_liked_items=False,
                                filter_items=None,
                                recalculate_user=True)]
    return res
```

In [18]:
```
%%time

result['als'] = result['user_id'].apply(lambda x: get_recommendations(x, model=model, N=5))

result.apply(lambda row: precision_at_k(row['als'], row['actual']), axis=1).mean()
```

```
CPU times: user 25.7 s, sys: 1.45 s, total: 27.2 s
Wall time: 13.8 s
```

Out[18]:
```
0.15760924158713993
```

In [19]:
```
result.head(2)
```

Out[19]:

	user_id	actual	als
0	1	[879517, 934369, 1115576, 1124029, 5572301, 65...	[901062, 1033142, 1005186, 878996, 1024306]
1	3	[823704, 834117, 840244, 913785, 917816, 93870...	[5569327, 1106523, 908531, 951590, 1092026]

Embeddings

In [20]:
```
model.item_factors.shape
```

Out[20]:
```
(5001, 64)
```

In [21]:
```
model.user_factors.shape
```

Out[21]:
```
(2500, 64)
```

In [22]:
```
# model.rank_items()
```

In [23]:
```
fast_recs = model.user_factors @ model.item_factors.T
fast_recs.shape
```

Out[23]:
```
(2500, 5001)
```

In [24]:
```
fast_recs[0,:]
```

Out[24]:
```
array([-0.005852  ,  0.12186948,  0.04337381, ...,  0.09003468,
       -0.10183086, -0.09273487], dtype=float32)
```

In [25]:
```
%%time
recommendations = model.recommend_all(N=5,
                                      user_items=csr_matrix(user_item_matrix).tocsr(),
                                      filter_already_liked_items=True,
                                      filter_items=None,
                                      recalculate_user=True,
                                      show_progress=True,
                                      batch_size=500)
recommendations
```

```
100%                                    2500/2500 [00:17<00:00, 143.89it/s]
```

```
CPU times: user 17 s, sys: 161 ms, total: 17.2 s
Wall time: 17.4 s
```

Out[25]:
```
array([[ 822, 2685,  659,  191, 3941],
       [2297, 2747, 4337, 2134, 1170],
       [2747,  337, 1908,  557, 1505],
       ...,
       [4337, 2297, 2134, 3575,  557],
       [ 655, 2747, 2297,  298, 3695],
       [ 557, 2447, 4054, 3679, 1317]], dtype=int32)
```

In [26]:
```
recommendations.shape
```

Out[26]:
```
(2500, 5)
```

Оценка качества

2. TF-IDF взвешивание

In [27]:
```
user_item_matrix = tfidf_weight(user_item_matrix.T).T  # Применяется к item-user матрице !
```

In [28]:
```
%%time

model = AlternatingLeastSquares(factors=64,
                                regularization=0.05,
                                iterations=15,
                                calculate_training_loss=True,
                                num_threads=10)
```

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

```
100%                                    15/15 [00:09<00:00, 1.59it/s, loss=0.0803]


CPU times: user 5.98 s, sys: 1.11 s, total: 7.09 s
Wall time: 5.09 s
```

In [29]:

```
result['als_tfidf'] = result['user_id'].apply(lambda x: get_recommendations(x, model=model, N=5))

result.apply(lambda row: precision_at_k(row['als_tfidf'], row['actual']), axis=1).mean()
```

Out[29]:

```
0.1605223505775969
```

In [30]:

```
# result.to_csv('../predictions/predictions_mf.csv', index=False)  # mf - matrix factorization
```

In [31]:

```
# os.path.abspath(os.path.join(os.pardir))
```

Ищем оптимальные параметры (grid-search)

In [32]:

```
import itertools
import copy
```

In [33]:

```
def print_log(row, header=False, spacing=12):
    top = ''
    middle = ''
    bottom = ''
    for r in row:
        top += '+{}'.format('-'*spacing)
        if isinstance(r, str):
            middle += '| {0:^{1}} '.format(r, spacing-2)
        elif isinstance(r, int):
            middle += '| {0:^{1}} '.format(r, spacing-2)
        elif isinstance(r, float):
            middle += '| {0:^{1}.5f} '.format(r, spacing-2)
        bottom += '+{}'.format('='*spacing)
    top += '+'
    middle += '|'
    bottom += '+'
    if header:
        print(top)
        print(middle)
        print(bottom)
    else:
        print(middle)
        print(top)
```

In [34]:

```
def learning_curve(model, user_item_matrix, epochs, k=5, user_index=None):
    prev_epoch = 0
    user_item_precision = []

    headers = ['epochs', 'p@k user_item_matrix']
    print_log(headers, header=True)

    for epoch in epochs:
        model.iterations = epoch - prev_epoch
        model.fit(csr_matrix(user_item_matrix).T.tocsr(),  # На вход item-user matrix
                  show_progress=True)
        result['als_tfidf'] = result['user_id'].apply(lambda x: get_recommendations(x, model=model, N=k))

        user_item_precision.append(result.apply(lambda row: precision_at_k(row['als_tfidf'], row['actual'])
        row = [epoch, user_item_precision[-1]]
        print_log(row)
        prev_epoch = epoch

    return model, user_item_precision
```

In [35]:

```
def grid_search_learning_curve(base_model, user_item_matrix, param_grid,
                               user_index=None, patk=5, epochs=range(2, 40, 2)):
    """
    "Inspired" (stolen) from sklearn gridsearch
    https://github.com/scikit-learn/scikit-learn/blob/master/sklearn/model_selection/_search.py
    """
    curves = []
    keys, values = zip(*param_grid.items())
    for v in itertools.product(*values):
        params = dict(zip(keys, v))
        this_model = copy.deepcopy(base_model)
        print_line = []
        for k, v in params.items():
            setattr(this_model, k, v)
            print_line.append((k, v))
        print(' | '.join('{}: {}'.format(k, v) for (k, v) in print_line))

        _, user_item_patk = learning_curve(this_model, user_item_matrix,
                                                        epochs, k=patk, user_index=user_index)
        curves.append({'params': params,
                       'patk': {'user_item_matrix': user_item_patk}})
    return curves
```

In [39]:
```
# param_grid = {'num_factors': [10, 20, 40, 80, 120],
#               'regularization': [0.0, 1e-5, 1e-3, 1e-1, 1e1, 1e2],
#               'alpha': [1, 10, 50, 100, 500, 1000]}

param_grid = {'num_factors': [5, 10, 20],
              'regularization': [0.0, 1e-2, 1e-1, 1e1],
              'patk': [3, 5]
             }

user_index = range(user_item_matrix.shape[0])
```

In [40]:
```
base_model = AlternatingLeastSquares()
```

In [50]:

In [50]:

In [41]:
```
curves = grid_search_learning_curve(base_model, user_item_matrix,
                                    param_grid,
                                    user_index=user_index,
                                    patk=5)
```

```
num_factors: 5 | regularization: 0.0 | patk: 3
+------------+------------+
|   epochs   | p@k user_item_matrix |
+============+============+

100%                                    2/2 [00:04<00:00, 2.49s/it]


|     2      |  0.16695   |
+------------+------------+

100%                                    2/2 [00:07<00:00, 3.54s/it]


|     4      |  0.15821   |
+------------+------------+

100%                                    2/2 [00:07<00:00, 3.52s/it]
```

In [42]:
```
type(curves)
```

Out[42]:
```
list
```

In [45]:
```
curves[:5]
```

Out[45]:

```
[{'params': {'num_factors': 5, 'regularization': 0.0, 'patk': 3},
  'patk': {'user_item_matrix': [0.16695128076343324,
    0.15821195379206213,
    0.15570065293822008,
    0.15318935208437795,
    0.15047714716222818,
    0.15107985936715027,
    0.15007533902561343,
    0.15047714716222815,
    0.14987443495730599,
    0.1487694625816154,
    0.1482672024108469,
    0.14846810647915426,
    0.14866901054746162,
    0.1490708186840762,
    0.1486690105474615,
    0.1479658463083857,
    0.1485685585133079,
    0.1486690105474616,
    0.14856855851330794]}},
 {'params': {'num_factors': 5, 'regularization': 0.0, 'patk': 5},
  'patk': {'user_item_matrix': [0.173179306880962,
    0.15881466599698443,
    0.15479658463083676,
    0.1533902561526851,
    0.15258663987945562,
    0.14987443495730596,
    0.14836765444500055,
    0.14876946258161533,
    0.14957307885484492,
    0.14917127071823022,
    0.14907081868407646,
    0.1480662983425395,
    0.1476644902059247,
    0.1479658463083858,
    0.14796584630838575,
    0.1483676544450005,
    0.14907081868407626,
    0.14917127071822991,
    0.1489703666499225]}},
 {'params': {'num_factors': 5, 'regularization': 0.01, 'patk': 3},
  'patk': {'user_item_matrix': [0.17368156705173043,
    0.1685585133098924,
    0.16082370668005824,
    0.1584128578603696,
    0.15650426921144955,
    0.1529884480160705,
    0.1527875439477631,
    0.1526870919136093,
    0.15349070818683883,
    0.15308890005022405,
    0.15379206428929995,
    0.15288799598191663,
    0.15218483174284078,
    0.15168257157207227,
    0.15077850326468908,
    0.14987443495730587,
    0.15017579105976697,
    0.1492717227523838,
    0.148869914615769]}},
 {'params': {'num_factors': 5, 'regularization': 0.01, 'patk': 5},
  'patk': {'user_item_matrix': [0.17267704671019338,
    0.16223003515820994,
    0.1557006529382199,
    0.1546961325966829,
    0.15328980411853133,
    0.15097940733299653,
    0.15238573581114817,
    0.15268709191360913.
```

ВЫВОД

Лучшее значение метрики p@k имеем при следующих параметрах
'params': {'num_factors': 5, 'regularization': 0.1, 'patk': 3}

In [46]:
```
%%time

model = AlternatingLeastSquares(factors=5,
                                regularization=0.1,
                                iterations=15,
                                calculate_training_loss=True,
                                num_threads=10)

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

```
100%                                    15/15 [00:11<00:00, 1.35it/s, loss=0.105]
```

```
CPU times: user 4.62 s, sys: 1.3 s, total: 5.93 s
Wall time: 3.94 s
```

In [47]:

```
result['als_tfidf'] = result['user_id'].apply(lambda x: get_recommendations(x, model=model, N=3))

result.apply(lambda row: precision_at_k(row['als_tfidf'], row['actual']), axis=1).mean()
```

Out[47]:

```
0.18583626318432878
```

In [ ]: