代码解读

```
# import packages
import time, math, os
from tqdm import tqdm
import gc
import pickle
import random
from datetime import datetime
from operator import itemgetter
import numpy as np
import pandas as pd
import warnings
from collections import defaultdict
import collections
warnings.filterwarnings('ignore')
```

operator模块中的itemgetter()函数！

```
 a = [1,2,3]
>>> b=operator.itemgetter(1)      //定义函数b，获取对象的第1个域的值
>>> b(a)
2
>>> b=operator.itemgetter(1,0)   //定义函数b，获取对象的第1个域和第0个的值
>>> b(a)
(2, 1)
```

```
# 全量训练集
all_click_df = get_all_click_df(data_path, offline=False)
```

Pandas之drop_duplicates：去除重复项

DataFrame.drop_duplicates(subset=None, keep='first', inplace=False)

- subset : column label or sequence of labels, optional
  用来指定特定的列，默认所有列
- keep : {'first', 'last', False}, default 'first'
  删除重复项并保留第一次出现的项
- inplace : boolean, default False
  是直接在原来数据上修改还是保留一个副本

```
In [22]: import pandas as pd
         data=pd.DataFrame({'A':[1,1,2,2],'B':['a','b','a','b']})
         data.drop_duplicates('B','first',inplace = True)
         data
```

Out[22]:

	A	B
0	1	a
1	1	b

此地无银三百两

```python
def get_all_click_df(data_path='./data_raw/', offline=True):
    if offline:
        all_click = pd.read_csv(data_path + 'train_click_log.csv')
    else:
        trn_click = pd.read_csv(data_path + 'train_click_log.csv')
        tst_click = pd.read_csv(data_path + 'testA_click_log.csv')

        all_click = trn_click.append(tst_click)

    all_click = all_click.drop_duplicates((['user_id', 'click_article_id', 'click_timestamp']))
    # 笔者在后面才发现前面有问题 送命题请注意
    # ['user_id', 'click_article_id', 'click_timestamp']要求三者参数都一样才能达到去重效果
    return all_click
```

```python
i2i_sim = itemcf_sim(all_click_df)
```

```python
def itemcf_sim(df):
    """
        文章与文章之间的相似性矩阵计算
        :param df: 数据表
        :item_created_time_dict:  文章创建时间的字典
        return : 文章与文章的相似性矩阵
        思路: 基于物品的协同过滤(详细请参考上一期推荐系统基础的组队学习)， 在多路召回部分会加上关联规则的召回策略
```

```
    """
    # df是合并起来的数据集
    user_item_time_dict = get_user_item_time(df)
   # get_user_item_time(df) 见下一块代码

    # 计算物品相似度
    i2i_sim = {}
    item_cnt = defaultdict(int) # 初始化为0
    #  tqdm显示for循环完成的进度
    for user, item_time_list in tqdm(user_item_time_dict.items()):  #{user1: [(item1, time1), (item2, time2)..]...}
        # 在基于商品的协同过滤优化的时候可以考虑时间因素
        for i, i_click_time in item_time_list: # i:item   i_click_time:time
            item_cnt[i] += 1 # 统计物品i被不同的人买了几次
            i2i_sim.setdefault(i, {}) # i2i_sim空字典 {user1:{},user2:{},...}
            for j, j_click_time in item_time_list: # j=item  j_click_time=time
                if(i == j):
                    continue
                i2i_sim[i].setdefault(j, 0) # { item1:{item2:0,item3:0,...}, item2: {item1:0,item3:0,...},...}

                i2i_sim[i][j] += 1 / math.log(len(item_time_list) + 1) # 对活跃用户的惩罚 另一种计算相似度公式

    i2i_sim_ = i2i_sim.copy() # copy一份字典
    for i, related_items in i2i_sim.items(): # i=item1  related_items={item2: i2i_sim[i][j],item3: i2i_sim[i][j],...}
        for j, wij in related_items.items(): # j=item2   wij=i2i_sim[i][j]
            i2i_sim_[i][j] = wij / math.sqrt(item_cnt[i] * item_cnt[j])  # item_cnt[i]物品i被统计次数 item_cnt[j]物品j被统计次数 i! =j

    # 将得到的相似性矩阵保存到本地
    pickle.dump(i2i_sim_, open(save_path + 'itemcf_i2i_sim.pkl', 'wb')) # 它可以将对象转换为一种可以传输或存储的格式。

    return i2i_sim_
```

tqdm

获取 用户 - 文章 - 点击时间字典

```
# 根据点击时间获取用户的点击文章序列   {user1: [(item1, time1), (item2, time2)..]...}
def get_user_item_time(click_df):
    # click_df参数是pandas中pd合并后的数据集
    click_df = click_df.sort_values('click_timestamp')
    # 就是按照表头click_timestamp大小排序
    # 送命题  一定要有click_df =
    # 否则你print(click_df)和print(click_df.sort_values('click_timestamp'))输出不一样
```

```
    def make_item_time_pair(df):
        # 拉锁函数
        return list(zip(df['click_article_id'], df['click_timestamp']))

    user_item_time_df = click_df.groupby('user_id')['click_article_id',
'click_timestamp'].apply(
        lambda x: make_item_time_pair(x)) /
        .reset_index().rename(columns={0: 'item_time_list'})
    user_item_time_dict = dict(zip(user_item_time_df['user_id'],
user_item_time_df['item_time_list']))

    return user_item_time_dict
```

拉锁函数超链接 Pandas：apply方法与lambda、groupby结合、apply多参数传递

Pandas之sort_values isin使用技巧

```
import pandas as pd
df = pd.DataFrame([[1,2,3],[2,3,4],[2,4,3],[1,3,7]],
                  index = ['one','two','three','four'],columns = ['A','B','C'])
print df
#        A  B  C
# one    1  2  3
# two    2  3  4
# three  2  4  3
# four   1  3  7
df.sort_values(by=['A','B'],ascending=[0,1],inplace=True)
print df
#        A  B  C
# two    2  3  4
# three  2  4  3
# one    1  2  3
# four   1  3  7

```

```
# 定义
user_recall_items_dict = collections.defaultdict(dict)
# <class 'dict'>  dict
# defaultdict(<class 'dict'>, {})  user_recall_items_dict

# 获取 用户 - 文章 - 点击时间的字典
user_item_time_dict = get_user_item_time(all_click_df) # 参数是合并后的数据集

```

```
# 根据点击时间获取用户的点击文章序列   {user1: [(item1, time1), (item2, time2)..]...}
def get_user_item_time(click_df):

    click_df = click_df.sort_values('click_timestamp') # 排序

    def make_item_time_pair(df):
        return list(zip(df['click_article_id'], df['click_timestamp']))

    user_item_time_df = click_df.groupby('user_id')['click_article_id',
'click_timestamp'].apply(lambda x: make_item_time_pair(x)) /

.reset_index().rename(columns={0: 'item_time_list'})
    user_item_time_dict = dict(zip(user_item_time_df['user_id'],
user_item_time_df['item_time_list']))

    return user_item_time_dict
```

```
# 去取文章相似度
i2i_sim = pickle.load(open(save_path + 'itemcf_i2i_sim.pkl', 'rb'))

# 相似文章的数量
sim_item_topk = 10

# 召回文章数量
recall_item_num = 10

# 用户热度补全
item_topk_click = get_item_topk_click(all_click_df, k=50)
```

```
# 获取近期点击最多的文章
def get_item_topk_click(click_df, k):
    # value_counts() 相同click_article_id出现次数 并由大到小排序 就是同一篇文章出现次数由
大到小排列 index取出前50篇
    topk_click = click_df['click_article_id'].value_counts().index[:k] # 返回的是个列
表
    return topk_click
```

```
for user in tqdm(all_click_df['user_id'].unique()): # 可以理解为去重  就是里面有哪些不同的用户
    # 返回值是(item,相似系数),....
    user_recall_items_dict[user] = item_based_recommend(user, user_item_time_dict, i2i_sim,
                                                        sim_item_topk, recall_item_num, item_topk_click)
#   user_item_time_dict:{user1: [(item1, time1), (item2, time2)..]...}   i2i_sim 相似系数矩阵
#   sim_item_topk=10 相似文章的数量   recall_item_num = 10 召回文章数量   item_topk_click 获取近期点击最多的文章
```

给每个用户根据物品的协同过滤推荐文章

```
# 基于商品的召回i2i
def item_based_recommend(user_id, user_item_time_dict, i2i_sim, sim_item_topk, recall_item_num, item_topk_click):
    """
        基于文章协同过滤的召回
        :param user_id: 用户id
        :param user_item_time_dict: 字典, 根据点击时间获取用户的点击文章序列   {user1: [(item1, time1), (item2, time2)..]...}
        :param i2i_sim: 字典，文章相似性矩阵
        :param sim_item_topk: 整数， 选择与当前文章最相似的前k篇文章
        :param recall_item_num: 整数， 最后的召回文章数量
        :param item_topk_click: 列表，点击次数最多的文章列表，用户召回补全
        return: 召回的文章列表 {item1:score1, item2: score2...}
        注意: 基于物品的协同过滤(详细请参考上一期推荐系统基础的组队学习)， 在多路召回部分会加上关联规则的召回策略
    """

    # 获取用户历史交互的文章
    user_hist_items = user_item_time_dict[user_id] # 其中一个用户浏览信息 user_item_time_dict={user1: [(item1, time1), (item2, time2)..]...}
    # 13036 [(237822, 1508084188196), (257291, 1508084218196)]   user_hist_items= [(item1, time1), (item2, time2)..]
    #  [(199198, 1507030404344), (272143, 1507030434344)]
    user_hist_items_ = {user_id for user_id, _ in user_hist_items} # 应该是item 例如 {199198, 272143}

    item_rank = {}
    for loc, (i, click_time) in enumerate(user_hist_items): # 按照其编号排序 1 (item1, time1);2  (item2, time2)
        # j=item2, item3..   wij=i2i_sim[i][j],i2i_sim[i][j]...  排序后前10篇   注意是排序后的不一定第一个就是user1 只是方便理解
        for j, wij in sorted(i2i_sim[i].items(), key=lambda x: x[1], reverse=True)[:sim_item_topk]:
            if j in user_hist_items_: # 如果商品的item{199198, 272143}
                continue

```

```
            item_rank.setdefault(j, 0)  # {item j:0...}
            item_rank[j] += wij   # {item j:i2i_sim[i][j]...}

    # 不足10个，用热门商品补全
    if len(item_rank) < recall_item_num:
        for i, item in enumerate(item_topk_click): # item_topk_click 前50篇
            if item in item_rank.items(): # 填充的item应该不在原来的列表中    这个意思应该是如果用户浏览过，就pass
                continue
            item_rank[item] = - i - 100 # 随便给个负数就行
            if len(item_rank) == recall_item_num:
                break

    item_rank = sorted(item_rank.items(), key=lambda x: x[1], reverse=True)[:recall_item_num] # 相似度排序 逆序 高到低

    return item_rank
```

enumerate

召回字典转换成df

```
# 将字典的形式转换成df
user_item_score_list = []
# 注释 score是相似系数

#        item1  item2  item3
for user, items in tqdm(user_recall_items_dict.items()): # {user1 : item_rank}
            # user1  score  score   score
    for item, score in items:  # 构造一个新列表
            # user2  score  score   score
        user_item_score_list.append([user, item, score])

recall_df = pd.DataFrame(user_item_score_list, columns=['user_id', 'click_article_id', 'pred_score'])  # 我也不懂这里为啥是预测分数。好像全局都没预测
```

Pandas-DataFrame基础知识点

```python
# 获取测试集
tst_click = pd.read_csv(data_path + 'testA_click_log.csv')
tst_users = tst_click['user_id'].unique() # 返回不同的user_id

# 从所有的召回数据中将测试集中的用户选出来
tst_recall = recall_df[recall_df['user_id'].isin(tst_users)] # 在user_id列表中查找tst_users 返回值=['user_id', 'click_article_id', 'pred_score']

# 生成提交文件
submit(tst_recall, topk=5, model_name='itemcf_baseline')
```

精确查找isin

```python
# 生成提交文件
def submit(recall_df, topk=5, model_name=None):
    recall_df = recall_df.sort_values(by=['user_id', 'pred_score']) # 排序 如果user_id相同按照pre_score排序
    recall_df['rank'] = recall_df.groupby(['user_id'])['pred_score'].rank(ascending=False, method='first') # 名次 最小的的排名第一

    # 判断是不是每个用户都有5篇文章及以上
    tmp = recall_df.groupby('user_id').apply(lambda x: x['rank'].max()) # 取最大的名次是否大于5
    assert tmp.min() >= topk

    del recall_df['pred_score']
    submit = recall_df[recall_df['rank'] <= topk].set_index(['user_id', 'rank']).unstack(-1).reset_index()

    submit.columns = [int(col) if isinstance(col, int) else col for col in submit.columns.droplevel(0)]
    # 按照提交格式定义列名
    submit = submit.rename(columns={'': 'user_id', 1: 'article_1', 2: 'article_2',
                                    3: 'article_3', 4: 'article_4', 5: 'article_5'})

    save_name = save_path + model_name + '_' + datetime.today().strftime('%m-%d') + '.csv'
    submit.to_csv(save_name, index=False, header=True)
```