智能系统 Lab2 文档

17302010021 林晨

一、代码基本结构

基于用户和基于产品的协同过滤算法都采取首先计算相似矩阵的办法。

①user_based

cf_user_based_sim.py：根据助教所给的计算相似矩阵的算法，进行改进优化形成相似矩阵并保存在文件 **user_based_similar.csv**，方便下一步的操作。

cf_based_user：读取上述矩阵。由于助教表示对于 **user_based** 可以不区分测试集和训练集，因而我取所有打分数据作为训练集。接着利用循环对于所有打分相似度在一定范围以上并且已经对相应产品（即电影）有评分的电影 **ID** 即下标索引。由于助教取消了 **rmse** 的计算因而未打印。如果要打印也简单，只要调用打印 **rmse** 函数即可。

然后循环使用如下公式：

最终写入 **predict_user.csv**。

$$\hat{x}_{u,m} = \bar{x}_u + \frac{\sum_{v \in N_u} \mathrm{sim}(u,v)(x_{v,m} - \bar{x}_v)}{\sum_{v \in N_u} |\mathrm{sim}(u,v)|}$$

```
try:
    for i in tqdm(range(user_number)):
        for j in range(item_number):
            if train_data_matrix[i][j] == 0:
                ind = np.nonzero(sim_data_transform_matrix[i] * train_data_transpose_matrix[j])
                print("ind:",ind)
                # print(sim_data_matrix[i][ind])
                # print((train_data_transpose_matrix[j][ind]-mean_user[ind]))
                # print(np.sum(sim_data_matrix[i][ind]*(train_data_transpose_matrix[j][ind]-mean_user[ind])))
                # print(np.sum(np.fabs(sim_data_transform_matrix[i][ind])))
                # 分母
                abs_sim_sum = np.sum(np.fabs(sim_data_transform_matrix[i][ind]))
                # 对分母进行微小量添加
                if abs_sim_sum == 0:
                    abs_sim_sum += 0.0000001
                result[i][j] = mean_user[i] + np.sum(sim_data_matrix[i][ind] * (train_data_transpose_matrix[j][ind] - mean_user[in
                # 修正预测打分结果
                if result[i][j] > 5:
                    result[i][j] = 5
                elif result[i][j] < 0:
                    result[i][j] = 0
            else:
                result[i][j] = train_data_matrix[i][j]
        print(result[i])
```

值得注意的是，我对于分母出现 0 的意外情况进行了讨论。分母出现零，一方面可是是由于冷启动问题，此时用 0 来表示后面的项。还有可能是 u 或者 v 的方差是 0，也就是说，u，v 这个用户对电影都分别是一个固定的打分（如 u:3,3,3;v:4,4,4）--此时算出来相似度的分母是 0。

②item_based

cf_item_based_sim.py：根据助教所给的计算相似矩阵的算法，去除其中的均值项，形成相似矩阵并保存于 item_based_similar.csv，方便下一步操作。

cf_based_item：读取上述矩阵。由于助教表示对于 item_based 可以不区分测试集和训练集，因而我取所有打分数据作为训练集。接着利用循环对于所有打分相似度在一定范围以上并且已经对相应有评分的用户 ID，即下标索引。由于助教取消了 rmse 的计算因而未打印。如果要打印也简单，只要调用打印 rmse 函数即可。

然后循环以下过程：

结果保存在 predict_item.csv

$$\hat{x}_{m,u} = \frac{\sum_{m' \in I_m} \mathrm{sim}(m,m') x_{m',u}}{\sum_{m' \in I_m} |\mathrm{sim}(m,m')|}$$

```
try:
    for i in tqdm(range(item_number)):
        for j in range(user_number):
            if train_data_transpose_matrix[i][j] == 0:
                ind = np.nonzero(sim_data_transform_matrix[i] * train_data_matrix[j])
                abs_sim_sum = np.sum(np.fabs(sim_data_transform_matrix[i][ind]))
                if abs_sim_sum == 0:
                    abs_sim_sum += 0.0000001
                result[i][j] = np.sum(sim_data_transform_matrix[i][ind] * train_data_matrix[j][ind])/abs_sim_sum
                # 修正预测打分结果
                if result[i][j] > 5:
                    result[i][j] = 5
                elif result[i][j] < 0:
                    result[i][j] = 0
            else:
                result[i][j] = train_data_transpose_matrix[i][j]
    import csv
    tran_result = np.transpose(result)
    df = pd.DataFrame(tran_result)
    df.to_csv("item_based_result.csv", header=True, index=True)
```

值得注意的是，我对于分母出现 0 的意外情况进行了讨论。分母出现零，一方面可是是由于冷启动问题，此时用 0 来表示后面的项。还有可能是 m1 或者 m2 的方差是 0，也就是说，m1，m2 这个电影相对每个用户都分别是一个固定的打分（如 u:3,3,3;v:4,4,4）--此时算出来相似度的分母是 0。

CF 优点：

不需要任何的用户或者内容的属性特征信息

推荐效果的惊喜度、准确度、多样性较好

随着用户行为增多，推荐效果也会提升

CF 缺点：

需要大量、准确的用户行为数据；

冷启动问题（新用户/物品没有行为，也找不到类似用户）

灰羊问题（个别口味特殊的用户找不到口味类似的用户）

因为采用历史行为数据，不易修改，不够灵活

覆盖率较低（数据稀疏问题，很多物品没有行为）

③PMF-ALS

```
if __name__ == "__main__":
    # run(train_data_matrix)
    import time
    from tqdm import tqdm
    #  k, rsme
    res = [0,10]
    t1 = time.time()
    #  获得user和item纬度的字典--实际上是一样的，只是为了便于查找
    dict_list = get_dict_u()
    dict1 = dict_list[0]
    dict2 = dict_list[1]
    result = get_random_dict(dict_list)  # 划分
    # print(result)  #  训练用户字典，测试用户字典，训练产品字典，测试产品字典
    train_u = result[0]
    test_u = result[1]
    train_i = result[2]
```

将数据划分为测试集和训练集按照 2:8 的比例

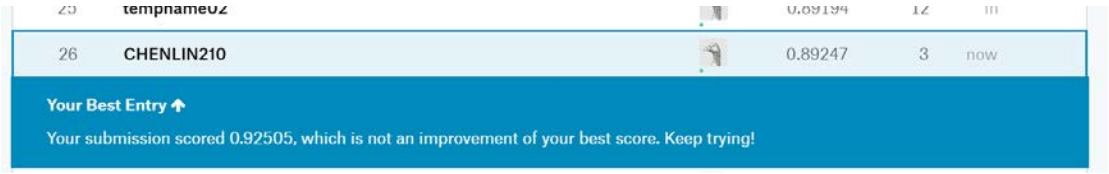
接下来为 pmf 具体实现：

PMF.py：循环分解为 1-50 的的 K 值，利用训练集得到的 F 和 G，分别求取其对于测试集合 rmse 的值，得到最佳 K 值。（同理，还有 K 值）

④SVD++算法：

SVD++.py：根据 SVD++算法我们初始化 alpha 和 lbd，然后选取不同的值求借 F 和 G 的收敛值和打分矩阵值。最后 lbd 取 0.3，alpha 取 1。rmse=0.88

```
lis = get_dict_u()
for m in range(1,6):
    alpha = 0.005 * m
    for l in range(1,6):
        lbd = 0.1 * l
        for z in tqdm(range(500)):
            for i in lis[0].keys():
                for j in lis[0][i]:
                    F[i] = (1-alpha*lbd)*F[i] + alpha*G[j]*(train_data_matrix[i][j] - (miu + bu + bm + np.dot(F[i],(G[j].T))))
                    G[j] = (1-alpha*lbd)*G[j] + alpha*F[i]*(train_data_matrix[i][j] - (miu + bu + bm + np.dot(F[i],(G[j].T))))
                    bu = (1-alpha*lbd)*bu + alpha*(train_data_matrix[i][j] - (miu + bu + bm +np.dot(F[i],(G[j].T))))
                    bm = (1-alpha*lbd)*bm + alpha*(train_data_matrix[i][j] - (miu + bu + bm + np.dot(F[i],(G[j].T))))
                    # print(F[i])
                    # print(G[j])
                    # F[i] = fi
                    # G[j] = gj
                    # bu = bu_t
                    # bm = bm_t
        print(F)
        print(G)
```



二、参数、算法比较

①算法比较

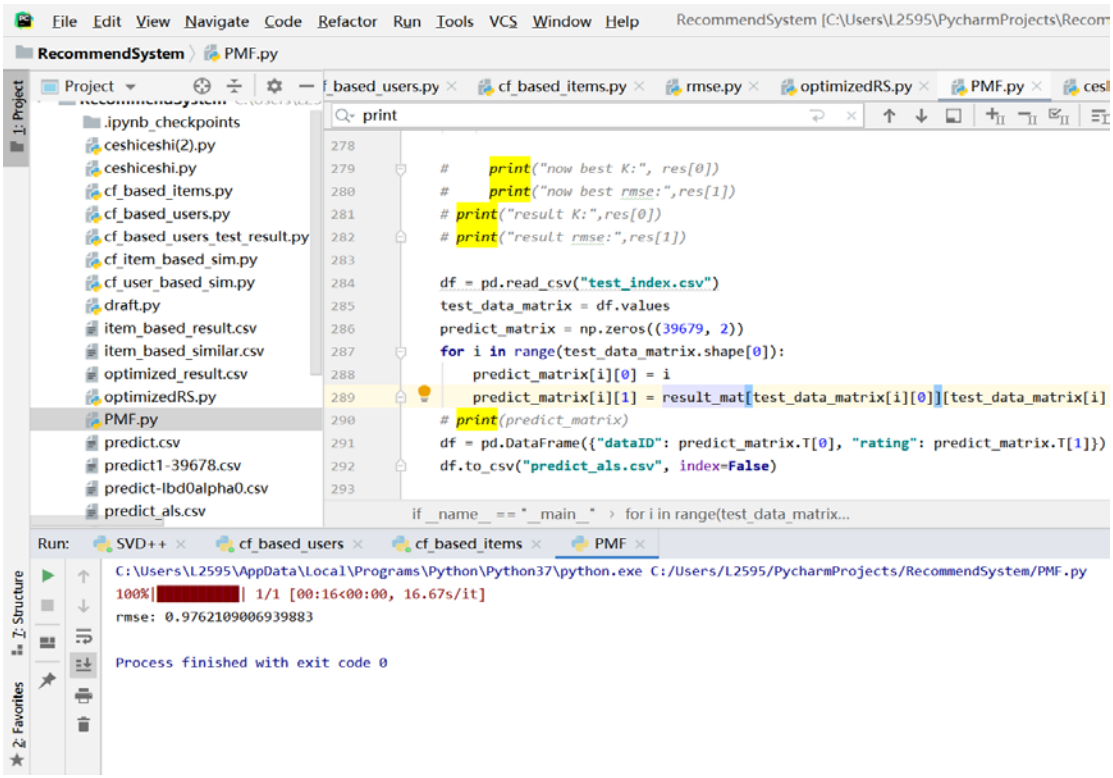
从目前的 rmse 提交测试结果来看，RMSE(item_based) > RMSE(user_based) > RMSE(ALS) > RMSE(SVD++)，分别约为 1.2,1.1,0.97,0.88。虽然这也收到参数的影响，但是整体上还是呈现上述的 rmse 排名趋势。

②参数比较

1、user:选取的最小的相似度为 0.2 左右（sim>= 0.2）得到的 rmse 相对来说是比较小的，但是这个波动并不大，根本上是算法本身的限制（还有数据集中部分用户评分过少等原因）。

2、item:选取所有的 item 进行计算，得到的 rmse 值本身较大，变化非常小，参数调整意义不大。

3、ALS:使用 K 的变量组合选取取得最佳的 K 值。最终确定 K 取 11 效果不错（lbd 在 0.2 左右）。因为其 rsme 相对最小，约为 0.976 结果如下（为缩短文档长度，取了前 14 条 K 的数据，lbd 同理使用循环求最佳）：



born rmse: 0.9771560318741384
now best K: 1
now best rmse: 0.9771560318741384

born rmse: 0.984755183575681
now best K: 1
now best rmse: 0.9771560318741384

born rmse: 1.0128120633227335
now best K: 1
now best rmse: 0.9771560318741384

born rmse: 1.0491261732166681
now best K: 1
now best rmse: 0.9771560318741384

born rmse: 1.0785924776799825
now best K: 1
now best rmse: 0.9771560318741384

born rmse: 1.0959014568735463
now best K: 1
now best rmse: 0.9771560318741384

born rmse: 1.1227138017414493
now best K: 1
now best rmse: 0.9771560318741384

born rmse: 0.9776657037287566
now best K: 1
now best rmse: 0.9771560318741384

born rmse: 0.9771682188927695
now best K: 1
now best rmse: 0.9771560318741384

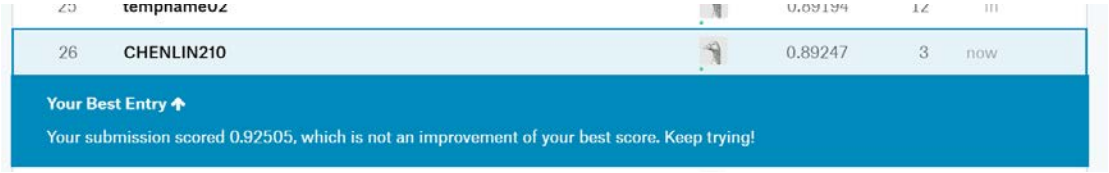
born rmse: 0.976536498864334
now best K: 10
now best rmse: 0.976536498864334

born rmse: 0.9764728746242117
now best K: 11
now best rmse: 0.9764728746242117

born rmse: 0.9765590015422806
now best K: 11
now best rmse: 0.9764728746242117

born rmse: 0.976760009400246
now best K: 11
now best rmse: 0.9764728746242117

4、SVD++：使用循环变化参数求得不同的 rmse，由于使用助教给的网站测试 rmse 即可，最终在 lbd 为 0.3，alpha 为 1 左右。



三、改进思路、实验

①减小参数的步长。可以选取 0.001 的步长来求取最佳参数，这样得到的 rmse 会更小，但是要花很多时间运行。
②梯度下降算法：可以尝试一下看看是否会得到更小的 rmse
③块处理：ppt 中对于网络中大量数据使用块处理，对于大量数据可能更为有效，可以一试。
④基于内容的推荐（Content-based Recommendation）是信息过滤技术的延续与发展，它是建立在项目的内容信息上作出推荐的，而不需要依据用户对项目的评价意见，更多地需要用机 器学习的方法从关于内容的特征描述的事例中得到用户的兴趣资料。在基于内容的推荐系统中，项目或对象是通过相关的特征的属性来定义，系统基于用户评价对象 的特征，学习用户的兴趣，考察用户资料与待预测项目的相匹配程度。用户的资料模型取决于所用学习方法，常用的有决策树、神经网络和基于向量的表示方法等。 基于内容的用户资料是需要有用户的历史数据，用户资料模型可能随着用户的偏好改变而发生变化。--不适合这个 lab
⑤基于效用的推荐（Utility-based Recommendation）是建立在对用户使用项目的效用情况上计算的，其核心问题是怎么样为每一个用户去创建一个效用函数，因此，用户资料模型很大程度上是由系统所采用的效用函数决定的。基于效用推荐的好处是它能把非产品的属性，如提供商的可靠性（Vendor Reliability）和产品的可得性（Product Availability）等考虑到效用计算中。--需要更多的数据

改进实验：调节 ALS 和 SVD 的 lbd--结论：效果一般
梯度下降算法--结论：效果差不多
结论--要想 rmse 小雨 0.8 很难，**我们需要更多的信息。**