Web Lab3

PB17111585 张永停

一、实验要求

- 数据来源为豆瓣电影的评分记录。 请根据训练数据中的用户评分信息，判断用户偏好，并为测试数据中 user-item 对进行评分。
- 训练集中所有的打分信息都以如下格式保存在"training.dat"文件中：

  ```
  User ID, Movie ID, Rating (0-5), Timestamp[, Tag 1, Tag 2, …]
  ```

- 用户之间的社交关系都以如下格式保存在"relation.txt"文件中：

  ```
  A:B,C,D 表示A关注了BCD
  ```

- 测试集中的用户与训练集一致，相比训练集而言抹去了打分信息，以如下格式保存在"testing.dat"中：

  ```
  User ID, Movie ID, Timestamp[, Tag 1, Tag 2, …]
  ```

二、算法设计

Part1 数据预处理

本次实验的数据预处理较为简单，只是为了训练方便，将原数据集中的TAG给删去

```python
def get_simple(origin_data_path, obj_data_path, is_test=False):
    with open(obj_data_path, 'w', encoding='utf-8') as fw:
        with open(origin_data_path, 'r', encoding='utf-8') as fr:
            if not is_test:
                for line in fr:
                    fw.write(','.join(line.split(',')[0:4]) + '\n')
            else:
                for line in fr:
                    fw.write(','.join(line.split(',')[0:3]) + '\n')
```

Part2 简单的推荐系统尝试

本次实验全程使用`surprise`库

SVD

为了验证模型效果，我使用了三折交叉验证

```
def train(train_data_path, save_path):
    reader = Reader(line_format='user item rating timestamp', sep=',')
    train_data = Dataset.load_from_file(train_data_path, reader=reader)

    model = SVD()
    cross_validate(model, train_data, measures=['RMSE', 'MAE'], cv=3,
verbose=True)
    dump.dump(save_path, algo=model)
```

SVD算法的效果为

```
svd
Evaluating RMSE, MAE of algorithm SVD on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.3083  1.3095  1.3069  1.3082  0.0011
MAE (testset)     1.0156  1.0173  1.0151  1.0160  0.0009
Fit time          87.77   100.41  107.13  98.44   8.02
Test time         8.95    8.70    8.84    8.83    0.10
```

KNN UserBased

代码此处就不贴了，就把上面的svd改一下model即可

我试了 k=10 / 20 / 30 / 40, 效果最好的为k=40

```
Evaluating RMSE, MAE of algorithm KNNWithMeans on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.3026  1.3040  1.3027  1.3031  0.0006
MAE (testset)     1.0225  1.0239  1.0235  1.0233  0.0006
Fit time          32.19   30.67   30.37   31.08   0.80
Test time         265.79  244.43  232.83  247.68  13.65
```

KNN ItemBased

只需要更改

```
model = surprise.KNNWithMeans(sim_options={'user_based':False})
```

由于UserBased效果最好的为k=40, 对于ItemBased我仅试了30， 40， 仍旧是k=40效果最好

ItemBased Knn需要非常大的内存， 因此该部分我是用服务器跑的(128GB)

```
Evaluating RMSE, MAE of algorithm KNNWithMeans on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    1.3821  1.3843  1.3851  1.3838  0.0013
MAE (testset)     1.1073  1.1088  1.1090  1.1084  0.0007
Fit time          224.57  235.39  215.72  225.22  8.04
Test time         452.49  468.17  455.30  458.65  6.83
```

测试结果

这里需要注意模型的预测结果是浮点数，为了满足实验提交格式，我们用`round`进行四舍五入取整。(直接向下或者向上取整的话，结果没有四舍五入好，emm其实也很合理)

knn_item_around.txt	1.56684231471759
knn_user_around.txt	1.4962112460314432
svd_around.txt	1.4879589947824896

可以看到svd的rmse是最低的

Part3 Voting

由于~~时间紧迫(期末考试)~~(拖延症)， 我并没有实现很复杂的ensemble, 只是对以上三个模型做了简单的线性加权，使用softvoting的方法预测用户评分。

这里的softvoting指的是

- 使用以上三个模型的浮点预测结果
- 根据不同的权重加和得到一个新的预测结果， 然后四舍五入取整

soft_voting_svd_user_item_221.txt	1.4723620933465498
soft_voting_svd_user_item_212.txt	1.4629138070571719
soft_voting_svd_user_item_313.txt	1.462400854661506
soft_voting_svd_user_item_213.txt	1.4639805383947648
soft_voting_svd_item_user_303.txt	1.4649886959472123
knn_network_40.txt	1.517986527448515

这里提交文件的后三位表示三种方法占的权重(前4个文件名命名错误了，应该是svd_item_user)

经过一系列尝试，最终发现 svd : item : user = 2 : 1 : 2时rmse最低

Part4 Social Network

本部分我根据relation.txt这个文件， 使用关注同个人的数目来计算两个用户之间的相似度

$$sim(u,v) = \frac{|follow(u) \cap follow(v)|}{\sqrt{(|follow(u)|+1)(|follow(v)|+1)}}$$

其中分母加一是为了防止没有关注别人的用户计算出现`div 0`的问题

这里的u, v即为用户id的int数字， 我称之为`raw_id`(借鉴surprise)

用户相似度计算完成后，按照传统方法，应该是聚类， 但如上所述，时间紧迫(误)， 且surprise这么好的库放在面前，便想着能不能借助surprise库来实现

在查询了surprise库的代码后，我惊喜的发现，surprise在fit的时候并不会执行聚类，他只是计算了similarity存储，只用predict的时候才聚类。于是我决定将上面训练好的user_based_knn的model的similarity matrix矩阵替换为通过社交网络计算的similarity matrix， 然后执行predict。

但上述方法存在一个问题， surprise的similarity matrix的行列使用的是inner id， 我需要将raw_id转换为inner_id。 查询了surprise的代码后，我发现可以直接通过

```
raw2inner = model.trainset._raw2inner_id_users
```

来得到raw2inner， 如果需要inner2raw, 只需要

```
inner2raw = {inner: raw for (raw, inner) in iteritems(raw2inner)}
```

到此，我们已经可以愉快的使用surprise库来实现我们的knn辣(吐槽surprise库similarity matrix竟然不是私有的)

```
def test(model_path, network_path, test_data_path):
    model = dump.load(model_path)[1]
    network_sim = np.load(network_path)

    usersim = model.sim
    inner_net = np.zeros(usersim.shape)

    raw2inner = model.trainset._raw2inner_id_users
    inner2raw = {inner: raw for (raw, inner) in iteritems(raw2inner)}

    for i in range(usersim.shape[0]):
        for j in range(usersim.shape[1]):
            rawid_i = int(inner2raw[i])
            rawid_j = int(inner2raw[j])
            inner_net[i, j] = network_sim[rawid_i, rawid_j]


    model.sim = (inner_net * 10 + usersim * 0) / 10

    with open("submission.txt", "w", encoding="utf-8") as fw:
        with open(test_data_path, "r", encoding="utf-8") as fr:
            for line in fr:
                item = line.split(',')
                fw.write(str(((model.predict(item[0], item[1], item[2]).est))) +
'\n')
```

结果是非常让我惊讶的，仅仅使用如此简单的方法计算相似度，最后的rmse也有1.5

knn_network_40.txt	1.517986527448515
network_user_19.txt	1.4962904012386649

Part 5 Voting.v2

其实这部分也比较无聊啦，就是把以上四个方法ensemble到一起，然后预测

network_user_19.txt	1.4962904012386649
network_user_55.txt	1.4970645404883653
soft_voting_svd_item_user_network_3131.txt	1.4628273505439617
soft_voting_svd_item_user_network_3121.txt	1.4621783972338167
soft_voting_svd_item_user_network_3103.txt	1.4675515244464064
soft_voting_svd_item_user_network_3153.txt	1.4684484795788546
soft_voting_svd_item_user_network_4021.txt	1.4653939590264882
soft_voting_svd_item_user_network_3211.txt	1.4646557696594242

可以看到最好的是 svd : item : user : network = 3:1:2:1