WebInfo-exp3 实验报告

组员

廖佳怡 PB19151776

谢新格 PB19081644

实验内容

基于豆瓣上的一些用户和电影、书籍、音乐的交互数据，以及用户之间相互关注的社交数据，来为豆瓣社区的用户推荐音乐。设计模型，为每个用户提供音乐方面的个性化推荐，生成Top-N列表。

数据集包含交互与评分、社交、跨域等信息，共5个文件`DoubanMusic.txt`，`DoubanSocial.txt`，`DoubanSocialFUll.txt`，`DoubanBook.txt`，`DoubanMovie.txt`。

本实验采用留一法进行评测，有四个评测指标：`HR@20`，`HR@100`，`NDCG@20`，`NDCG@100`。

数据分析

- 首先我们分析了数据的规模，用户数量23599，音乐数量21602，用户数和物品数近似相等，如果我们用稠密矩阵存储每个用户-音乐对，且使用8字节的double类型，内存开销约3~4GB，这是可以接受的，同时我们在模型中会全程注意这样的大矩阵，保证所有的运行时都最多只存在这样一个大矩阵，以防止因内存开销过大而导致程序崩溃。
- 之后我们观察到有很多未评分项，而且也有很多用户一个评分项都没有，之后我们统计了所有用户-音乐对，发现有约140w个条目，平均每个用户交互了60多首音乐，但其中有约30%的音乐是交互但未评价的。如果我们直接删去这30%数据，将会损失至少30%的信息量，且显然那些没有评过分的用户就完全无法预测了。所以认为这种做法是不可取的。
- 另外，这次实验任务是预测用户可能看的电影，而不是预测用户对电影的评分，我们认为这次实验的主要数据是用户交互过哪写电影，而评分数据，社交数据，跨邻域数据只是辅助作用。

设置验证集

我们使用留一法将训练集`DoubanMusic.txt`划分出`train.txt`和`dev.txt`，用做新的训练集和验证集，而测试集就是助教提供在测试网站上的。`train.txt`的格式和`DoubanMusic.txt`完全相同，同时，我们写了`tester.py`测试脚本来计算验证集上的准确率`hit@20`和`hit@100`。

模型建立

设计思想

- 我们现在推荐系统中最基本的模型——协同过滤出发。首先我们注意到以下事实，协同过滤往往适用于矩阵较稠密的情况，但我们这次实验的数据是相当稀疏的。同时，协同过滤是在评分预测，和这次实验的交互预测有所区别。所以我要对协同过滤进行修改，以适用本次实验。
- 由于是交互预测，一个直观的感受是，一个用户并不一定总是去交互评分高的音乐（如果这样，这个用户很可能不会打低分，都是高分的话也没什么区分度了），而且，实验采用的留一法，留的那

个“一”，并不都是高分音乐，也存在很多低分项，因此我们不拿“评分高低“去预测用户是否交互，这样我们直接删除了所有评分数据，而只保存交互数据，整个矩阵变为了0-1矩阵。

- 观察实际的推荐系统，如B站上推荐的视频，新推荐的视频和我以前看过的视频有很高的相似度，于是我们也使用相似度这个概念来：音乐-音乐的相似度，或者用户-用户的相似度。对于0-1向量的相似度计算，最直接的当然是Jaccard相似度，最终基于相似度去推荐。

具体实现一：基于物品

修改协同过滤的基于物品的方法，有许多不同之处，在步骤中说明。

步骤如下

1. 数据预处理和数据结构：将音乐Map到交互过的用户列表，本质是音乐-用户的邻接表

```
    MapMusicID = {}

    f = open(data_path, 'r')
    lines = f.readlines()
    f.close()

    for line in lines:
        temp = line.split()
        UserID = int(temp[0])
        for pair in temp[1:]:
            MusicID = int(pair.split(',')[0])
            if MusicID not in MapMusicID:
                MapMusicID[MusicID] = [UserID]      You, 2 days ago • add utils and UserBase
            else:
                MapMusicID[MusicID].append(UserID)
    return MapMusicID
```

   稀疏矩阵的邻接表，内存开销很小。

2. 不进行归一化，直接计算物品间的相似度，并保存相似度矩阵（因为同个相似度可能被使用多次，这么做可以减小下一步骤时间），用Jaccard方法计算，A, B两个物品间的相似度如下

$$Sim(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

```
def JaccardSim(list1:list, list2:list):
    intersection = len(set(list1).intersection(set(list2)))
    union = len(list1) + len(list2) - intersection
    return intersection/union
```

   为了提高运行效率，我们将列表转成set，并用Python自带的set交运算换成分子的计算，并且分母不必再求并运算，而是直接通过加减就能计算得到。实际证明，我们这么做与手动合并相比，大大减少了运行时间，生成矩阵时间约40min。由于相似度矩阵是个稠密的大矩阵，内存开销约3.6GB，在可接受范围内。对应矩阵生成文件在`MusicBase`目录下的`MatSim.py`文件

3. 对于每一个用户，预测其可能的交互情况，由于内存中已经存在了相似度矩阵这个大矩阵，我们预测好一个用户后，直接写入数据。

   对于每一个用户，对其没交互过的音乐，计算一个交互可能性，再进行排序，对最高的100个结果进行数据。假设一个用户已知交互过的音乐集合为S，对任意一个未交互的音乐M，其交互可能性为

$$pred(M) = \sum_{N \in S} Sim(M,N) \qquad M \notin S$$

```
    for UserID in UserMap:
        if UserID < count:
            continue
        predict_list = []
        for MusicID in range(MusicNum):
            if MusicID in UserMap[UserID]:
                continue
            predict = 0
            for UserMusic in UserMap[UserID]:
                predict += MatSim[UserMusic][MusicID]
            predict_list.append((predict,MusicID))
        predict_list.sort(reverse=True)
```

注意，和一般的协同过滤不同的是，我们不采用k近邻，而是取了S集合中所有元素。这么做是有原因的，从数据上看，S集合的大小从个位数到千位数分布很不均匀，使用小的k会损失很多信息，并且会增加一轮排序，即还会提高时间复杂度。最主要的是，我们从验证集上发现考虑S中全部元素的效果最好。

对于不同用户，其S集合的大小可能是不同的，但是这里我们不需要平均化，即不需要除以S集合的大小，因为我们是要的是排序结果，除以S的大小对排序不会有影响。

另外，我们在这个步骤中采用时间换空间的策略（上个步骤是空间换时间），运行一次预测约2h，为了可中断，我们设置了`count`参数，在程序中断后，可以从`count`点继续运行，这能让我们更好地跑出结果。

对应文件为`MusicBase`下的`main.py`

实验结果

提交时间	文件名称	Hit@20	Hit@100	NDCG@20	NDCG@100
2022-01-17 23:18:58	192_1642432738_MusicBase.txt	0.242468	0.422476	0.123137	0.155744

这个实验结果相当的出色，hit@100达到了0.42，并且四个指标全部超过了Baseline。

具体实现二：基于用户

基于用户的模型采用的是用户-用户的相似度，在数学矩阵的角度上，基于用户和基于物品是一个互为转置的关系，流程和基于物品的模型比较相似，这里不在赘述，直接展示结果：

2022-01-20 14:11:56	192_1642659115_UserBaseMode1.txt	0.198356	0.362685	0.105863	0.135349

hit@100为0.36，低于基于物品时的0.42，但也有不错的预测效果。事实上，从协同过滤本身而言，基于物品的准确率往往会高于基于用户，本次实验的情况也体现了这一点

其他模型和方法

我们还尝试了其他模型和改进方法，但并未获得良好结果，我们分析了其他模型失败的可能原因，还是能收获一些思考和对推荐系统的认识。具体过程写在实验总结之后的附录中。

实验总结

在本次实验中，我们合理的分析了数据集，在协同过滤的基础上，加入了自己的思考和方法，改造了协同过滤以适应本次实验，并取得了较好的预测效果。我们一共尝试了3种模型，2个获得了成功，其中基于物品的模型hit@100达到了0.42，四个指标均超过了Baseline，满足了本次实验的预期。

最佳结果：

提交时间	文件名称	Hit@20	Hit@100	NDCG@20	NDCG@100
2022-01-17 23:18:58	192_1642432738_MusicBase.txt	0.242468	0.422476	0.123137	0.155744

附录：其他尝试

附录一：变化相似度方法

在上述模型中，度量相似度的方法可以是自定义的。在现实中考虑，一个看的人很多的物品，更可能被推荐到，那么在Jaccard相似度的分母中，热门物品其实可能会被稀释，因此我们定义了在此基础上修改Jaccard相似度为

$$Sim(A,B)=\frac{|A\cap B|}{|A\cup B|^{mode}}$$

这里mode为指数，mode越小，认为这种热门的稀释性越低，在实验中，我们分别取0，0.5，1，并将预测统计结果如下，指标为hit@100

mode	MusicBase	UserBase
0	0.328	0.328
0.5	0.379	0.346
1	0.422	0.362

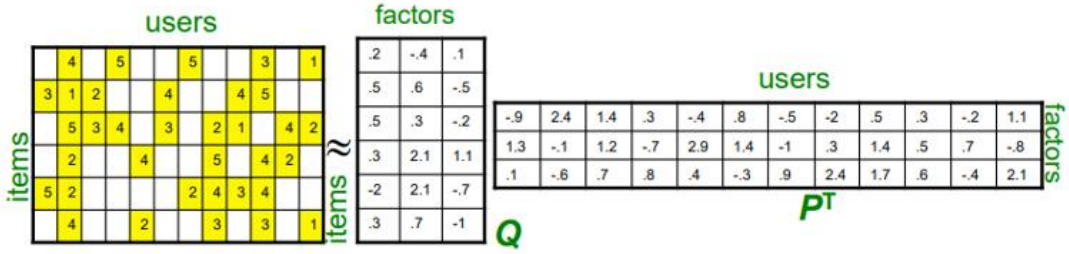
从结果上来看，还是原来的Jaccard相似度效果最好，但是从中可以横向对比的是，UserBase情况下受到分母的影响较小，也就是说，一个用户交互的音乐多少，对其和其他人的的相似度影响很小，重要的是两个用户交互的音乐之间，有多少重合。

附录二：Funk-SVD

由于原始矩阵的稀疏性，我们考虑用矩阵分解来填补矩阵空缺，从而进行分数预测和基于该分数的推荐。并且为了防止过拟合，这里加入正则项，总体公式如下：

$$argmin\sum_{(i,j)\in K}=(m_{ij}-q_j^Tp_i)^2+\lambda(||p_i||_2^2+||q_j||_2^2)$$

- 既然用户评分是根据潜在因子的乘积所得，那么，基于这种方法得到的用户评分，应与历史评分记录尽可能接近，即 $\min_{P,Q} \sum_{(i,x)\in R}(r_{xi} - q_i \cdot p_x)^2$



$$\min_{P,Q} \underbrace{\sum_{training}(r_{xi} - q_i p_x)^2}_{\text{“error”}} + \underbrace{\left[\lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2\right]}_{\text{“length”}}$$

我们考虑了三种情况如下，均没有得到理想的效果，故我们这样分析：

- 不考虑评分，只考虑用户浏览历史，如文件 `BaseMF.py`：

  这样的原始矩阵为零一矩阵，由于SVD只拟合原始矩阵中的非零元素，故其全拟合的是1，即不同位置的相等元素，所以并不会达到期望的推荐效果，最后运行结果和随机推荐差别不大。

- 考虑评分，如文件 `BaseRatingMF.py`：

  这里考虑评分数据，对于浏览过且未评分的数据，采用该用户所有评分的平均分。由于有评分的数据并不多，且考虑到测试方式，即交互预测，那么测试集中需要我们命中的物品，并不一定是高分物品，也有可能存在低分物品，故考虑评分的模型也并不能得到理想的结果，最后运行结果还不如随机推荐。

- 加入社交数据，如文件夹 `SocialBaseMF`：

  我们知道社交在用户与物品的交互中也发挥着重要影响，故我们试图利用社交数据来提高模型准确率，这里采用了 `DoubanSocial.txt` 的follow关系，将每个用户关注的用户所浏览过的物品加入到该用户的浏览历史中。但也许是因为社交在本数据集中发挥的作用不大，故最后结果并不理想。