Web信息处理课程实验三-实验报告

小组成员：PB18051081李钊佚、PB18051061黄育庆

本次实验引用相关信息说明：本次实验我们仅使用了python3.7的编程环境和math数学包（没使用numpy），参考资料为徐老师上课slides-chapter14协同过滤算法部分相关内容，没有参考或者使用其他代码、paper、blogs、或者是其他open-source packets。

目录：

@1：协同过滤算法（based on items）介绍和我们的代码实现

协同过滤算法(collborative filtering algos)是一类简单、朴素、经典、有效的根据用户群与商品群的历史交互信息来预测用户对具体商品的喜好情况并筛选推荐的算法。区别于基于内容的推荐方案仅仅基于单一用户记录对该用户做出推荐，协同过滤的本质思想在于我们相信，往往在实际应用中，其他用户的浏览行为对当前我们需要分析的用户有借鉴作用。

推荐系统的本质是矩阵补全问题，协同过滤的思想在于基于矩阵的其他行协助填补本行的空缺。

我们选用的是基于内存（based on memory）的协同过滤推荐

基于内存的协同过滤推荐又主要分为基于用户的（based on users）、基于物品的（based on items）这两类，一般情况下，基于物品的协同过滤推荐效果又要好于基于用户的协同过滤推荐，又考虑到原始数据中存在一部分用户的评分缺失情况，因此我们考虑实现基于物品的协同过滤推荐算法。

算法（CF based on Items）

(1)计算任意两物品之间的相似度（与PPT一致，采用Pearson相似度）

(2)利用某物品与其他物品的相似度加权和某用户对其他物品的打分预测该用户对该物品的打分情况

(3)把所有未打分物品的打分预测值汇总排序，选择打分预测值高的优先推荐

---

(1)计算任意两物品之间的相似度（与PPT一致，采用Pearson相似度）

$$sim(a,b)=\frac{\sum_{p\in product(P)}(r_{a,p}-\bar{r}_a)(r_{b,p}-\bar{r}_b)}{\sqrt{\sum_{p\in product(P)}(r_{a,p}-\bar{r}_a)^2}\sqrt{\sum_{p\in product(P)}(r_{b,p}-\bar{r}_b)^2}}$$

我们的代码实现：

```
def get_sims(infos):
    for id1 in infos.keys():
        for id2 in infos.keys():
            if norm_table[id1] == 0.0 or norm_table[id2] == 0.0:
                sim_table[encode(id1,id2)] = 0.0
            else:
                sim_table[encode(id1,id2)] =
get_inner_product(infos,id1,id2) * 1.0 / (norm_table[id1] * norm_table[id2])
'''
Tips：我们这里infos里面的数据已经去中心化（已经减减过均值了，因此在这里直接计算范数和内积即可）。
'''
```

(2)利用某物品与其他物品的相似度加权和某用户对其他物品的打分预测该用户对该物品的打分情况

- 基于物品推荐的计算公式

$$r_{ix} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{jx}}{\sum s_{ij}}$$

  - 其中，相似度计算与User-based类似
    - 即构造两个向量，每维减去平均值，然后计算Pearson相关系数
    - 此外，同样可以基于平均分对分数估计进行修正
      - 如果未评分，则直接设为0，不用减去平均数
      - 思考：最终的r_{ix}为何不需要平均分修正?

我们的代码实现：

```
def predict(infos, user, music, avg_dict,N10):
    '''
    predicts scores which this user gave for this music
    '''
    sum = 0.
    for m_id in N10[music]:
        if user in infos[m_id].keys():
            sum += get_sim(encode(music,m_id)) * (infos[m_id][user] +
avg_dict[m_id]) * 1.

    temp = 0.
    for m_id in N10[music]:
        temp += get_sim(encode(music,m_id))
    if temp == 0.0:
        score = 0.0
    else:
        score = float(sum)/ temp
    return score
```

(3)把所有未打分物品的打分预测值汇总排序，选择打分预测值高的优先推荐

根据本次实验要求，要求给出预测值top100的推荐：

```
musiclist4user = infos_pre[i].keys()
            music4predict = list(set(music_list)-set(musiclist4user))
            for music in music4predict:
                scores[music] = predict(infos,i,music,avg,N10)
            '''
            sort for this dict(score) according to score-value
            '''
            top100 = dict(sorted(scores.items(),key=lambda
e:e[1],reverse=True)[:100])
            top100 = list(top100.keys())
```

@2：实验中对数据的各种处理方式（to decrease memory cost or to accelerate program）

我们会提及以下几个对数据的处理：

(1):预处理，将以用户划分的原数据重新整理成方便执行基于物品的协同过滤推荐算法的新数据结构

这样方便我们后边直接去计算不同音乐之间的相似度。

```
def transinfo_user2music(infos):

    infost = {}
    for user in infos.keys():
        for music in infos[user].keys():
            infost[music] = {}

    for user in infos.keys():
        for music in infos[user].keys():
            infost[music][user] = infos[user][music]
    return infost

"""
infost_mat =
{
    (key=1(music_id):value={(key=3(user_id):value=2(score)),......}),
    ...,
    ...
}
"""
```

(2):预处理，将原缺失评分数据改记为0，因为原始缺失数据并不会很大程度上影响预测结果。

```
def read_userfile(filename):
    infos_mat = {}

    with open(filename,"r") as fr:
        for line in fr.readlines():
            line = line.strip().split('\t')
            len_user = len(line) - 1
```

```
            user_id = int(line[0])
            infos_mat[user_id] = {}
            for i in range(len_user):
                music4user = line[i+1].strip().split(',')
                music,score = int(music4user[0]),int(music4user[1])

                if score == -1:
                    infos_mat[user_id][music] = 0
                else:
                    infos_mat[user_id][music] = score

    return infos_mat

```

(3):预先计算并保存相似度矩阵信息，并简化相似度矩阵信息，降低内存需求，缩减算法执行时间

```
def save_sims_table(filename):

    with open(filename,"w")as fs:
        for key in sim_table.keys():
            fs.write(str(key)+'\t'+str(sim_table[key])+'\n')

def load_sims_table(filename):
    print("---load_sim_table---")
    cnt = 0
    with open(filename,"r")as fr:
        for line in fr.readlines():
            cnt += 1
            if cnt%10000 == 0:
                print(cnt/10000)
            line = line.strip().split('\t')
            key,value = int(line[0]),float(line[1])
            sim_table[key]=value
```

我们做存储优化的代码：这样可以降低一半的内存用量

```
def read_simfile(filename1,filename2):
    sims_table = {}
    cnt = 0
    with open(filename1,"r") as fr,open(filename2,"w")as fw:
        for line in fr.readlines():
            cnt += 1
            if(cnt%10000==0):print(cnt/10000)
            line_proc = line.strip().split('\t')
            if line_proc[1] == "0.0":
                continue
            else:
                fw.write(line)


filename1=r"C:\Users\Strawberry\Desktop\Web_info_lab3\sim_table.txt"
filename2=r"C:\Users\Strawberry\Desktop\Web_info_lab3\sim_table2.txt"
read_simfile(filename1,filename2)
```

(4):采用10-NN（10近邻）大大缩减算法执行时间

```
def get_neiborghoods(infos,save_filepath):
    neiborghoods = {}
    cnt = 0
    for id1 in infos.keys():
        cnt += 1
        if(cnt%10 == 0):print(cnt)
        temp = {}
        for id2 in infos.keys():
            temp[id2] = get_sim(encode(id1,id2))
        top10 = dict(sorted(temp.items(),key=lambda e:e[1],reverse=True)[:10])
        top10 = list(top10.keys())
        neiborghoods[id1] = top10
    with open(save_filepath,"w") as fw:
        for key in neiborghoods.keys():
            fw.write(str(key)+'\t'+str(neiborghoods[key])+'\n')
```

原始有20000多个音乐，如果我们每个推荐计算时都用所有音乐进行相似度加权计算，那么计算时间将会变得巨长无比，通过10-NN近似，我们将计算时间开销节省了近2000倍，使得预测计算变得高效，可以接受。

@3：OJ上评测结果

您已经提交 **1** 次，剩余提交次数为 **9** 次。

提交时间	文件名称	Hit@20	Hit@100	NDCG@20	NDCG@100
2022-01-22 23:13:53	60_1642864433_result.txt	0.158693	0.268401	0.079091	0.099225

与baseline对比，

Model	$HR@20$	$HR@100$	$NDCG@20$	$NDCG@100$
Baseline	0.2249	0.4183	0.1092	0.1442

可以明显得出结论，虽然我们的简单模型的效果逊色于baseline的model但是也算明显有效地起到了推荐效果。