智能推荐系统第一次作业：
基于协同过滤的用户评分预测

徐珑珊
10195102506

2022 年 4 月 3 日

目录

1 算法介绍

协同过滤 (Collaborative Filtering, CF) 的核心想法是利用某兴趣相投、拥有共同经验之群体的喜好来推荐用户感兴趣的信息。通过分析目标用户与其他用户的相似度、目标物品与其他物品的相似度，过滤出相似的用户和物品，通过他们的历史行为，帮助预测目标用户对目标物品的喜爱程度。过滤不一定局限于特别相似的，特别不相似信息的记录也相当重要。

协同过滤分为两种，一种是基于用户的协同过滤 (User-based Collaborative Filtering, UCF)，另一种是基于物品的协同过滤 (Item-based Collaborative Filtering, ICF)。

1.1 基于用户的协同过滤算法

UCF 是通过不同用户对同一件物品的评分来评测用户之间喜好的相似性，基于用户之间的相似性做出推荐，这种算法给用户推荐和他兴趣相投的其他用户喜欢的物品。

基于用户的协同过滤步骤如下：

1. 给定用户集合 S_U、物品集合 S_I、 $user-item$ 评分矩阵 $mat[|S_U|][|S_I|]$，对 S_U 中的每一对用户 (u_a, u_b)，找到他们都评过分的共同物品，加入集合 S_i；
2. 考虑 u_a、u_b 的个人评分均值和对 $item \in S_i$ 的评分，计算 u_a、u_b 的相似度，可采用计算 Pearson Correlation、Jaccard Similarity、Consine Similarity 等方法, 构造相似度矩阵 $sim[|S_U|][|S_U|]$；
3. 给定目标用户 u 和目标物品 i，对于 $k \in S_U$，$k \neq u$，如果 k 评价过目标物品 i，那么将 k 加入 u 的邻居集合 S_n；
4. 对于 $n \in S_n$，用 $sim[u][n]$ 和 $mat[n][i]$，结合 u 的个人评分习惯，来预测 $mat[u][i]$，可采用平均值、加权平均值等方法。

1.2 基于物品的协同过滤算法

ICF 是通过用户对不同物品的评分来评测物品之间的相似性，基于物品之间的相似性做出推荐，即，给用户推荐和他之前喜欢的物品相似的物品。

基于物品的协同过滤步骤如下：

1. 给定用户集合 S_U、物品集合 S_I、 $user-item$ 评分矩阵 $mat[|S_U|][|S_I|]$，对 S_I 中的每一对物品 (i_a, i_b)，找到用户 $user$，满足 $user$ 给 i_a、i_b 都打过分，将这样的 $user$ 加入集合 S_u；
2. 对于 $user \in S_u$，根据 $mat[user][i_a]$ 和 $mat[user][i_b]$，计算 i_a、i_b 的相似度，可采用计算 Consine Similarity 的方法，构造相似度矩阵 $sim[|S_U|][|S_U|]$；
3. 给定目标用户 u 和目标物品 i，对于 $k \in S_I$，$k \neq i$，如果 u 评价过 k，那么将 k 加入 i 的邻居集合 S_n；
4. 对于 $n \in S_n$，用 $sim[i][n]$ 和 $mat[u][n]$，来预测 $mat[u][i]$，可采用平均值、加权平均值等方法。

2 核心代码注解

本节解释代码的核心部分，所列函数非完整代码。主要代码语言为 Python，考虑到计算相似度的部分由于复杂度较高，用 Python 计算时间较长，不方便调试，后改用 C++ 编写。

2.1 验证集划分

函数定义： split_valid_data(self, path, to_path, rate, round)

参数含义：

- path：train.csv 路径
- to_path：划分好的验证集输出文件路径
- rate：验证集条目数占 train.csv 总条目数比例
- round：改变验证集挑选起点

引入用户活跃度和物品活跃度。用户活跃度定义为该用户评分过的物品总数，物品活跃度定义为给该物品评过分的用户总数。

为了保证训练数据的完整性，先读入 train.csv，构造 user-item 评分矩阵，并计算用户、物品活跃度。只挑选用户活跃度和物品活跃度均不低于5的用户-物品-评分条目，选入验证集，并从训练集中去除，更新用户、物品活跃度。

```
    unum, inum = self.user_dict[i[0]], self.item_dict[i[1]]
    if ( self.user_active[unum] >= 5 and self.item_active[inum] >= 5 ):
        # write to the validation set
        writer.writerow(i)
        # remove from the training set
        self.user_active[unum] -= 1
        self.item_active[inum] -= 1
        self.matrix[unum][inum] = 0
        cnt += 1
    if ( cnt == total ): # make the validation set size of total
        break
```

由于下一步计算相似度转入 C++，划分完训练集后需要将 user-item 评分矩阵写入 .csv 文件中。

```
matwriter = csv.writer(open(matrix_path,'w',newline=''))
for i in range (len(self.matrix)):
    matwriter.writerow(self.matrix[i])
```

2.2 相似度计算

由于相似度计算的复杂度为 $O(m^2n)$ （对于 UCF，m 为用户数量，n 为物品数量；对于 ICF，m 为物品数量，n 为用户数量），试过用 Python 计算时间较长，因此改成 C++。

由于 UCF、ICF 的相似度计算过程类似，只是循环的含义不同，此处仅解释 UCF 的用户相似度计算过程。

首先需要读入划分完验证集之后在 Python 中计算得到的 user-item matrix、用户打分平均值 user_ave、物品得分平均值 item_ave。

调用函数：

void read_matrix(string path);

void read_list(string path, double * dst);

```
void read_list(string path, double * dst){
    ifstream inFile(path, ios::in);
    int i = 0;
    string line, field;
    while (getline(inFile, line)){// read by row
        string field;
        istringstream sin(line);
        getline(sin, field, ',');
        dst[i] = atof(field.c_str());
        ++ i;
    }
    inFile.close();
    cout << "total lines: " << i << endl;
    cout << "finish reading" << endl;
}
```

然后计算用户相似度。

函数定义：void user_sim_cal(string to_path);

参数含义：to_path：保存相似度矩阵的文件路径

```
    for ( int ua = 0; ua < U; ++ ua ){
        sim[ua][ua] = 1; // self-self similarity := 1
        for ( int ub = ua + 1; ub < U; ++ ub ){
            for ( int item = 0; item < I; ++ item ){
                if ( r_ai > 0 && r_bi > 0 ){ // a and b => i
                    ++ simcnt;
                    E += ( r_ai - r_a ) * ( r_bi - r_b );
                    Sa += ( r_ai - r_a ) * ( r_ai - r_a );
                    Sb += ( r_bi - r_b ) * ( r_bi - r_b );
                }
            }
            if ( simcnt > 5 && Sa * Sb != 0 )
                sim[ua][ub] = E / sqrt(Sa * Sb);
            else    // no enough info for sim calculation
                sim[ua][ub] = 0;
```

```
16              sim[ub][ua] = sim[ua][ub];
17          }
18      }
```

2.3 用户评分预测

同样以 UCF 为例。按相似度由高到低遍历所有邻居，如果该邻居给目标物品打过分，那么记录相关数值。遍历完后，根据相应公式计算预测值。将预测出来的评分限制到 [1, 5] 的区间。

```
1  def predict_1rating(self, unum, inum):
2      for u in range(len(self.user_dict)): # all users
3          unum_b = int(self.neighbor[unum][u]) # current neighbor
4          if ( unum == unum_b ):
5              continue
6          if ( r_bi != 0 ): # neighbor => item
7              total += ( r_bi - r_b ) * self.sim[unum][unum_b]
8              weigh += self.sim[unum][unum_b]
9              cnt += 1
10         if ( cnt == 3 ): # top-cnt neighbors
11             break
12     if ( weigh == 0 ):
13         pred_ui = self.user_ave_score[unum] # self.all_user_ave
14         self.disable += 1
15     else:
16         pred_ui = r_u + total / weigh
17     if ( pred_ui < 1 ):
18         pred_ui = 1
19     if ( pred_ui > 5 ):
20         pred_ui = 5
21     return np.squeeze(pred_ui)
```

3 结果分析

尝试了 UCF、ICF 的不同公式、参数、组合。榜单上的提交大致如下：

1. UCF, Pearson, top-3 neighbors, 退化成用户个人打分均值 (single user average, sua), RMSE 1.02865 (3.1节)
2. UCF, Cosine, all neighbors, sua, RMSE 1.00378 (3.1节)
3. ICF, Cosine & popular users penalized, sua, top-20 neighbors, RMSE 1.04457 (3.1节)
4. ICF, Cosine, similarity > 0.1, 退化成所有用户打分均值 (all user average, aua), RMSE 1.00263 (3.1节)
5. UCF, Pearson & Floyd, top-3 neighbors, aua, RMSE 1.00460 (3.2节)
6. UCF, Pearson & Floyd, top-1 neighbor, aua, RMSE 1.07210 (3.2节)

7. ICF, Cosine, sim > 0.8, sua, RMSE 0.95481 (3.3节)
8. ICF, Cosine, common users > 5, sim > 0.8, sua, RMSE 0.94520 (3.3节)
9. ICF, Cosine, common users > 5, sim > 0.8, aua, RMSE 0.90687 (3.3节)
10. UICF, Pearson for UCF & Cosine for ICF, common > 1, sim > 0.1 for UCF, sim > 0.6 for ICF, aua, RMSE 0.92198 (3.4节)

下面将这 10 次提交分为 4 部分作进一步的说明。

3.1 基本公式

前 4 次提交尝试了 UCF、ICF 及其基本的相似度计算公式。对 UCF 来说，相似度计算可以采用 Pearson Correlation 和 Cosine similarity 来计算，分别对应第 1、2 次提交。第 1 次使用 Pearson Correlation 作为相似度，考虑最相似的 3 位邻居；第 2 次使用余弦相似度，考虑所有邻居，按相似度加权平均。

对 ICF 来说，相似度计算可以采用余弦相似度（提交 4 ），课件上另有一个降低活跃用户权重的余弦相似度计算公式，对应提交 3。

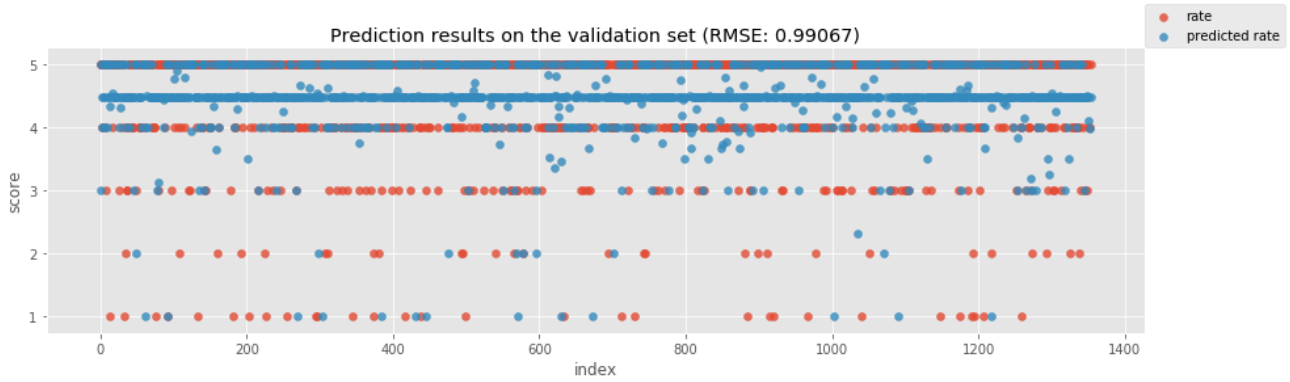

图 1: 第 4 次提交在验证集上的预测结果

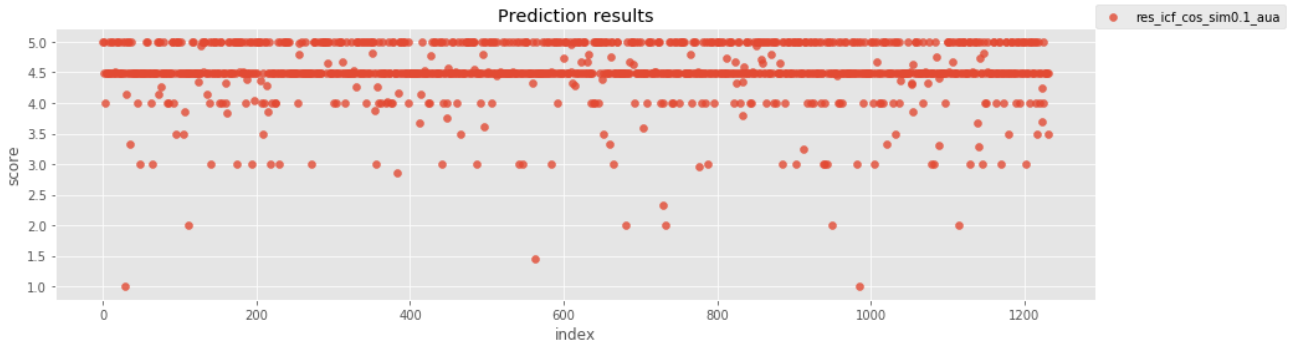

图 2: 第 4 次提交在测试集上的预测结果

这 4 次提交总得来说差别不大，预测点比较分散，能涵盖 $[1,5]$ 的评分取值区间。同时预测错误带来的误差比较大。

3.2 Floyd 算法扩展相似度

考虑到数据集较小，相似度矩阵十分稀疏，想尝试能否按照“A 和 B 相似，B 和 C 相似，那么 A 和 C 相似” 以及“A 和 B 相似，B 和 C 不相似，那么 A 和 C 不相似” 的逻辑来填充相似度矩阵。第 5、6 次提交对此做了尝试。

在 Floyd 最短路径算法的基础上修改，用 sim[A][B] * sim[B][C] 来更新 sim[A][C]。

代码如下：

```
void floyd(string to_path){
    for ( int k = 0; k < I; ++ k )
        for ( int i = 0; i < I; ++ i )
            for ( int j = 0; j < I; ++ j )
                if ( sim[i][j] == 0 && sim[i][k] * sim[k][j] > 0 )
                    sim[i][j] = sim[i][k] * sim[k][j];
}
```

使用这种方法确实能使得相似度矩阵不那么稀疏，但是相似度相乘后得到的相似度往往值很小，参考价值不大。根据在 kaggle 上的测试结果，这种方法对预测准确度没有明显提高。

3.3 退化成均值

由于相似度矩阵比较稀疏，且尝试填充没有起到较好的效果，只能考虑公式的退化（第 7、8、9 次提交）。对于大部分用户、物品之间计算出的相似度是算不出来的，找不到可以使用的邻居，预测全部采用用户个人打分均值或所有用户打分均值。

另外，由于数据集太小，可以认为共同交互比较少的用户或物品之间算出来的相似度并不准确，不予考虑，仍旧退化成均值。

第 8、9 次提交中的 common users > 5 就是对共同交互的次数做了一个限制：对于两个物品，如果没有超过 5 个用户同时给它们打过分，那么就认为无法计算它们之间的相似度。

这一约束导致验证集/测试集上只有个别点采用了预测，其他都由均值填充。

和图1相比，相当于几乎没做预测，但由于更符合数据集整体的分布规律， RMSE 的值比较小。

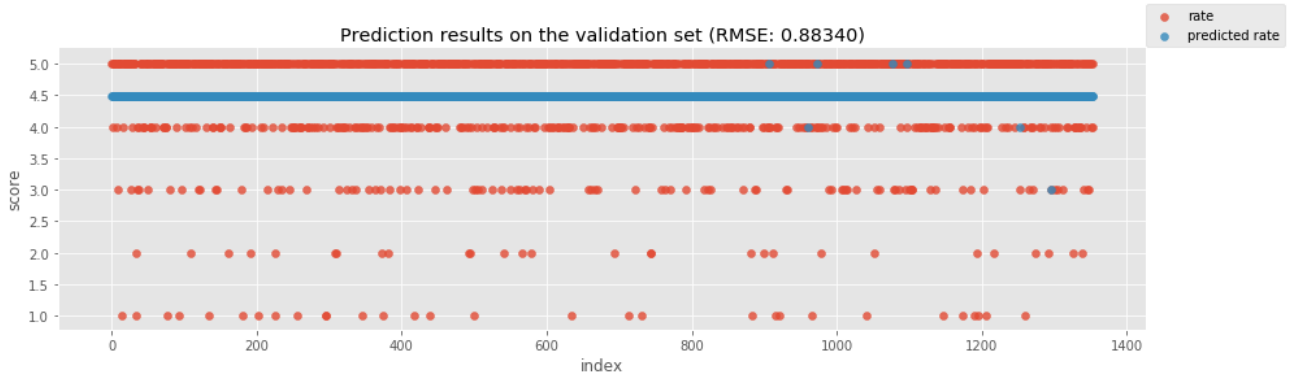

图 3: 第 9 次提交在验证集上的预测结果

3.4 UCF-ICF 结合

最后的第10次提交尝试了综合上述提交，UCF 与 ICF 结合的方法。用户相似度的计算采用 Pearson Correlation，物品相似度的计算采用余弦相似度，计算相似度时，共同交互至少为 2，认为能够计算相似度。 UCF 的预测将相似度阈值定在 0.1（考虑到基于用户的预测是在用户个人打分均值基准上浮动，相似度小的用户影响本身就比较小）， ICF 的预测将相似度定在 0.6。无法获得有效邻居的退化成所有用户打分均值。最后将 UCF 得到的预测值和 ICF 得到的预测值相加取平均。

结合 UCF、ICF 后，可以从图 4 看出，有一部分的预测点偏离了所有用户均值，向 4 分、5 分靠拢。猜想在调参到一个合适的数值后，能够更准确得预测准 4 分、5 分的情况，也就是大部分评分的情况，从而进一步减小 RMSE。

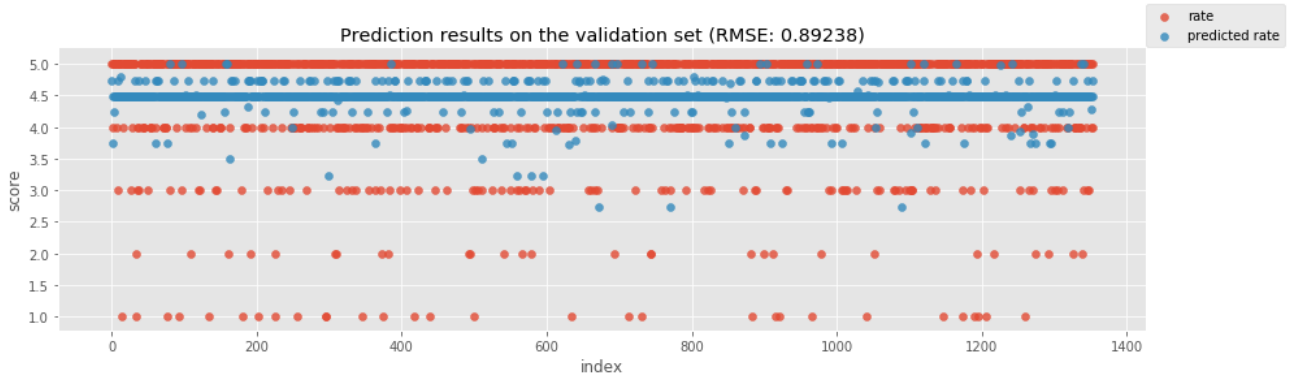

图 4: 第 10 次提交在验证集上的预测结果

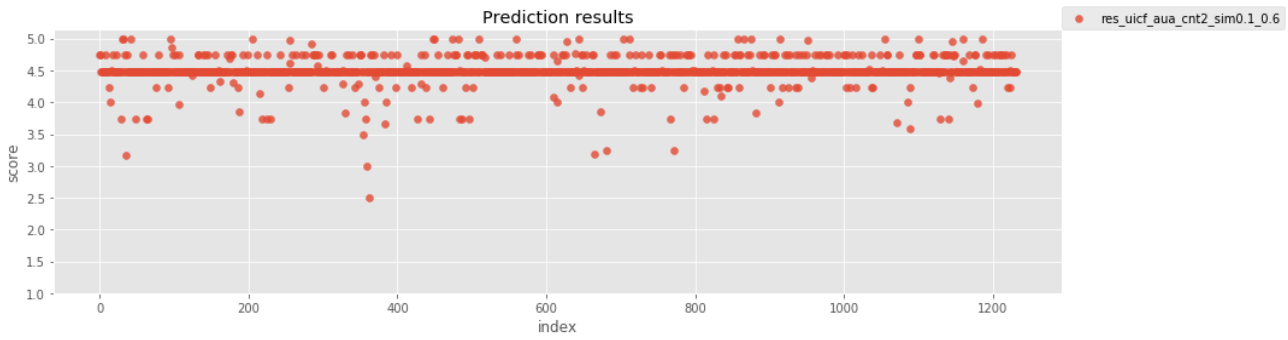

图 5: 第 10 次提交在测试集上的预测结果

4 代码运行

4.1 环境

Python 环境：

- Anaconda3-5.2.0-Windows-x86_64
- Python 3.6

C++ 环境：

- gcc version 8.2.0 (MinGW.org GCC-8.2.0-3)

运行文件夹目录：

- Assign-1-data
  - train.csv
  - test.csv
- Assign-1-result
- usimcal.cpp
- isimcal.cpp
- ucf.ipynb
- icf.ipynb
- uicf.ipynb

4.2 运行

代码以 .ipynb 和 .cpp 格式的文件提交。

UCF 运行方式：

依次执行 ucf.ipynb 中的 cell，完成导入数据集、划分验证集、计算平均值等工作。

直到写有“# 计算 similarity C++”的 cell，去运行 usimcal.cpp，计算并导出用户相似度矩阵。

然后回到 ucf.ipynb，继续执行 cell，完成导入相似度矩阵、邻居排序，以及在验证集上验证、在测试集上测试推荐效果，并且可视化推荐结果。

ICF 运行方式：

依次执行 icf.ipynb 中的 cell，完成导入数据集、划分验证集、计算平均值等工作。

直到写有“# 计算 similarity C++”的 cell，去运行 isimcal.cpp，计算并导出用户相似度矩阵。

然后回到 icf.ipynb，继续执行 cell，完成导入相似度矩阵、邻居排序，以及在验证集上验证、在测试集上测试推荐效果，并且可视化推荐结果。

UICF 运行方式：

依次执行 uicf.ipynb 中的 cell，完成导入数据集、划分验证集、计算平均值等工作。

直到写有“# 计算 user/item similarity C++”的 cell，去运行 usimcal.cpp 和 isimcal.cpp，计算并导出用户、物品相似度矩阵。

然后回到 uicf.ipynb，继续执行 cell，完成导入相似度矩阵、邻居排序，以及在验证集上验证、在测试集上测试推荐效果，并且可视化推荐结果。

备注：

- 相似度矩阵文件导出、导入路径需匹配；
- simcal.cpp 中有多个计算相似度的函数可选，代表不同的计算方法；
- 训练集不改变的话，调整预测用的相似度阈值，不需要重新计算相似度矩阵；
- 代码中涉及到修改预测所用阈值的部分已框出。

5 提交文件列表

5.1 Code

- ucf.ipynb：基于用户的协同过滤算法代码
- icf.ipynb：基于物品的协同过滤算法代码
- uicf.ipynb：基于用户、物品混合的协同过滤算法代码
- usimcal.cpp：计算用户相似度矩阵的代码
- isimcal.cpp：计算物品相似度矩阵的代码

5.2 SimMatrix

- usim.csv：用户相似度矩阵
- isim.csv：物品相似度矩阵