# 华东师范大学计算机科学技术系上机实践报告

**课程名称:** 智能推荐系统　　**年级:** 2018　　**上机实践成绩:**
**指导教师:** 张伟　　**姓名:** 张翼鹏　　**创新实践成绩:**
**实验编号:** 01　　**学号:** 10185102105　　**上机实践日期:** 2021/04/04
**实验名称:** Memory-based CF for Rating Prediction

## 1 问题重述

本次实践要求使用基于记忆的协同过滤算法对指定用户对指定物品的打分做出预测，其中给出的数据包括 train.csv 文件和 test.csv 文件，前者每行数据包含了用户、商品、打分分值和打分时间，后者为测试文件，要求将预测结果输出到该文件中。

## 2 算法原理

基于记忆的协同过滤算法 (Memory-based Collaborative Filtering) 直接使用整个已知的 (所谓“记忆”) 评分矩阵去预测用户对未知项目的评分或者推荐，又可以分为基于用户的协同过滤算法 (User-based nearest-neighbor Collaborative Filtering, UCF) 和基于物品的协同过滤算法 (Item-based nearest-neighbor Collaborative Filtering, ICF)。它最基本的优势在于实现简单，推荐精度较高; 它主要的劣势是对数据稀疏敏感，可扩展性不强。本次实验将使用这两种核心算法进行对用户商品评分的预测。

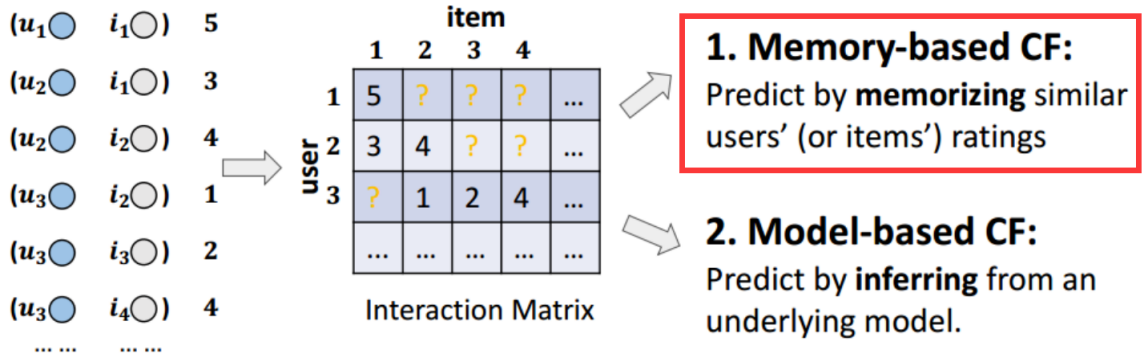



图 1: 协同过滤算法

### 2.1 UCF

#### 2.1.1 算法思想

基于用户的协同过滤算法 *(UCF)* 是推荐系统中最古老的算法。可以不夸张地说，这个算法的诞生标志了推荐系统的诞生。该算法在 *1992* 年被提出，并应用于邮件过滤系统，*1994* 年被 *GroupLens* 用于新闻过滤。在此之后直到 *2000* 年，该算法都是推荐系统领域最著名的算法。本节将对该算法进行详细介绍，首先介绍最基础的算法，然后在此基础上提出不同的改进方法，并通过真实的数据集进行评测... 在一个在线个性化推荐系统中，当一个用户 *A* 需要个性化推荐时，

可以先找到和他有相似兴趣的其他用户，然后把那些用户喜欢的、而用户 $A$ 没有听说过的物品推荐给 $A$。这种方法称为基于用户的协同过滤算法。

——《推荐系统实践》$P44$，项亮著，人民邮电出版社出版

#### 2.1.2 基本步骤

从上面的描述中可以看到，基于用户的协同过滤算法主要包括两个步骤：

(1) 找到和目标用户兴趣相似的用户集合。

(2) 找到这个集合中的用户喜欢的，且目标用户没有听说过的物品推荐给目标用户。

本任务要求预测评分，因此第二条在表述上可以调整为“根据这个集合中对待预测物品打过分的用户的历史打分，按照一定权重（比如相似度越大、间隔时间越短权重越大）预测分值”。

因此可以看出整个算法的关键就是计算两个用户的兴趣相似度。这里协同过滤算法主要利用用户行为的相似度计算兴趣的相似度。给定用户 $a$ 和用户 $b$，令 $sim(a,b)$ 表示二者之间的相似度，采取如下两种方式进行计算：

(1) Pearson 相似度：

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

其中，集合 $P$ 表示用户 $a$ 和用户 $b$ 都参与过评分的商品集合，$r_{a,p}, r_{b,p}$ 分别表示用户 $a$ 和用户 $b$ 对商品 $p$ 打的分值，$\bar{r}_a, \bar{r}_b$ 分别表示用户 $a$ 和用户 $b$ 对所有打过分的商品的历史平均打分值。

(2) 改进的余弦相似度：

$$\mathrm{sim}(a,b)=\frac{\sum_{i\in N(a)\cap N(b)}\frac{1}{1+\alpha|t_{ai}-t_{bi}|}}{\sqrt{|N(a)||N(b)|}}$$

其中，集合 $N(A)$ 和 $N(b)$ 分别表示用户 $a$ 和用户 $b$ 给予过积极评价的商品集合，$t_{ai}, t_{bi}$ 分别表示用户 $a$ 和用户 $b$ 对商品 $i$ 打分的时间，$\alpha$ 为常参数。

注意计算相似度的方法还有很多，如 Jaccard 等，这里根据数据的特征只考虑了这两种方法。最终预测用户 $a$ 对商品 $p$ 的评分采用如下公式：

$$\mathrm{pred}(a,p)=\left(\frac{\sum_{b\in N}\mathrm{sim}(a,b)*\left(r_{b,p}-\overline{r_b}\right)*\frac{1}{1+\alpha|t_0-t_{b,p}|}}{\sum_{b\in N}\mathrm{sim}(a,b)}\right)$$

其中，$N$ 表示与用户 $a$ 相似的用户集合，$sim(a,b)$ 表示用户 $a$ 和用户 $b$ 的相似度，$r_{b,p}$ 表示用户 $b$ 对物品 $p$ 的打分，$\overline{r_b}$ 表示用户 $b$ 对其他所有打过分的物品的历史平均打分，$t_0$ 表示预测任务的时间，$t_{b,p}$ 表示用户 $b$ 对物品 $p$ 评分的时间，代入相应数据即可预测用户 $a$ 对商品 $p$ 的评分。

### 2.2 ICF

#### 2.2.1 算法思想

基于物品的协同过滤算法 *(ICF)* 是目前业界应用最多的算法。无论是亚马逊网，还是 *Netflix*、*Hulu*、*YouTube*，其推荐算法的基础都是该算法。本节将从基础的算法开始介绍，然后提出算法的改进方法，并通过实际数据集评测该算法。基于用户的协同过滤算法在一些网站 *(*如 *Digg)* 中得到了应用，但该算法有一些缺点。首先，随着网站的用户数目越来越大，计算用户兴趣相似度矩阵将越来越困难，其运算时间复杂度和空间复杂度的增长和用户数的增长近似于平方关系。其次，基于用户的协同过滤很难对推荐结果作出解释... 而该算法 *(ICF)* 并不利用物品的内容属性计算

*物品之间的相似度，它主要通过分析用户的行为记录计算物品之间的相似度。该算法认为，物品 A 和物品 B 具有很大的相似度是因为喜欢物品 A 的用户大都也喜欢物品 B... 基于物品的协同过滤算法可以利用用户的历史行为给推荐结果提供推荐解释。*

*——《推荐系统实践》P51-52，项亮著，人民邮电出版社出版*

### 2.2.2 基本步骤

基于物品的协同过滤算法主要分为两步：

(1) 计算物品之间的相似度。

(2) 根据物品的相似度和用户的历史行为给用户生成推荐列表。

类似 2.1.2 节，根据任务要求，第二条在表述上可以调整为“根据这个集合中对待其他用户的历史打分，按照一定权重（比如相似度越大、间隔时间越短权重越大）预测分值”，给定商品 $a$ 和商品 $b$，令 $sim(a,b)$ 表示二者之间的相似度，同样给出两种计算商品间相似度的方法：

(1) 改进的余弦相似度：

$$\mathrm{sim}(a,b)=\frac{\sum_{u\in N(a)\cap N(b)}\frac{1}{\log(1+|N(u)|)}}{\sqrt{|N(a)||N(b)|}}$$

其中 $N(a)$ $N(b)$ 分别表示给商品 $a$ $b$ 积极评价的用户集合，$N(u)$ 表示用户 $u$ 打分过的商品集合。

(2) 引入时间因素的余弦相似度：

$$\mathrm{sim}(a,b)=\frac{\sum_{u\in N(a)\cap N(b)}\frac{1}{1+\alpha|t_{ua}-t_{ub}|}}{\sqrt{|N(a)||N(b)|}}$$

其中 $N(a)$ $N(b)$ 分别表示给商品 $a$ $b$ 积极评价的用户集合，$t_{ua}$ $t_{ub}$ 分别表示用户给商品评分的时间。

最终预测用户 $u$ 对商品 $p$ 的评分采用如下公式：

$$\mathrm{pred}(u,p)=\frac{\sum_{i\in ratedItem(u)}\mathrm{sim}(i,p)*r_{u,i}*\frac{1}{1+\beta|t_0-t_{ui}|}}{\sum_{i\in ratedItem(u)}\mathrm{sim}(i,p)}$$

其中集合 $ratedItem(u)$ 表示和物品 $p$ 相似、且曾被用户 $u$ 打过分的商品集合。

# 3 算法实现

首先设置与用户的交互模式，通过主程序调用目录下不同的文件，主程序如下：

```
import os
print('执行中，请稍等...')
print('请输入数字选择模式，1 代表调参模式，2 代表预测模式：')
print('（备注：测试模式不写 csv，按照 8：2 划分数据集，输出 MAE 和 RMSE；\
预测模式将把预测结果写入 test.csv）')
op1 = int(input())
while(op1 != 1 and op1 != 2):
    print('输入数字无效，请重新输入！')
    print('请输入数字选择核心算法：1 代表 UCF，2 代表 ICF：')
    op1 = int(input())
```

```
print('请输入数字选择核心算法: 1 代表 UCF, 2 代表 ICF: ')
op2 = int(input())
while(op2 != 1 and op2 != 2):
    print('输入数字无效, 请重新输入! ')
    print('请输入数字选择核心算法: 1 代表 UCF, 2 代表 ICF: ')
    op2 = int(input())
print('数据初始化中, 请稍等...')
if(op1 == 1):
    if(op2 == 1):
        os.system('UCF_test.py')
    else:
        os.system('ICF_test.py')
else:
    if(op2 == 1):
        os.system('UCF.py')
    else:
        os.system('ICF.py')
```

## 3.1 UCF

根据“算法原理”章节中叙述的内容，具体实现 UCF，下文将分别叙述数据结构和核心程序。

### 3.1.1 数据结构

通览整个算法流程可以看出，UCF 的实现有很多不确定性，比如计算用户间的相似度、引入时间元素后的 $\alpha$ 系数等，为了初步把握这些不确定因素，考虑在实现 UCF 前先粗略评估一下这两种相似度计算方法和 $\alpha$ 的范围，因此将给出的数据集按照 8:2 的比例划分为训练集和测试集，根据测试集评估效果（通常取 7:3 的比例，考虑到数据量有限、用户评分矩阵高度稀疏，这里加大训练集占比）。

主要的数据结构包括：

- Data 类

  将 csv 文件中的每一行抽象成一条 Data，其中包括用户 id、商品 id、打分时间、分值

- User 类

  包括用户打过分的物品集合 rated_business、用户历史平均评分 avg_star、与用户相似的用户集合 sim_userset

- dataset，存放全部数据
- trainset，存放训练集数据
- testset，存放测试集数据
- userset，存放用户数据

- sim_matrix，用户相似度矩阵，sim_matrix[i][j]=sim_matrix[j][i] 表示用户 $i$ 和 $j$ 之间的相似度
- scoring_matrix，打分矩阵，scoring_matrix[i][j] 表示用户 $i$ 给物品 $j$ 打分的分值
- time_matrix，打分时间矩阵，scoring_matrix[i][j] 表示用户 $i$ 给物品 $j$ 打分的时间
- business_avg_star，物品被打的平均分值

### 3.1.2 核心程序

首先为了便于索引，将 csv 中的用户和商品字符串都映射到整形 id 上，从零开始编号（下文 ICF 中将不再叙述）：

```
# 读取 train.csv 文件
f = open('train.csv')
df = pd.read_csv(f)
f.close()
# 构造用户和商品的 id 字典
user_index = 0
business_index = 0
user_dict = {}
business_dict = {}
for index, row in df.iterrows():
    user_id = row["user_id"]
    business_id = row["business_id"]
    if user_id not in user_dict:
        user_dict[user_id] = user_index
        user_index += 1
    if business_id not in business_dict:
        business_dict[business_id] = business_index
        business_index += 1
```

计算用户相似度（这里可供选择计算方式，在命令行输入）：

```
# 选择相似度计算方式
print('请输入数字选择相似度计算方式，1 代表 Pearson 相似度，2 代表余弦相似度：')
op2 = int(input())
while(op2 != 1 and op2 != 2):
    print('输入数字无效，请重新输入！')
    print('请输入数字选择相似度计算方式，1 代表 Pearson 相似度，2 代表余弦相似度：')
    op2 = int(input())

print('计算中，请稍等...')
for i in range(user_index-1):
    for j in range(i+1, user_index):
```

```
        # 计算 Pearson 相似性，将非负值纳入计算范围
        if(op2 == 1):
            P = list(set(userset[i].rated_business).
            intersection(set(userset[j].rated_business)))
            if(len(P) > 0):
                up = 0.0
                downa = 0.0
                downb = 0.0
                for business in P:
                    up += (scoring_matrix[i][business]-userset[i].avg_star)
                    *(scoring_matrix[j][business]-userset[j].avg_star)
                    downa += (scoring_matrix[i][business]-userset[i].avg_star)**2
                    downb += (scoring_matrix[j][business]-userset[j].avg_star)**2
                downa = downa**0.5
                downb = downb**0.5
                if(downa*downb<1e-6):
                    continue
                sim_matrix[i][j] = up/(downa*downb)
                sim_matrix[j][i] = sim_matrix[i][j]
                if sim_matrix[i][j] > 0.5:
                    userset[i].sim_userset.append(j)
                    userset[j].sim_userset.append(i)
        # 计算余弦相似性
        else:
            Na = []
            Nb = []
            '''
            用户 a 和 b 给出积极评价的商品集合
            这里以大于等于用户的历史平均打分为评价标准
            '''
            for business in userset[i].rated_business:
                if(scoring_matrix[i][business] >= userset[i].avg_star):
                    Na.append(business)
            for business in userset[j].rated_business:
                if(scoring_matrix[j][business] >= userset[j].avg_star):
                    Nb.append(business)
            N = list(set(Na).intersection(set(Nb)))
            if(len(N) > 0):
                up = 0
                down = (len(Na)*len(Nb))**0.5
                for business in N:
                    up += 1/(1+alpha*math.fabs(time_matrix[i][business]
```

```
                    -time_matrix[j][business]))
                if(down > 1e-6):
                    sim_matrix[i][j] = up/down
                    sim_matrix[j][i] = sim_matrix[i][j]
```

预测部分核心代码:

```
result = []
for index, row in df.iterrows():
    user = row["user_id"]
    business = row["business_id"]
    date = row["date"]
    user_id = user_dict[user]
    business_id = business_dict[business]
    timeStamp = time.mktime(datetime.strptime(date, "%Y/%m/%d %H:%M").
    timetuple())/(30*24*3600)
    sim_userset = userset[user_id].sim_userset
    data = Data(user_id, business_id, timeStamp, 0)
    '''
    处理冷启动问题:
    如果某用户没有任何行为数据，则预测值将取所有其他用户对该物品打分的平均值
    '''
    # 若该用户之前对该物品打过分，直接取原来的分值
    if scoring_matrix[user_id][business_id] > 1e-6:
        preq = scoring_matrix[user_id][business_id]
    elif(userset[user_id].avg_star > 1e-6):
        up = 0.0
        down = 0.0
        preq = userset[user_id].avg_star
        for sim_user in sim_userset:
            if(scoring_matrix[sim_user][business_id] > 1e-6):
                up += sim_matrix[user_id][sim_user]*(scoring_matrix[sim_user]
                [business_id]-userset[sim_user].avg_star)*(1/(1+alpha
                *math.fabs(data.timeStamp-time_matrix[sim_user][business_id])))
                down += sim_matrix[user_id][sim_user]
        # python 精度问题，防止除零
        if(down > 1e-6):
            preq += up/down
    else:
        preq = business_avg_star[business_id]
    result.append([user, business, date, "{:.2f}".format(preq)])
```

## 3.2 ICF

### 3.2.1 数据结构

类比 UCF 的对称部分，ICF 相似度的计算也给出了两种方式，综合分析后，主要的数据结构包括：

- Data 类

  将 csv 文件中的每一行抽象成一条 Data，其中包括用户 id、商品 id、打分时间、分值

- User 类

  包括用户打过分的物品集合 rated_business、用户历史平均评分 avg_star

- Business 类

  包括与商品相似的商品集合 sim_business 、给该商品过积极评价的用户集 user_with _positive_feedback

- dataset，存放全部数据
- trainset，存放训练集数据
- testset，存放测试集数据
- userset，存放用户数据
- sim_matrix，商品相似度矩阵，sim_matrix[i][j]=sim_matrix[j][i] 表示商品 $i$ 和 $j$ 之间的相似度
- scoring_matrix，打分矩阵，scoring_matrix[i][j] 表示用户 $i$ 给物品 $j$ 打分的分值
- time_matrix，打分时间矩阵，scoring_matrix[i][j] 表示用户 $i$ 给物品 $j$ 打分的时间
- business_avg_star，物品被打的平均分值

### 3.2.2 核心程序

计算商品相似度（这里可供选择计算方式，在命令行输入）：

```
# 构造商品相似度矩阵
sim_matrix = np.zeros([business_index, business_index])
alpha = 1.0 / 18 # 希望相差 18 个月，影响降为一半
beta = 1.0 / 18 # 希望相差 18 个月，影响降为一半

# 选择相似度计算方式
print('请输入数字选择相似度计算方式，\
1 代表改进的余弦相似度，2 代表引入时间因素的余弦相似度：')
op2 = int(input())
while(op2 != 1 and op2 != 2):
    print('输入数字无效，请重新输入！')
    print('请输入数字选择相似度计算方式，1 代表改进的余弦相似度，\
```

```
    2 代表引入时间因素的余弦相似度: ')
    op2 = int(input())

print('计算中, 请稍等...')
for i in range(business_index-1):
    for j in range(i+1, business_index):
        # 计算改进的余弦相似性, 将非负值纳入相似范围
        Na = business_set[i].user_with_positive_feedback
        Nb = business_set[j].user_with_positive_feedback
        N = list(set(Na).intersection(set(Nb)))
        up = 0.0
        down = (len(Na) * len(Nb)) ** 0.5
        if(op2 == 1):
            for user in N:
                Nu = userset[user].rated_business
                if(len(Nu) > 1e-6):
                    up += 1/(math.log(1+len(Nu)))
        # 计算引入时间的余弦相似性, 将非负值纳入相似范围
        else:
            for user in N:
                up += 1/(1+alpha*math.fabs(time_matrix[user][i]-time_matrix[user][j]))
        if (down > 1e-6):
            sim_matrix[i][j] = up / down
            sim_matrix[j][i] = sim_matrix[i][j]
        # 值非负认为相似
        if(sim_matrix[i][j] > 1e-6):
            business_set[i].sim_business.append(j)
            business_set[i].sim_business.append(j)
```

预测部分核心代码:

```
result = []
for index, row in df.iterrows():
    user = row["user_id"]
    business = row["business_id"]
    date = row["date"]
    user_id = user_dict[user]
    business_id = business_dict[business]
    timeStamp = time.mktime(datetime.strptime(date, "%Y/%m/%d %H:%M")
    .timetuple())/(30*24*3600)
    sim_business = business_set[business_id].sim_business
    data = Data(user_id, business_id, timeStamp, 0)

```

```
    '''
    处理冷启动问题:
    如果某用户没有任何行为数据，则预测值将取所有其他用户对该物品打分的平均值
    如果要预测的用户和物品都没有任何数据，取所有用户平均打分
    '''
    # 若该用户之前对该物品打过分，直接取原来的分值
    if scoring_matrix[user_id][business_id] > 1e-6:
        preq = scoring_matrix[user_id][business_id]
    else:
        preq = business_avg_star[business_id]
        rated_item = []
        for sim_business in userset[user_id].rated_business:
            if(sim_matrix[business_id][sim_business] > 1e-6):
                rated_item.append(sim_business)
        up = 0.0
        down = 0.0
        for sim_business in rated_item:
            up += sim_matrix[business_id][sim_business]*scoring_matrix[user_id]
            [sim_business]*(1/(1+beta*math.fabs
            (timeStamp-time_matrix[user_id][sim_business])))
            down += sim_matrix[business_id][sim_business]
        if(down > 1e-6):
            preq = up/down
    result.append([user, business, date, "{:.2f}".format(preq)])
```

# 4 实验结果

## 4.1 文件结构

文件结构和占比如下图所示:

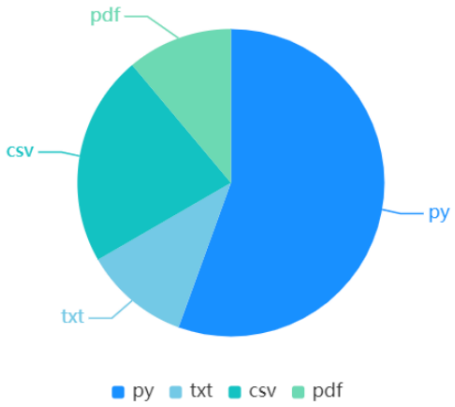



图 2: 文件结构

目录结构如下图所示：

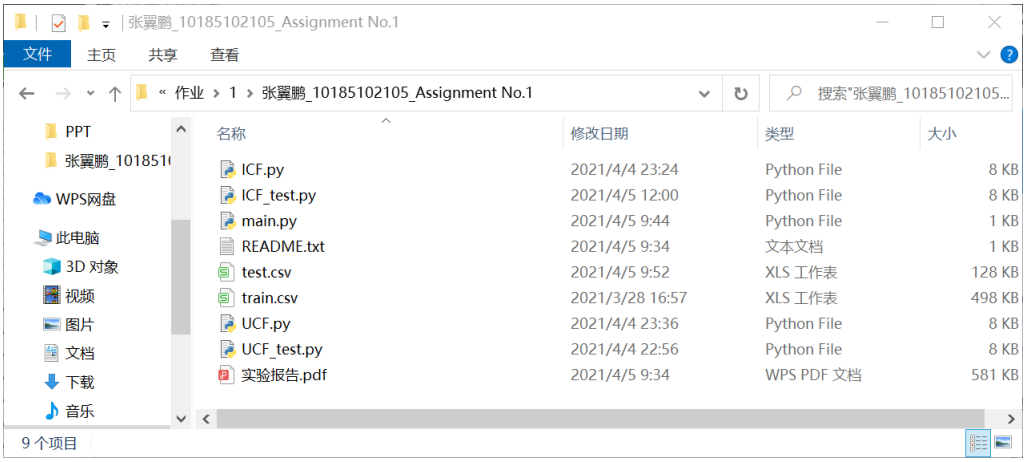



图 3: 目录结构

其中 main.py 是需要执行的主程序文件，其可以调用'UCF.py'、'ICF.py'、'UCF_test.py'、'ICF_test.py' 四个文件，前两个用于预测并填表，后两个用户调参和评估，'README.txt' 记录了运行方法，' 实验报告.pdf(即本文件)' 记录了本次实践的相关内容。

## 4.2 运行方式

在程序文件夹下命令行执行'python main.py'，交互选择运行模式和相似度计算方式：

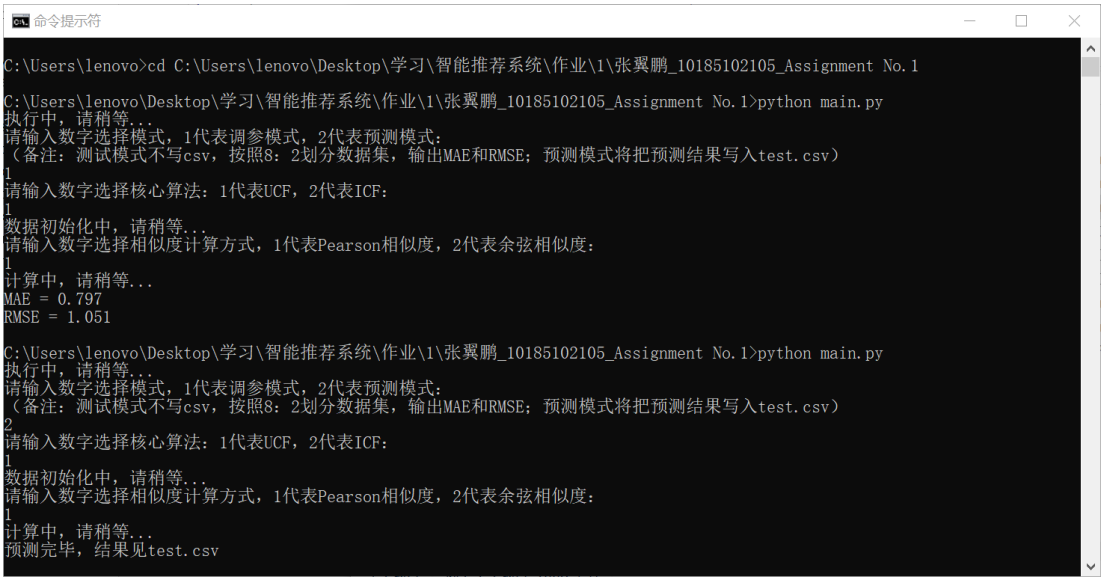
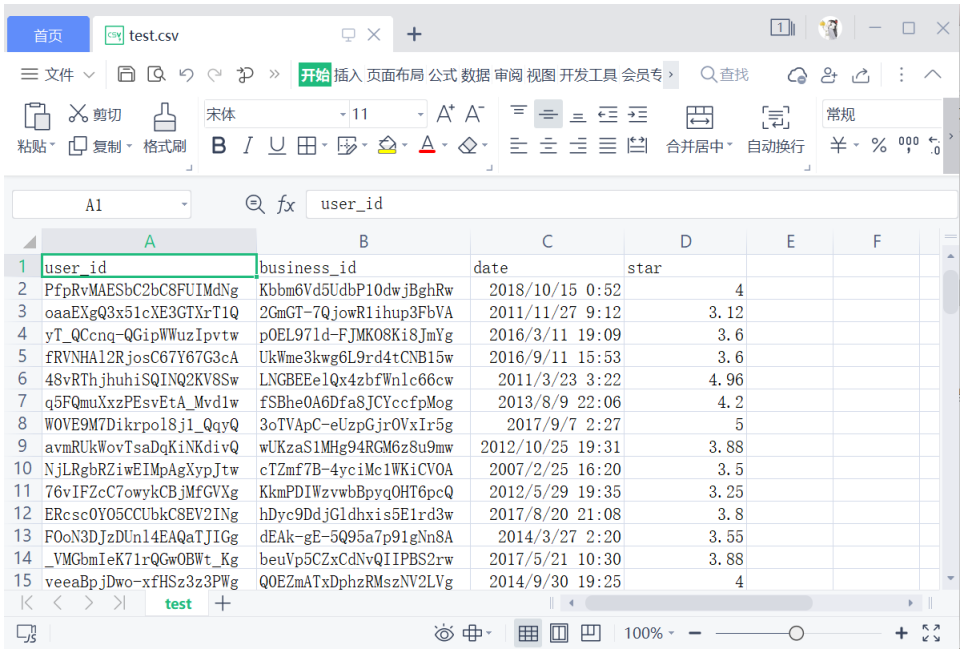



图 4: 执行实例

## 4.3 输出结果

以预测模式下 UCF 算法的 Pearson 相似度为例，输出结果到 test.csv 中：



| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| 1 | user_id | business_id | date | star | | |
| 2 | PfpRvMAESbC2bC8FUIMdNg | Kbbm6Vd5UdbP10dwjBghRw | 2018/10/15 0:52 | 4 | | |
| 3 | oaaEXgQ3x51cXE3GTXrT1Q | 2GmGT-7QjowR1ihup3FbVA | 2011/11/27 9:12 | 3.12 | | |
| 4 | yT_QCcnq-QGipWWuzIpvtw | pOEL971d-FJMKO8Ki8JmYg | 2016/3/11 19:09 | 3.6 | | |
| 5 | fRVNHAl2RjosC67Y67G3cA | UkWme3kwg6L9rd4tCNB15w | 2016/9/11 15:53 | 3.6 | | |
| 6 | 48vRThjhuhiSQINQ2KV8Sw | LNGBEEelQx4zbfWnlc66cw | 2011/3/23 3:22 | 4.96 | | |
| 7 | q5FQmuXxzPEsvEtA_Mvd1w | fSBheOA6Dfa8JCYccfpMog | 2013/8/9 22:06 | 4.2 | | |
| 8 | WOVE9M7Dikrpol8j1_QqyQ | 3oTVApC-eUzpGjrOVxIr5g | 2017/9/7 2:27 | 5 | | |
| 9 | avmRUkWovTsaDqKiNKdivQ | wUKzaS1MHg94RGM6z8u9mw | 2012/10/25 19:31 | 3.88 | | |
| 10 | NjLRgbRZiwEIMpAgXypJtw | cTZmf7B-4yciMc1WKiCVOA | 2007/2/25 16:20 | 3.5 | | |
| 11 | 76vIFZcC7owykCBjMfGVXg | KkmPDIWzvwbBpyqOHT6pcQ | 2012/5/29 19:35 | 3.25 | | |
| 12 | ERcscOYO5CCUbkC8EV2INg | hDyc9DdjGldhxis5E1rd3w | 2017/8/20 21:08 | 3.8 | | |
| 13 | FOoN3DJzDUnl4EAQaTJIGg | dEAk-gE-5Q95a7p91gNn8A | 2014/3/27 2:20 | 3.55 | | |
| 14 | _VMGbmIeK71rQGwOBWt_Kg | beuVp5CZxCdNvQIIPBS2rw | 2017/5/21 10:30 | 3.88 | | |
| 15 | veeaBpjDwo-xfHSz3z3PWg | QOEZmATxDphzRMszNV2LVg | 2014/9/30 19:25 | 4 | | |

图 5: 预测实例-UCF_Pearson

# 5 评估与思考

## 5.1 简要评估

对于不同算法的不同相似度计算方式做了简要的评估，篇幅所限，记录每种组合结果测试 7 次如下：

| 算法/相似度公式 | 测试 1 | 测试 2 | 测试 3 | 测试 4 | 测试 5 | 测试 6 | 测试 7 |
|---|---|---|---|---|---|---|---|
| UCF/Pearson 相似度 | 0.810 | 0.802 | 0.839 | 0.794 | 0.801 | 0.794 | 0.836 |
| UCF/引入时间的余弦相似度 | 0.809 | 0.806 | 0.790 | 0.821 | 0.805 | 0.820 | 0.826 |
| ICF/改进的余弦相似度 | 1.030 | 1.110 | 1.060 | 1.088 | 1.040 | 1.081 | 1.092 |
| ICF/引入时间的余弦相似度 | 1.464 | 1.115 | 1.083 | 1.047 | 1.017 | 1.178 | 1.468 |

表 1: 不同组合下的 MAE 值

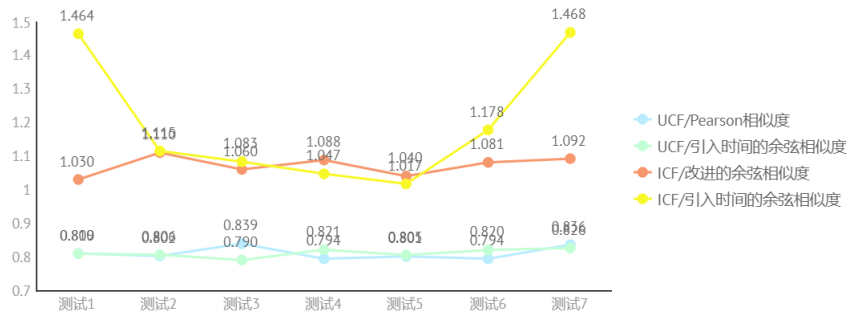



图 6: MAE 折线图

| 算法/相似度公式 | 测试 1 | 测试 2 | 测试 3 | 测试 4 | 测试 5 | 测试 6 | 测试 7 |
|---|---|---|---|---|---|---|---|
| UCF/Pearson 相似度 | 1.064 | 1.047 | 1.086 | 1.041 | 1.053 | 1.033 | 1.101 |
| UCF/引入时间的余弦相似度 | 1.045 | 1.070 | 1.049 | 1.079 | 1.059 | 1.085 | 1.090 |
| ICF/改进的余弦相似度 | 1.425 | 1.521 | 1.461 | 1.490 | 1.426 | 1.470 | 1.468 |
| ICF/引入时间的余弦相似度 | 1.707 | 1.522 | 1.504 | 1.435 | 1.386 | 1.469 | 1.730 |

表 2: 不同组合下的 RMSE 值

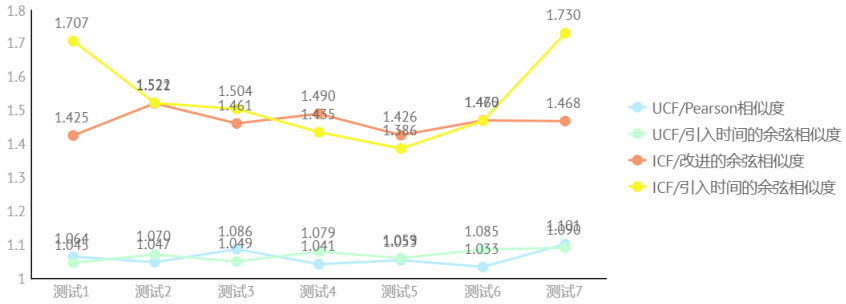



图 7: RMSE 折线图

观察以上图表内容可以发现，UCF 算法在本次任务中表现更为优秀。

## 5.2 思考问题

### 5.2.1 参数估计

本次实践中计算相似度和计算预测值时由于引入了时间数据，涉及到了常参数 $\alpha$ 和 $\beta$，这里将时间戳都转换为月份，两个时间戳的差值表示月份的差值，规定行为时间差在 18 个月（一年半）时的影响降为一半，故取 $\alpha = \beta = \frac{1}{18}$，这里也许采用更精确的方法对参数进行预测。

### 5.2.2 冷启动问题

由于测试时将 train.csv 进行了划分，在预测 testset 中的评分时可能会遇到用户或物品没有历史数据的情况，针对这种冷启动问题，采用如下方式处理：

(1) 如果用户没有行为数据而商品有，则采用其他对商品评分过的用户的平均分作为预测值；

(2) 如果商品没有历史数据而用户有，则采用该用户对其他商品打分的平均分作为预测值；

(3) 如果用户和商品都没有历史数据，采用打分矩阵中所有非零值的平均值作为预测值。

也许此处可以改进。

# 6 实践总结

本次实践基本实现了两种算法 UCF 和 ICF，并设置了交互选择模式和相似度计算方式，但由于数据范围有限，最终预测的结果不是很理想，同时对参数的估计和；冷启动问题的处理没有采取很科学的措施，留待改进。