# LAB 2 文档

### 一、User Based Recommendation System

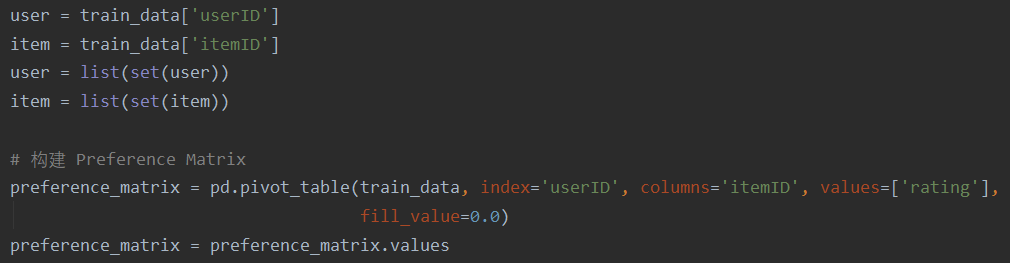
#### 1.1 代码结构

|  |  |
| --- | --- |
| **函数或属性** | **作用** |
| **工具函数utils** | |
| read\_file(path,train=true) | 读取处理csv文件  train为True时候，处理训练数据的csv文件，返回preference\_ matrix，user，item  train为False时候，处理测试数据的csv文件，将测试数据读取到矩阵 |
| save\_csv\_from\_rating(matrix,path) | 根据读取的测试数据集，将计算过rating的数据重新写入csv文件 |
| **类：userBasedRecommSys** | **基于用户相似度计算推荐系统实现** |
| **属性：** | |
| self.preference\_matrix = []  self.user = []  self.item = []  self.user\_n = 0  self.item\_m = 0  self.k\_nearest = 0  self.average\_rate\_array = []  self.user\_similarity\_matrix = []  self.similarity\_uv\_k = []  self.similarity\_index\_k = [] | 用户评分矩阵 n\*m  用户的数组  商品的数组  用户数 n  商品数 m  最近邻居数 k  用户的平均评分矩阵 n  用户的相似度矩阵 n\*n  User的K个最近邻居的依次相似度  User的K个最近邻居的依次index  (与uv\_k一一对应) |
| **函数：** | |
| set\_up(path,k) | 初始化用户评分矩阵等所有属性  path是训练数据的相对路径  k是所取的k个相似度最大的用户邻居 |
| get\_average\_rating | 计算用户的平均评分，返回值数组 |
| get\_user\_similarity\_matrix | 利用numpy. corrcoef计算用户的相似度矩阵(Pearson 相关性)，返回值n\*n的矩阵 |
| k\_neighbors(u) | 计算用户u最相似的k个用户，返回值相似度数组，index数组 |
| get\_k\_neighbors\_matrix | 计算所有用户的最相似的k个用户的矩阵返回值n\*k矩阵 |
| predict\_without\_k | 不过滤用户，使用与其他所有用户的相似度来预测用户的评分，返回值n\*m全满的矩阵 |
| predict\_with\_index | 对提供的测试集针对性地进行数据预测，返回带有预测评分的矩阵 |
| predict\_to\_csv(test\_path,save\_path) | 将预测结果写入csv文件 |

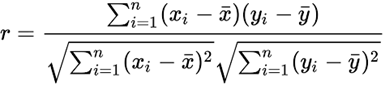
#### 1.2 算法过程

**第一步：读取csv文件，构建用户评分矩阵n\*m。**

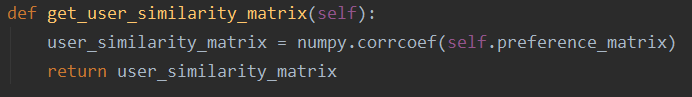
其中n为用户个数，m为商品个数。同时将用户的index，商品的index分别保存到一个数组里面。因为测试的时候发现，商品的index与商品的ID不是一一对应的，因为有些商品是没有评分的。



**第二步：**由用户的评分矩阵，**根据Pearson 相关性得到用户之间的相似度矩阵。**

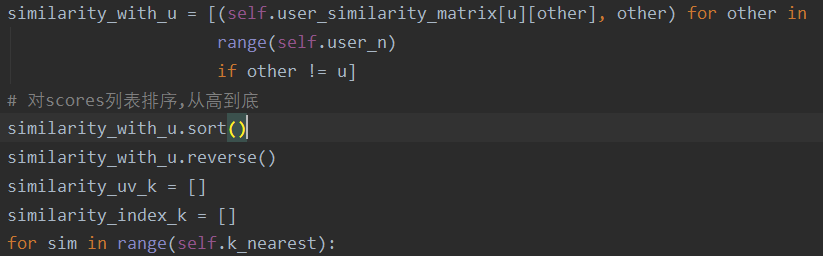
公式如下：

在本实验的实现中，我使用numpy.correcf 来计算相似度矩阵，得到的是一个n\*n的矩阵，其中n是用户的个数。user\_similarity\_matrix[u][v]表示用户u，v之间的相似度，其中user\_similarity\_matrix[u][u]为1.0。



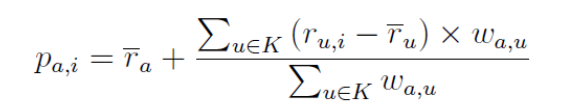
**第三步：**根据相似度矩阵，**找出前k最相近的邻居。**

对用户评分相似度矩阵的每一行进行排序，同时保存前k邻居用户的评分，和前k邻居用户的index。一一对应，因为在下一步预测的时候会用到。



**第四步：对用户评分进行预测。**

考虑user-item 如果两个用户相似，自然他们对某个物品评分应该接近，但如果总有些用户a喜欢给物品评很高的分，这时候即使两个用户不相似，也能占到很高的权重，这就相当于一种干扰，一种噪声，所以，对每个用户评分做一个平均，标准化。



#### 1.3 不同算法，参数的实验

**1.3.1在这里说一下在实验过程中遇到的一些问题，也是不断在改进的过程：**

**问题一：计算用户相似度循环时间太长**

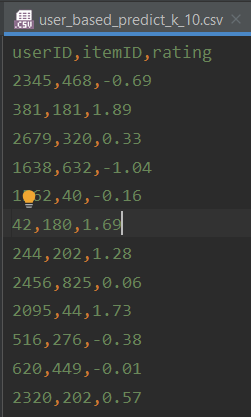
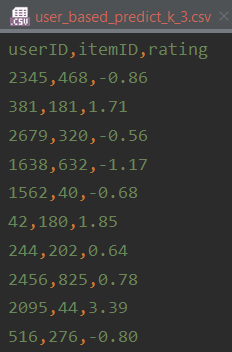
刚开始直接根据PPT上面的公式手写for循环，对矩阵进行计算，效率实在太低了，一直跑不出来。

**原因：**没有合理的利用numpy的multiple，dot等方法。

**解决办法：**使用numpy的correcf方法计算用户的相似度。

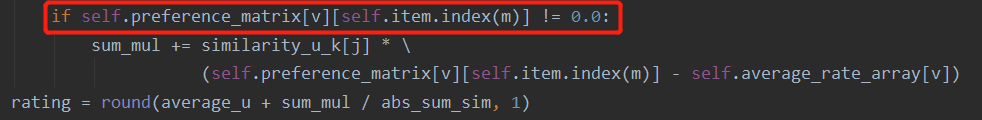
**问题二：评分分布在0附近**

预测评分基本在0 附近，并且负值结果不少，而且k值的修改对结果是不起作用的（依旧在0附近分布较多），而根据对原始数据的观察，用户评分还是比较多的在3，4左右，所以当前预测评分应该是不准确的，如下图：

**原因：**检查之后发现是在预测的时候，有个地方的代码写的有问题：在预测的时候没有加上前面的一句非零的判断，就是预测用户对该商品的评分的时候，**排除没有对该商品评分的用户，**否则的话在预测公式里面（Ru，I – Ru的平均值）得到的就是一个负数，又因为数据中有很多用户并没有对该商品进行评分，所以会导致最后的结果大多分布在0 左右。

**解决办法：**在预测评分的时候加上下面的一行代码



**问题三：item id找不到**

**原因:**

**1.3.2在解决以上问题的基础上，对k值进行调节观察预测结果的效果。**

① k = 3

② k = 5

因为在解决第一个问题的基础上，有可能用户得相似度比较高，但是因为此时的k比较小，并不一定与该用户相似度高的用户评价了该商品，用户评分矩阵是比较稀疏的。所以Preference\_matrix[v][m]的值为0，而在最后一步进行预测计算的时候，用该值减去平均评分得到的就是负值，所以对于预测结果并不友好。

**解决办法：**在第一种方法的基础上，不考虑k个最大邻居，即利用所有其他用户与该用户的相似度，对该用户进行预测。第二种想到的解决方法，增大k值，增大到50发现依然有效改善，所以就采用第一种方法。

### 二、Item Based Recommendation System

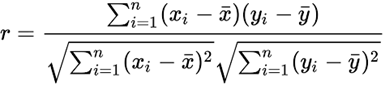
#### 1.1 代码结构

|  |  |
| --- | --- |
| **函数或属性** | **作用** |
| **类：userBasedRecommSys** | **基于用户相似度计算推荐系统实现** |
| **属性：** | |
| self.preference\_matrix\_T = []  self.user = []  self.item = []  self.user\_n = 0  self.item\_m = 0  self.item\_similarity\_matrix = [] | 用户评分矩阵的转置 m\*n  用户的数组  商品的数组  用户数 n  商品数 m  商品的相似度矩阵 m\*m |
| **函数：** | |
| set\_up(path) | 初始化用户评分矩阵等所有属性  path是训练数据的相对路径 |
| get\_item\_similarity\_matrix | 利用numpy. corrcoef计算商品的相似度矩阵(Pearson 相关性)，返回值m\*m的矩阵 |
| predict\_with\_index | 对提供的测试集针对性地进行数据预测，返回带有预测评分的矩阵 |
| predict\_to\_csv(test\_path,save\_path) | 将预测结果写入csv文件 |

#### 1.2 算法过程

**第一步：读取csv文件，构建用户评分矩阵的转置m\*n。**

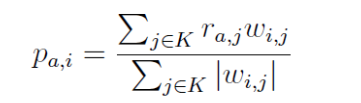
**第二步：**由用户的评分矩阵，**根据Pearson 相关性得到商品之间的相似度矩阵。**

公式如下：

在本实验的实现中，我使用numpy.correcf 来计算相似度矩阵，得到的是一个m\*m的矩阵，其中m是商品的个数。item\_similarity\_matrix[m1][m2]表示商品m1，m2之间的相似度，其中item\_similarity\_matrix[m1][m2]为1.0。

**第三步：对用户评分进行预测。**

如果物品m与某个物品b相似度较高，那么 与b有交互的用户k对b的评分会很接近于m，最简单的是m=b，那么评分相等，可以给出公式来预估k对m的评分，其中分母起到正则化的作用

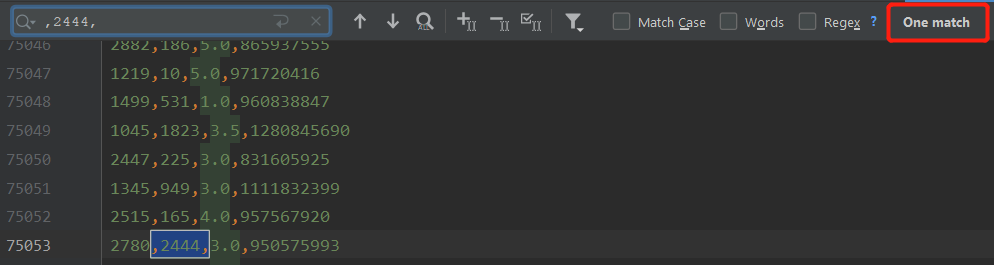


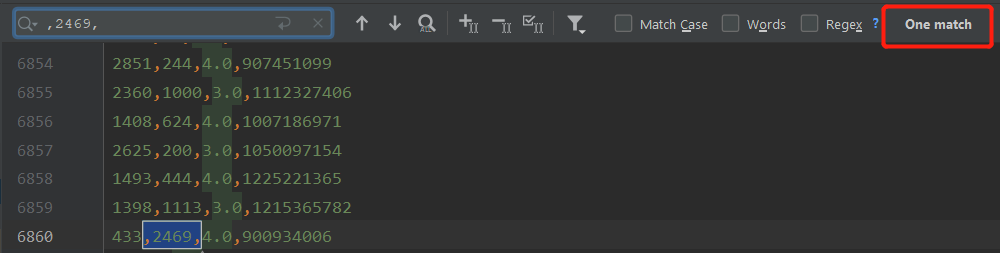
问题：



原因：





解决：



效果：

### 三、PMF base on ALS

### 四、自己对不同算法的理解