# LAB 2 文档

### 一、User Based Recommendation System

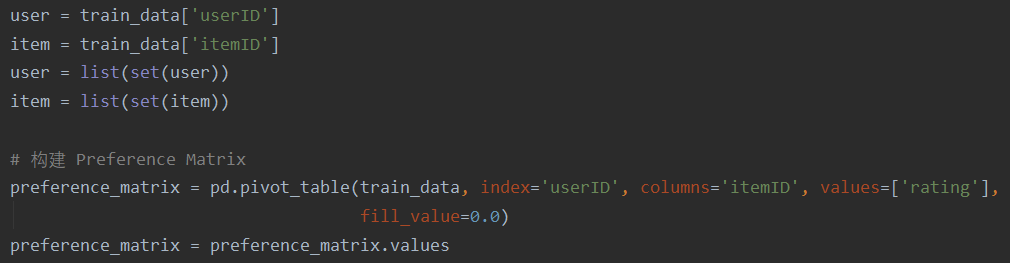
#### 1.1 代码结构

|  |  |
| --- | --- |
| **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是训练数据集文件的相对路径  k是最相似用户数 |
| **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  path是测试数据集文件的相对路径 |
| **predict\_without\_k(path)** | 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  path是测试数据集文件的相对路径 |
| **predict\_with\_k(path)** | 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，k=True)** | 将预测结果写入csv文件  k表示指定的最相似邻居用户数目 |

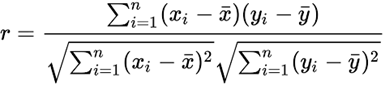
#### 1.2 算法过程

**第一步：读取csv文件，构建用户评分矩阵n\*m。**

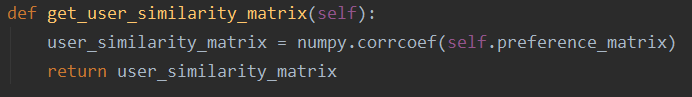
其中n为用户个数，m为商品个数。同时将用户的index，商品的index分别保存到一个数组里面。因为测试的时候发现，商品的index与商品的ID不是一一对应的，因为有些商品是没有评分的。



**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。**

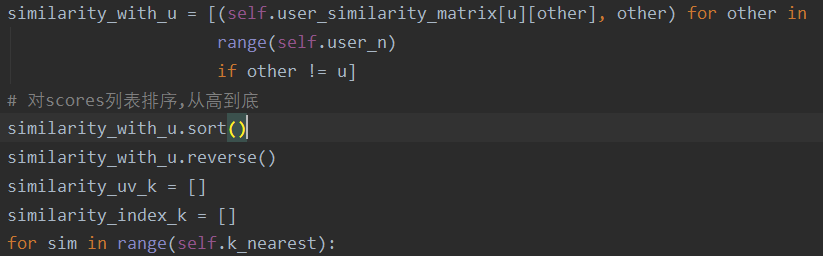
公式如下：

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。



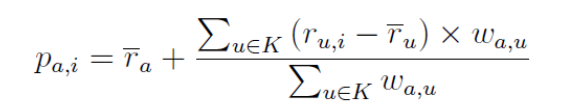
**第三步：**根据相似度矩阵，**找出前k最相近的邻居。**

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，参数的实验结果比较

**1.3.1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**

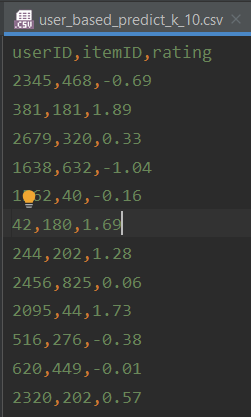
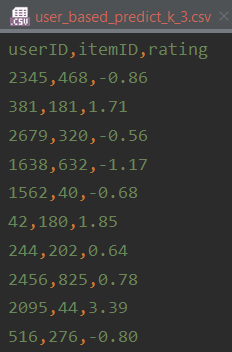
刚开始直接根据PPT上面的公式手写for循环，对矩阵进行计算，效率实在太低了，一直跑不出来。

**原因：**没有合理的利用numpy的multiple，dot等方法。

**解决：**使用numpy的correcf方法计算用户的相似度。

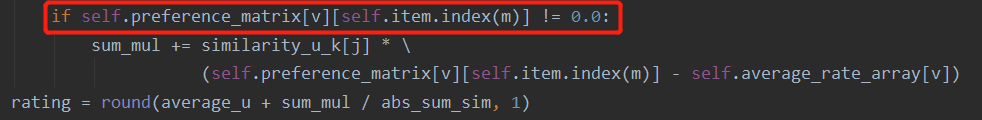
**问题二：评分分布在0附近**

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，所以推测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 附近。

**解决：**在预测评分的时候加上下面的一行代码

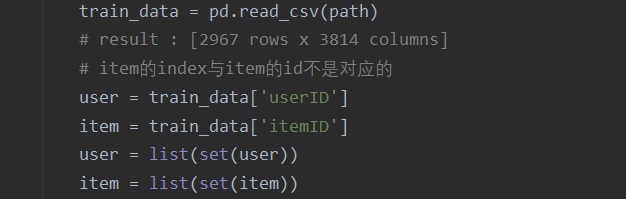


**问题三：item id找不到**

对测试集数据进行预测的时候报错，某个item id在preference\_matrix中找不到。

**原因：**虽然item的id最大是，但是查看item id的所有数据时发现item的index与id不是一一对应的，因为id是不连续的，有一些item没有用户对其评分。如下图：

**解决：**在构建preference\_matrix的时候，同时用一个item数组记录所有item的id。



**1.3.2不同k值的预测结果**

在解决以上问题的基础上，使用不同的k值进行预测。由于没有测试数据的结果集，所以只通过观察对预测结果做一些个人的定性分析，也是在调整实验参数过程中的一些发现。

1. **k = 3**

|  |  |  |
| --- | --- | --- |
| **部分评分** | **评分区间** | **大致数目** |
|  | 0.0 – 0.9 |  |
| 1.0 – 1.9 |  |
| 2.0 – 2.9 |  |
| 3.0 – 3.9 |  |
| 4.0 – 4.9 |  |
| 5.0 |  |

1. **k = 5**

|  |  |  |
| --- | --- | --- |
| **部分评分** | **评分区间** | **大致数目** |
|  | 0.0 – 0.9 |  |
| 1.0 – 1.9 |  |
| 2.0 – 2.9 |  |
| 3.0 – 3.9 |  |
| 4.0 – 4.9 |  |
| 5.0 |  |

1. **k = 10**

|  |  |  |
| --- | --- | --- |
| **部分评分** | **评分区间** | **大致数目** |
|  | 0.0 – 0.9 |  |
| 1.0 – 1.9 |  |
| 2.0 – 2.9 |  |
| 3.0 – 3.9 |  |
| 4.0 – 4.9 |  |
| 5.0 |  |

1. **k = 20**

|  |  |  |
| --- | --- | --- |
| **部分评分** | **评分区间** | **大致数目** |
|  | 0.0 – 0.9 |  |
| 1.0 – 1.9 |  |
| 2.0 – 2.9 |  |
| 3.0 – 3.9 |  |
| 4.0 – 4.9 |  |
| 5.0 |  |

1. **k = 50**

|  |  |  |
| --- | --- | --- |
| **部分评分** | **评分区间** | **大致数目** |
|  | 0.0 – 0.9 |  |
| 1.0 – 1.9 |  |
| 2.0 – 2.9 |  |
| 3.0 – 3.9 |  |
| 4.0 – 4.9 |  |
| 5.0 |  |

1. **k = 100**

|  |  |  |
| --- | --- | --- |
| **部分评分** | **评分区间** | **大致数目** |
|  | 0.0 – 0.9 |  |
| 1.0 – 1.9 |  |
| 2.0 – 2.9 |  |
| 3.0 – 3.9 |  |
| 4.0 – 4.9 |  |
| 5.0 |  |

1. **Without k**

|  |  |  |
| --- | --- | --- |
| **部分评分** | **评分区间** | **大致数目** |
|  | 0.0 – 0.9 |  |
| 1.0 – 1.9 |  |
| 2.0 – 2.9 |  |
| 3.0 – 3.9 |  |
| 4.0 – 4.9 |  |
| 5.0 |  |

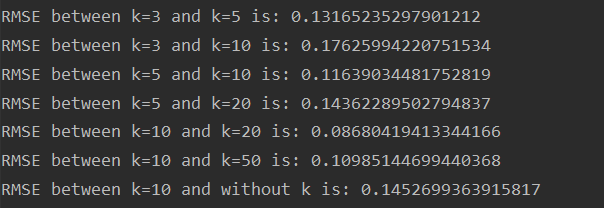
**总结：**

通过观察以上数据，发现，预测评分较多分布在3.0 – 3.9之间，猜测是与平均评分相关，并且k取值越大的时候，评分数据越往中间靠拢，两边的数据（评分区域1.0 – 1.9和5.0）不断在减少。K取值越大，越多与被预测用户u本身相关性不那么大的用户的评分数据被用来参与预测。然后计算得到的训练集数据中所有用户的平均评分如下：



所以k取值越大，更容易使预测的评分向平均评分靠近。

因为没有实际评分结果来对比，所以这里就计算一下不同k值得到的预测结果的RMSE**。**



K值越大之后，与较小k值时候的预测结果的RMSE差就越大。

最终选择k = 10作为预测结果。

### 二、Item Based Recommendation System

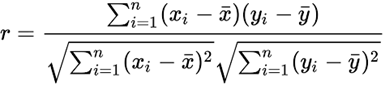
#### 1.1 代码结构

|  |  |
| --- | --- |
| **函数或属性** | **作用** |
| **类：itemBasedRecommSys** | **基于商品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.k\_nearest = 0**  **self.average\_rate\_array = []**  **self.item\_similarity\_matrix = []**  **self.similarity\_m12\_k = []**  **self.similarity\_index\_k = []** | 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  商品的最近邻居数  商品的平均评分数组  商品的相似度矩阵 m\*m  商品的K个最近邻居的依次相似度  商品的K个最近邻居的依次index  (与m12\_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是训练数据集文件的相对路径  k是最相似商品数 |
| **get\_average\_rating** | 计算商品的平均评分，返回值数组 |
| **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 |
| **k\_neighbors(m)** | 计算商品m最相似的k个商品，返回值相似度数组，index数组 |
| **get\_k\_neighbors\_matrix** | 计算所有闪频的最相似的k个用户的矩阵返回值m\*k矩阵 |
| **predict(path, k = False)** | 预测用户对商品的评分  path表示测试数据集文件的相对路径  k表示是否筛选商品的k个相似商品  默认为False |
| **predict\_u\_m(u, m，k=False)** | 预测用户u对商品m的评分 |
| **predict\_to\_csv(test\_path, k = False)** | 将预测结果写入csv文件  test\_path表示测试数据集文件的相对路径  k默认为False |

#### 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。**

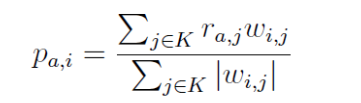
公式如下：

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。

\*(实验步骤)：根据相似度矩阵，**找出前k最相近的邻居。**这个步骤狮子啊实验过程中为了比较不同实验参数的预测结果来写的。

**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



#### 1.3 不同算法，参数的实验结果比较

**1.3.1实验过程中遇到的一些问题，也是不断在改进的过程**

在基于用户的预测的基础上，遇到的问题主要在预测结果上面

**问题一：预测结果出现负数和0.0值**

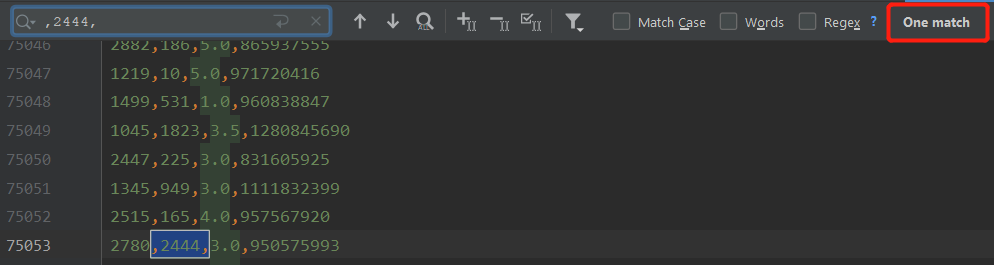
如下图所示：

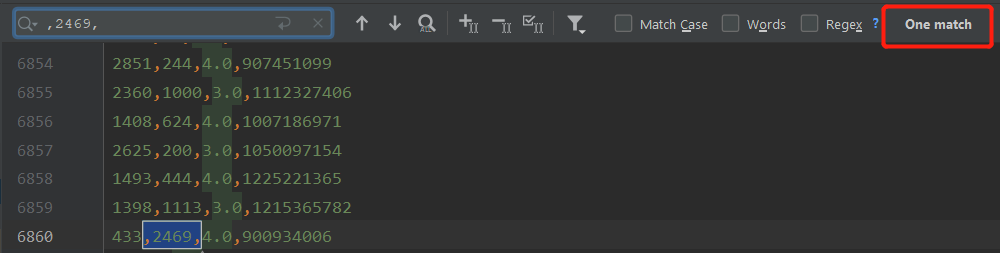


**原因：**

检查训练数据发现，这些出现负值商品基本上都有一个共同的问题，那就是训练集中用户对该商品的评分呢的数量非常少，有些只有1个，或者几个。这就导致在预测的时候，同时对该商品评分又对其他商品评分的用户就更少了，所以再加上如果两个商品的相似度是负相关的，就会出现上面这样负值的结果，或者0.0的结果。





**解决：**

**对于预测评分小于或者等于0的结果，赋值为所有用户对该商品的平均评分：**



**效果如下：**

**1.3.2不同k值的预测结果**

因为基于商品的预测是不筛选与商品的相似度最高的k个商品的，所以也想试一下，如果像基于用户的预测那样对商品筛选出k个最相似商品，预测结果是怎样的。

1. k = 10



1. k = 500



通过以上两组测试结果发现，对商品做筛选之后，出现0.0的结果更加多了，并且k值越小，出现0.0的结果越多，增大k值0.0的结果有效减少。分析原因，因为对商品的相似度计算使用的方法与计算用户相似度的方法是一样的，而用户评分矩阵 [2967 rows x 3814 columns]，商品的数量多于用户的数量，相比较用户而言，从商品的角度纵向来看，用户对某个商品的评分是更加稀疏的。所以还是不需要k值来辅助预测。

### 三、PMF base on ALS

### 四、自己对不同算法的理解