# LAB 2 文档 闫世艺 16302010076

### 一、User Based Recommendation System

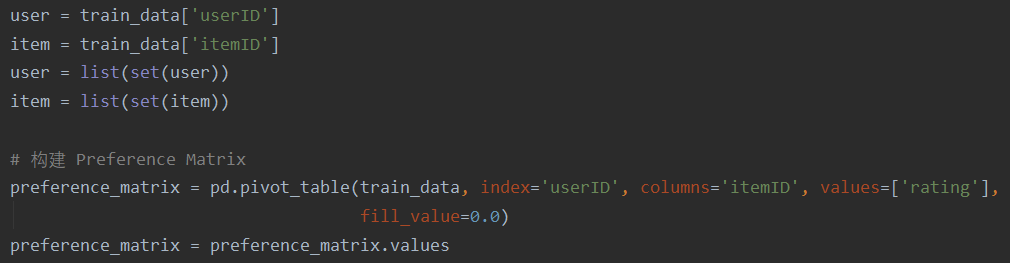
#### 1.1 代码结构

|  |  |
| --- | --- |
| **函数或属性** | **作用** |
| **工具函数**util | |
| 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 |
| 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 |
| generate\_random(row, col) | 生成0-1的随机数的矩阵 row\*col |
| get\_rmse(predict,target) | 计算两个数组的rmse |
| get\_rmse\_with\_csv(predict\_path,target\_path) | 先处理文件文件，再计算rmse |
| draw(iter\_array,rmse\_array) | 画图，iter\_times - rmse |
| draw\_train\_test(iter\_array,train\_rmse,test\_rmse) | 画图，iter\_times – train\_rmse / test\_rmse |
| Train\_test\_split(path) | 读取csv文件分割数据集为训练集和测试集 |
| **类：**UserBased | **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 n  商品的数组 m  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 |

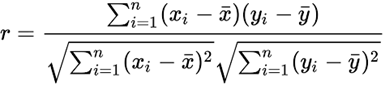
#### 1.2 算法过程

**第一步：读取csv文件，构建用户评分矩阵n\*m。**

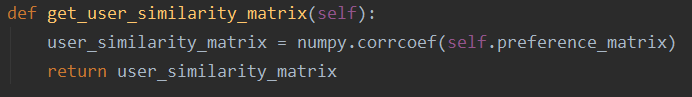
其中n为用户个数，m为商品个数。同时将用户的index，商品的index分别保存到一个数组里面。因为测试的时候发现，商品的index与商品的ID不是一一对应的，因为有些商品是没有评分的。



**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。**

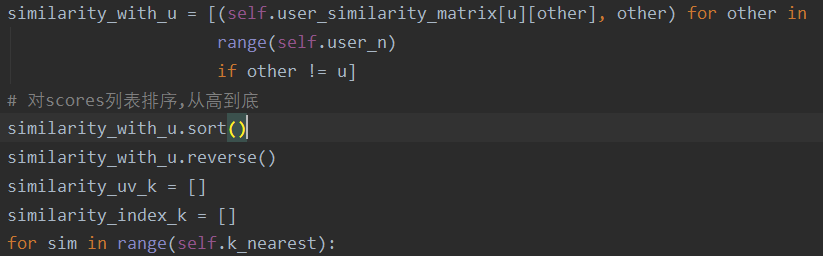
公式如下：

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。



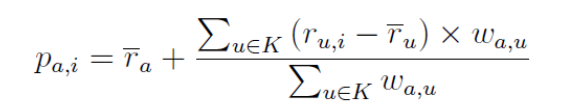
**第三步：**根据相似度矩阵，**找出前k最相近的邻居。**

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u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，所以对每个用户评分做一个平均，标准化。



#### 1.3 不同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**

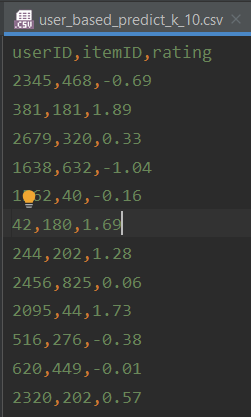
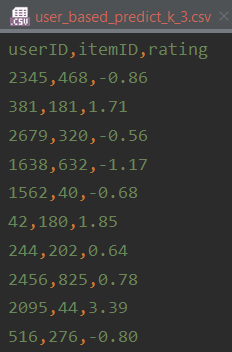
刚开始直接根据PPT上面的公式手写for循环，对矩阵进行计算，效率实在太低了，一直跑不出来。

**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。

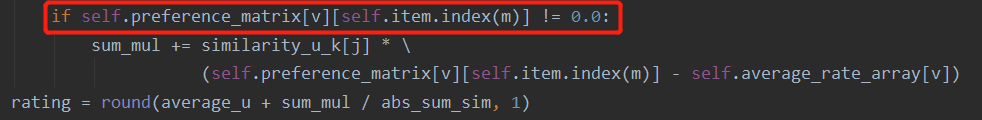
**问题二：评分分布在0附近**

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

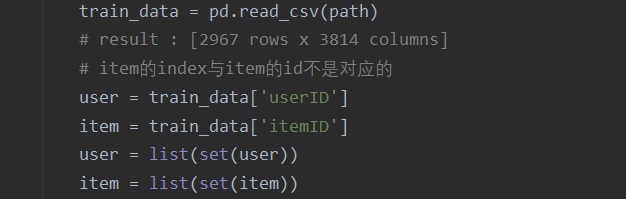


**问题三：item id找不到**

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对test\_index.csv, 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：**

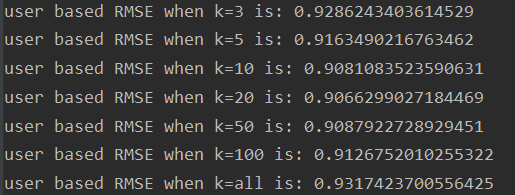
通过观察以上数据，发现，预测评分较多分布在3.0 – 3.9之间，猜测是与平均评分相关，并且k取值越大的时候，评分数据越往中间靠拢，两边的数据（评分区域1.0 – 1.9和5.0）不断在减少。K取值越大，越多与被预测用户u本身相关性不那么大的用户的评分数据被用来参与预测。然后计算得到的训练集数据中所有用户的平均评分如下：



所以k取值越大，更容易使预测的评分向平均评分靠近。

**计算不同k值预测结果与原来数据集的RMSE误差**

对train.csv文件即作为训练集，也作为测试集，对其中的数据并进行预测，输出文件<\\data\\self\_test\\user\_based\_……>，计算RMSE, **结果如下图：**



K取值为20的时候，RMSE最小，所以选择k = 20进行预测，输出out\_1.csv作为结果。

Kaggle上的Score：



### 二、Item Based Recommendation System

#### 2.1 代码结构

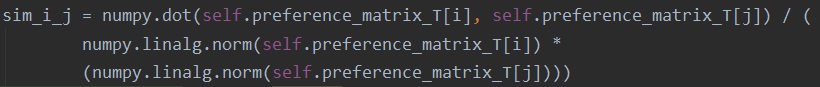
|  |  |
| --- | --- |
| **函数或属性** | **作用** |
| **类：ItemBased** | **基于商品相似度计算推荐系统实现** |
| **属性：** | |
| self.preference\_matrix­\_T = []  self.user = []  self.item = []  self.user\_n = 0  self.item\_m = 0  self.average\_rate\_array = []  self.item\_similarity\_matrix = [] | 用户评分矩阵的转置 m\*n  用户的数组 n  商品的数组m  用户数 n  商品数 m  商品的平均评分数组m  商品的相似度矩阵 m\*m |
| 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 |
| 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 |

#### 2.2 算法过程

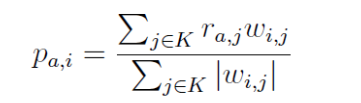
**第一步：读取csv文件，构建用户评分矩阵的转置m\*n。**

**第二步：由用户的评分矩阵，根据余弦相关性得到商品之间的相似度矩阵。**

公式如下：

**第三步：对用户评分进行预测。**

如果物品m1与某个物品m2相似度较高，那么 与m2有交互的用户U对m1的评分会很接近于m­2，如果m1=m2，那么评分相等，预测u对m的评分，其中分母起到正则化的作用。



#### 2.3 不同参数的实验结果比较

**2.3.1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**

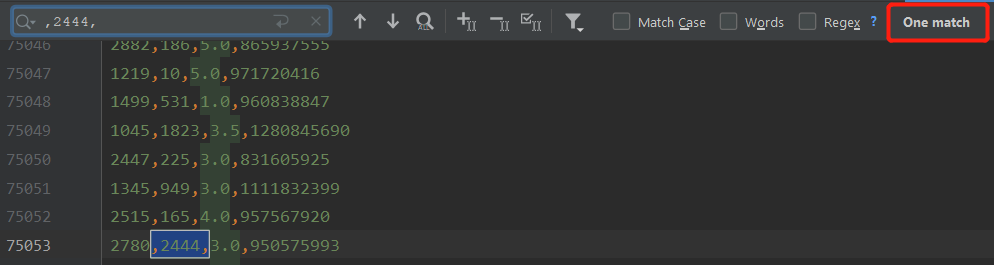
如下图所示：

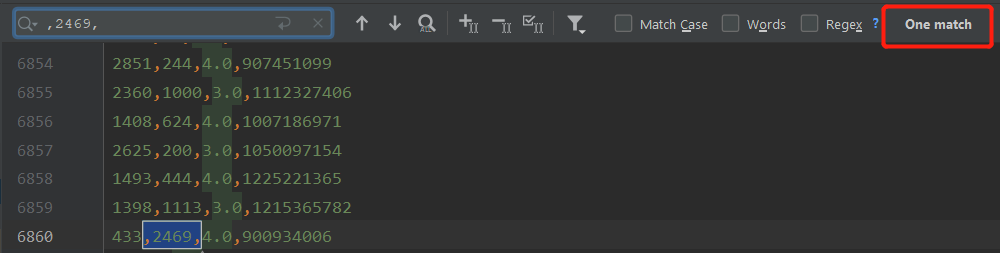


**原因：**

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：**

**2.3.2 预测结果**

对train.csv文件即作为训练集，也作为测试集，对其中的数据并进行预测，输出文件<\\data\\self\_test\\item\_based\_predict\_cos.csv>，计算RMSE, **结果如下图：**



使用以上基于商品的预测方法对测试数据进行预测，输出out\_2.csv作为结果。

item based的测试结果比user based还要好一点。

Kaggle上的Score:



### 三、PMF Base on ALS

#### 3.1 代码结构

|  |  |
| --- | --- |
| **函数或属性** | **作用** |
| **类：PMF\_ALS** | **基于ALS算法的PMF推荐系统** |
| **属性：** | |
| self.preference\_matrix = []  self.user\_k\_n = []  self.item\_k\_m = []  self.user = []  self.item = []  self.user\_n = 0  self.item\_m = 0  self.k\_factors = 0 | 用户评分矩阵的转置 m\*n  用户带隐藏因子的数组矩阵n\*k  商品带隐藏因子的矩阵m\*k  用户的数组 n  商品的数组 m  用户数 n  商品数m  隐藏因子k |
| 函数： | |
| set\_up(path, k\_factors) | 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\_factors是隐藏因子数 |
| als\_function(latent\_matrix, fixed\_matrix) | ALS核心算法，固定一个隐藏因子矩阵，优化另一个隐藏因子矩阵 |
| train(iterators，reg) | 训练，iterators训练次数，reg正则化因子 |
| predict() | 预测所有用户对所有商品评分，n\*m |
| predict\_with\_index(path,iter\_times) | 读取测试文件，预测相应u，m的评分 |
| calculate\_rmse(iter\_array) | 计算不同迭代次数的预测结果的rmse |

#### 3.2算法原理

**3.2.1 普通最小二乘法（OLS）**

损失函数：L1（U，I） = Σ（R-UTI）2

**3.2.2** 在普通最小二乘法（OLS）的基础上**加入正则化，避免过拟合问题**

损失函数：**L2（U，I） = Σi，j（Ri,j -UTI）2 + λ（|U|2+|I|2）**

L2（U，I）对Un求导，令导数为0，求解得到：

**∑(i，m)(IiT Ii + λE) Un = ∑(i, m)Ri，j Ii**

即**Un = （ITI + λE）-1 Ri I …… ①**

同理L2（U，I）对Im求导，求解得到：

**∑(j，n)(UjT Uj + λE)Im = ∑(j, m)Ri, j Uj**

即**Im = （UTU + λE）-1 RjT U …… ②**

公式**①** **②**就是ALS算法训练的重点，迭代计算矩阵U，I，迭代一定的次数或者直到损失函数收敛为止。

Ru(1×n)是R的第u行,Ri(1×m)是R的第i列， I是k×k的单位矩阵。

代码中实现：ITI + λE 或者 UTU + λE



**Un：**



**Im：**

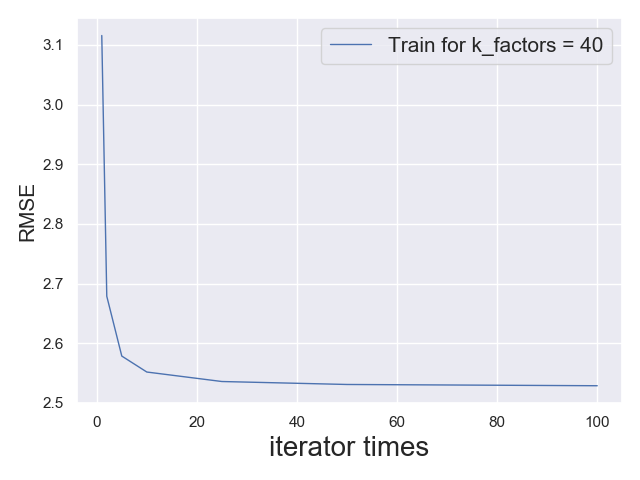
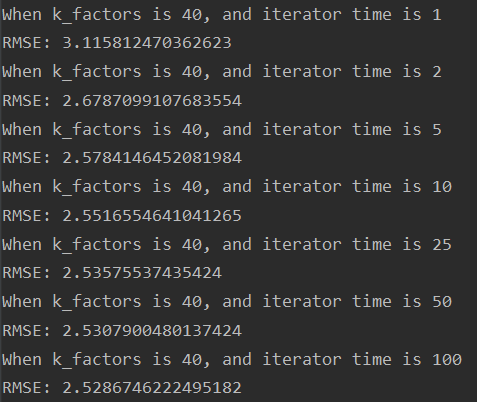


* + 1. **迭代过程**

首先随机初始化U，I，先固定I更新U,  然后固定U更新I,  直到均方根误差RMSE很小或者到达一定的迭代次数。

#### 3.3不同参数的实验结果比较

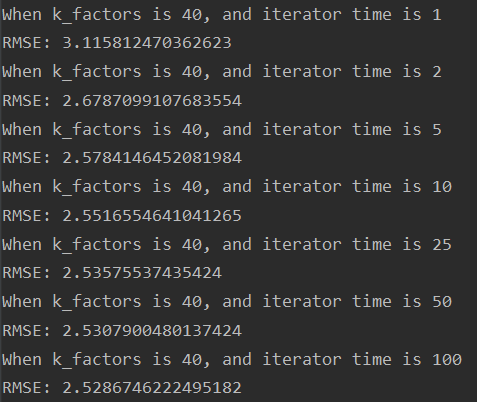
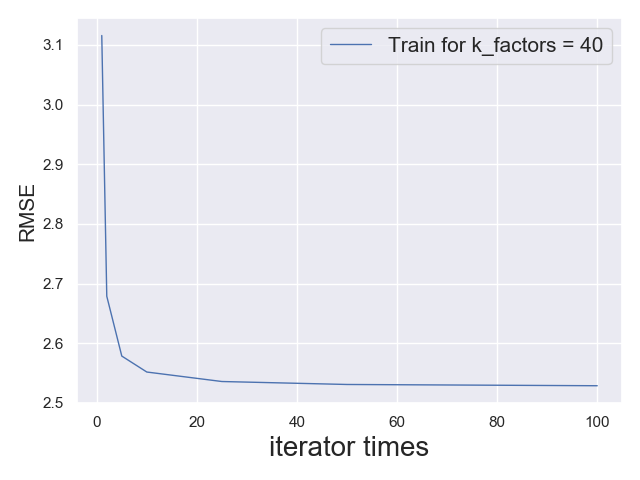
**3.3.1 固定**K，reg = 1**，迭代次数-**RMSE**图:**

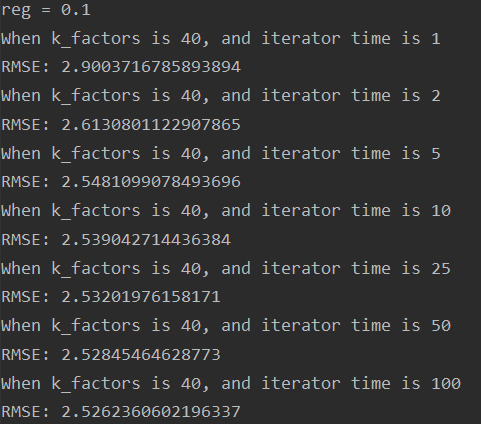
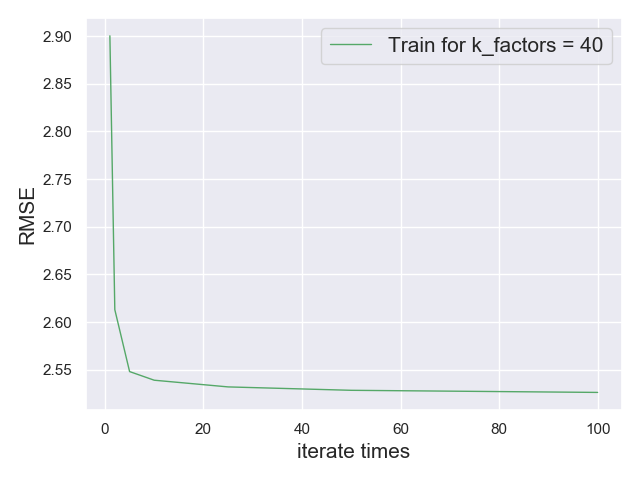
观察上图发现，在训练的迭代次数达到50以上的时候，RMSE已经收敛。

**3.3.2 固定**K = 40**，不同**reg

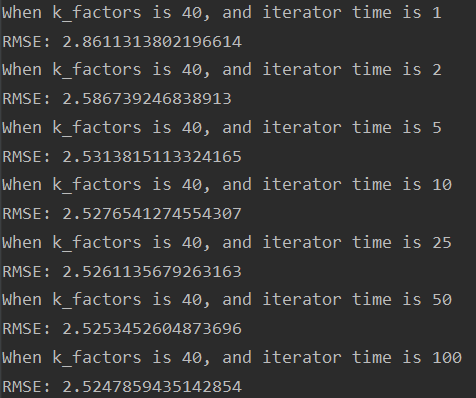
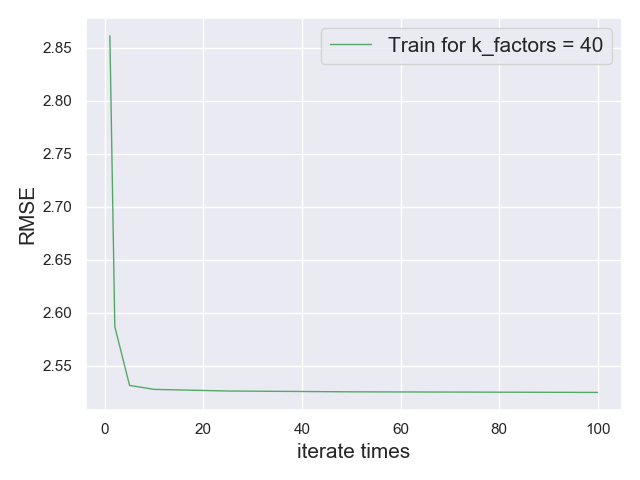
① reg = 1



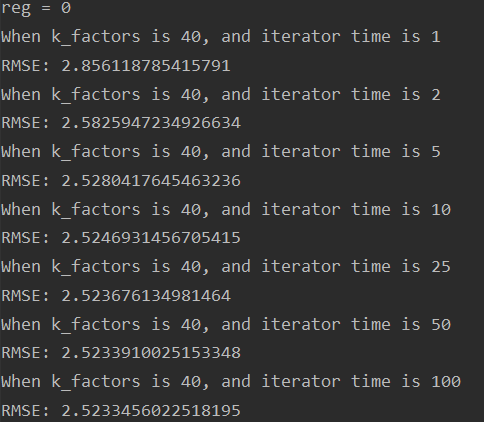
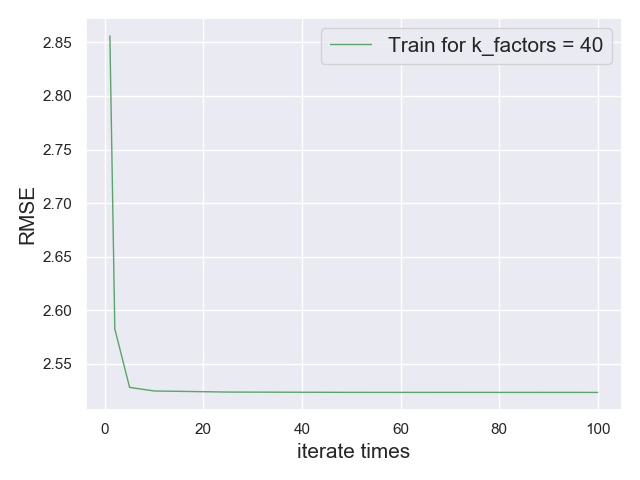
② reg = 0.1



③ reg = 0.01



④ reg = 0

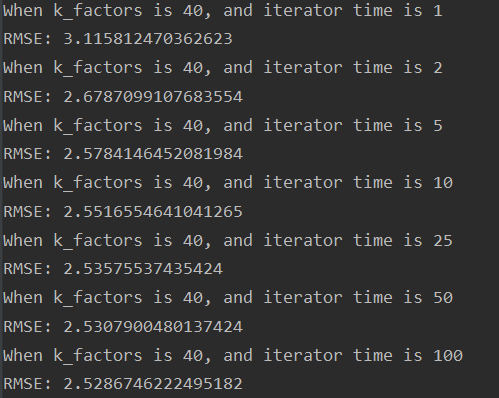
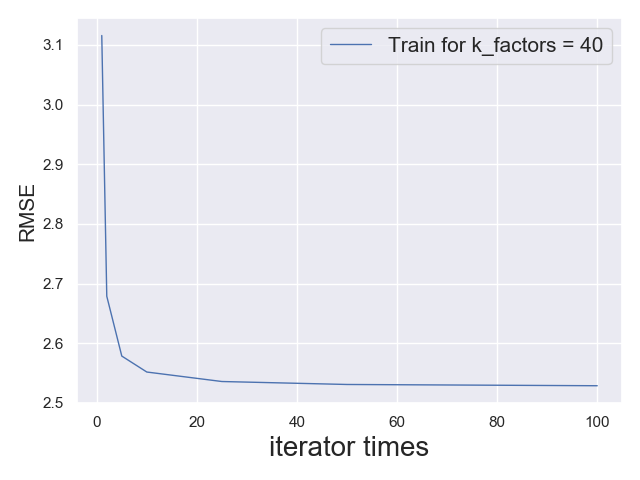


reg减小，RMSE更快的收敛，比较明显的是起始值的减小，收敛的最终效果有略微减小，不明显。

**3.3.3** **不同**K**值的收敛效果**

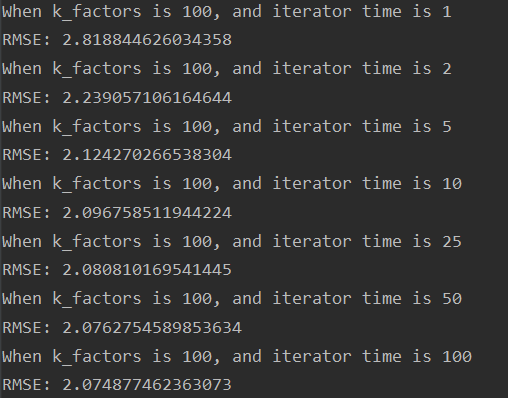
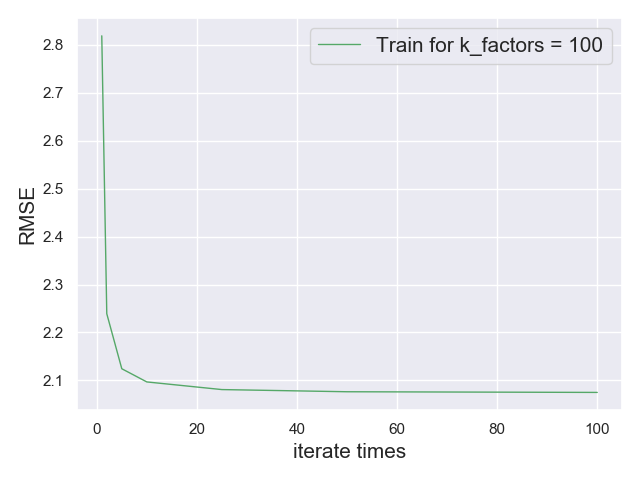
分别选择k值为40，100，200，500，进行训练。

1. k = 40



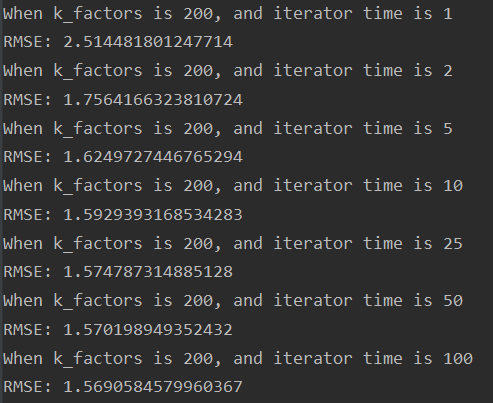
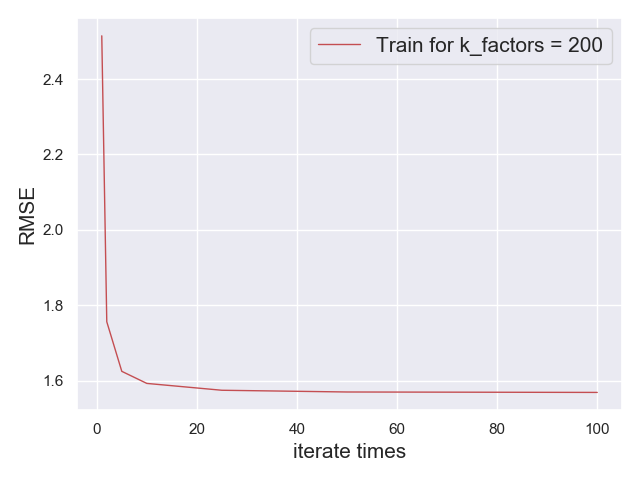
如图所示：训练次数达到100，RMSE收敛到2.5左右。

1. K= 100



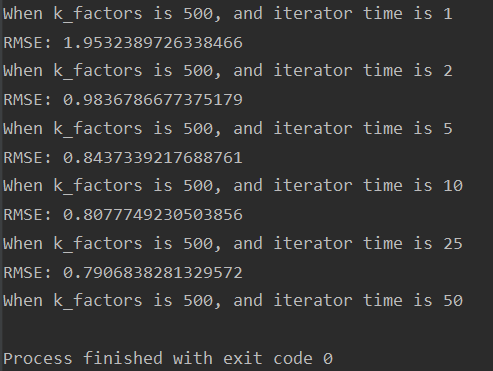
如图所示：训练次数达到100，RMSE收敛到2.0左右。

1. K = 200



如图所示：训练次数达到100，RMSE收敛到1.5左右。

1. K = 500

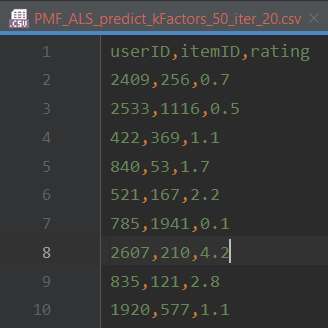
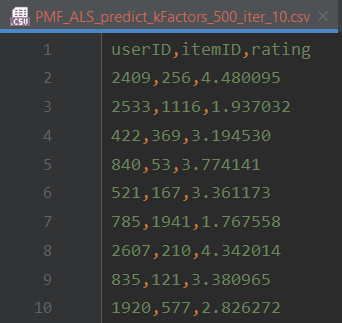


如图所示：训练次数达到25的时候，RMSE收敛到0.8一下。（由于时间原因，k = 500时，训练时间特别长，并且观察也可以发现此时RMSE已经逐渐收敛，所以不再增加迭代次数iter\_times。）

另外下面是对train.csv训练的时候，不同参数的输出，也可以看出来数据在逐渐向train.csv靠近。

左图：kFactors = 50，迭代20次

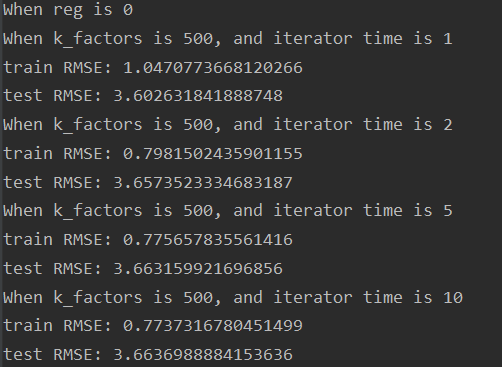
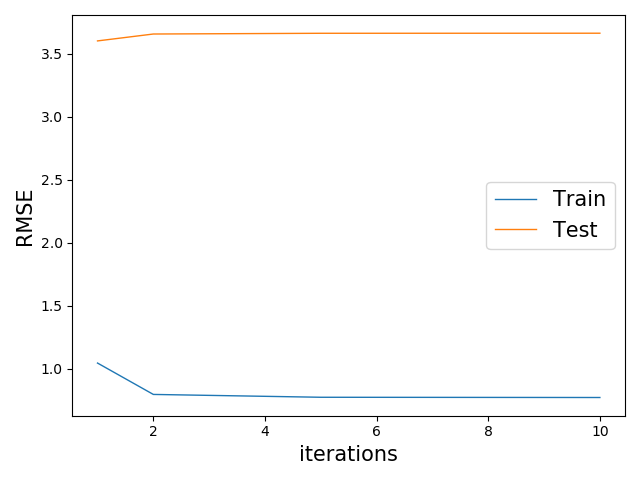
右图：kFactors = 500，迭代10次

以上都是在全集的训练集train.csv上做的测试，然后开始对测试对局进行预测的时候突然发现问题了。**测试集的RMSE不下降，或者RMSE还在较高水平的时候就收敛了。**

**如下图所示：**

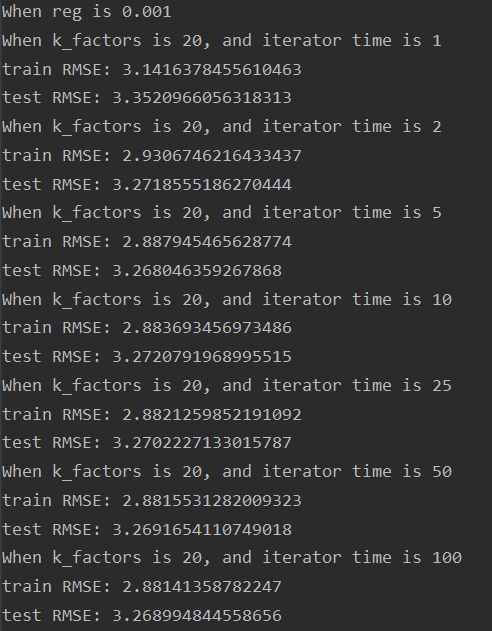
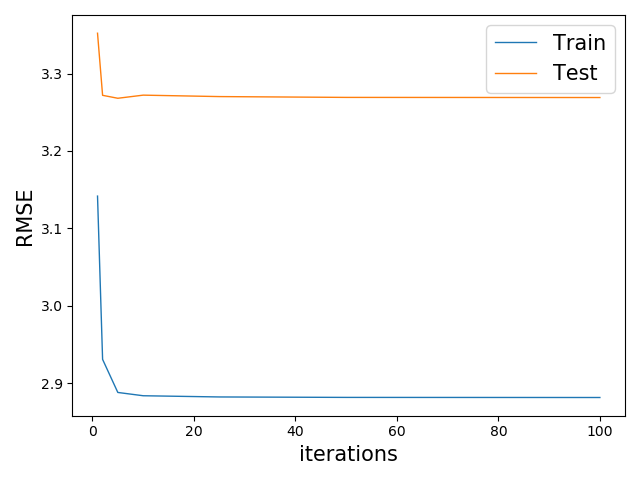
**参数：reg = 0 K = 500**

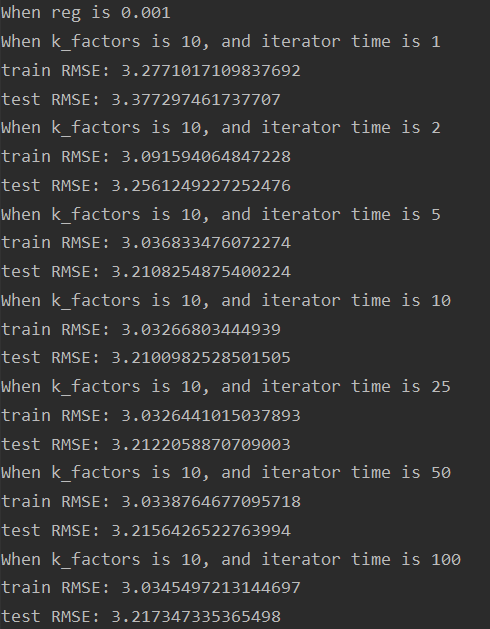
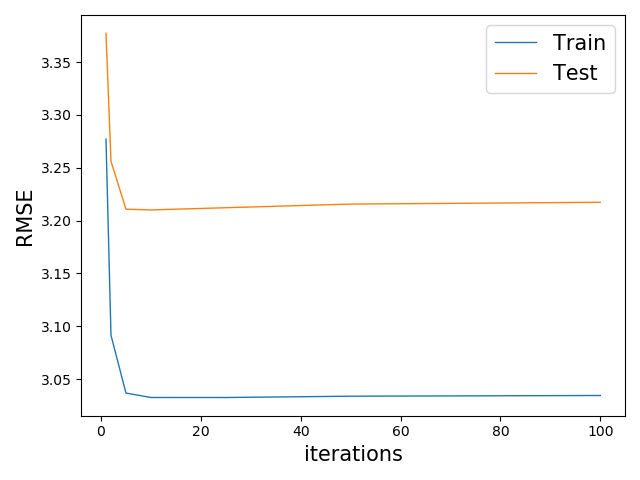


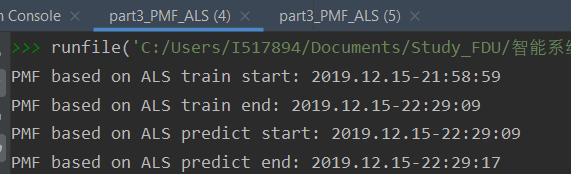
然后经搜索发现问题可能是由于**过拟合**：

修正方法有：增大数据规模、减小数据特征数（维数）、增大正则化系数λ

减小k，增大λ





**总结：**最开始没有分割测试集与训练集，以为对训练集做好训练，测试集的RMSE也就会减小到差不多的程度，但是使用测试数据集进行预测的时候发现预测结果并没有很好，可能与自己分割的测试集有关。当K取很大的时候，训练集能有效的收敛到较低水平，但是测试集的RMSE却随着迭代次数增加而变大，应该是K值太大了，影响预测效果。**最后又不断调整参数**，根据kaggle平台上的评分，所以最后选取K = 800，reg= 0.01,训练迭代次数10次的参数对测试集进行测试，输出out\_3.csv作为结果。运行时间较长：

Kaggle上的Score：



### 四、Optimized Recommendation System

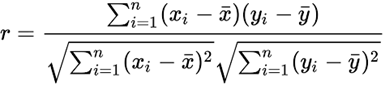
#### 4.1 代码结构

|  |  |
| --- | --- |
| **函数或属性** | **作用** |
| **类：ItemBasedOpt** | **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 n  商品的数组m  用户数 n  商品数 m  商品的最近邻居数 k  商品的平均评分数组m  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 |

#### 4.2 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。**

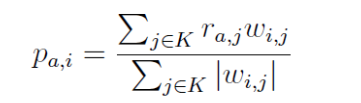
公式如下：

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



#### 4.3 不同参数的实验结果比较

因为在user based 推荐系统中，会考虑用户的前k相似用户，所以想探究一下是否对item based 推荐系统前k相似商品也会有影响。实验如下：

**4.3.1对test\_index.csv, 不同k值预测结果**

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



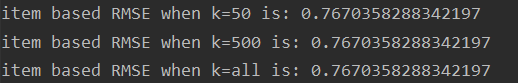
1. k = 500



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。

**定量分析: 计算不同k值预测结果与原来数据集的RMSE**

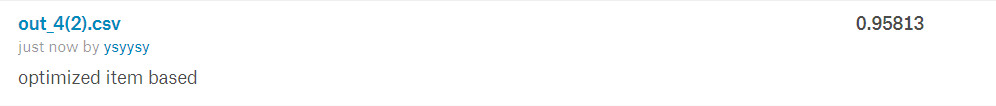
对train.csv文件即作为训练集，也作为测试集，对其中的数据并进行预测，输出文件<\\data\\self\_test\\item\_based\_……>，计算RMSE, **结果如下图：**



**总结:**

对train.csv作为测试数据，换了相似度计算的函数之后，训练集的RMSE有所减小。综合考虑前面对其他数据集的测试结果，还是选择不考虑k值的预测，输出out\_4.csv作为结果。

Kaggle上的Score：



### 五、总结

本次实验中，基于用户和基于商品的推荐系统，主要是找到一个计算用户或者商品相似度的方法，然后根据相似度对未知的评分进行预测。主要是矩阵的运算，可能存在的问题就是时间可能有些长。使用最小二乘法进行矩阵分解的算法对用户评分进行分析的方法没有达到很好的效果，可能存在的原因是正则化因子reg，与k值的选取等参数的影响。首先矩阵分解是降维，最小二乘法的基本思想就是固定其中一个变量，更新另一个变量，迭代，使得最后的RMSE达到收敛。