推荐系统实验报告

一、任务概览

（一）数据集信息

- train.txt：包含用户、物品、评分,共 5021342 行，其中共 19835 名用户，455309 个商品和 5001506 个用户-商品-评分三元组；
- text.txt：包含用户和物品，需要自己给出评分；
- itemAttribut.txt：中给出了物品和属性值（共两个属性）。

（二）目标

预测一个用户对某个商品的评分。

二、实验原理

（一）协同过滤

1.基于用户的协同过滤 UserCF

（1）算法思想

找到和目标用户兴趣相似的用户集合。找到这个集合中的用户喜欢的物品认为具有更高的分数，推荐给目标用户。

（2）实现原理

a．计算两个用户的相似度

当知道两个用户对一系列产品的评分向量 X,Y，计算两者相似度可以用皮尔逊相关系数，公式如下：

$$sim(X,Y) = \frac{Cov(X,Y)}{\sqrt{D(X)D(Y)}}$$

其中$Cov(X,Y)$是 X 与 Y 的协方差，$D(X)$和$D(Y)$为 X 和 Y 的方差。当$sim(X,Y)$的绝对值越接近于 1，表示两用户越相近。

b．为相似的用户预测商品

得到用户的相似度矩阵后，就可以按照相似度值，计算加权平均，作为被预测用户对商品的预测分数：

$$\widehat{R_{x_i}} = \frac{\sum_{y \in N} s_{xy} R_{y_i}}{\sum_{y \in N} s_{xy}}$$

其中$s_{xy} = sim(X,Y), R_{x_i}$为 x 用户对物品 i 的评分。

2.基于物品的协同过滤 ItemCF

基于物品的协同过滤与基于用户的方法相似，区别是寻找相似的物品，预测评分，将基于用户的方法中用户和物品的互换即可得到。

不过相对基于用户的协同过滤，因为物品直接的相似性相对比较固定，所以可以预先在线下计算好不同物品之间的相似度，把结果存在表中，当推荐时进行查表，计算用户可能的打分值。

3.分数偏置值

考虑到用户打分可能存在严苛与松的分别，为最终预测分数加上一个 bais：

$$b_{xi} = \mu + b_x + b_i$$

其中μ为总体平均值，b_x为用户 x 打分平均值与μ的差距，b_i为物品 i 分数平均值与μ的差距，得到预测值为：

$$\widehat{R_{x_i}} = b_{xi} + \frac{\sum_{y \in N} s_{xy}(R_{y_i} - b_{xi})}{\sum_{y \in N} s_{xy}}$$

（二）SVD 奇异值分解

SVD 可以最小化误差平方和（SSE）,又因为 RSME 与 SSE 单调正相关，所以可以通过 SVD 来最小化 RSME.

1.理论基础：

SVD 可以很容易得到任意矩阵的满秩分解，用满秩分解可以对数据做压缩，对任意 M*N 的矩阵均存在如下分解：

$$A_{m*n} \rightarrow X_{m*k} Y_{k*n}$$

实际上：

$$A_{m*n} = U \Sigma V^T$$

2.实现原理：

因为用户*物品矩阵有很多缺失值（分数未知），所以需要学习分解后的 q、p 矩阵，再利用这些学习后的 q、p 矩阵来相乘得到误差最小的缺失值的合理值。

3.避免过拟合

过拟合的表现是在训练集上性能良好而在测试集上则不好，即鲁棒性、泛化性不强。解决方法是利用正则化因子矫正高维度的拟合。即在目标函数中增加一项，专门用来惩罚过高维度的拟合。

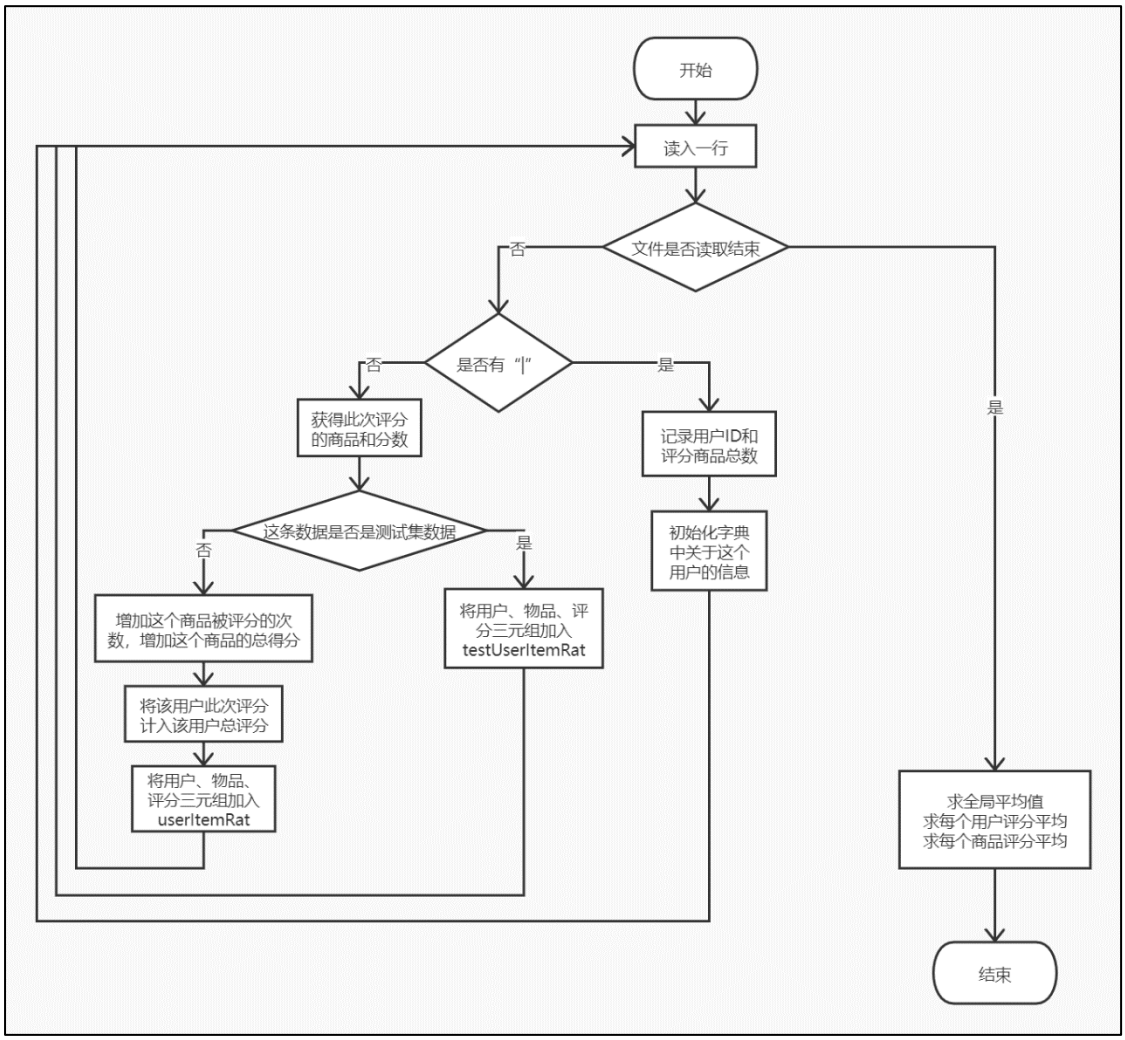
三、关键部分代码解析

（一）预处理

对于 train.txt，划分训练集和验证集，记录相应的用户-商品-分数三元组，并同时计算总评分和总评分次数，之后用于计算相较于平均分的偏置

变量名称	说明
userItemRat	记录训练集用户-商品-分数三元组
testUserItemRat	记录验证集用户-商品-分数三元组
userBias	记录用户 ID 对应的较平均分的偏置
itemBias	记录商品 ID 对应的较平均分的偏置
userTotalRatings	记录用户 ID 对应的总评分

userTotalItems	记录用户 ID 对应的总商品数
itemTotalRatings	记录商品 ID 对应的总评分
itemTotalUser	记录商品 ID 对应的总评分用户数
AttributeOfItem	商品的属性
UserToAttri	用户对属性的[总评分,评分次数]
AttriToUser	属性的[总评分,被评分次数]
UserBiasA	用户对属性评分的偏置项
AttriBias	该属性相对于平均分的偏置项



预处理部分代码流程图

（二）SVD 奇异值分解

1. 基本算法

核心函数为 train()：

```
def train(self, k1=210, lambda1=0.01, stuRate=0.001, steps=12, choose_step=False)
```

参数解释：k 为隐藏因子个数，lambda 为正则化因子，stuRate 为学习率，steps 为迭代次数，choose_step 用于网格调参时选择迭代次数。

a. 初始化矩阵

```
# q 是用户矩阵
self.qMatrix = 0.1 * np.random.randn(self.numOfUsers, k1) / np.sqrt(k1)
# p 是商品矩阵
self.pMatrix = 0.1 * np.random.randn(k1, self.numOfItems) / np.sqrt(k1)
```

b. 随机梯度下降

```
predict_rating = np.dot(self.qMatrix[user, :], self.pMatrix[:, self.items[item]])+self.overallMean+self.userBias[user]+self.itemBias[item]
thisLoss = rating-predict_rating

q_change = Rate*(thisLoss*self.pMatrix[:, self.items[item]]-lambda1*self.qMatrix[user, :])
p_change = Rate*(thisLoss*self.qMatrix[user, :]-lambda1*self.pMatrix[:, self.items[item]])
self.qMatrix[user, :] += q_change
self.pMatrix[:, self.items[item]] += p_change
```

2. 增加偏置项（bias）

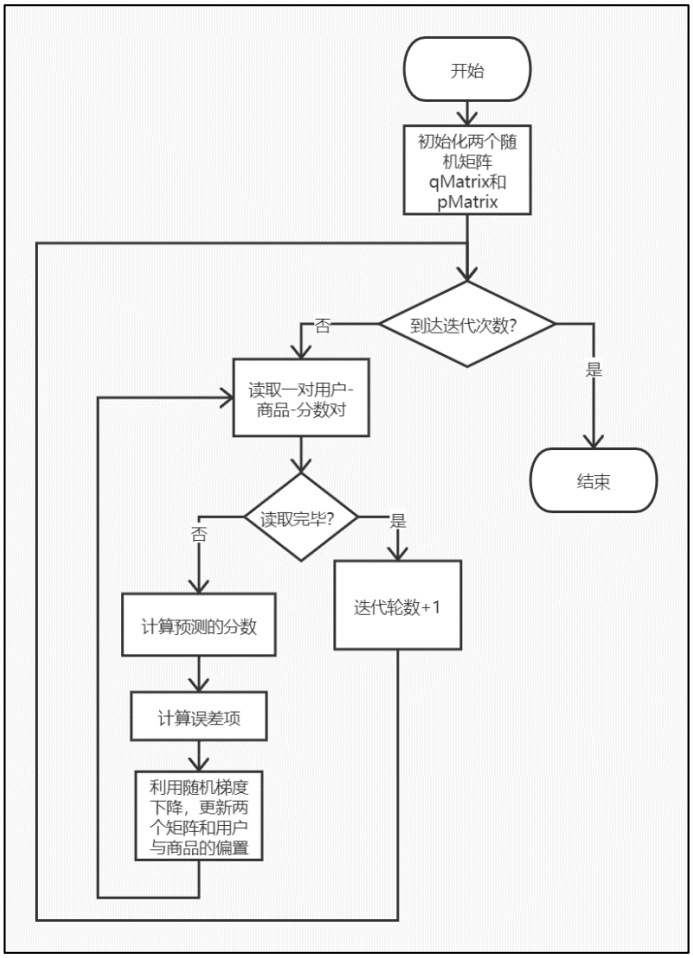
作图后发现用户-物品-评分中有许多横线，即有些用户对于大部分商品的评分都较一致，因此可以说明用户的评分习惯（严格或宽松）是较好的衡量指标，因此考虑偏置 bias

a. 计算偏置项

```
# 计算商品获得的总评分与被评分次数之商
for item in itemTotalRatings:
    if itemTotalUser[item] != 0:
        self.itemBias[item] = itemTotalRatings[item] / itemTotalUser[item]
    else:
        self.itemBias[item] = 0.0
# 计算用户给出的总评分与评分次数之商
for user in userTotalRatings.keys():
    if userTotalItems[user] != 0:
        self.userBias[user] = userTotalRatings[user] / userTotalItems[user]
    else:
        self.userBias[user] = 0.0
# 根据平均分对 bias 进行修正
self.overallMean = np.mean(self.userItemRat[:, 2])
for i in self.userBias.keys():
    self.userBias[i] -= self.overallMean
for j in self.itemBias.keys():
    self.itemBias[j] -= self.overallMean
```

b.带偏置项进行梯度下降

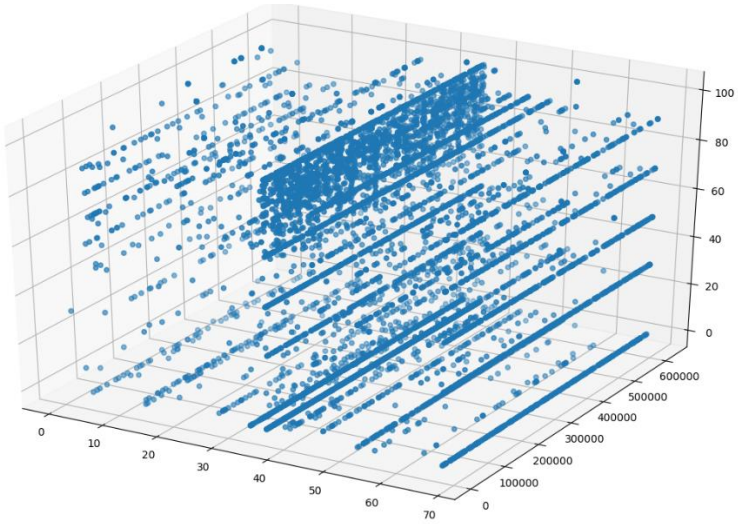
```
self.userBias[user] += Rate * (thisLoss - lambda1 * self.userBias[user])
self.itemBias[item] += Rate * (thisLoss - lambda1 * self.itemBias[item])
```



增加偏置后的 train()函数流程图

3. 利用属性信息进行改进

通过对 itemAttribute.txt 的数据整合，得到用户-属性 1（或属性 2）值-评分的矩阵。



经随机取样绘图发现，除了比较明显的用户打分习惯外，用户-属性值间也存在较大的相关关系，但这种相关关系有不是线性或者多项式的。因此，基于传统 SVD 推荐算法的启发，决定以属性值（离散值）为矩阵一维度，用户编号为另一维度，做矩阵分解。

相比于传统的 SVD 推荐，因为属性值往往少于商品数（很多商品的某一属性值相同），这种基于属性的 SVD 中矩阵维度更小。

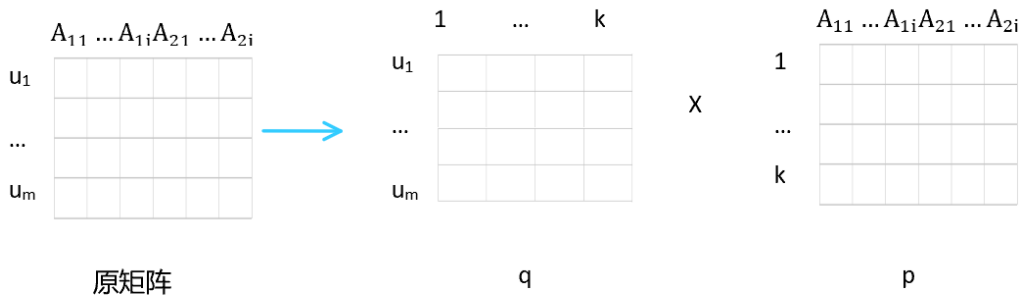
同时，很多商品刚上市时并未被用户打过分、或用户打分数目较少，但可基于其具有的属性而获得目标用户对它的打分估计。因此，也一定程度上贡献于“冷启动”问题。

下面以图为例进行阐述：

假设用户为$u_1, \ldots, u_m$

属性 1 值为$A_{11}, \ldots, A_{1i}$

属性 2 值为$A_{21}, \ldots, A_{2i}$



（用户*属性矩阵分解为用户*k 维矩阵和 k*属性维矩阵）

```
for user, attri, rating in self.userAttriRat:
    count+=1
    loss = 0.0
    user = int(user)
    attri = int(attri)

    predict_rating = np.dot(self.qMatrix2[user, :], self.pMatrix2[:, self.attris[attri]])+self.overallMean2+self.UserBiasA[user]+self.AttriBias[attri]
    thisLoss = rating-predict_rating

    q_change = Rate*(thisLoss*self.pMatrix2[:, self.attris[attri]]-lambda2*self.qMatrix2[user, :])
    p_change = Rate*(thisLoss*self.qMatrix2[user, :]-lambda2*self.pMatrix2[:,self.attris[attri]])
    self.qMatrix2[user, :] += q_change
    self.pMatrix2[:, self.attris[attri]] += p_change

    self.UserBiasA[user] += Rate * (thisLoss - lambda2 * self.UserBiasA[user])
```

```
16.     self.AttriBias[attri] += Rate * (thisLoss -
    lambda2 * self.AttriBias[attri])
```

4. 网格调参

对于超参数正则化因子、学习率和潜在因子数 k,利用网格调参（adjust 函数），以对用户-商品矩阵为例：

```
1.  self.minLoss1 = 300
2.  for i in range(18, 22):  # 对 k1 调参
3.      for sturate in [0.001, 0.0003]:  # 对学习率调参
4.          for lambda1 in [0.01, 0.1]:  # 对正则化因子调参
5.              self.train(k1=i*10, lambda1=lambda1, stuRate=sturate, steps=15,
    choose_step=True)
6.  print("best k1:"+str(self.k1))
7.  print("best study rate:"+str(self.stuRate1))
8.  print("best lambda1:"+str(self.lambda1))
9.  print("best step1:"+str(self.steps1))
```

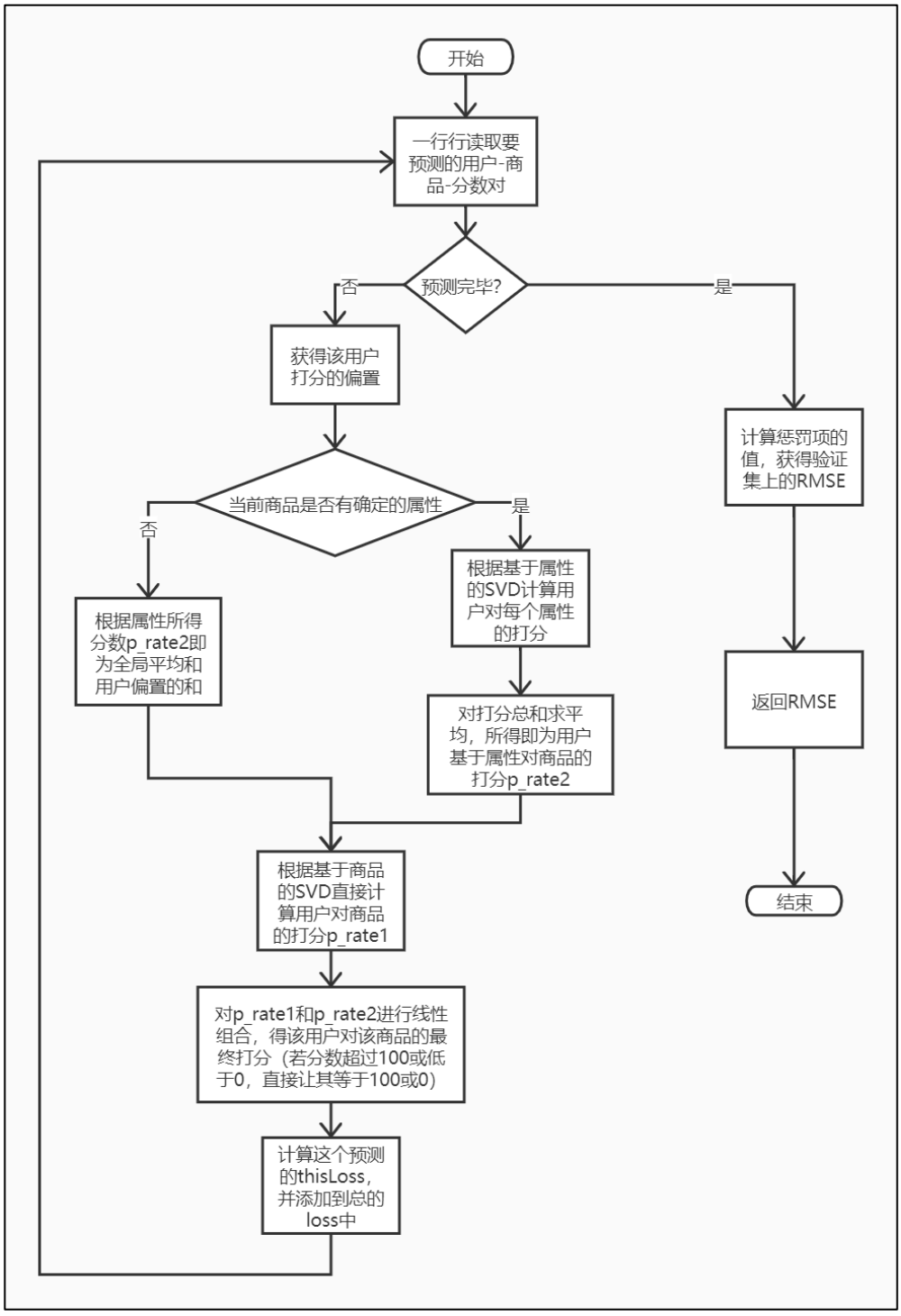
（三）验证集测试

利用用户-商品矩阵和用户-属性值矩阵进行测试（注意到商品属性值在属性 1 与属性 2 上无交叉，因此共用一个矩阵）

```
1.  def validate(self, coe1=0.04, coe2=0.96, choose_coe=False):  # 基于商品-属性-
    用户进行打分预测
2.      loss = 0
3.      loss_item = 0
4.      loss_attri = 0
5.      for user, item, rating in self.testUserItemRat:
6.          count = 0
7.          user = int(user)
8.          item = int(item)
9.
10.         # 计算用户对属性的打分
11.         p_rate2 = 0
12.         # 获得该用户打分的偏置
13.         if user not in self.UserBiasA.keys():
14.             userb = 0
15.         else:
16.             userb = self.UserBiasA[user]
17.
18.         if item not in self.AttributeOfItem.keys():  # 商品没有属性
19.             p_rate2 = self.overallMean2+userb
20.         else:  # 商品可能有属性
21.             for attri in self.AttributeOfItem[item]:
```

```
                count += 1
                if user not in self.UserBiasA.keys():  # 这个用户没打过分
                    if attri not in self.AttriBias.keys():
                        p_rate2 += self.overallMean2
                    else:
                        p_rate2 += self.overallMean2+self.AttriBias[attri]
                else:  # 这个用户打过分
                    if attri not in self.AttriBias.keys():
                        p_rate2 += self.overallMean2+userb
                    else:
                        p_rate2 += np.dot(self.qMatrix2[user, :], self.pMatrix2[:, self.attris[attri]])+self.overallMean2+self.UserBiasA[user]+self.AttriBias[attri]

            if count == 0:
                p_rate2 = self.overallMean2+userb
            else:
                p_rate2 /= count  # 用户对该商品属性打分的平均

        # 计算用户对商品的打分
        if item not in self.itemBias.keys():
            if user not in self.userBias.keys():
                p_rate1 = self.overallMean
            else:
                p_rate1 = self.overallMean+self.userBias[user]
        else:
            if user not in self.userBias.keys():
                p_rate1 = self.overallMean+self.itemBias[item]
            else:
                p_rate1 = np.dot(self.qMatrix[user, :], self.pMatrix[:, self.items[item]])+self.overallMean+self.userBias[user]+self.itemBias[item]
        predict_attri = p_rate2
        predict_item = p_rate1
        # 线性组合
        predict_rating = coe1*p_rate1+coe2*p_rate2

        predict_attri = round(predict_attri/10)*10
        predict_item = round(predict_item/10)*10
        predict_rating = round(predict_rating/10)*10

        if predict_rating > 100:
            predict_rating = 100
        if predict_rating < 0:
            predict_rating = 0
```

```
63.         if predict_attri > 100:
64.             predict_attri = 100
65.         if predict_attri < 0:
66.             predict_attri =0
67.         if predict_item > 100:
68.             predict_item = 100
69.         if predict_item < 0:
70.             predict_item = 0
71.
72.         thisLoss = rating-predict_rating
73.         loss += np.square(thisLoss)/500
74.         loss_attri += np.square(rating-predict_attri)/500
75.         loss_item += np.square(rating-predict_item)/500
76.
77.     num = len(self.testUserItemRat)
78.     loss1 = coe1*self.lambda2 * (((self.qMatrix2 * self.qMatrix2).sum())/num
    ) + \
79.         coe2*self.lambda1 * (((self.qMatrix * self.qMatrix).sum())/num)
80.     loss2 = coe1*self.lambda2 * (((self.pMatrix2 * self.pMatrix2).sum())/num
    ) + \
81.         coe2*self.lambda1 * (((self.pMatrix * self.pMatrix).sum())/num)
82.     loss3 = coe1*self.lambda2*((sum(list(map(lambda num: num*num, self.UserB
    iasA.values()))))/num)\
83.         + coe2*self.lambda1*((sum(list(map(lambda num: num*num, self.userBia
    s.values()))))/num)
84.     loss4 = coe1*self.lambda2*((sum(list(map(lambda num: num*num, self.Attri
    Bias.values()))))/num)\
85.         + coe2*self.lambda1*((sum(list(map(lambda num: num*num, self.itemBia
    s.values()))))/num)
86.     rmse = np.sqrt(loss/num*500+loss1+loss2+loss3+loss4)
87.     if not choose_coe:
88.         print('validation RMSE (only item) :'+str(np.sqrt(loss_item/num*500+
    loss1+loss2+loss3+loss4)))
89.         print('validation RMSE (only attribution) :'+str(np.sqrt(loss_attri/
    num*500+loss1+loss2+loss3+loss4)))
90.         print('validation RMSE (both item and attribution) :'+str(rmse))
91.     else:
92.         return rmse
```



validate()函数的流程图

四、实验相关统计信息

(一)实验结果

1.验证集

验证集进行预测并计算 RMSE。其中仅使用基于属性的 SVD 得到的 RMSE 大约为 35，仅使用基于商品的 SVD 得到的 RMSE 大约为 27.3，同时使用属性和商品的 SVD 得到的 RMSE 大约为 26.3，可见加入属性可有效提高算法的表现。

2.测试集

测试集的结果，见“实验结果”中的“result.txt 文件”。

(二)算法性能

1.所用时间

我们的程序有三种获得结果的模式，第一种是直接调用预先训练好的模型，第二种是使用默认的参数从头训练模型，第三种是通过网格调参函数，获得这个验证集上最好的参数，再使用这个参数进行训练。

（1）使用预先训练好的模型，程序需要花费 1 分钟左右。

（2）使用默认参数从头开始训练模型，需要 30 分钟左右。

（3）进行网格调参，根据本次最好参数训练模型，需要 13-14 个小时。当前训练函数中的默认参数是跑过多次网格调参后选出的，所以可以直接使用默认参数，已经没有必要再选择调参模式。

2.模型所占内存

当前模型所占内存约为 1GB。

五、程序运行说明

运行程序前，请先看“可执行文件”文件夹下的“README.txt”。