

推荐系统第一次作业实验报告

学号： 10185102136

姓名： 张靖

专业名称： 计算机科学与技术

学生年级： 2018 级本科

指导教师： 张伟

课程性质： 专业任意选修

研修时间： 2020~2021 学年第 2 学期

华东师范大学计算机科学与技术学院

2021 年 4 月 1 日

目 录

摘要与文件说明	3
一：协同过滤基本算法	4
一、算法原理.....	4
1.基于用户的协同过滤算法.....	4
2.基于商品的协同过滤算法.....	4
二、相似度计算.....	5
三、实验使用环境.....	6
四、核心代码注释.....	6
数据读入.....	6
模型预测.....	7
五、实验结果比对.....	8
二：协同过滤改进方法	9
一、分段函数法.....	9
1.残差网络思想.....	9
2.偏移截断法.....	9
二、COS 中心偏移.....	10
添加用户平均分.....	10
Z-score	10
item 平均分修正.....	11
尝试采用 ITEM-CF（理论上效果不好）.....	11
三：基于神经网络的协同过滤	12
一、网络架构以及优势.....	12
二、核心代码注释.....	13
构建 CF 类：	13
构建 MODEL 类：	14
三、实验结果分析.....	14
四：使用平均值的 slope-one 算法	15
一：大致思路.....	15
二：核心代码与最后结果.....	15

摘要与文件说明

名称	修改日期	类型	大小
ITEM-CF	2021/4/1 22:05	文件夹	
SLOPE-ONE (使用这个来Predict)	2021/4/1 22:00	文件夹	
USER-CF (改良后的)	2021/4/1 22:01	文件夹	
神经网络方法	2021/4/1 22:01	文件夹	
predict.csv	2021/3/30 16:00	Microsoft Excel ...	163 KB
test.csv	2021/3/24 11:19	Microsoft Excel ...	129 KB
train.csv	2021/3/24 11:19	Microsoft Excel ...	543 KB
推荐系统第一次作业_10185102136_张...	2021/4/1 22:11	Microsoft Word ...	2,013 KB

一共分为如上 7 个文件，train 和 test 是作业里的训练和测试集，predict 是按照 test 预测得到的分数，最终结果在 test 中。每个文件夹有对应方法的 jupyter notebook 文件，可以用 prompt 打开里面的代码。一共有 4 个方法。

本次任务是协同过滤回归任务，主要是基于数据的预测，由于矩阵比较稀疏，所以要多多尝试用哪个方法最好。在实验过程中可能有代码优化不够，或者方法还不够到位的地方。报告中也有可能出现解释错误，解释不够的情况。但是每个步骤都是我努力研究，花时间花精力去努力的结果。而且在本地测试集，我也用了最佳的 slope-one，认为这个最好去预测，如果结果过拟合不太好，我也觉得这是一个提升的过程，但是不代表这个模型不好。

用的方法有：基于 USER-CF, ITEM-CF 的 COS, JACCARD, PEARSON 相关系数方法，协同过滤优化，分段函数，COS 中心偏移方法，神经网络方法，SLOPE-ONE 方法。

最后，感谢指导我的老师张伟和助教对数据集的提供与修正，还有一起交流的同学，大家都给我对方法产生了思考与影响，也是能够让我能想到各种方法的渠道，希望最后能够得到好成绩！

2021 年 4 月 1 日

ECNU 18 计科一班 10185102136 张靖

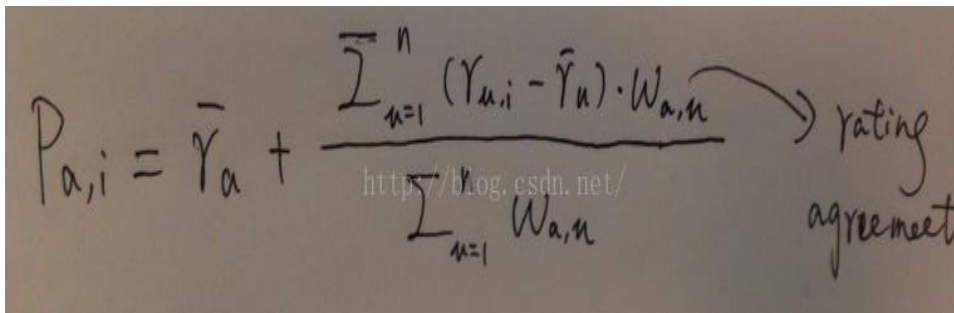
一：协同过滤基本算法

一、算法原理

1.基于用户的协同过滤算法

协同过滤可以看做是机器学习中的 K 近邻算法，选取相似的几个用户，用这些已经评分过的商品得分去计算当前用户没有评分过的商品，从而预测出对未知商品的喜好程度。

首先要计算用户间的相似度，在 user-item 矩阵中，可以通过皮尔逊相关系数，COS 相似度，JACCARD 相似度去计算各个用户的相似度，在相似度中选取相应的比较高的相似度值，用得分函数取计算用户的得分，具体公式如下：


$$P_{a,i} = \bar{r}_a + \frac{\sum_{n=1}^n (r_{a,i} - \bar{r}_a) \cdot W_{a,n}}{\sum_{n=1}^n W_{a,n}}$$

rating agreement

2.基于商品的协同过滤算法

基于商品的协同过滤算法与基于用户协同过滤算法，是相似的。主要区别在于计算相似度的时候，是以 ITEM 作为主体，可以看做 user-item 矩阵中，求了转置，用新的转置矩阵去计算相似度。

在计算预测的得分时，是根据用户自己对其他的打分，来计算位置商品的打分，与用户的协同过滤稍有不同。计算公式如下：

■ Another view: Item-item

- For item i , find other similar items
- Estimate rating for item i based on ratings for similar items
- Can use same similarity metrics and prediction functions as in user-user model

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

s_{ij} ... similarity of items i and j
 r_{xj} ...rating of user x on item j
 $N(i;x)$... set items rated by x similar to i

总体看来，如果用户数量>物品数量，选择 USER-CF 会好一点。如果是物品数量>用户数量，那么选择 ITEM-CF 会好一点。具体看对相似度产生影响最小的，就选哪一个。

观察结果，对比 train 和 test 文件可知，本任务是个热启动问题。而且用户数量小于商品数量，采用 USER-CF

二、相似度计算

相似度计算中，USER-CF 和 ITEM-CF 是一样的。

首先是 COS 相似度：

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

其次是皮尔逊相关系数：

Pearson Correlation Coefficient

$$r_{a,n} = \frac{\sum_{i=1}^m (r_{a,i} - \bar{r}_a)(r_{u,i} - \bar{r}_u)}{\sqrt{\sum_{i=1}^m (r_{a,i} - \bar{r}_a)^2} \sqrt{\sum_{i=1}^m (r_{u,i} - \bar{r}_u)^2}}$$

- range of $[-1, 1]$
- compared with Spearman Ranking Correlation
- m ratings in common

可以看到皮尔逊相关系数实际上就是先对数据进行归一化，然后用 cos 相似度计算。

然后是 JACCARD 相似度（广义）：

$$J(A, B) = \frac{\sum_i \min(x_i, y_i)}{\sum_i \max(x_i, y_i)}$$

最后以平均值计算作为参考：

$$p_{a,i} = \frac{\sum_{u=1}^n r_{u,i}}{n}$$

prediction

user a

item i

ratings

user u rated item i

<http://blog.csdn.net/>

三、实验使用环境

python3.6 版本以上, anaconda-prompt 最新版本, pytorch 最新版本, 不用 gpu 也行 (后面神经网络要用), numpy, pandas 最新版本。sklearn 最新版本 (因为要用包来快速计算相似度)

IDE 推荐 pycharm, 或者 jupyter notebook

四、核心代码注释

数据读入

首先是数据读入部分, 数据读入部分比较繁琐, 按照 pandas 读取数据, 之后建立字典, 将数据中的字符串一一映射到数字当中, 同时由于是热启动问题, 所以不需担心映射不到数字。读取所有数据, 并且字典当中保存 string2int 的数据, 然后列表保存了 user-item-star 的结构。然后用 user_count 和 business_count 来记录有多少个用户, 商品。

```
In [3]: user_dict={}
business_dict={}
#print(dataset)
# 对于每一行, 通过列名name访问对应的元素
users=[]
items=[]
stars=[]
time_year=[]
user_count=0
business_count=0
for i in range(0, len(dataset)):
    user=dataset.iloc[i]["user_id"]
    item=dataset.iloc[i]["business_id"]
    if(user not in user_dict):
        user_dict[user]=user_count
        user_count=user_count+1
    if(item not in business_dict):
        business_dict[item]=business_count
        business_count=business_count+1
    users.append(user_dict[user])
    items.append(business_dict[item])
    stars.append(float(dataset.iloc[i]["stars"]))
    time_year.append(float(dataset.iloc[i]["date"][0:4]))
print(user_count)
print(len(items))
print(max(time_year))
print(min(time_year))

1103
7932
2019.0
2005.0
```

然后是计算相似度, 这里以 COS 相似度为例子, 其他类似。

```
In [7]: #user训练, 包括Matrix和cos
import math
matrix = np.zeros((user_count, business_count))
#0->6999是训练集\n
for i in range(7000):
    matrix[users[i]][items[i]]=stars[i]
from sklearn.metrics.pairwise import cosine_similarity
cos_users =cosine_similarity(matrix)
```

新建一个 matrix 数组, 代表了 user-item, 之后用 sklearn 的包快速计算每个 user-user 的相似度。

模型预测

最后是模型的预测, 模型的预测需要有 2 个判断:

- 1: 如果是冷启动 (经过检查, 发现不存在这个情况), 这种是用户不存在, 商品也不存在, 那么就直接评分 3
- 2: 如果只是用户冷启动, 那么对商品取平均分, 直接取平均分

```
for j in range(0, business_count):
    if(matrix[require_user][j]!=0):
        aresult=areult+matrix[require_user][j]
        afenmu=afenmu+1
    if(afenmu==0):
        ra=3
    else:
        ra=areult/afenmu
```

判断完以上 2 种特殊情况时候, 就可以按照公式直接计算了。

```

# 遍历用户
for j in range(0, user_count):
    if ((require_user != j) and (cos_users[require_user][j] > 0.999) and (matrix[j][require_item] != 0)):
        temp_fenmu = 0
        temp_result = 0
        rb = 0
        for k in range(0, business_count): # 遍历j对象所有的物品，取评分平均值
            if (matrix[j][k] != 0):
                temp_result = temp_result + matrix[j][k]
                temp_fenmu = temp_fenmu + 1
        rb = temp_result / temp_fenmu
        fenmu = fenmu + cos_users[require_user][j]
        # print(result)
        result = result + cos_users[require_user][j] * (matrix[j][require_item] - rb)
if (fenmu == 0):
    result = ra
else:

```

红色划线部分代表了阈值，整段代码的意思是，遍历出相似度高的用户，而且对这个商品评分不为 0，之后遍历这个用户的评分平均值，然后按照下列的公式，计算出 result 值即可。

$$s(u, i) = \bar{r}_u + \frac{\sum_{v \in V} (r_{vi} - \bar{r}_v) * w_{uv}}{\sum_{v \in V} w_{uv}}$$

这个公式代表了是修正的公式，考虑到了每个用户评分的评价程度。有的用户评分相对高，有的评分相对低，所以用这个公式比较合理。经过计算，修正的 USER-CF 分会比没修正的 USER-CF 高。所以只采用这个公式。

五、实验结果比对

使用 0~6999 作为训练集，7000~len(users) 作为测试集

基本的调参方式有：设置 TOP-N，阈值，还有一些其他的调整。

我用的是阈值方法，不用 TOP-N。

大致得到最佳情况如下：

使用方法	平均值	COS	PEARSON	JACCARD
LOSS	1.105023153645934	1.1581796 365994226	约等于 1.13	约等于 1.2
参数调整	作为参考	设置相似度 阈值>0.2	设置相似度 阈值>0.2	设置相似度阈 值>0.4

可以看见，简单用协同过滤计算发现准确度还不如用平均值的，（也有可能是我算法自己出问题了，不过检查了很久，发现应该没有问题），所以必须加以改进。

二：协同过滤改进方法

考虑到实现简便，统一采用 USER-CF，并且使用 COS 相似度去优化。

一、分段函数法

分段函数方法是我自创的方法，总体思想是：

用户不管对于什么物品来说，收到主要影响的因素仍然是自己对评分的估计程度，所以总的来看，如果基于预测的方法，预测出跟自己评分的差异很大，那么就取分段函数，分段有几个方法：

1.残差网络思想法

可以认为，最后一定要连接一个用户对所有的平均值，这样能保证结果至少不会比平均值差

2.偏移截断法

偏移截断法的思想就是，如果预测的得分比用户差异很大，那么大的就直接不要掉预测的得分，或者乘 0.1，这样结果不会受到特殊情况的影响，或者说明偏移太大是数据量不够导致算不准。

另外，这个“太大”需要分类讨论，比如 $\text{abs}(\text{用户平均分}-\text{预测的得分}) > 0.5$ ，那么 $\text{result} = \text{用户平均分} + \text{预测的得分} * 0.1$ ，然后 $\text{abs}(\text{用户平均分}-\text{预测的得分}) > 0.2$ ， $\text{result} = \text{用户平均分} + \text{预测的得分} * 0.6$ 这样。

基于这两个思想，去改良如下代码：

```
if(fenmu==0):
    result=ra
else:
    result=result/fenmu
    # result=result+ra
    if(abs(result)>0.4):
        result=ra+result*0.1
    elif(abs(result)>0.2 and abs(result)<0.4):
        result=ra+0.5*result
    elif(abs(result)>0.1 and abs(result)<0.2):
        result=ra+0.8*result
    else:
        result=ra+result
```

数据集测试	平均分 LOSS 参考	协同过滤优化
>7000	1.105023153645934	1.0635761261855121
>7500	1.081204217112636	1.0707239427104152
>7920	1.98	1.8333421651602115

发现准确率是有一定提升的。

之后在基于分段函数中，去改良下面的 COS 中心偏移

二、COS 中心偏移

第二种改进方法是 COS 中心偏移。中心偏移有 3 种方法。中心偏移是为了解决矩阵稀疏的情况。

添加用户平均分

思路：把没有评分过的商品，统统添加上自己的平均分，这种方法比较粗暴，而且没有很强的理论性，只是单纯加上平均分为了不稀疏而已。但是发现效果很好。具体代码添加：

```
matrix_cos[matrix_cos==0] = user_mean
#第一种方法，没有的值用平均值填补
for i in range(user_count):
    fenzi=0
    fenmu=0
    for j in range(0,business_count):
        if(matrix_cos[i][j]!=0):
            fenzi=fenzi+matrix[i][j]
            fenmu=fenmu+1
    result=fenzi/fenmu
    for j in range(0,business_count):
        if(matrix_cos[i][j]==0):
            matrix_cos[i][j]=result
```

只要添加这行代码，就可以使得准确率提升很多，设置 COS 阈值>0.99
用 0~数据集测试的数据去训练，可得：

数据集测试	平均分 LOSS 参考	协同过滤优化
>7000	1.105023153645934	1.0635761261855121
>7500	1.158506767986353	1.0707239427104152
>7920	1.9841086785188515	1.8333421651602115

Z-score

思路：

■ Normalize ratings by subtracting row mean

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	$2/3$			$5/3$	$-7/3$		
B	$1/3$	$1/3$	$-2/3$				
C				$-5/3$	$1/3$	$4/3$	
D		0					0

如上图，可以把所有有评分过的商品，全部映射到平均分为 0，这样为 0 的数据就不会产生影响了。也可以考虑方差也映射到 1 去。这样计算相似度比较合理。但是经过计算，准确率更低了，直接淘汰。

item 平均分修正

思路：item 修正考虑到不能每个用户都用平均分去填补，用商品的平均分去填补，这样就能综合考虑用户的平均分，商品的平均分去填补，解决稀疏问题。

虽然感觉理论上会优于第一种改良，但是经过实践证明，是不如第一种的，所以综上，采用第一种+分段函数优化。

尝试采用 ITEM-CF（理论上效果不好）

ITEM-CF 算法跟 USER-CF 算法是差不多的。改进的点在如下：

```
5]: #item训练, matrix2包含了item-user矩阵
matrix2=matrix.T
cos_items=cosine_similarity(matrix2)
```

首先是对矩阵求了转置，将原来的 USER-ITEM 变成 ITEM-USER，然后计算相似度。

```

afermu=0
for j in range(0,business_count):
    if(matrix2[j][require_user]):
        afermu=afermu+1
        afenzi=afenzi+matrix2[j][require_user]
ra=afenzi/afermu
#print(ra)
for j in range(0,business_count):
    if((require_item!=j) and (cos_items[require_item][j]>0.5) and (matrix2[j][require_user]!=0)):
        temp_fenmu=0
        temp_result=0
        fenmu=fenmu+cos_items[require_item][j]
        #print(result)
        result=result+cos_items[require_item][j]*matrix2[j][require_user]

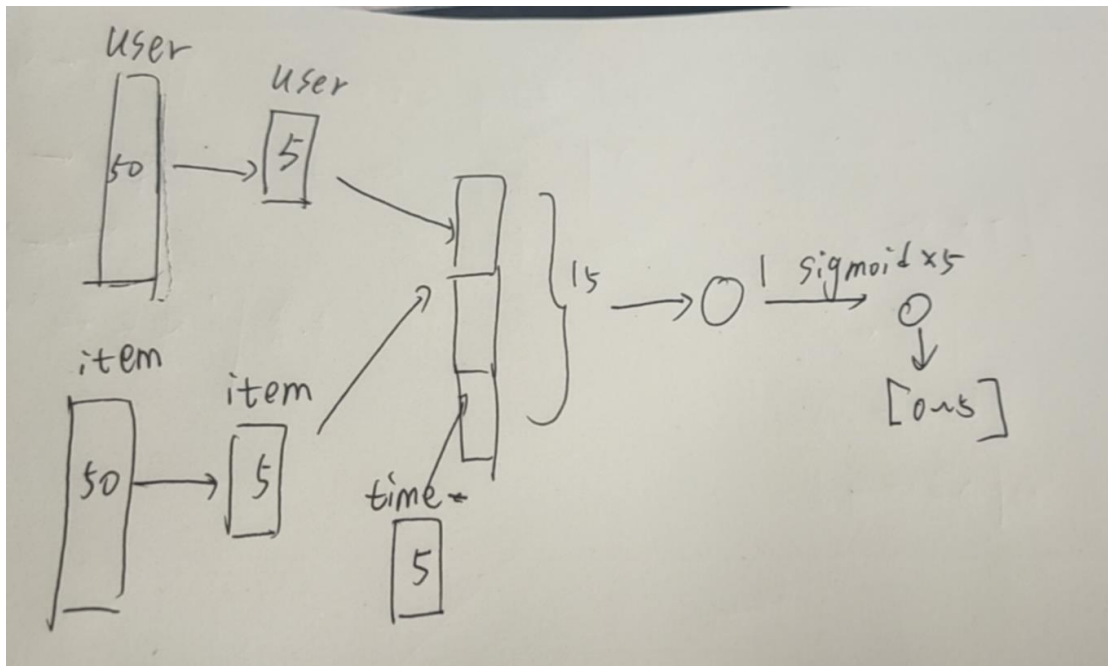
```

剩下的部分也差不多，就是基于 ITEM 部分的 COS 进行计算，不多阐述。
而且均方误差： $=1.1061273940578693$ ，基本没有改进，放弃研究这种方法，也不过多说明。

三：基于神经网络的协同过滤

一、网络架构以及优势

定义网络如下：



利用神经网络的 embedding，将 user 变成 50 维的向量，item 也是。之后经过全连接层，变成 5 维向量，之后 3 个 5 维向量拼接，经过 $\text{sigmoid} \times 5$ 变成 $0 \sim 5$ 的值。
利用神经网络的优势：

1. 可以利用时间。用 USER-CF 的话，时间处理比较麻烦，而且不一定有提升，现在用 year 来映射 embedding，不需要特殊处理，可以自动学习内在特征。
2. 解决了稀疏问题。由于 USER-ITEM 的表，稀疏性太强，算相似度会不太准，用神经网络自学习可以很好解决这个问题。
3. 热启动问题有优势。神经网络会容易过拟合，如果是冷启动怕会有问题，但是

本问题是热启动问题，所以不用太担心。

4. 可以自学习内在特征。用户与用户之间的关系是有内在反映的。单纯评分可能无法考虑到交叉特征，采用神经网络是有优势的。

二、核心代码注释

构建 CF 类：

```
class CF(torch.nn.Module):
    """
    Linear Regressoin Module, the input features and output
    features are defaults both 1
    """
    def __init__(self):
        """
        Initialize the Linear Model
        """
        super().__init__()
        self.user_embedding=torch.nn.Embedding(1103,50)
        self.item_embedding=torch.nn.Embedding(1733,50)
        self.time_embedding=torch.nn.Embedding(15,5)
        self.user_linear=torch.nn.Linear(50,5)
        self.item_linear = torch.nn.Linear(50, 5)
        self.final=torch.nn.Linear(15,1)
        self.sigmoid=torch.nn.Sigmoid()
    def forward(self, user_id, item_id, time_id):
        x=self.user_embedding(user_id)
        x=self.user_linear(x)
        y=self.item_embedding(item_id)
        y=self.item_linear(y)
        z=self.time_embedding(time_id)
        out=torch.cat((x, y, z), 1)
        out=self.final(out)
        out=self.sigmoid(out)*5
        #print(out)
        return out
```

CF 类里面，有 Embedding 层，全连接层，SIGMOID 函数。前向传播网络跟图是一样的。而且顺序依然是相同的，这里不做过多的阐述。

构建 MODEL 类:

```
class Model():
    def __init__(self):
        self.create_model()
        self.loss_function = torch.nn.MSELoss()
        self.optimizer = torch.optim.Adam(self.model.parameters(), lr=0.000005)
    def create_model(self):
        self.model = CF()
    def train(self, user_ids, item_ids, stars, time_id):
        #self.model.load_state_dict(torch.load("linear.pth"))
        #self.optimizer.zero_grad()
        temp_total=100
        for epoch in range(1000000):
            total=0
            for j in range(0, 7000, 50):
                # print(input_ids[j, j+50])
                prediction = self.model(user_ids[j:j+50], item_ids[j:j+50], time_id[j:j+50][:, 0])
                # prediction=torch.tensor(prediction, dtype=torch.float32)
                # labels=list(labels)
                # labels = torch.tensor(labels[j:j+50], dtype=torch.float32)
                #print(prediction)
                loss = self.loss_function(prediction, stars[j:j+50])
                self.optimizer.zero_grad()
                loss.backward()
                self.optimizer.step()
                #torch.save(self.model.state_dict(), "linear.pth")
                print(j)
                print("epoch: 0, loss is: 0".format(epoch, loss.item()))
                total=total+loss.item()
            print("epoch: 0, loss is: 0".format(epoch, loss.item()))
            print("均方误差: =0".format(total / 140))
            if(temp_total<total/140):
                break
            temp_total=total/140
            torch.save(self.model.state_dict(), "linear.pth")

    def test(self, user_ids, item_ids, stars, time_id):
        self.model.load_state_dict(torch.load("linear.pth"))
        prediction = self.model(user_ids, item_ids, time_id[:, 0])
        loss = self.loss_function(prediction, stars)
        print("均方误差: =0".format(loss.item()))
```

MODEL 类部分也不难，设置 batchsize=50, 然后放到 CF 类中去训练。大概要迭代到 loss=0.9 比较完美，否则接下去会过拟合。为了防止训练出问题，设置 ADAM 优化器去优化，用 MSE 均方误差来优化，跟任务目标是一致的。

三、实验结果分析

由于神经网络训练有点久，所以统一用 0~6999 训练，改变测试集来测试

数据集测试	平均分 LOSS 参考	协同过滤优化	神经网络
>7000	1.105023153645934	1.06357612618 55121	1.0052713155746 46
>7500	1.158506767986353	1.08496325619 1523	0.9828014373779 297
>7920	1.966831265730693 2	1.94862842591 87856	1.6706066131591 797

可以看到神经网络准确率算是比较高的。

四：使用平均值的 slope-one 算法

一：大致思路

目前发现使用隐变量模型的准确率较高，但是其实有一种很简单的算法，而且发现用在这个数据集上，效果非常的好。结果是 slope-one 算法。

算法的思路很简单，对于一个用户 i ，商品 J ，他的预测得分就是用户 i 的平均分加上商品 J 的平均分/2 即可，当然这种比率是可以调整的，而且结果非常之好。但是不知道放在训练集是否会出现问题，但是我还是以这个算法作为最后的 predict。

二：核心代码与最后结果

算法比较简单，与前面的代码改动部分如下：

```
for j in range(0,business_count):
    if(matrix[require_user][j]!=0):
        aresult=aresult+matrix[require_user][j]
        afenmu=afenmu+1
    if(afenmu==0):
        ra=3
    else:
        ra=aresult/afenmu

    #计算ra
    #print(ra)
    rb=0
    temp_fenmu=0
    temp_result=0
    for j in range(0,user_count):
        if((require_user!=j) and (matrix[j][require_item]!=0)):
            temp_fenmu=temp_fenmu+1
            temp_result=temp_result+matrix[j][require_item]
    if(temp_fenmu==0):
        rb=3
    else:
        rb=temp_result/temp_fenmu

    # if(abs(result)>0.4):
    #     result=ra+result*0.1
    # elif(abs(result)>0.2 and abs(result)<0.4):
    #     result=ra+0.5*result
    # elif(abs(result)>0.1 and abs(result)<0.2):
    #     result=ra+0.8*result
    # else:
    #     result=ra+result
    #result=(ra+rb)/2

result=(ra+rb)/2
```

思路很简单，改下设置成平均值就可以了，但是效果却出奇的好。可能还是由于数据集不全，然后矩阵过于稀疏，这种情况往往使用相似度，隐变量感觉考虑太过复杂，反而对模型产生不好的影响。

由于神经网络训练有点久，所以统一用 0~6999 训练，改变测试集来测试

对比结果如下：

数据集测试	平均分 LOSS 参考	协同过滤优化	神经网络	SLOPE-ONE
>7000	1.105023153645934	1.0635761261855121	1.005271315574646	0.9509809160756498
>7500	1.158506767986353	1.084963256191523	0.9828014373779297	0.9599962475039179
>7920	1.9668312657306932	1.9486284259187856	1.6706066131591797	1.3838358221488483

有时候，大道至简，用最简单的方式，有时候就是最好的方式。