



南开大学  
Nankai University

南 开 大 学  
网 络 空 间 安 全 学 院  
大数据计算与应用实验报告

---

实验 2：推荐系统

---

罗功成 1910487

年级：2019 级

专业：信息安全

2022 年 6 月 2 日

## 摘要

运用 Pearson 相关系数计算相似度，基于商品相似度的协同过滤实现推荐系统预测打分

关键字：Pearson 相关系数，协同过滤，推荐系统

## 目录

一、 实验目的	1
二、 实验环境	1
三、 数据集说明	1
四、 程序运行说明	1
五、 数据集统计信息	2
六、 代码实现思路	2
(一) 数据预处理	2
(二) 基于 pearson 相关系数的协同过滤	2
(三) 基于商品相似度的预测评分	3
七、 核心代码	4
八、 实验结果	13
九、 结果分析	13
十、 对比实验	15
十一、 总结和收获	16

## 一、 实验目的

对给定数据集，预测用户  $u$  对商品  $i$  的打分

## 二、 实验环境

windows10+vs2022+release 编译。

## 三、 数据集说明

1. 数据集文件 exe 文件放在同一个文件夹（推荐系统）下。

2. 可执行文件采用 release 方式编译

程序源码：recommend.cpp, 数据预处理.h, Pearson.h, 平均值计算.h

可执行文件：recommend.exe

训练集：train.txt; 测试集：test.txt; 其他依赖文件：itemAttribute.txt

实验结果：最终结果.txt，为避免覆盖，在执行文件时另建一个空白的 result.txt 来存放输出结果。

实验报告：.pdf 文档

## 四、 程序运行说明

将 release 版本的 exe 文件和训练集、测试集等文件放在同一个文件夹后，直接点击运行 exe.

经实测，在 win10,win11 平台下的其他电脑均可正常运行该程序。

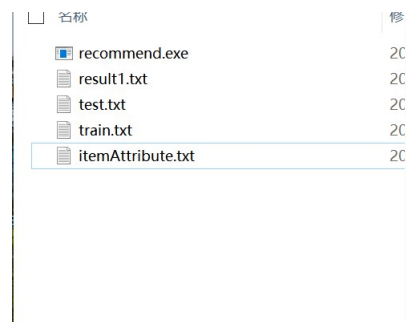


图 1: 可执行文件，数据集，结果集等

## 五、 数据集统计信息

1. 用户总数: 19835

2. 商品总数: 624961

对于用户和商品, 下标最大值分别为 19834, 624960. 但由于是从 0 开始的, 因此总数上要加 1.

3. 评分记录条数: 5001507

4. 商品的平均得分: 49.5046

## 六、 代码实现思路

### (一) 数据预处理

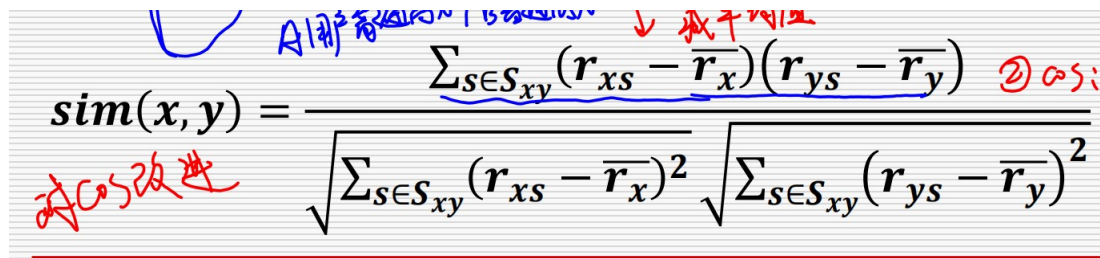
读取数据后进行分块处理, 然后统计商品和用户总数, 将每块原始内容按稠密矩阵存储,  $(u, i)$  作为对应下标关系存放得分情况。然后以稀疏矩阵的方式对其进行转化, 并计算总体的得分。然后根据 Pearson 相关系数和协同过滤等的公式需要可对应行列进行运算。

由于商品的序号是无序的, 并且分布比较均匀, 所以用插值查找查看用户看过哪些商品/商品被哪些用户评分

在统计表的基础上构建一个存放 pearson 相关系数的表, 用于下面协同过滤和预测评分。

### (二) 基于 pearson 相关系数的协同过滤

pearson 相关系数有效的解决了 Jaccard 中由于非 0 即 1 的粒度过粗的问题, 以及 cos 相似度中可能会出现由于没有观看而没有打分, 被认为不喜欢而打成负分的问题。



The image shows the formula for Pearson correlation coefficient with handwritten annotations in red and blue. The formula is:

$$sim(x, y) = \frac{\sum_{s \in S_{xy}} (r_{xs} - \bar{r}_x)(r_{ys} - \bar{r}_y)}{\sqrt{\sum_{s \in S_{xy}} (r_{xs} - \bar{r}_x)^2} \sqrt{\sum_{s \in S_{xy}} (r_{ys} - \bar{r}_y)^2}}$$

Annotations include: "A1" in blue, "我平均值" (my average) in red, "对cos改进" (improvement on cos) in red, and "cos:" in red.

图 2: Pearson 相关系数的计算

pearson 相关系数本质上是对 cos 相似度的改进, 它将用户的打分减去用户历史评分的平均值, 这样解决了一些用户普遍高分或普遍低分的影响, 根据打分相较于自己打分平均值的偏差程

度来确定用户对某商品的喜好程度。

### (三) 基于商品相似度的预测评分

由于商品相似度相对于用户的相似度关系更为稳定，因此采用对商品相似度协同过滤。

此外，根据给出的 itemAttribute.txt 中的商品属性，可以采用启发式算法解决新商品问题：

对于那些新加入的商品，由于其打分人数过少而造成的无法找到足够的相似商品，可以用商品属性的方式来确定哪些商品的评分情况可以被用来预测当前商品。

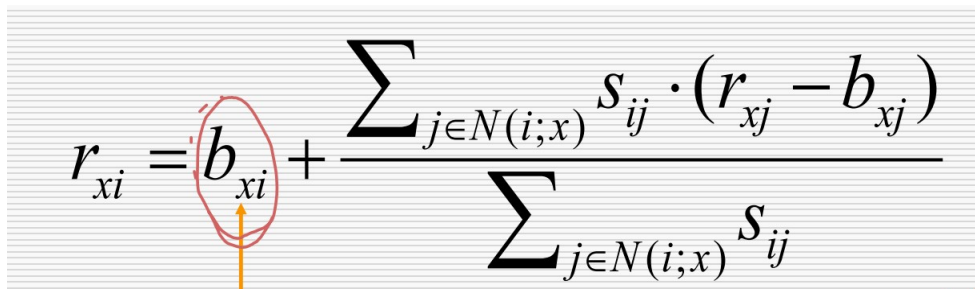
$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$


图 3: 预测评分公式

1.  $b_{xi} = u + b_x + b_i$ : 加权平均后的用户/商品个性化偏差

(1) 其中， $u$  为所有商品被所有用户打分后的总体打分平均值， $b_x$  为用户  $X$  的喜好造成的偏差（即用户  $X$  打分平均值减去所有商品在所有用户打分后的总体平均值）， $b_i$  为某个商品  $i$  得分的平均值减去总体平均值  $u$  的差值。

(2) 实现方式：先读取全部得分后，计算总体平均值；再将用户和商品作为二维数组 `uiscore1` 的两个维度，将对应的  $(u, i)$  填入对应下标位置；分别按行按列求用户打分平均值和商品得分平均值。

注意，空缺位置不填直接跳过，认为用户只是没有看过这部电影而不是不喜欢这个电影，只对得分的位置进行平均值统计

2. 右侧式子记为  $r$ : 排除打分偏差后的加权平均值。

(1) 分母统计了和商品  $i$  相似度最高的  $k$  个商品数量，分子中排除掉偏差后的得分值作为权重乘上对应的商品，进行加权累加和，最后得到一个加权平均。

(2) 首先根据相似度得到  $k$  个匹配最佳的商品，然后利用 1 中得到的偏差值，将要进行预测的部分对应的得分减去偏差值得到权重，然后计算加权平均。

## 七、 核心代码

### 分块处理

```

1 void blockSaving()
2 {
3     int count = usernum/1000;//每1000个用户分成一组
4     ofstream* out = new ofstream[count];
5     char buf[50000];//按50000字符容量缓冲区准备
6     for (int i = 0; i < count; i++)
7     {
8         sprintf_s(buf, "train%d.txt", i);
9         out[i].open(buf);
10    }
11    ifstream in("train.txt");
12    int itemID, rate0;
13    int current = -1;//头节点, 放在第一个节点之前, 记为-1
14    int count6 = 0;//记录当前记录了多少条, 判断是否要换到下一个块记录数据
15    list<int> LinkList;
16    while (!in.eof())
17    {
18        int flag = s.find("|");//每个新用户记录都有自己的userid和“|”
19        if (flag != -1)//有竖线的是新的用户
20        {
21            continue;//直接进行下一步, 读取下面若干行用户的打分信息
22        }
23        in >> itemID >> rate0;//输入为产品号和初始的打分
24        if (current != itemID)
25        {
26            int t;
27            while (!LinkList.empty())
28            {
29                t = LinkList.front();
30                out[current / blocksize] << current << " " <<
31                    counttoNode[current - 1] << " " << t <<
32                    endl;
33                LinkList.pop_front();
34            }
35            current = itemID;
36            count6 = 0;
37        }
38        count6++;
39        LinkList.push_back(rate0);//数据写入链表尾部
40    }
41    cout << "完成分块" << endl;
42    cout << endl;

```

```

42 }
43 }
44 }

```

## 数据预处理

```

1  统计用户/电影总数:
2  int maxitem = 0; //求商品个数
3  int maxuser = 0; //求商品个数
4
5  if (maxitem < ItemID)
6      maxitem = itemID;
7  if (maxuser < UserID)
8      maxuser = UserID;
9
10 查看用户看过哪些商品/商品被哪些用户评分: 插值查找
11
12 //由于商品的序号是无序的, 并且分布比较均匀, 采用插值查找确定用户是否购买该商品
13 int insertfind(itemrate** ratetable, int userid0, int left, int right, int
    key) {
14
15     if (left > right || key < ratetable[userid0][0].itemID || key >
        ratetable[userid0][sizeof(ratetable[userid0]) / sizeof(int) - 1].
        itemID) {
16         return -1;
17     }
18     int mid = left + (right - left) * (key - ratetable[userid0][left].
        itemID) / (ratetable[right].itemID - ratetable[left].itemID);
19     int midvalue = ratetable[userid0][mid].itemID;
20     if (key > midvalue) {
21         return insertfind(ratetable, userid0, mid + 1, right, key);
22     }
23     else
24         if (key < midvalue) {
25             return insertfind(ratetable, userid0, 0, mid - 1, key);
26         }
27         else {
28             return mid;
29         }
30 }
31
32
33
34 /求总体平均值
35 double allitemaverage()
36 {
37

```

```

38     double sum = 0;
39     int count = 0;// (u, i) 对数
40
41     for (int i = 0; i < usernum; i++)//遍历所有用户
42     {
43         for (int j = 0; j < itemnumtable[i]; j++)//遍历每个用户对商品
44             打分
45             {
46                 sum = sum + data0[i][j].rate;//将分值累加
47                 count++;//每产生一个打分记录就加1
48             }
49     }
50     cout << "打分记录 (u,i) 对数有" << count << "条" << endl;
51     double x = sum / count;
52     return x;
53 }

```

## 转化稀疏矩阵

```

1 //构建稀疏矩阵
2 class sparsenode
3 {
4     int row, col;
5     int rate;
6 };
7 class sparsematrix
8 {
9     sparsenode ratadata[5001507];//数组大小为含有打分的总记录数, 5001507
10    int countrate, userID, itemID;//分别记录每个用户ID对商品ID的打分
11    countrate
12 public:
13     sparsematrix(int** a, int b, int c)
14     {
15         itemID = 0;
16         countrate = b;
17         userID = c;
18         int i, j;
19         for (i = 0; i < b; i++)
20         {
21             for (j = 0; j < c; j++)
22             {
23                 if (a[i][j] != 0)
24                 {
25                     ratadata[itemID].value = a[i][j];
26                     ratadata[itemID].row = i;
27                     ratadata[itemID].col = j;
28                     itemID++;
29                 }
30             }
31         }
32     }
33 }

```



```

30         }
31     }

```

## Pearson 相关系数

```

1 //求用户x对所有产品打分的平均值
2 double useraverage(int UserID)
3 {
4     double sum = 0;
5     for (int i = 0; i < itemnumtable[UserID]; i++)//遍历自己所有的商品
6     {
7         sum = sum + data0[UserID][i].rate;//打分相加
8     }
9     double x = sum / itemnumtable[UserID]; //平均
10    return x;
11 }
12
13 //求产品item的所有评分的平均值
14 double itemaverage(int itemID)
15 {
16     double sum = 0;
17     int count = 0;
18
19     for (int i = 0; i < usernum; i++) //遍历所有用户
20     {
21         //在该用户对这个商品的打分处进行累加计算
22         sum = sum + data0[i][itemID].rate;
23         count++;
24     }
25
26     double x = sum / count;
27     return x;
28 }
29
30 double rateitemsimilar(int item1, int item2)
31 {
32     //求rx*ry累加和后的分子
33
34     double rateitem0[10000]; //保存商品的打分
35     int count00 = 0; //保存是第几个商品
36
37
38
39     for (int i = 0; i < itemnumtable[item1]; i++) //遍历商品A的所有打分情况
40     {
41         int j = insertfind(pearson, item1, item2, itemnumtable[item2], pearson
42                             [item1][i].itemID);
43         if (j != -1) //在商品A中的打分，找商品B中是否也有打分
44         {

```

```

44     rateitem0[count00] = (pearson[item1][i].rate - //将那些有对应评分的进
        行计算
45     itemaverage(i)) * (pearson[item2][j].rate - itemaverage(i)); //(rx-avg)
        *(ry-avg)
46     count00++;
47 }
48     }
49
50
51
52     //rx*ry的各项累加和做分子
53     double PeaNumerator = 0;
54     for (int i = 0; i < count00; i++)
55         PeaNumerator = PeaNumerator + rateitem0[i];
56
57
58     //rx-avg累加平方开根 乘 根号下ry-avg累加平方 开根号做分母
59
60
61     double countitemA; double countitemB;
62
63     for (int i = 0; i < itemnumtable[item1]; i++)
64         countitemA = countitemA + (pearson[item1][i].rate - itemaverage
            (i)) *
65         (pearson[item1][i].rate - itemaverage(i));
66
67
68     for (int i = 0; i < itemnumtable[item2]; i++)
69         countitemB = countitemB + (pearson[item2][i].rate -
            itemaverage(i)) *
70         (pearson[item2][i].rate - itemaverage(i));
71
72     //求最终的pearson系数
73     double pearson = PeaNumerator / (sqrt(countitemA) * sqrt(countitemB))
        ;
74
75     return pearson;
76 }

```

引入 attribute.txt 解决没有对应评分问题

```

1
2 double max1(double r, int itemID, double average, int itemID1) //取所有相似的商品
    中相似度最大的两个
3
4 {
5     double maxsimilar = 0;
6     double x11 = 0;
7     double x22 = 0;

```

```

8         for (int i = 0; i < 19835; i++)//遍历所有用户
9         {
10 int j = insertfind(data0, itemID, i,itemnumtable[i],itemID1);//找一下哪些用户
    买过这两个商品
11
12         if (j != -1)//如果该两个商品被同一个用户购买过
13         {
14             double a = rateitemsimilar(itemID, i);//计算两者的相似度
15             if (a > 0) //如果相似度大于0, 说明是有相似关系的
16             {
17                 x11 += a;
18                 x22 += a * (data0[i][j].rate - bxi(i, data0[i][j].
                    itemID, average));
19             }
20             if (maxsimilar < a)
21                 maxsimilar = a;
22         }
23     }
24     return maxsimilar;
25 }
26
27 void rateitemadd(itemrate** a1, int item1, int item2, double pearson,int
    user)
28 //如果没有足够多和商品i相似的, 就采用属性相似产品评分进行替代
29
30 {
31     ifstream attribute;
32     attribute.open("itemAttribute.txt");//读取商品属性
33
34     for (int i = 0; i < a1[item1][sizeof(a1[item1]) / sizeof(item1)].
        itemID; i++)
35
36     {
37         string s;
38         int itemID00=0;
39         int attribute1, attribute2;
40         double rate=max1(rateitemsimilar(item1, item2),item1,item2,
            pearson);
41         if (rate < 0.2)//如果连与之相似的商品最大的相似度都不到0.2,
42 //说明只通过用户的打分情况, 不能找到很合适的相似商品, 需要借助商品属性来确定
        相似的。
43
44         {
45             while (getline(attribute, s)) //逐行进行读取
46             {
47                 int flag = s.find("|"); //查找该行中第一次出现"|"所在位置
48
49                 string itemstr = s.substr(0, flag);//商品号为|前的数字

```

```

50         itemID00 = stoi(itemstr);
51
52         string attributestr = s.substr(flag + 1, s.length()-1); //后面的
           属性字符串，中间|隔开
53
54         int flag1 = attributestr.find("|"); //第二个|分割两个商品类似
           的属性
55         string attributelstr = attributestr.substr(flag + 1, flag1 -
           1);
56         attributel = stoi(attributelstr);
57         string attribute2str = attributestr.substr(flag1 + 1, s.
           length() - 1);
58         attribute2 = stoi(attribute2str); //将属性号转换为整数
59
60         a1[user][item1].rate = (a1[user][attributel].rate+a1[user][attribute2
           ].rate) / 2;
61
62         }
63     }
64 }
65 attribute.close();
66
67 }

```

### 基于商品相似度协同过滤预测评分

```

1  double bxi(int userID, int itemID, double average)
2  {
3      double bx = useraverage(userID) - average;
4      double bi = itemaverage(itemID) - average;
5      double bxi = average + bx + bi;
6      return bxi;
7  }
8
9
10 //最后产品得分中的加权平均r的值
11 double rxi(int userID, int itemID, double average)
12 {
13
14     double additem12 = 0;
15     double additem123 = 0;
16     int itemID2=pearson[userID][itemID].itemID;
17
18     for (int i = 0; i < itemnum; i++)//遍历所有产品
19     {
20         int j = insertfind(data0, userID,i, itemnumtable[i], itemID);
           //查找另一个相似产品是否也被该用户打分
21
22         if (j != -1)//如果该用户也给这个产品打分

```

```

23         {
24             double a = rateitemsimilar(userID, i); //求两个产品相似度
25             if (a > 0)
26             {
27                 if (a > 0.2) //如果两产品相似，且相关程度pearson系数大于0.2时
28                 {
29                     additem12 += a;
30                     additem123 += a * (data0[i][j].rate - bxi(i, data0[i][j].
                        itemID, average));
31                                     //直接使用相似度计算来加权平均
32                 }
33             }
34             else //否则，引入商品属性来完成推荐
35             {
36                 rateitemadd(data0, itemID, itemID2, average, userID);
37             }
38         }
39     }
40 }
41 }
42 }
43 }
44 }
45
46 //预测评分
47 double ratepredict(int UserID, int ItemID, double average)
48 {
49     //利用相似产品得分情况，预测当前商品会被该用户打多少分。
50     double rate = rxi(UserID, ItemID, average) + bxi(UserID, ItemID,
        average);
51     if (rate < 0) //预测得分小于零，就认为没看过，不打分，置为0
52         rate = 0;
53     return rate;
54 }

```

## RMSE 计算

```

1     ifstream infileresult;
2     infileresult.open("result1.txt");
3
4     string s;
5     double average = 49;
6     double sum = 0;
7     double temp = 0;
8     int count = 0;
9     double rate[120000];
10    while (getline(infileresult, s))
11    {
12        int flag = s.find("|");

```

```
13         if (s == "")//如果没数据就结束
14             break;
15         if (flag == -1)//如果有数据但没有|，说明是打分处
16         {
17             count++;
18             int flag1 = s.find(" ");//在打分处找空格，空格右侧是
19                 得分
20             //!!!!注意 下面是要看flag1!!
21             string s1 = s.substr(flag1 + 1, s.length() - flag1);
22             //从空格位置向后两个开始，
23             //读取长度为字符串长度减去空格及之前的长度
24             rate[count] = stod(s1);
25             //temp = sqrt((rate - average) * (rate - average));
26             sum = sum + rate[count];//从1开始
27         }
28         if (flag != -1)
29         {
30             continue;//找到竖线，说明进入新用户，直接跳过这行，进
31                 入下一行
32         }
33     }
34     temp = sum / count;//平均值
35     cout << temp << endl;
36     sum = 0;//sum用完 要清0!!!
37     for (int i = 1; i <= count; i++)
38     {
39         sum += (rate[i] - temp) * (rate[i] - temp);
40     }
41     sum = sqrt(sum / count);//RMSE
```

八、 实验结果



图 4: 实验命令行输出效果

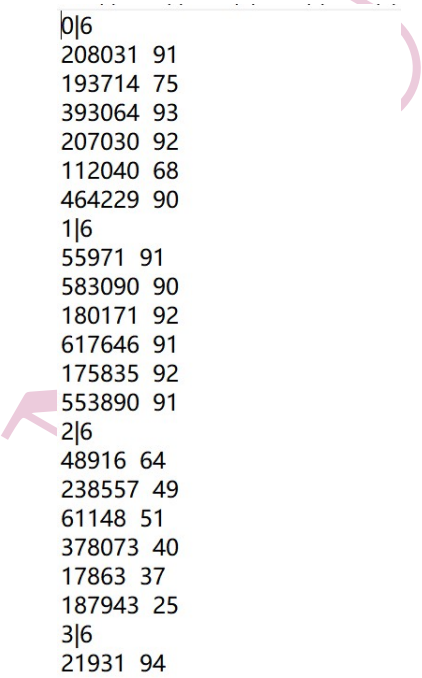


图 5: 输出结果的 txt 中部分截图

九、 结果分析

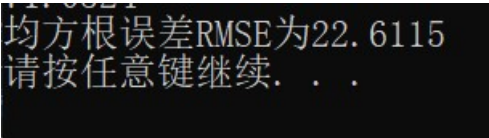


图 6: 基于商品 RMSE

基于商品的相似度时的推荐结果：

1.RMSE:22.6115

2. 时间复杂度：

插值查找：介于完全均匀分布  $O(\log(\log_n))$  到最坏情况  $O(n)$  之间；

几个平均值计算： $O(n)$

pearson 系数代替打分组成新的二维数组： $O(m*n)$  , $m,n$  分别为遍历用户和商品

各步骤分步进行，最后总体进行预测评分，总体的时间复杂度为累乘关系，约为  $O(n^3 \log n)$

3. 空间复杂度： $m$  为用户数， $n$  为产品数

存放初始打分表： $\text{sizeof}(\text{int}) * m * \text{sizeof}(\text{int}) * n$ ;

pearson 系数表： $\text{sizeof}(\text{double}) * n * \text{sizeof}(\text{int}) * m$ ;

预测打分表： $\text{sizeof}(\text{int}) * m * \text{sizeof}(\text{double}) * n$ ;

总体空间消耗，三者相加即可。

320660 90  
66468 0  
304992 0  
551779 0  
322107 0  
567279 0  
29955 0  
531250 0  
543514 0  
138209 0  
156081 0  
526095 0  
197538 70  
25429 0  
605150 0  
347755 0  
506978 0



```
253506 0
320120 0
120709 0
185112 0
321118 0
116230 0
103632 30
133873 0
299915 0
41714 0
168696 0
504556 0
255202 0
484024 0
554315 0
600232 0
503637 0
65831 0
```

图 7: 训练集中出现的大量的 0

RMSE 较大的主要可能是数据集中的 0 较多所导致的。由于，这使得相似程度高的商品（如 pearson 相关系数达到 0.8 甚至更多）的较少，大部分商品的相似程度并不高，甚至一些商品找不到对应的相似产品，在属性表里也找不到类似属性的商品等，这些因素使得预测评分的准确性收到了影响。

十、 对比实验

```
11.8821
均方根误差RMSE为25.4924
请按任意键继续. . .
```

图 8: 基于用户 RMSE

假若换成是基于用户的相似度时的推荐结果：

1.RMSE:25.4924

2. 时间复杂度和空间复杂度：

算法上和基于商品的基本一致，所以空间消耗和时间复杂度在算法的理论上是相近的，但是实际用时上，由于基于商品中采用了 attribute.txt 来类比给出分数，这使得基于商品相似度推荐实际耗时更少。

可见，基于商品的相似度在 RMSE 更小的基础上，减少了耗时，这说明在实际应用中，基于商品的协同过滤的效果更好。

如果对于现实生活中有很多新产品加入的情况时，暂时没有足够的顾客来进行购买，会产生很多的 0，基于用户的相似度计算时将很难找到，甚至于没有类似的用户对这个商品进行打分，因此无法对商品做出有效的预测。

此外，用户的需求不是一成不变的，以前需要购买的商品不意味着未来一定要再次购买，比如一些耐用品不需要经常买，比如以前家里有宠物需要买对应的饲料但现在不再养后就不需要了等等，这些都使得用户相似度并不是完全稳定的。

这时，如果采用基于商品的相似度的协同过滤，将会使用商品的特性来匹配已经进行销售很长时间的类似商品，由于它们已经积累了足够多的用户评分，并且商品的相似度属性在商品制作完成后基本固定了，这些因素使得基于商品的相似度的评分预测表现更好。

## 十一、 总结和收获

1. 在编程实践中，充分学习了解了推荐系统的相关知识和多种算法实现方式，并在此基础上完成了基于商品相似度的协同过滤的方式完成了推荐系统。
2. 对相似度计算部分的多种算法进行学习了解，认识到 cos 相似度，Jaccard 距离等算法存在的不足，最终选取了 Pearson 相关系数完成了相似度计算。
3. 通过对比基于用户和基于商品两种不同协同过滤方式和实验结果，充分认识到了基于商品的相似度表现更好的特点。