南开大学

Nankai University

南　开　大　学

计 算 机 学 院

大数据计算及应用期末报告

推荐系统实验报告

姜奕兵 1710218

皮春莹 1711436

周辰霏 1712991

年级 : 2017 级

专业 : 计算机科学与技术

2020 年 6 月 13 日

摘要

基于 C++ 实现了协同过滤算法，将给定的数据集 train.txt 划分为训练集和验证集，用训练集对模型进行训练，用验证集计算 RMSE，验证模型的预测结果，最后将 test.txt 作为模型输入，得到预测的分数。

关键字：推荐系统; 协同过滤算法

目录

一、 问题描述

使用学过的推荐算法预测 test.txt 文件中用户对电影的评分。在这里我们使用了基于用户的的协同过滤算法。

数据集描述:

- train.txt 文件存储了用户 ID、物品 ID 及对应的评分，用于训练和验证模型，包括 19835 个用户，共 5001513 个评分。
- test.txt 文件存储了用户 ID、物品 ID，用于测试模型，包括与 train.txt 中相同的 19835 个用户，每个用户需要计算对应 6 种物品的评分。
- itemAttribute.txt 用于优化模型，包括 624961 件物品的属性描述。

二、 算法细节

基于用户的协同过滤通过不同用户对物品的评分来评测用户之间的相似性。基于用户的相似性做推荐，即给用户推荐和他兴趣相投的其他用户喜欢的物品。算法实现流程在下面分别进行介绍。

(一) 数据预处理

(二) 划分验证集

定义一些全局变量 user_num、item_num，以及结构体 itemScore，用于存放物品 ID 和分数:

定义

```
//一共有0~19834个用户（19835）
const int user_num = 19835;
//存放每个用户的物品数目
int item_num[user_num];

struct itemScore
{
  int item_id;
  double item_sco;
};

//开19835行（用户）、不固定列的矩阵score_matrix，存放txt读入的数据
itemScore **score_matrix = new itemScore *[user_num];
```

划分

```
//分界线
int FineLine[user_num];
void div_read2buffer()
{
  //打开train.txt
  ifstream infile1;
```

```
    infile1.open("train.txt");
    //若失败,则输出错误消息,并终止程序运行
    assert(infile1.is_open());

    string s; //存放读取行数据
    string blank = " ";
    string line = "|";
    int UserID = -1; //user递增, 可以直接先赋值为-1

    int NumOfMovie = 0; //目前的用户所看过的物品数目
    int clock = 0;

    while (getline(infile1, s))//逐行读取数据
    {
      int LocLine = s.find(line); //查找该行中"|"所在位置
      if (s == "")
        break;

      if (LocLine != -1) //"|"存在, 新的用户, 读取他看过的物品数
      {
        UserID++;
        clock = 0;

        int len = s.length();
        string ss = s.substr(LocLine + 1, len - LocLine);
        NumOfMovie = stoi(ss); //该用户看过的物品数目

        item_num[UserID] = NumOfMovie;

        //动态初始化data数组中该UserID行
        score_matrix[UserID] = new itemScore[NumOfMovie];
      }

      else //"|"不存在, 还是上一个用户,现在s存的是物品号和评分
      {
        int LocBlack = s.find(blank); //空格所在位置
        string item_id = s.substr(0, LocBlack);
        int leng = s.length();
        string sco = s.substr(LocBlack + 2, leng - LocBlack);

        double score = stof(sco);//当前用户对目前行的物品的评分
        int MovieID = stoi(item_id);//当前物品编号

        struct itemScore MS = { MovieID,score };
        score_matrix[UserID][clock] = MS;
        clock++;
      }
    }
```

```
    infile1.close();
    cout << "traindata.txt已读取完毕" << endl;
}
```

然后调用 div_write2file() 根据上面统计的数量及存储的信息，将数据按 15% 的比例划分为两部分。

统计信息

```
void div_write2file()
{

  for (int i = 0; i < user_num; i++)
    FineLine[i] = int(item_num[i] * 0.85);
  cout << "Calculate FineLine." << endl;

  fstream file1;
  file1.open("train1.txt", std::ios::out | std::ios::app);
  cout << "opened train1.txt" << endl;
  for (int i = 0; i < user_num; i++)
  {
    for (int j = 0; j < FineLine[i]; j++)
    {
      if (!j)
      {
        file1 << i << "|" << FineLine[i] << endl;
        file1 << score_matrix[i][j].item_id << "  "
         << score_matrix[i][j].item_sco << endl;
      }
      else
        file1 << score_matrix[i][j].item_id << "  "
        << score_matrix[i][j].item_sco << endl;
    }
  }
  file1.close();

  file1.open("verify.txt", std::ios::out | std::ios::app);
  cout << "opened verify.txt" << endl;
  for (int i = 0; i < user_num; i++)
  {
    int k = item_num[i] - FineLine[i];
    for (int j = FineLine[i]; j < item_num[i]; j++)
    {
      if (j == FineLine[i])
      {
        file1 << i << "|" << k << endl;
        file1 << score_matrix[i][j].item_id << "  "
        << score_matrix[i][j].item_sco << endl;
      }
      else
```

```
            file1 << score_matrix[i][j].item_id << "  "
            << score_matrix[i][j].item_sco << endl;
        }
      }
      file1.close();
    }
```

1. 提取 txt 文本信息

为了便于计算，增加全局变量（如下所示），item_num 记录了 train1.txt 每个用户的物品数目，buffer 记录了 test.txt 中的用户号及对应物品号，ver_item_num 记录了 verify.txt 每个用户的物品数目，ver_buffer 记录了 verify.txt 中的用户号及对应物品号，itemScore 的定义在上面已经提过，score_matrix 用于记录 train1.txt 中的用户号、物品号及分数.

变量设定

```
//存放train1.txt每个用户的物品数目
int item_num[user_num];
int buffer[user_num][6];
//存放verify.txt每个用户的物品数目
int ver_item_num[user_num];
vector<vector<int>>ver_buffer;
itemScore **score_matrix = new itemScore *[user_num];
```

score_matrix 计算过程由 load_score() 函数实现，代码如下：

分数矩阵

```
//读取数据到score_matrix矩阵
void load_score()
{
        //打开train1.txt
        ifstream infile1;
        infile1.open("train.txt");
        assert(infile1.is_open());    //若失败,则输出错误消息,并终止程序运行


        string s; //存放读取行数据
        string blank = " ";
        string line = "|";
        int user_id = -1; //user递增, 可以直接先赋值为-1

        int item_count = 0; //目前的用户的物品数目
        int clock = 0;

        while (getline(infile1, s))//逐行读取数据
        {
                int line_pos = s.find(line); //查找该行中"|"所在位置
                if (s == "")
                        break;
```

```
23
24                  if (line_pos != -1) //"|"存在，新的用户，读取他的物品数
25                  {
26                          user_id++;
27                          clock = 0;
28
29                          int len = s.length();
30                          string ss = s.substr(line_pos + 1, len - line_pos);
31                          item_count = stoi(ss); //该用户的物品数目
32
33                          item_num[user_id] = item_count;
34
35                          //动态初始化data数组中该user_id行
36                          score_matrix[user_id] = new itemScore[item_count];
37                  }
38
39                  else //"|"不存在，还是上一个用户,现在s存的是物品号和评分
40                  {
41                          int blank_pos = s.find(blank); //空格所在位置
42                          string item_id = s.substr(0, blank_pos);
43                          int leng = s.length();
44                          string sco = s.substr(blank_pos + 2, leng - blank_pos
                                );
45
46                          double score = stof(sco);//当前用户对目前行的物品的评
                                分
47                          int itemID = stoi(item_id);//当前物品编号
48
49                          struct itemScore MS = { itemID,score };
50
51                          //将该行数据存入效用矩阵对应位置
52                          score_matrix[user_id][clock] = MS;
53                          clock++;
54                  }
55          }
56          infile1.close();
57          cout << "train1.txt已读取完毕" << endl;
58  }
```

（三） 计算用户的相似度

这里采用的是皮尔逊相关系数（Pearson Correlation Coefficient），它是余弦相似度在维度值缺失情况下的一种改进。

用户 x 和用户 y 之间的 Pearson 相似度的计算公式如图1所示。

Pearson 系数的取值范围和意义如图2所示。

由于 Pearson 相似度的分子部分需要多次地找两个用户共同的物品 ID，为了更快速地进行二

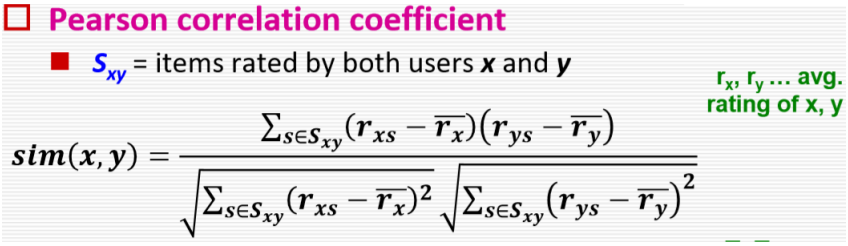


图 1: Pearson 相似度的计算公式

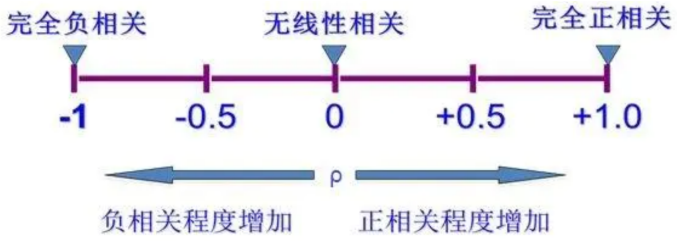


图 2: Pearson 系数的取值范围和意义

分查找，先将 score_matrix 按照物品按编号从小到大升序排列:

二分查找共同 ID

```
//将每个用户的物品按编号从小到大升序排列
void sort_item_id()
{
        for (int i = 0; i < user_num; i++)
        {
                sort(score_matrix[i], score_matrix[i] + item_num[i], compare)
                    ;
        }
}

//二分查找
int binary_search(itemScore **R, int uid, int n, int K)
{
        //在有序表R[0..n-1]中进行二分查找，成功时返回结点的位置，失败时返回-1
        int left = 0, right = n - 1, mid;       //置当前查找区间上、下界的初值
        while (left <= right)
        {
                if (R[uid][left].item_id == K)
                        return left;
                if (R[uid][right].item_id == K)
                        return right;           //当前查找区间R[left..right]非
                            空
                mid = left + ((right - left) / 2);
                //使用(left+right)/2会有整数溢出的问题
                if (R[uid][mid].item_id == K)
                        return mid;             //查找成功返回
                if (R[uid][mid].item_id<K)
```

```
                left = mid + 1;                    //继续在R[mid+1..right]
                    中查找
            else
                right = mid - 1;                   //继续在R[left..mid-1]中
                    查找
        }
        if (left>right)
            return -1;//当left>right时表示所查找区间内没有结果, 查找失败
}
```

编程实现 Pearson 相似度的计算公式:

根据 Pearson 相似度的计算公式,里面需要计算某一用户的所有评分的平均值,编写 get_avg_user(int user_id) 来实现这一功能:

计算分数

```
//求用户x对所有产品打分的平均值
double get_avg_user(int user_id)
{
        double avg = 0;
        double Sum = 0;
        for (int i = 0; i < item_num[user_id]; i++)
        {
            Sum = Sum + score_matrix[user_id][i].item_sco;
        }
        avg = Sum / item_num[user_id];
        return avg;
}
```

为了便于计算任意两个用户之间的 Pearson 相似度，声明一个全局变量 pearson_matrix 系数矩阵，pearson_matrix[i][j] 表示第 i 个用户的第 j 个评分的物品 ID 以及评分减去用户 i 打分的平均值。由 get_pearson_matrix() 函数来计算 pearson_matrix。

pearson 矩阵

```
itemScore **pearson_matrix = new itemScore *[user_num];
//将score_matrix矩阵的数据转成pearson_matrix系数矩阵
void get_pearson_matrix()
{
        for (int i = 0; i < user_num; i++)
        {//外层遍历所有用户
            double avguser = get_avg_user(i);
            pearson_matrix[i] = new itemScore[item_num[i]];

            for (int j = 0; j < item_num[i]; j++)
            {//内层遍历当前用户看过的所有物品
                pearson_matrix[i][j].item_id = score_matrix[i][j].
                    item_id;
      pearson_matrix[i][j].item_sco =
      score_matrix[i][j].item_sco - avguser;
            }
```

```
        }
}
```

求用户 A 和用户 B 的 pearson 相似度时，分子部分需要对 AB 都有的物品所有物品编号 s 进行遍历，求出用户 A 对物品 s 的评分与用户 B 对物品 s 的评分的乘积之和，存放在 pear_sum 变量中。分母部分需要求用户 A、B 对所有物品的评分各自平方求和开根号在相乘。最后得出用户 A 和用户 B 的 pearson 相似度，代码如下：

求解用户相似度

```
// 求用户A和用户B的pearson相似度
double pear_AB(int UserA, int UserB)
{
        //求出AB都的物品a的编号
        //计算：用户A对物品a的评分*用户B对物品a的评分，存在multi_res
        double multi_res[10000];
        int clock = 0;

        if (item_num[UserA] <= item_num[UserB])
        {
                for (int i = 0; i < item_num[UserA]; i++) //外层遍历A用户的物品
                {
      int j = binary_search(pearson_matrix, UserB,
      item_num[UserB], pearson_matrix[UserA][i].item_id);
                        if (j != -1)
                        {
        multi_res[clock] = pearson_matrix[UserA][i].item_sco
        * pearson_matrix[UserB][j].item_sco;
                                clock++;
                        }
                }
        }

        if (item_num[UserA] > item_num[UserB])
        {
                for (int i = 0; i < item_num[UserB]; i++) //外层遍历B用户的物品
                {
      int j = binary_search(pearson_matrix, UserA,
      item_num[UserA], pearson_matrix[UserB][i].item_id);
                        if (j != -1)
                        {
        multi_res[clock] = pearson_matrix[UserB][i].item_sco
        * pearson_matrix[UserA][j].item_sco;
                                clock++;
                        }
                }
        }
```

```
38
39          //将multi_res数组里的各数字求和，作为pearson系数的分子
40          double pear_sum = 0;
41          for (int i = 0; i < clock; i++)
42                  pear_sum = pear_sum + multi_res[i];
43
44          //求pearson系数的分母
45
46          //求用户A对所有物品的评分各自平方求和开根号
47          double sum_userA = 0;
48          for (int i = 0; i < item_num[UserA]; i++)
49      sum_userA = sum_userA + pearson_matrix[UserA][i].item_sco
50       * pearson_matrix[UserA][i].item_sco;
51          double sqrt_userA = sqrt(sum_userA);
52
53          //求用户B对所有物品的评分各自平方求和开根号
54          double sum_userB = 0;
55          for (int i = 0; i < item_num[UserB]; i++)
56      sum_userB = sum_userB + pearson_matrix[UserB][i].item_sco
57       * pearson_matrix[UserB][i].item_sco;
58          double sqrt_userB = sqrt(sum_userB);
59
60
61          //求pearson系数
62          double pearson = pear_sum / (sqrt_userA * sqrt_userB);
63
64          return pearson;
65  }
```

（四） 预测最终分数

需要预测用户 x 对物品 i 的打分，如果仅从用户之间的相关性考虑，这个分数可以记为 r_{ij}，计算公式如图3所示。

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

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

图 3: 计算 r_{ij} 的公式

但仅依靠用户相关性预测出来的分数，没有考虑到物品的特点以及特定用户打分的习惯，因此需要作出调整，加入基准，调节预测分数，如图4所示，将公式的加法第二个操作数记为修正后的 r_{ij}，把这一最终的分数记为 End_{xi}。下面分 b_{ij} 和 r_{ij} 两部分来计算 End_{xi}。

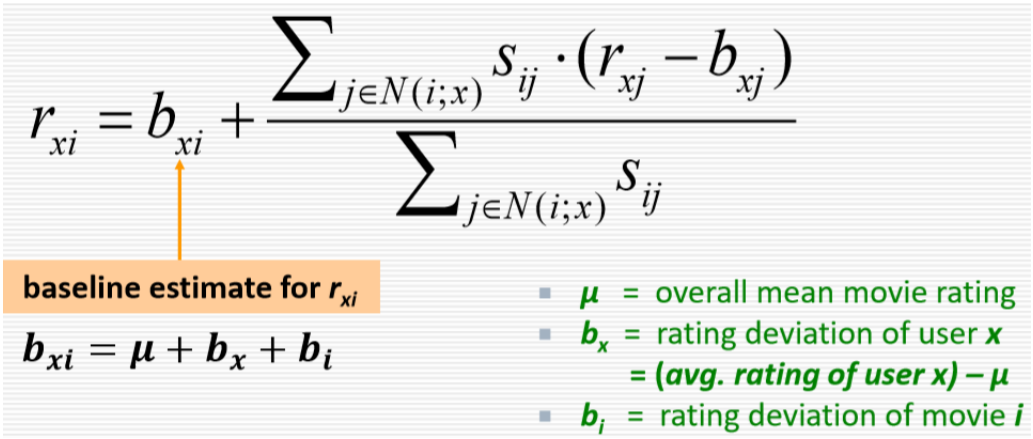


图 4: 计算最终分数的公式

1. 计算 baseline estimate

为了计算总体平均值 μ，编写函数 get_avg_all()，将计算结果保存在变量 u 中：

计算均值

```
//求总体平均值
double get_avg_all()
{
  double avg = 0;
  double Sum = 0;
  int count = 0;

  for (int i = 0; i < user_num; i++)
  {
    for (int j = 0; j < item_num[i]; j++)
    {
      Sum = Sum + score_matrix[i][j].item_sco;
      count++;
    }
  }

  avg = Sum / count;
  return avg;
}
double u = get_avg_all();
```

公式中需要求某一用户评分的平均值，编写函数 get_avg_item(int it_id) 来计算：

计算产品全部均值

```
//求产品i的所有评分的平均值
double get_avg_item(int it_id)
{
    double avg = 0;
    double Sum = 0;
    int count = 0;
```

```
7
8           for (int i = 0; i < user_num; i++) //外层遍历所有用户
9           {
10                  int j = binary_search(score_matrix, i, item_num[i], it_id);
11                  //for (int j = 0; j < item_num[i]; j++) //内层遍历每一用户看
                        过的所有物品
12                  //{
13                  if (j != -1)
14                  {
15                          Sum = Sum + score_matrix[i][j].item_sco;
16                          count++;
17                  }
18                  //}
19          }
20
21          avg = Sum / count;
22          return avg;
23  }
```

计算 b_{xi}:

计算 bxi 均值

```
1   //最后产品得分中的bxi的值
2   double bxi(int user_id, int it_id, double u)
3   {
4           double bx = get_avg_user(user_id) - u;
5           double bi = get_avg_item(it_id) - u;
6
7           double bxi = u + bx + bi;
8
9           return bxi;
10  }
```

2. 计算修正后的 r_{xi}

现在的 r_{xi}，分母是所有与目标用户有关、且看过目标物品的其他用户，与目标用户的相似度之和，分子是看过目标物品的其他用户与目标用户的相似度，与看过目标物品的其他用户对目标物品的打分与相应 baseline 之差的乘积之和。计算过程中需要对无关联或关联很小的用户进行特殊处理，避免出现分母为 0 的情况。代码如下:

rxi

```
1   //最后产品得分中的rxi的值
2   double rxi(int user_id, int it_id, double u)
3   {
4       //求所有与目标用户有关、且看过目标物品的其他用户，与目标用户的相似度之和
5       //求(看过目标物品的其他用户与目标用户的相似度)*
6       (看过目标物品的其他用户对目标物品的打分)之和
7       double sum_sij = 0;
```

```
    double sum_sij_rjx = 0;

    for (int i = 0; i < user_num; i++)//外层遍历用户
    {
      int j = binary_search(score_matrix, i, item_num[i], it_id);

      if (j != -1)//如果该用户也看过目标物品
      {
        double a = pear_AB(user_id, i);
        if (a > 0) //如果两用户是相似用户
        {
          sum_sij += a;
          sum_sij_rjx += a * (score_matrix[i][j].item_sco
          - bxi(i, score_matrix[i][j].item_id, u));
        }
      }
    }

    if (sum_sij != 0 && sum_sij_rjx != 0)
    {
      double rix = sum_sij_rjx / sum_sij;
      return rix;
    }

    else
    {
      double rix = 1000;
      return rix;
    }
  }
```

3. 最终预测分数 End_{xi}

最终得到的预测分数即为 b_{xi} 与 r_{xi} 求和的结果，但要对上面 r_{xi} 说到的特殊情况进行判断和处理。

最终分数

```
  //最终产品的终极得分
  double End(int user_id, int it_id, double u)
  {
    double End = 0;
    double rxi1 = rxi(user_id, it_id, u);

    if (rxi1 == 1000) //如果目标用户的相似用户都没有看过目标物品
    {
      //以目标用户打分平均值作为结果
      End = get_avg_user(user_id);
    }
```

```
    else
    {
      End = rxi1 + bxi(user_id, it_id, u);
      if (End > 100)
        End = 100;
      if (End < 0)
        End = 0;
    }
    cout << "user_id = " << user_id << ", it_id = " << it_id << ", End = " <<
         End << endl;
    return End;
  }
```

4. 对所有数据进行测试

需要进行预测的是验证集 verify.txt 和测试集 test.txt，调用 read2buffer() 将需要测试的数据读入，分别存在 vec_buffer 和 buffer 数组，并在 ver_item_num 数组中存放 verify.txt 每个用户的物品数目：

获取数据

```
//把需要测试的数据（test.txt和verify）读入
void read2buffer()
{
  ifstream infile;
  infile.open("test.txt");
  assert(infile.is_open());

  string s; //存放读取行数据
  string line = "|";
  int user_id = -1; //user递增，可以直接先赋值为-1
  int Row = -1;
  int i = 0;

  while (getline(infile, s))
  {
    int line_pos = s.find(line); //查找该行中"|"所在位置
    if (s == "")
      break;
    if (line_pos != -1)
    {
      Row++;
      i = 0;
    }
    else
    {
      buffer[Row][i] = stoi(s);
      i++;
    }
```

```
    }
    infile.close();

    infile.open("verify.txt");
    assert(infile.is_open());
    string blank = " ";
    user_id = -1; //user递增, 可以直接先赋值为-1
    Row = -1;

    while (getline(infile, s))
    {
      int line_pos = s.find(line); //查找该行中"|"所在位置
      if (s == "")
        break;
      if (line_pos != -1)
      {
        ver_buffer.push_back({});
        Row++;
        string ss = s.substr(line_pos + 1, s.length() - line_pos);
        //该用户的物品数目
        ver_item_num[Row] = stoi(ss);
      }
      else
      {
        int blank_pos = s.find(blank); //空格所在位置
        string item_id = s.substr(0, blank_pos);
        ver_buffer[Row].push_back(stoi(item_id));
      }
    }
    infile.close();
  }
```

最后便是对每条数据调用End()函数计算分数,再按照格式规定写入文本文件,由write2file(double u)函数实现:

计算全部分数

```
//把读入的数据进行测试, 并写入结果文档 (result.txt)
void write2file(double u)
{
  fstream file1;
  file1.open("test_result.txt", std::ios::out | std::ios::app);
  for (int i = 0; i < user_num; i++)
  {
    for (int j = 0; j < 6; j++)
    {
      if (!j)
      {
        file1 << i << "|" << "6" << endl;
```

```
            file1 << buffer[i][j] << "  " << int(End(i, buffer[i][j], u)) <<
                endl;
          }
          else
            file1 << buffer[i][j] << "  " << int(End(i, buffer[i][j], u)) <<
                endl;
        }
      }
      file1.close();

      //计算验证集分数
      file1.open("verify_result.txt", std::ios::out | std::ios::app);
      for (int i = 0; i < user_num; i++)
      {
        for (int j = 0; j < ver_item_num[i]; j++)
        {
          if (!j)
          {
            file1 << i << "|" << ver_item_num[i] << endl;
            file1 << ver_buffer[i][j] << "  " << int(End(i, ver_buffer[i][j], u
                )) << endl;
          }
          else
            file1 << ver_buffer[i][j] << "  " << int(End(i, ver_buffer[i][j], u
                )) << endl;
        }
      }
      file1.close();
    }
```

主函数

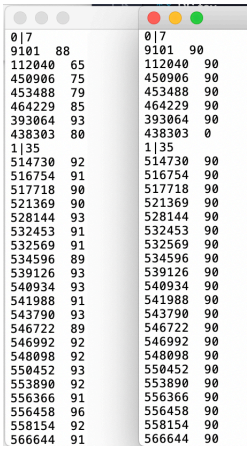
```
int main()
{
        load_score();
        read2buffer();
        double u = get_avg_all();
        sort_item_id();
        get_pearson_matrix();
        write2file(u);

        return 0;
}
```

三、 实验结果

在实验过程中我们对验证集测试结果与原结果对比最终得到 rmse 结果为 28.591，说明整体结果较为可靠。训练时间整体较长，计算验证集用时 6 天，由于我们没有渠道使用大型云服务器

只能使用小型云主机故耗时较长，而测试集结果用时 2 天，整体上和测试数据的大小成正比。内存占用比例约为 30%-40%.



0|7	0|7
9101 88	9101 90
112040 65	112040 90
450906 75	450906 90
453488 79	453488 90
464229 85	464229 90
393064 93	393064 90
438303 80	438303 0
1|35	1|35
514730 92	514730 90
516754 91	516754 90
517718 90	517718 90
521369 90	521369 90
528144 93	528144 90
532453 91	532453 90
532569 91	532569 90
534596 89	534596 90
539126 93	539126 90
540934 93	540934 90
541988 91	541988 90
543790 93	543790 90
546722 89	546722 90
546992 92	546992 90
548098 92	548098 90
550452 93	550452 90
553890 92	553890 90
556366 91	556366 90
556458 96	556458 90
558154 92	558154 90
566644 91	566644 90

图 5: 验证结果与原数据对比

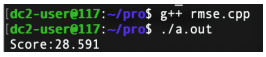


图 6: rmse 测试结果

四、 实验分析

计算 RMSE

```
//默认验证集的原有分数和预测分数的位置是对应的
vector<int> predict, rating;

void getScore()
{
  ifstream infile1;
  infile1.open("verify.txt");
  assert(infile1.is_open());    //若失败,则输出错误消息,并终止程序运行
  string s; //存放读取行数据
  string blank = " ";
  string line = "|";
  while (getline(infile1, s))//逐行读取数据
  {
    int LocLine = s.find(line); //查找该行中"|"所在位置
    if (s == "")
      break;

    if (LocLine == -1)
    {
      int LocBlack = s.find(blank); //空格所在位置
      string sco = s.substr(LocBlack + 2, s.length() - LocBlack);
```

```
            double score = stof(sco);
            rating.push_back(score);
        }
    }
    infile1.close();

    infile1.open("verify_result.txt");
    assert(infile1.is_open());    //若失败,则输出错误消息,并终止程序运行
    while (getline(infile1, s))//逐行读取数据
    {
        int LocLine = s.find(line); //查找该行中"|"所在位置
        if (s == "")
            break;
        if (LocLine == -1)
        {
            int LocBlack = s.find(blank); //空格所在位置
            string sco = s.substr(LocBlack + 2, s.length() - LocBlack);
            double score = stof(sco);
            predict.push_back(score);
        }
    }
    infile1.close();
}
double calcuRMSE()
{
    getScore();
    double res = 0;
    int N = predict.size();
    for (int i = 0; i < N; i++)
    {
        res += (pow((predict[i] - rating[i]), 2));
    }
    res = pow(res / N, 0.5);
    return res;
}
```


参考文献

[1] 推荐系统 1-2