推荐系统之协同过滤的原理及C++实现

1. 引言

假如你经营着一家网店，里面卖各种商品（Items），有很多用户在你的店里面买过东西，并对买过的Items进行了评分，我们称之为历史信息，现在为了提高销售量，必须主动向用户推销产品，所以关键是要判断出用户除了已经买过的商品之外还会喜欢哪些商品，这就需要利用用户购买商品过程产生的历史信息。协同过滤通常分为基于用户的协同过滤和基于商品的协同过滤。

- 基于用户的协同过滤：利用用户之间的相似度进行推荐
- 基于物品的协同过滤：利用物品之间的相似度进行推荐

2. 原理

关于协同过滤的原理网上到处都有，思想很简单，这里就不赘述，下面举一个简单的实例来说明基于用户的协同过滤：

```
1  5 3 0 4 0 0 1 2 4 4
2  3 1 2 0 0 2 0 0 1 2
3  4 0 1 2 1 0 1 2 4 0
4  0 2 3 0 0 4 2 1 1 2
5  1 3 4 1 2 0 0 0 0 2
6  2 0 0 0 2 3 5 1 4 0
7  4 1 1 0 2 0 1 0 0 3
8  1 2 0 3 0 0 2 1 0 3
9  3 1 0 3 0 4 0 0 1 0
10 2 1 3 1 0 1 0 0 2 1
```

上面每一行代表一个用户，每一列代表一个商品，比如第2行第一列的3表示用户2对商品1的评分为3，0代表对应的用户还没有购买过该商品，现在想预测用户2对商品4的评分：

- 找出对商品4评过分的用户：用户1, 3, 5, 8, 9, 10，评分分别为：4， 2， 1， 3， 3， 1
- 分别计算用户2与用户1, 3, 5, 8, 9, 10之间的相似度，相似度的计算方法有很多，常用的分为3类：欧氏距离，余弦相似度，皮尔逊相关系数，网上很容易查到，这里以常用的余弦相关系数说明：

要计算用户2与用户1之间的相似度，首先找到二者都评过分的商品为：商品1， 2， 9， 10，用户1对这4个商品的评分向量为r1=[5 3 4 4]，用户2对这4个商品评分向量为r2=[3 1 1 2]；所谓余弦相似度就是利用两个向量之间夹角的余弦值来衡量两个向量之间的相似度，显然夹角越小，余弦值就越大，两个向量就越靠近，即二者越相似，于是用户2和用户1之间的相似度就为sim2_1=(5*3 + 3*1 + 4*1 + 4*2)/（||r1|| * ||r2||）= 0.953， 其中||r||代表向量r的模长或者2范数，类似地分别计算出用户2与用户3 5 8 9 10之间的sim2_3, sim2_5, sim2_8, sim2_9, sim2_10

- 最后利用相似度加权得到用户2对商品4的预测评分：predict = 4*sim2_1 + 2*sim2_3 + 1*sim2_5 + 3*sim2_8 + 3*sim2_9 + 1*sim2_10
- 基于物品相似度就是与上面计算过程几乎相似，只是计算的是物品之间的相似度

3. 实现

关于Matlab的实现可以参考：http://blog.csdn.net/google19890102/article/details/28112091，这里我用C++实现，并用movielens.rar进行测试，这个数据集是包括训练集和测试集，已经处理成矩阵形式。

- 首先给出读取训练数据和保存预测结果的头文件

```
#ifndef LOAD_H
#define LOAD_H
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

using namespace std;

template <typename T>
vector<vector<T> > txtRead(string FilePath, int row, int col)
{
    ifstream input(FilePath);
    if (!input.is_open())
    {
        cerr << "File is not existing, check the path: \n" <<  FilePath << endl;
        exit(1);
    }
    vector<vector<T> > data(row, vector<T>(col, 0));
    for (int i = 0; i < row; ++i)
    {
        for (int j = 0; j < col; ++j)
        {
            input >> data[i][j];
        }
    }
    return data;
}

template<typename T>
void txtWrite(vector<vector<T> > Matrix, string dest)
{
    ofstream output(dest);
    vector<vector<T> >::size_type row = Matrix.size();
    vector<T>::size_type col = Matrix[0].size();
    for (vector<vector<T> >::size_type i = 0; i < row; ++i)
    {
        for (vector<T>::size_type j = 0; j < col; ++j)
        {
```

```
            output << Matrix[i][j];
        }
        output << endl;
    }
}
#endif
```

- 再给出评价预测好坏的计算RMSE的头文件

```
#ifndef EVALUATE_H
#define EVALUATE_H
#include <cmath>
#include <vector>

double ComputeRMSE(vector<vector<double> > predict, vector<vector<double> > test)
{
    int Counter = 0;
    double sum = 0;
    for (vector<vector<double> >::size_type i = 0; i < test.size(); ++i)
    {
        for (vector<double>::size_type j = 0; j < test[0].size(); ++j)
        {
            if (predict[i][j] && test[i][j])
            {
                ++Counter;
                sum += pow((test[i][j] - predict[i][j]), 2);
            }
        }
    }
    return sqrt(sum / Counter);
}

#endif
```

- 最后给出主函数：

```
#include "load.h"
#include "evaluate.h"
#include <vector>
#include <string>
```

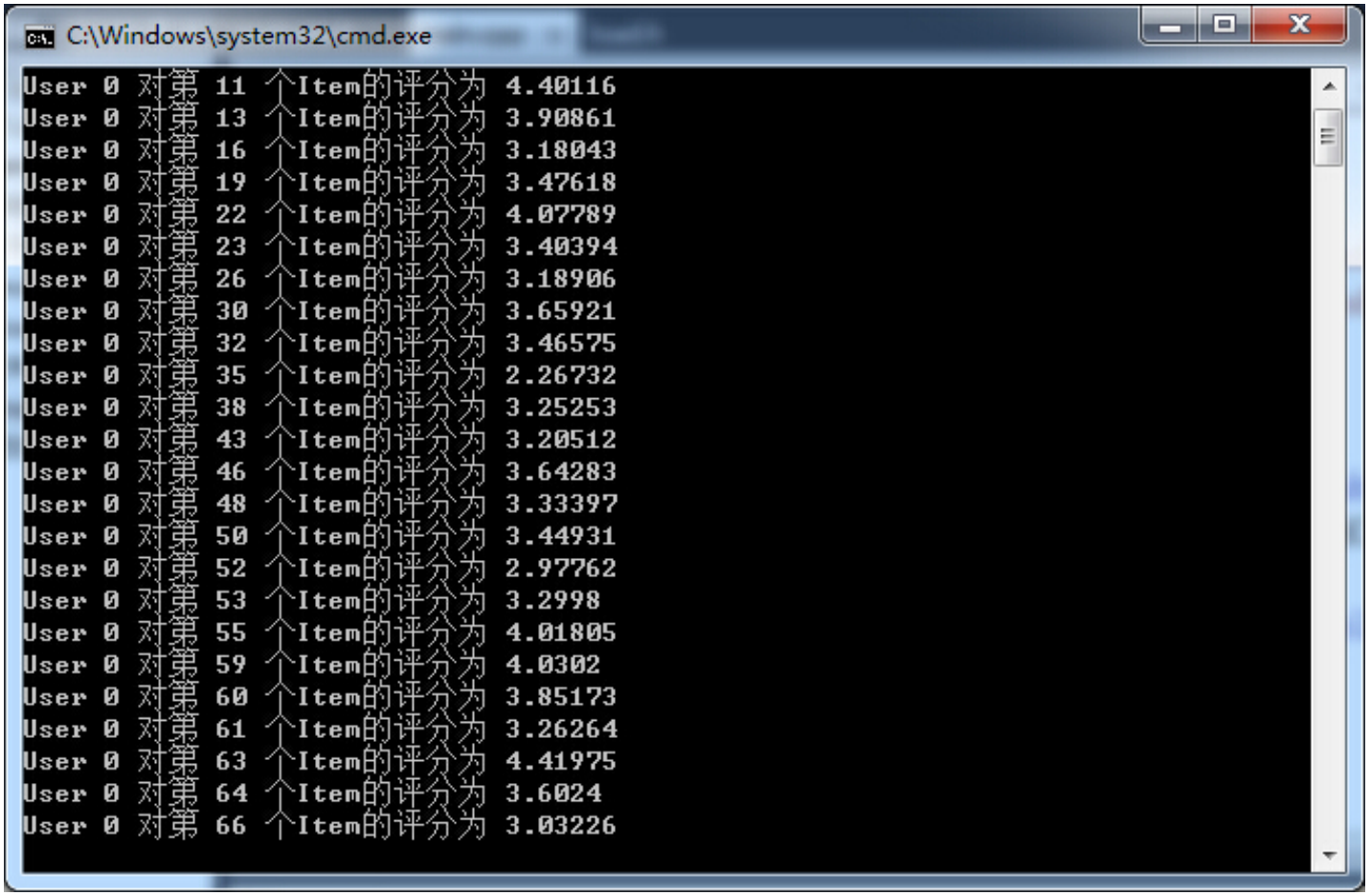
```cpp
#include <cmath>
#include <assert.h>
using namespace std;

double norm(vector<double> A)
{
    double res = 0;
    for(vector<double>::size_type i = 0; i < A.size(); ++i)
    {
        res += pow(A[i], 2);
    }
    return sqrt(res);
}

double InnerProduct(vector<double> A, vector<double> B)
{
    double res = 0;
    for(vector<double>::size_type i = 0; i < A.size(); ++i)
    {
        res += A[i] * B[i];
    }
    return res;
}

double ComputeSim(vector<double> A, vector<double> B, int method)
{
    switch (method)
    {
    case 0://欧氏距离
        {
            vector<double> C;
            for(vector<double>::size_type i = 0; i < A.size(); ++i)
            {
                C.push_back((A[i] - B[i]));
            }
            return 1 / (1 + norm(C));
            break;
        }
    case 1://皮尔逊相关系数
        {
            double A_mean = 0;
            double B_mean = 0;
            for(vector<double>::size_type i = 0; i < A.size(); ++i)
            {
                A_mean += A[i];
```

```
                B_mean += B[i];
            }
            A_mean /= A.size();
            B_mean /= B.size();
            vector<double> C(A);
            vector<double> D(B);
            for(vector<double>::size_type i = 0; i < A.size(); ++i)
            {
                C[i] = A[i] - A_mean;
                D[i] = B[i] - B_mean;
            }
            assert(norm(C) * norm(D));
            return InnerProduct(C,D) / (norm(C) * norm(D));
            break;
        }
    case 2:
        {
            assert(norm(A) * norm(B));
            return InnerProduct(A,B) / (norm(A) * norm(B));
            break;
        }
    default:
        {
            cout << " Choose method:" << endl;
            cout << "0:欧氏距离\n1:皮尔逊相关系数\n2:余弦相似度\n";
            return -1;
        }
    }

}

void FindCommon(vector<double> A, vector<double> B, vector<double> &C, vector<double> &D)
{
    for(vector<double>::size_type i = 0; i < A.size(); ++i)
    {
        if (A[i] && B[i])
        {
            C.push_back(A[i]);
            D.push_back(B[i]);
        }
    }
}


```

```
vector<vector<double> > UserBasedCF(vector<vector<double> > train, int usersNum, int itemsNum)
{
    vector<vector<double> > predict(usersNum, vector<double>(itemsNum, 0));
    for (int i = 0; i < usersNum; ++i) //对每个用户进行预测
    {
        //找出user i未评分的item j，预测user i 对item j的评分
        for (int j = 0; j < itemsNum; ++j)
        {


            if (train[i][j])
                continue;
            //如果item j没有被user i评过分,找出对 item j评过分的用户
            else
            {
                vector<double> sim;
                vector<double> historyScores;
                for (int k = 0; k < usersNum; ++k)
                {
                    //如果user k对item j 评过分，计算user k与user i的相似度

                    if (train[k][j])//找出对item j 评过分的user k
                    {
                        // 为了计算user k与user i的相似度，必须找出二者共同评过分的items
                        // 把二者对共同评过分的items的评分分别存储在两个vector中
                        vector<double> commonA,commonB;
                    FindCommon(train[i], train[k], commonA, commonB);
                        //如果二者存在共同评过分的items,计算相似度
                        if (!commonA.empty())
                        {
                            sim.push_back(ComputeSim(commonA, commonB, 2));
                            // 把user k对item j 的历史评分记录下来
                           historyScores.push_back(train[k][j]);
                        }
                    }

                }
                // 计算出所有与user i存在共同评过分的items的users与user i之间的相似度，
                // 保存在sim中,这些users对目标items j(即user i没有评过分)的历史评分记
                // 录在historyScores中。利用这两个vector，计算出相似度加权平均分作为预
                // 测user i对item j的评分
                double SimSum = 0;
                if (!sim.empty())
                {
```

```
138                     for(vector<double>::size_type m = 0; m < sim.size(); ++m)
139                     {
140                         SimSum += sim[m];
141                     }
142                 predict[i][j] = InnerProduct(sim, historyScores) / (SimSum);
143                 cout << "User "<< i << " 对第 " << j << " 个Item的评分为 " <<
predict[i][j] << endl;
144                 }
145             }
146         }
147     }
148     return predict;
149 }
150
151 int main()
152 {
153     string FilePath1("E:\\Matlab code\\recommendation
system\\data\\movielens\\train.txt");
154     string FilePath2("E:\\Matlab code\\recommendation
system\\data\\movielens\\test.txt");
155
156     int row = 943;
157     int col = 1682;
158     vector<vector<double> > train = txtRead<double>(FilePath1, row, col);
159     vector<vector<double> > predict = UserBasedCF(train, row, col);
160     txtWrite(predict, "predict.txt");
161     vector<vector<double> > test = txtRead<double>(FilePath2, 462, 1591);
162     double rmse = ComputeRMSE(predict,test);
163     cout << "RMSE is " << rmse <<endl;
164     return 0;
165 }
```

4. 运行



由于程序没有优化，循环比较多，时间比较长，程序没写好，如果读者有兴趣帮我优化，请联系我，多谢，欢迎有兴趣的可以自己构造一个小点的数据集试一试，以前我用这个数据在Matlab中运行的RMSE是1左右，所以如果读者运行结果得到测试集上的RMSE是0.9-1.3之间问题应该不大，如果偏离太多，程序设计可能就有问题。