并行程序设计实践GPU项目一

项目名称：CUDA设计一个KNN分类算法

完成成员名单：

班级、姓名和学号:计算机2018-5,尹浩男,201801051827

班级、姓名和学号:计算机2018-5, 乔丹 ,201801050516

班级、姓名和学号:计算机2018-5,贾志愚,201801020607

班级、姓名和学号:计算机2018-5,刘羽翯,201801120916

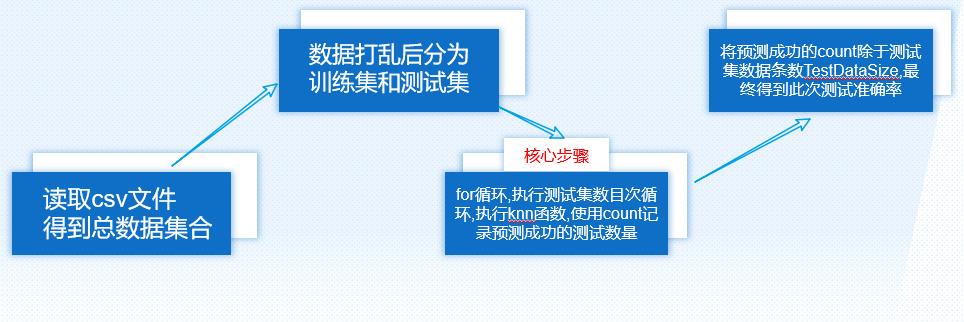
时间：2020年10月30日

## 1.1 项目要求

使用cuda设计一个KNN分类算法(K近邻算法)程序。

## 1.2 项目介绍

### 1.2.1项目的核心流程



### 1.2.2 读取csv文件

我从kaggle上下载了很多数据集，最终发现数据集以csv文件的格式居多，所以我就萌生了写一下通用的数据类来自动化读取csv文件，想要的结果是，不需要手动的配置多少行多少列，只要格式符合我的要求，所有的参数都可以直接拿到，包括数据有多少行，多少列，最终预测的不重复结果集有什么。

第一步读取数据集有了自己的类之后就非常方便。直接调用写好的接口即可。

### 1.2.3 数据打乱后分为训练集和测试集

上一步获取数据之后，因为将其分为训练集和测试集费时又费力，所以我实际上做的是用下标假分开数据，通过下标来标注区分好数据集，随后我将数据随机打乱，每次执行的数据就不一样了

### 1.2.4 for循环,执行测试集数目次循环,执行knn函数,使用count记录预测成功的测试数量

这一步是整个项目的核心代码，所有的cuda执行语句都在这里，包括数据的预处理，核函数执行程序和数据的后期处理。下面就来介绍KNN函数。

#### 1.2.4.1 knn函数的介绍

##### 函数原型

bool knn(vector<double> \* testPiece, //测试集的一行

    int position , //测试集的位置

    vector<vector<double>> \*doubleDataVector, // 所有的数据

    vector<string> \* resultVector, //结果的数据集

    set<string>\* resultSet); //结果的所有可能值 的 set(不重复)

训练集行数 trainDataSize

测试行是columnSize列

##### Knn 函数执行步骤

1. **申请空间** (显存和内存)。

2. **执行第一个核函数**,算出每一个测试行和所有行差值的平方矩阵

afterSubAndSquareResultArray,矩阵规模是 trainDataSize \* columnSize 。

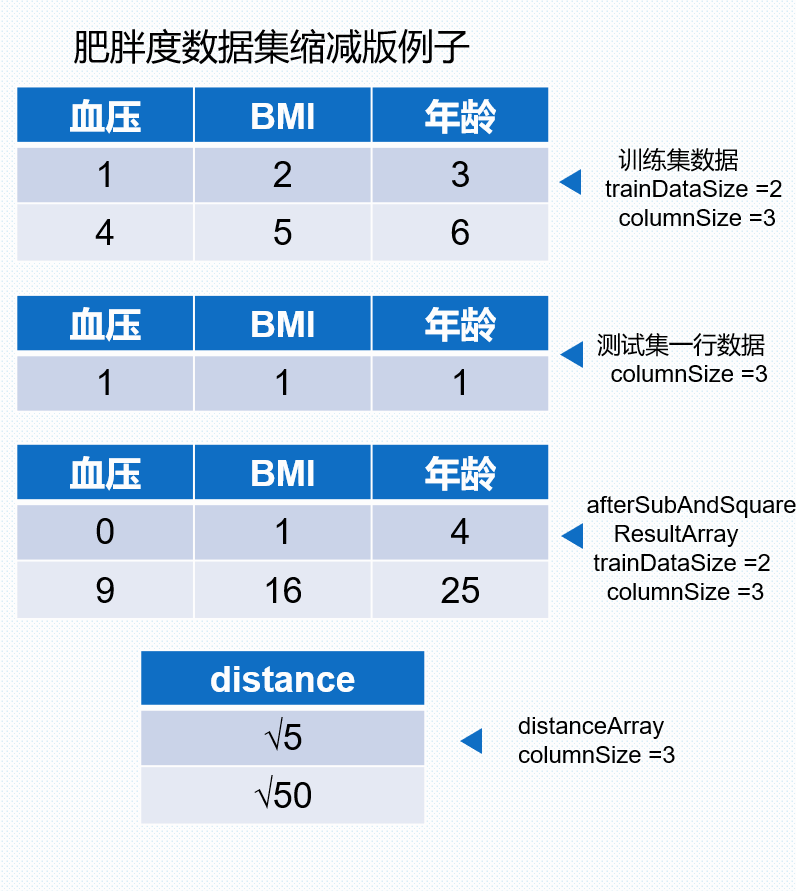
3. **执行第二个核函数**, 算出afterSubAndSquareResultArray每一行的值开方之后的值。此时的矩阵是一个一维矩阵distanceArray(有trainDataSize列),其每一个值代表的就是该测试行距离所有训练集的欧式距离

4. **加权** 将distanceArray中的数据排序,取前k个距离,然后根据这k个数据进行加权累加到所有可能的值上去（比如癌症的诊断数据集就只有得病和不得病两个选项),

5. **选择权值最高的可能值作为预测值** 最终可以从这多个可能的集合中挑选一个权值最高的，得到的肯定是可能性最高的。这样做的好处就是多考虑了排名对于最终预测值的影响,排名越靠前,影响越大。

6.**返回预测结果和真实值的对比，以便于后期统计**

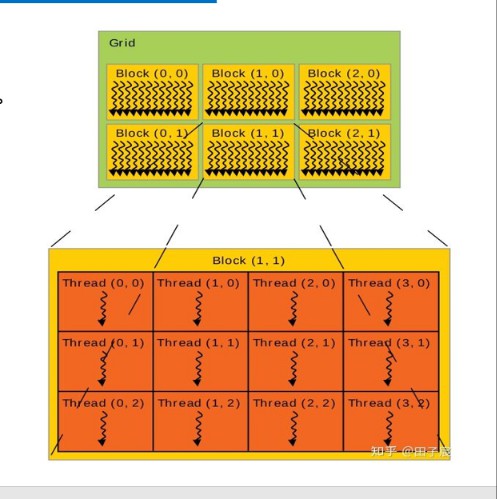
##### 一个小例子



#### 1.2.4.2 核函数介绍

knn函数的重点 就是两个核函数的执行

因为第一个核函数执行的cuda核心线程分布如下图所示



Grid 中block的值是 二维排布的 ThreadSize \* ThreadSize

Block中 thread是二维排布的 rowSize/ThreadSize columnSize/ThreadSize

所以定位一个线程需要四个参数

其实写cuda的核函数的关键在于定位到每一个thread，让每一个thread去执行同一段代码，对于矩阵的运算，cuda具有先天优势，那么一个二维矩阵的每一个元素的运算怎么映射到cuda上的线程，显然，需要将cuda的维度也降到二维。   下图是如何定位

图片包含 图表, 图示, 箱线图

描述已自动生成

##### 第一个核函数 计算一个二维矩阵每一行减去一个一维矩阵的对应位置数据后平方

//一次性算整个数组   测试集中的一行 都被训练集中的每一行先做减法然后平方

\_\_global\_\_ void

MatrixSubAndSquare

(double \*trainSet,  //传入二维数组

 double \*oneRowOftestSet, //需要计算距离测试集的某一行

 double \*afterSubAndSquareResultArray, //经过减法计算和平方计算后的中间数据

 int columnSize

){

// 设想的grid布 rowSize/ThreadSize columnSize/ThreadSize

// ThreadSize=32 ThreadSize=32

// rowSize/ThreadSize \* ThreadSize = rowSize

// columnSize/ThreadSize = columnSize

    int row = blockDim.x \* blockIdx.x + threadIdx.x;

    int col = blockDim.y \* blockIdx.y + threadIdx.y;

    double value = trainSet[row \* columnSize + col]  - oneRowOftestSet[col]; //把平方后的值放在新数组中

    afterSubAndSquareResultArray[row \* columnSize + col] = value \* value;

}

\_\_global\_\_ void sumMatrix(double \*aa,double \*distance,int columnSize){ //计算每行的和 然后开方

    int x = blockIdx.x \*blockDim.x + threadIdx.x;

    double value = 0;

    for(int i=0;i<columnSize;i++){

        value += aa[x \* columnSize + i];

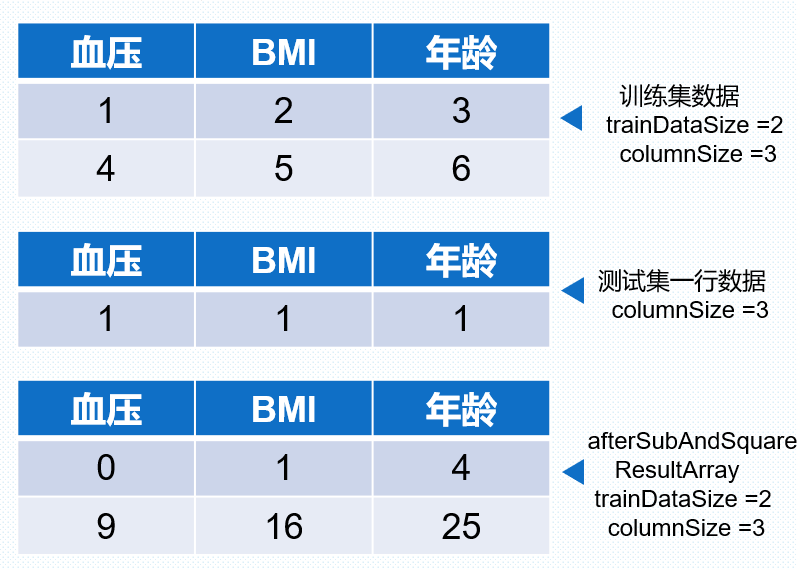
    }

    distance[x] = sqrt(value);

}

###### 下边是一个小例子

训练集数据的每行减去一行测试集数据后平方后的值就是第一个核函数执行的内容



##### 第二个核函数 将每行数据所有列的数据累加后开方得到距离数组

\_\_global\_\_ void sumMatrix(double \*aa,double \*distance,int columnSize){ //计算每行的和 然后开方

    int row = blockIdx.x \*blockDim.x + threadIdx.x;

    double value = 0;

    for(int i=0;i<columnSize;i++){

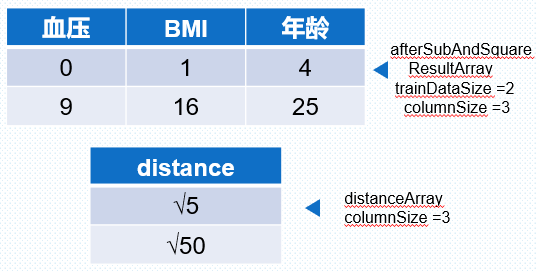
        value += aa[row \* columnSize + i]; // 计算累加和

    }

    distance[row] = sqrt(value); //开方 每行得到distance数组中的一个元素

}

###### 以下是一个小例子



#### 1.2.4.3 数据后期处理

将前k个距离挑选出来，距离测试数据距离越近的行权重越大，将其加到对应值上，然后比较所有的可能值，看那个数最大，预测值就是它了，比如 得病 (3.28) 不得病 (1.26)

那么可能性最大就是得病了。

如果预测正确 返回true 否则返回false。

## 1.3 源代码

### 1.3.1读取csv文件代码

#### Yhn.csv 文件

//

// Created by 尹浩男 on 2020/9/24.

//

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <set>

#include <algorithm>

#ifndef HELLO\_CSV\_H

#define HELLO\_CSV\_H

using namespace std;

class Csv {

private:

    ifstream \*file;

    string stringOfResult;  //这个是需要预测的列的名字

    set<string> \*realitySet; //# 这个是用来存储结果可能的所有种类 比如预测是否得病只有 两种可能 得病和不得病

    const char symbol = ',';  //定义用什么符号分割 默认读取csv文件 用逗号分割

    vector<string> \*rowVector;    //里边存放了所有的字符串数据 需要进行转换

    vector<string> \* header;  //存放头的信息 不包括预测的名字

    vector<vector<double>> \*doubleDataArray; //二维数组 用来存放数据

    vector<string> \*stringResultVector;  //用于存储结果列中的所有数据

    void init();  // 初始化 用来动态开辟空间的函数

    /\*\*

     \* @brief 传入字符串和分隔符 返回一个string的vector  相当于python或者java的split函数

     \* @param (string) str   要分割的字符串

     \* @param (char) symbol  分隔符

     \* @return (vector<string>\*) 返回一个string的vector

     \*/

    vector<string>\* getStringVector(string str,char symbol);

    /\*\*

     \* @brief 将string

     \* @param strVector

     \* @return

     \*/

    vector<double>\* stringVectorToDoubleVector(vector<string> \*strVector);

public:

    Csv(ifstream \*file);  //构造函数

    virtual ~Csv();  // 析构函数  用来free一些new的内容

    vector<vector<double>> \*  getDoubleData();  //获取数据的而二维数组vector

    vector<string> \* getHeaderNameVector();

    vector<string> \* getResultVector();

    set<string> \* getResultSet();

    void printHeaderVector();  //打印 Header列的名字 不包括结果列的名字

    void printDoubleDataVector();  //打印所有数据

    void printResultInformation(); //打印有关结果列的信息 包括结果列的名称 和 结果列中所有可能的值(无重复)

    void printResultVector();   //打印结果列的所有数据

};

#endif //HELLO\_CSV\_H

#### Yhncsv.cpp

//

// Created by 尹浩男 on 2020/9/24.

//

#include "yhncsv.h"

void Csv::init() {

    doubleDataArray = new vector<vector<double>>; //二维数组 用来存放数据

    stringOfResult = "";  //这个是需要预测的列的名字

    realitySet = new set<string>; //# 这个是用来存储结果可能的所有种类 比如预测是否得病只有 两种可能 得病和不得病

    rowVector = new vector<string>;    //里边存放了所有的字符串数据 需要进行转换

    stringResultVector = new vector<string>; // 里边存放了最后一行的result数据

    if(!file->is\_open()){

        cout<<"打开文件失败 open file failure"<<endl;

        exit(-1);

    }

}

vector<string> \*Csv::getStringVector(string str, char symbol) {

    auto \*stringVector = new vector<string>;

    auto \* positionOfSymbolVector = new vector<int>;

    int positionOfSymbolVectorSize = 0;

    for(int i=0;i<str.length();i++){

        if(str[i] == symbol){

            positionOfSymbolVector->push\_back(i);

        }

    }

    positionOfSymbolVectorSize = positionOfSymbolVector->size(); //获取vector大小

    //先取出第一个 字符串

    string temp = str.substr(0,positionOfSymbolVector->at(0));

    stringVector->push\_back(temp);

    //取出中间的部分

    for(int i=0;i<positionOfSymbolVectorSize -1;i++){

        temp = str.substr(positionOfSymbolVector->at(i)+1,positionOfSymbolVector->at(i+1) - positionOfSymbolVector->at(i) -1);

        stringVector->push\_back(temp);

    }

    //取出最后一个字符串

    temp = str.substr(positionOfSymbolVector->at(positionOfSymbolVectorSize -1) + 1,

                      str.size()- positionOfSymbolVector->at(positionOfSymbolVectorSize -1));

    stringVector->push\_back(temp);

//    cout<<"size of stringVector:"<<stringVectorSize<<endl;

    free(positionOfSymbolVector); //释放内存防止内存泄漏

    return stringVector;

}

vector<double> \*Csv::stringVectorToDoubleVector(vector<string> \*strVector) {

    auto \*doubleVector = new vector<double>;

    for(string ele : \*strVector){

        doubleVector->push\_back(atof(ele.c\_str()));

    }

    return doubleVector;

}

Csv::Csv(ifstream \*ffile) :file(ffile){

    init();  //初始化new 一些参数

    string row;

    while (getline(\*file,row)){

        // cout<<row<<endl;

        // 拿到的数据是这样的 RI,Na,Mg,Al,Si,K,Ca,Ba,Fe,Type

        //1.52101,13.64,4.49,1.1,71.78,0.06,8.75,0,0,1

        rowVector->push\_back(row);  //把每一行字符串填入vector

    }

    //获取第一行header的信息

    header = getStringVector(rowVector->at(0),symbol);

    stringOfResult = header->back();  //获取 需要预测的列的名字

    header->pop\_back();  //弹出最后的元素

    // 获取之后的所有数据 去除了结果行

    for(int i=1;i<rowVector->size();i++){

        vector<string> \* tempStringVector = getStringVector(rowVector->at(i),symbol);  //先统一转换为字符串vector

        string temp = tempStringVector->back();

        stringResultVector->push\_back(temp);  //把所有的结果存成一个vector 用来计算准确率的时候用

        realitySet->insert(temp); //把最后一列 需要预测的种类 全部存入不可重复的set中去

        tempStringVector->pop\_back(); //弹出最后一个元素

        vector<double> \* tempDoubleVector = stringVectorToDoubleVector(tempStringVector); //然后再进行从字符串vector向doubleVector的转换

        doubleDataArray->push\_back(\*tempDoubleVector);   //将得到的转换完成的doubleVector 存入二维数组中

        free(tempStringVector);  //释放到临时的stringVector

    }

    free(rowVector); //用完rowVector就释放掉

}

vector<vector<double>> \*Csv::getDoubleData() {

    return doubleDataArray;

}

vector<string> \*Csv::getResultVector() {

    return stringResultVector;

}

Csv::~Csv() {

    free(stringResultVector); //释放掉

    free(realitySet);  //释放set

    while(doubleDataArray->empty()){  //从尾部依次释放申请的空间

        free(&doubleDataArray->back());  //释放掉一维数组

        doubleDataArray->pop\_back(); // 将元素弹出

    }

    free(doubleDataArray); //最终释放二维数组

}

vector<string> \*Csv::getHeaderNameVector() {

    return header;

}

set<string> \*Csv::getResultSet() {

    return realitySet;

}

void Csv::printHeaderVector() {

    cout<<"判断依据有 : ";

    for(string temp : \*header){

        cout<<temp<<" , ";

    }

    cout<<endl;

}

void Csv::printDoubleDataVector() {

    for(vector<double> doubleTempVector : \*doubleDataArray){

        for(double temp : doubleTempVector){

            cout<<temp<<" ";

        }

        cout<<endl;

    }

}

void Csv::printResultInformation() {

    cout<<"结果所在的列名称是："<<stringOfResult<<" ";

    cout<<"需要预测的值的所有可能参数(无重复)有 ";

    for(string str : \*realitySet){

        cout<<str<<" ";

    }

    cout<<endl;

}

void Csv::printResultVector() {

    for(string temp:\*stringResultVector){

        cout<<temp<<" ";

    }

    cout<<endl;

}

### 1.3.2 主函数执行代码

#include <vector>

#include <string>

#include <ctime>

#include <cstdlib>

#include "yhncsv.h"

#include "common.h"

#include <map>

using namespace std;

// 全局变量

double trainDataProportion = (float )2/3;  //用于规定训练集占总数据的比例

int dataSize = 0;

int trainDataSize = 0; //用于记录训练集的大小

int testDataSize = 0; //用于记录测试集的大小

int columnSize = 0;  //用于记录数据列的数量

int threadSize = 2; //一个块中 线程数是32 \* 32  =1024 最大值

int k=14; //  用来设置取前 k 个距离最近的数据

double allKernelFunctionCostTime = 0;

double allCostTime = 0;

string fileName = "F://fashion-mnist\_train.csv";  //定义文件名字

//一次性算整个数组   测试集中的一行 都被训练集中的每一行先做减法然后平方

\_\_global\_\_ void

MatrixSubAndSquare

(double \*trainSet,  //传入二维数组 每一个都可以

 double \*oneRowOftestSet, //需要计算距离测试集的某一行

 double \*afterSubAndSquareResultArray, //经过减法计算和平方计算后的中间数据

 int columnSize){

//设想的grid 分布rowSize/ThreadSize columnSize/ThreadSize

// ThreadSize=32 ThreadSize=32

// rowSize/ThreadSize \* ThreadSize = rowSize

// columnSize/ThreadSize = columnSize

    int row = blockDim.x \* blockIdx.x + threadIdx.x;

    int col = blockDim.y \* blockIdx.y + threadIdx.y;

   double value=trainSet[row \* columnSize + col]oneRowOftestSet[col];

//把平方后的值放在新数组中

    afterSubAndSquareResultArray[row \* columnSize + col] = value \* value;

}

\_\_global\_\_ void sumMatrix(double \*aa,double \*distance,int columnSize){ //计算每行的和 然后开方

    int x = blockIdx.x \*blockDim.x + threadIdx.x;

    double value = 0;

    for(int i=0;i<columnSize;i++){

        value += aa[x \* columnSize + i];

    }

    distance[x] = sqrt(value);

}

//这个knn函数的一次运行 算出的是  一行测试数据 距离  所有训练集所有行的距离 然后根据最近的k个数据来预测值

bool knn(vector<double> \* testPiece, int position ,vector<vector<double>> \*doubleDataVector,vector<string> \* resultVector,set<string>\* resultSet){

    //1. 初始化一些变量

    //1.1 基本变量的初始化

    double maxWeight = -1;  // 用来存储最大权重

    string maxWeightStr = "";  // 用来存储最大权重的字符串  也就是预测值

    bool flag = false;   // 返回给主函数 用于判断预测是否正确

    double sum = 0;  //这个数据用于之后计算权值的时候用 谁离得最近 权值越高

    auto \*doubleArrayA = new double[trainDataSize \* columnSize]; // 用于在传输数据的时候用数组临时存储训练集数组 vector行不通 把值逐个赋值给数组 vector不行

    auto \*doubleArrayB = new double[columnSize]; //是用来存储测试集的一行数据的 和上方一样

    //    double \* doubleArrayResult = new double[trainDataSize\*columnSize];  //用于存储中间数据 经过相减平方后的数据 调试时可以输出

    //1.2 映射指针的创建 定义指针 用来映射显存中的数据

    double \*cudaDoubleArray; //整个数据二维数组 显存中的数据

    double \*cudaTestArrayPiece; //一行测试集数据 显存中的数据

    double \*cudaAfterSubAndSquareDoubleArrayResult; //中间数据  显存中的数据

    //1.3 核函数规模的定义 第一个是矩阵减法 和 平方的核函数

    dim3 firstBlocksPerGrid(trainDataSize/threadSize,columnSize/threadSize);

    dim3 firstThreadsPerBlock(threadSize,threadSize);

    dim3 secondBlocksPerGrid(trainDataSize/threadSize);

    dim3 secondThreadsPerBlock(threadSize);

    //2. 申请空间

    cudaMalloc((void\*\*)&cudaDoubleArray,sizeof(double) \* trainDataSize \* columnSize ); //申请显存中二维数组的空间 用于存放训练集数据

    cudaMalloc((void\*\*)&cudaTestArrayPiece,sizeof(double) \* columnSize);  //申请测试数据一维数组的空间 二维训练集的每一行都减去一维测试集的对应位的数据 然后平方

    cudaMalloc((void\*\*)&cudaAfterSubAndSquareDoubleArrayResult,sizeof(double) \* trainDataSize \* columnSize ); //申请中间结果的显存空间 规模和训练集一样

    //3.拷贝数据进入显存

    //3.1拷贝训练集显存

    //目前是没有什么好办法 只能挨个进行赋值 应该也不慢 但是肯定比那些直接进行内存整块拷贝的慢

    //3.1.1 先把数据弄到一个double数组中去

    for(int i=0;i<trainDataSize;i++){

        for(int j =0;j<columnSize;j++){

            doubleArrayA[i\* columnSize +j] = doubleDataVector->at(i).at(j);

        }

    }

    //3.1.2 执行cuda显存拷贝函数

    cudaMemcpy(cudaDoubleArray,doubleArrayA,sizeof(double)  \* columnSize \* trainDataSize ,cudaMemcpyHostToDevice); //将训练集的数据拷入到显存中

    //3.2 拷贝测试集数据 只有一行 所以可以用copy函数

    //3.2.1 从vector<double> 转换为 double 数组

    copy(testPiece->begin(),testPiece->end(),doubleArrayB);  //分别表示 要复制的vector的头, 要复制的vector的尾 , 目标数组

    //3.2.2 执行cuda显存拷贝函数

    cudaMemcpy(cudaTestArrayPiece,doubleArrayB ,sizeof(double) \* columnSize ,cudaMemcpyHostToDevice); //将test的数据传入

    //4. 执行第一个核函数

    cudaEvent\_t start1,stop1;

    float elapsedTime1 = 0;

    cudaEventCreate(&start1);

    cudaEventCreate(&stop1);

    cudaEventRecord(start1,0);

    MatrixSubAndSquare<<<firstBlocksPerGrid,firstThreadsPerBlock>>>(cudaDoubleArray,cudaTestArrayPiece,cudaAfterSubAndSquareDoubleArrayResult,columnSize);

    cudaEventRecord(stop1,0);

    cudaEventSynchronize(stop1);

    cudaEventElapsedTime(&elapsedTime1,start1,stop1);

//    cout<<"first kennel function cost time:"<<elapsedTime1<<endl;

    //将结果拷贝回来  这一步是中间步骤 调试的时候排错用

//    cudaMemcpy(doubleArrayResult,cudaAfterSubAndSquareDoubleArrayResult,trainDataSize \* columnSize \*sizeof(double)  ,cudaMemcpyDeviceToHost); //将训练集的数据拷入到显存中)

//    //打印计算的中间结果 中间步骤 调试使用

//    for(int i=0;i<trainDataSize;i++){

//        for(int j=0;j<columnSize;j++){

//            cout<< doubleArrayResult[i\*columnSize + j]<<"    ";

//        }

//        cout<<endl;

//    }

    //5. 释放一部分显存和内存  注意没有释放 cudaAfterSubAndSquareDoubleArrayResult 因为中间结果还需要使用

    //5.1 释放显存

    cudaFree(cudaDoubleArray); //释放 二维数组(排布为一维) 训练集数据

    cudaFree(cudaTestArrayPiece); //释放 一维数组 测试集的一行数据

    //5.2 释放内存

    free(doubleArrayA);

    free(doubleArrayB);

    //6 为执行第二个核函数准备空间(内存和显存)

    auto \*distanceArray = new double[trainDataSize]; //申请内存空间 用来存放距离数组

    double \*cudaDistanceArray; //申请空间 映射显存空间 用来存放距离数组

    cudaMalloc((void\*\*)&cudaDistanceArray,sizeof(double) \* trainDataSize ); //申请存放距离显存空间

    //7 执行第二个核函数

    cudaEvent\_t start2,stop2;

    float elapsedTime2 = 0;

    cudaEventCreate(&start2);

    cudaEventCreate(&stop2);

    cudaEventRecord(start2,0);

    sumMatrix<<<secondBlocksPerGrid,secondThreadsPerBlock>>>(cudaAfterSubAndSquareDoubleArrayResult,cudaDistanceArray,columnSize);

    cudaEventRecord(stop2,0);

    cudaEventSynchronize(stop2);

    cudaEventElapsedTime(&elapsedTime2,start2,stop2);

//    cout<<"second kennel function cost time:"<<elapsedTime2<<endl;

//    cout<<"two kennel function cost time:"<<elapsedTime1+elapsedTime2<<endl;

    allKernelFunctionCostTime += elapsedTime1+elapsedTime2;

    //8 将最后的距离数组拷贝回内存 以便后边使用

    cudaMemcpy(distanceArray,cudaDistanceArray,sizeof(double) \*trainDataSize ,cudaMemcpyDeviceToHost);

    //9 释放掉所有显存 因为以后用不到了

    cudaFree(cudaAfterSubAndSquareDoubleArrayResult);  //释放掉中间数据数组

    cudaFree(cudaDistanceArray);  //释放掉距离数组

//    for(int i=0;i<trainDataSize ;i++ ){  //打印距离数据

//        cout<<distanceArray[i]<<endl;

//    }

    //10 最后的数据处理

    //10.1 初始化一些stl 以后会用到

    auto \*realityAndDistanceMap = new multimap<double,string>;  //距离作为key 真实值为value 这样做的好处是自动排序 需要采用multimap 虽然距离一般不能一样 但是就怕巧了

    auto \*weightMap = new map<string,double>;  //权重map   后边double数据可以作为依据 key不可能重复 所以放心用 map

    set<string>::iterator setItr ;  //用来遍历所有结果(resultSet) 这个set里存放了结果集的所有可能 比如判断是否得病的数据集 只有得病或者不得病两种 已经是排好序的

    map<double,string>::iterator mapIter; //用于遍历

    //10.2 计算好距离之后需要把数据和真实值对应起来  之后用于统计权值的时候回用到 realityAndDistanceMap multimap<double,string>

    for(int i=0;i<trainDataSize;i++){

        realityAndDistanceMap->insert(pair<double,string>(distanceArray[i] ,resultVector->at(i)));

    }

    free(distanceArray); //顺手把distanceArray释放掉 以后不会再用了

    //10.3.初始化权重map 把value都设成0  统计权重的目的是综合考虑k个最近的点的影响 约接近的点权重越高

    setItr = resultSet->begin();  //resultSet set<string> 的迭代器

    for(;setItr!=resultSet->end();setItr++){

        weightMap->insert(pair<string,double>(\*setItr,0));

    }

    //10.4 计算前k个最近的点的总距离sum 用于算权重

    mapIter = realityAndDistanceMap->begin();

    for(int i=0;i<k;i++,mapIter++){ //计算sum值 计算出来sum值

        sum  += mapIter->first;

    }

    //10.5 分别计算前k个点的权值 根据其真实值 加到所有可能的值上 比如得病权重5.4 不得病权重为 8.4 所以可以判断大概率是不得病

    mapIter = realityAndDistanceMap->begin();

    for(int i=0;i<k;i++ ,mapIter++){  //前k个元素的权重算出来

        (\*weightMap)[mapIter->second] += 1 - (mapIter->first / sum) ; //距离越近 权重越高  注意这里是 1- xxx

    }

    //10.6 找到最高的那个权重的值 比如是不得病 然后将它赋值给maxWeightStr

    for(pair<string,double> p :\*weightMap ){

        if(p.second > maxWeight){

            maxWeight = p.second;

            maxWeightStr = p.first;

        }

    }

    //10.7 根据测试集的真实值和预测值对比是否一样  如果一样说明预测成功

    if(resultVector->at(position).compare(maxWeightStr)  == 0)

        flag = true;  //flag 后边会return回去

    else

        flag = false;

    //11 最终释放掉所有的内存(显存已经全部释放)

    free(realityAndDistanceMap); //释放真实值和距离map

    free(weightMap); //释放权重map

    return flag; //返回预测结果和真实值是否匹配

}

int main(int argc,char \* argv[]) {

    clock\_t allTimeBegin = clock();

    if(argc > 1){

        fileName = argv[1];

        cout<<"已输入参数， csv文件为 :   "<<argv[1]<<endl<<endl;

    }else{

        cout<<"未输入参数！！！ 默认csv文件是:"<<fileName<<endl;

    }

    //1. 初始化所有参数

    vector<vector<double>> \*doubleDataVector = nullptr; //二维数组 用来存放训练集和测试集的所有数据

    vector<string> \* resultVector = nullptr;  //结果集 里边全是字符串

    set<string> \* realitySet = nullptr;  //无重复的所有结果集

    ifstream inputFile;   //定义文件输入流

    //2.处理i/o流 打开文件 然后调用yhnCsv类来读取csv文件的数据

    inputFile.open(fileName);  //打开文件

    Csv \* csvReader = new Csv(&inputFile);  //把文件句柄传进去

    //3. 获取数据

    //3.1 数组或者集合的获取

    realitySet = csvReader->getResultSet();  //获取不重复的数据集

    resultVector = csvReader->getResultVector(); //拿到所有的结果

    doubleDataVector = csvReader->getDoubleData(); //把所有的训练集和测试集的数据拿到

    //3.2 获取全局变量

    columnSize = doubleDataVector->at(0).size();  //数据列的数量

    dataSize = doubleDataVector->size();  //记录总数据集的行数

    trainDataSize = trainDataProportion \* dataSize;  //记录训练集的行数  比例乘以 总数据集的行数

    testDataSize = dataSize - trainDataSize; //记录测试集的行数

    //4. 同时对全部数据集和结果数据集进行随机 如果随机数不相同 那么就交换  这样能同时进行多个数组的交换 swap函数很好使

    srand((unsigned int)time(NULL));  //以时间为基准进行随机

    for (int i = 0; i < dataSize; ++i) {  //最多交换 总数据集大小 其实一般就行 不过无所谓了

        int n1 = (rand() % dataSize);//产生n以内的随机数  n是数组元素个数

        int n2 = (rand() % dataSize);

        if (n1 != n2) { //若两随机数不相等 则下标为这两随机数的数组进行交换

            swap(doubleDataVector->at(n1),doubleDataVector->at(n2));

            swap(resultVector->at(n1),resultVector->at(n2));

        }

    }

    csvReader->printHeaderVector();  //打印头的所有字符串 不包括结果列的名字

    csvReader->printResultInformation(); //打印result信息

    cout<<"k值为："<<k<<",总数据集有"<<dataSize<<"条，"<<"训练集有"<<trainDataSize<<"条,"<<"测试集有"<<testDataSize<<"条"<<endl;

    int count = 0; //用来统计成功预测的数量

    for(int i=0;i<testDataSize;i++){

        bool flag =

 knn(&doubleDataVector->at(trainDataSize + i), //测试集的一行

                         trainDataSize +i  , //测试集的位置

                         doubleDataVector, // 所有的数据

                         resultVector, //结果的数据集

                         realitySet); //结果的所有可能值 的 set(不重复)

        if(flag){

            count++;

        }

    }

    clock\_t allTimeEnd = clock();

    cout<<"此次随机的数据集的准确率为: "<<(float )count/testDataSize \*100 <<"%"<<endl ;

    cout<<"all Kernel Function Cost Time: "<<allKernelFunctionCostTime<<" ms"<<endl;

    cout<<"total Cost Time:"<<(allTimeEnd - allTimeBegin)/CLOCKS\_PER\_SEC <<" s"<<endl;

    free(csvReader);

}

## 1.4 数据分析

### Python串行运行结果

读取文件为: glass.csv

根据以下参数来预测结果(列的名称) ： ['RI', 'Na', 'Mg', 'Al', 'Si', 'K', 'Ca', 'Ba', 'Fe']

结果所在的列名称是：Type 他的种类有： ['1', '2', '3', '5', '6', '7']

准确率为 61.111111111111114

程序运行时间：0.066445 s

### C++ CUDA运行结果

未输入参数！！！ 默认csv文件是:../glass.csv

判断依据有 : RI , Na , Mg , Al , Si , K , Ca , Ba , Fe ,

结果所在的列名称是：Type 需要预测的值的所有可能参数(无重复)有 1 2 3 5 6 7

k值为：14,总数据集有214条，训练集有142条,测试集有72条

此次随机的数据集的准确率为: 68.0556%

所有核函数执行时间为 : 0.799776 ms

total Cost Time:0 s

**运行速度比较 0.799776 ms << 0.066445 s**

### 结论

Cuda的优化是很明显的 ，我利用python写了一个串行的程序，会发现用时明显缩短，cuda核函数的执行时间甚至只有0.7毫秒，可以说是质的提升 。

为了测试这个cuda程序的兼容性和健壮性，我们使用了不止一个数据集，我们会发现，knn不是对于任何数据集都使用，尤其是那种结果集分类多的数据集，预测正确率只有不到50%，算是我们找的所有数据集中准确率低的离谱的了

为了测试内存的申请释放会不会出现问题，我们还测试了超大的数据，测试没有问题，但是我们发现内存的拷贝和其他步骤的执行占用大部分时间，以后可以考虑优化这一方面的内容