并行程序设计实践MPICH项目一

项目名称：基于OpenMP、MPI、MPI+OpenMP的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

**题目：**

用OpenMP、MPI、MPI+OpenMP设计一个KNN分类算法（K近邻算法）程序。

说明：在特征空间中查找K个最相似或者距离最近的样本，然后根据K个最相似的样本对未知样本进行分类。

通过训练集和测试集给出算法的正确率。

**程序核心代码：**

|  |
| --- |
| #include <iostream>  #include <cmath>  #include "yhncsv.h"  #include <vector>  #include <string>  #include <ctime>  #include <cstdlib>  #include "yhncsv.h"  #include <map>  #include <omp.h>  #include <mpi.h>  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 k=15; // 用来设置取前 k 个距离最近的数据  string fileName = "../fashion-mnist\_test.csv"; //定义文件名字  int threadNum = 2;  //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 \*distanceArray = new double[trainDataSize]; //申请内存空间 用来存放距离数组  //我们要求的数据有 距离数组  omp\_set\_num\_threads(threadNum);  #pragma omp parallel for  for(int i=0;i<trainDataSize;i++){  double sum = 0;  omp\_set\_num\_threads(threadNum);  for(int j=0;j<columnSize;j++){  sum += pow(testPiece->at(j) - doubleDataVector->at(i).at(j),2); //求出平方累加  }  distanceArray[i] = sqrt(sum);  }  //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[]) {  //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->printDoubleDataVector();  // 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  int count = 0; //用来统计成功预测的数量  //mpi 开始了  int rank,num\_procs; // rank 是当前运行线程编号 proc\_len 是进程名字长度 num\_procs 是进程数量  double startTime = 0,endTime = 0; //记录时间  MPI\_Init(&argc,&argv);  //用来获取正在调用进程的通信子中的进程号的函数  MPI\_Comm\_rank(MPI\_COMM\_WORLD,&rank);  //用来得到通信子的进程数的函数  MPI\_Comm\_size(MPI\_COMM\_WORLD,&num\_procs);  if(rank == 0){ // 开始计时  cout<<"threadNum = "<<threadNum<<endl;  startTime = MPI\_Wtime();  cout<<"The k value is："<<k<<",the all dataSet has "<<dataSize<<" pieces of data，"<<"the train Set has "<<trainDataSize<<",the test set has "<<testDataSize<<endl;  printf("total %d processes\n",num\_procs);  }  int partCount = 0;  for(int i=rank;i<testDataSize;i+=num\_procs){  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){  partCount++;  }  }  MPI\_Reduce(&partCount,&count,1,MPI\_DOUBLE,MPI\_SUM,0,MPI\_COMM\_WORLD);  if(rank == 0){  endTime = MPI\_Wtime();  cout<<"this time the accuracy of dataSet is "<<(float )count/testDataSize \*100 <<"%"<<endl ;  double seconds = endTime - startTime ;  cout<<"cost time "<<seconds<<" seconds"<<endl; //将最终的消耗时间进行打印  }  MPI\_Finalize();  delete csvReader;  } |

**程序运行结果：**

**CPU串行：**

no input value!!!! the default csv file is:../KNN\_Data.csv

The k value is：15,the all dataSet has 1000 pieces of data，the train Set has 666,the test set has 334

this time the accuracy of dataSet is 80.2395%

cost time 3.14609 seconds

threadNum = 1

**OpenMP：**

no input value!!!! the default csv file is:../KNN\_Data.csv

The k value is：15,the all dataSet has 1000 pieces of data，the train Set has 666,the test set has 334

this time the accuracy of dataSet is 80.5389%

cost time 0.679237 seconds

threadNum = 10

**MPI：**

mpirun -np 10 ./mpiKnn   
The k value is：15,the all dataSet has 1000 pieces of data，the train Set has 666,the test set has 334  
total 10 processes  
this time the accuracy of dataSet is 79.3413%  
cost time 0.738929 seconds

**MPI+OpenMP:**

mpirun -np 28 ./hybridMpiAndOpenMPKnn   
threadNum = 2  
The k value is：15,the all dataSet has 1000 pieces of data，the train Set has 666,the test set has 334  
total 28 processes  
this time the accuracy of dataSet is 79.9401%  
cost time 0.496747 seconds

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | CPU串行 | OpenMP | MPI | MPI＋OpenMP |
| 时间（s） | 3.14609 | 0.679237 | 0.738929 | 0.496747 |
| 准确率（%） | 80.2395 | 80.5389 | 79.3413 | 79.9401 |

