KNN.
第一行输入依次为：k(k<=10000)，特征向量的长度 L(L<=100)，训练数据行数 M(M>k, M<=10000)，测试数据行数 N(N<=10000)为测试数据行数。
之后是 M 行训练数据和 N 行测试数据。每行中数据使用空格分隔。
输出：
预测类别

```
测试数据:
3 5 16 2
tran.txt
0.19 0.04 0.06 0.22 0.11 A 0.28 0.42 0.38 0.39 0.44 B 0.71 0.61 0.54 0.52 0.54 C
0.98 0.82 0.92 0.98 0.97 D 0.05 0.03 0.15 0.01 0.11 A 0.33 0.29 0.33 0.47 0.27 B
0.72 0.52 0.61 0.71 0.68 C 0.78 0.86 0.91 1.0 0.76 D 0.01 0.17 0.14 0.15 0.2 A
0.44 0.36 0.32 0.32 0.35 B 0.67 0.65 0.57 0.58 0.52 C 0.87 0.92 0.8 0.83 0.77 D
0.01 0.11 0.14 0.12 0.07 A 0.33 0.43 0.43 0.45 0.38 B 0.57 0.54 0.75 0.7 0.64 C
0.9 0.94 0.83 0.96 0.77 D
test.txt
0.29 0.29 0.42 0.36 0.27 0.56 0.67 0.71 0.66 0.7
输出:
B C
*/
#include <iostream>#include <vector>#include <math.h>
#include <map>#include <algorithm>#include <fstream>
using namespace std;
  struct sample {
      char label;
      double distance;
  };
  bool cmp(sample a, sample b)
  {     return a.distance < b.distance;  }
  void readTrainData(vector<vector<double> > &trainData,
vector<char> &trainLabel, int L, int M)
  {
      ifstream inFile;  //打开训练数据 txt 文件
      inFile.open("E://Program Files//tran.txt");
      if(!inFile.is_open())
            {  cout << "can not read file tran" << endl;  }
      for (int i = 0; i < M; ++i) {
          // 读入训练数据特征
          vector<double> lineData;
          double tmpData;
          for (int j = 0; j < L; ++j) {
                inFile>>tmpData;
                lineData.push_back(tmpData);
          }
          trainData.push_back(lineData);
          // 读入训练数据标注
          char label;
          inFile>>label;
          trainLabel.push_back(label);
       }
       inFile.close();            //关闭文件输入流
   }
  void readTestData(vector<vector<double> >
&testData,vector<char> &testLabel, int L, int N)
  {
      ifstream inFile;  //打开测试数据 txt 文件
      inFile.open("E://Program Files//test.txt");
      if(!inFile.is_open())
            {cout << "can not read file test" << endl; }
         //读测试数据
      for (int i = 0; i < N; ++i) {
          vector<double> lineData;
          double tmpData;
          char tlabel;
          for (int j = 0; j < L; ++j) {
              inFile>>tmpData;
              lineData.push_back(tmpData);
          }
          testData.push_back(lineData);
          //读取测试标注
          inFile>>tlabel;
          testLabel.push_back(tlabel);
      }
       inFile.close();            //关闭文件输入流
  }
  double calcDistance(vector<double> data1, vector<double>
data2)
  {
      int length = data1.size();
      double distance = 0.0;
      for (int i = 0; i < length; ++i)
   distance += (data1[i] - data2[i]) * (data1[i] - data2[i]);
      return sqrt(distance);
  }

  void KNN(vector<vector<double> > trainData,
vector<vector<double> > testData,
           vector<char> trainLabel, vector<char>testLabel,int
k, int M, int N)
  {
      float accuracy;//识别率          float d=0;          float allaccuracy=0;
      int x=0;//整张图片识别率判断          int x1=0;
      for (int i = 0; i < N; ++i) {
          // 计算每一个测试样本与所有训练样本的距离，并排序。
          vector<sample> distances;
          for (int j = 0; j < M; ++j) {
              sample tmpDistance;
              tmpDistance.distance = calcDistance(testData[i],
trainData[j]);
              tmpDistance.label = trainLabel[j];
              distances.push_back(tmpDistance);
          }
          sort(distances.begin(), distances.end(), cmp);//排序
          // 选择前 k 个样本的距离，确定前 k 个点所在类别出现频率。
          map<char, int> labelMap;
          for (int j = 0; j < k; ++j) {
              if (!labelMap[distances[j].label])
                  labelMap[distances[j].label] = 0;
                  labelMap[distances[j].label] += 1;
          }
          // 找到 labelMap 中值最大的类别
          int maxVal = 0;
          char resLabel;
          for (map<char, int>::iterator it = labelMap.begin();
it != labelMap.end(); ++it)
              if (it->second > maxVal) {
                  maxVal = it->second;
                  resLabel = it->first;
              }
          // 输出类别。
          cout << resLabel << endl;//测试
        //识别率
        x++;//单张字符识别率
        if( testLabel[i] == resLabel)       {     x1++;     }
        if(x1==6)             { cout<<"number is"<< i<<endl;    }
        if(x==6&&x1==6)      {      allaccuracy++;                    cout<<"allaccuracy is"<<
allaccuracy<<endl;           }
        if(x==6)             {    x1=0;    x=0;       }
        //整张字符识别率
        if( testLabel[i] == resLabel){        d++;          }
      }
      accuracy=d/N;
      cout <<"signal character accuracy is "<<accuracy<< endl;
      allaccuracy=allaccuracy*6/N;
      cout <<"The whole picture recognition rate is "<<   allaccuracy <<endl;
  }
  int main()
  {   int k=3;      int L=5;//特征向量长度
     int M=6;//训练数据行数      int N=12;//测试行数
      vector<vector<double> > trainData;
      vector<vector<double> > testData;
      vector<char> trainLabel;
      vector<char> testLabel;
      readTrainData(trainData, trainLabel, L, M);
      readTestData(testData,testLabel, L, N);
      KNN(trainData, testData, trainLabel,testLabel,k,M,N);
      return 0;
  }
```

KNNData.java

```
public class KNNData implements Comparable<KNNData>{
    double c1;      double c2;      double c3;      double distance;      String type;
    public KNNData(double c1, double c2, double c3, String type) {
        this.c1 = c1;           this.c2 = c2;           this.c3 = c3;           this.type = type;
    }
     @Override
    public int compareTo(KNNData arg0) {
        return Double.valueOf(this.distance).compareTo(Double.valueOf(arg0.distance));
    }
}
```

KNN.java

```
import java.util.Collections;import java.util.HashMap;import java.util.Iterator;import java.util.List;
import java.util.Map;import java.util.Set;
public class KNN {
     //训练集
    private List<KNNData> KNNDS = null;
     public KNN(List<KNNData> KNNDS) {
        this.KNNDS = KNNDS;
    }
     //欧式距离
    private static double disCal(KNNData i, KNNData td) {
        return Math.sqrt((i.c1 - td.c1)*(i.c1 - td.c1)+(i.c2 - td.c2)*(i.c2 - td.c2)+
                (i.c3 - td.c3)*(i.c3 - td.c3));
    }
    private static String getMaxValueKey(int k, List<KNNData> ts){
        //只保留前 k 个元素
         while(ts.size() != k) {          ts.remove(k);    }
         String sKey;
        //保存 key 以及出现次数
        HashMap<String,Integer> keySet = new HashMap<String,Integer>();
        keySet.put(ts.get(0).type,1);
        for (int x = 1; x < ts.size(); x++) {
            sKey = ts.get(x).type;
            if (keySet.containsKey(sKey)) {
                keySet.put(sKey, keySet.get(sKey)+1);
            } else {            keySet.put(sKey, 1);         }
        }
        Set<Map.Entry<String,Integer>> set = keySet.entrySet();
        Iterator<Map.Entry<String,Integer>> iter = set.iterator();
        int mValue = 0;
        String mType = "";
        while (iter.hasNext()){
            Map.Entry<String,Integer> map = iter.next();
            if (mValue < map.getValue()) {      mType = map.getKey();      mValue =
map.getValue();    }
        }
        return mType;
    }

    public static String knnCal(int k, KNNData i, List<KNNData> ts) {
        //保存距离
        for (KNNData td : ts) {      td.distance = disCal(i, td);       }
        Collections.sort(ts);
        return getMaxValueKey(k, ts);
    }
}
```

KNNTest.java

```
import java.util.ArrayList;import java.util.List;
public class KNNTest {
    public static void main(String[] args) {
         List<KNNData> kd = new ArrayList<KNNData>();
        //训练集
        kd.add(new KNNData(1.2,1.1,0.1,"A"));            kd.add(new KNNData(1.2,1.1,0.1,"A"));
        kd.add(new KNNData(7.1,5,0.1,"B"));            kd.add(new KNNData(6,1.2,0.1,"B"));
        kd.add(new KNNData(2,2.6,0.1,"C"));            kd.add(new KNNData(2,2.6,0.1,"C"));
        kd.add(new KNNData(2,2.6,0.1,"C"));            kd.add(new KNNData(100,1.1,0.1,"D"));
        System.out.println(KNN.knnCal(3, new KNNData(1.1,1.1,0.1,"N/A"), kd));
    }
}
```

1 初始化距离为最大值 2 计算未知样本和每个训练样本的距离 dist 3 得到目前 k 个最邻近样本中的最大距离 maxdist 4 如果 dist 小于 maxdist，则将该训练样本作为 K 最近邻样本 5 重复步骤 234 直到未知样本和所有训练样本的距离都算完 6 统计 K 个最近邻样本中的每个类别出现的次数 7 选择出现频率最大的类别作为未知样本的类别

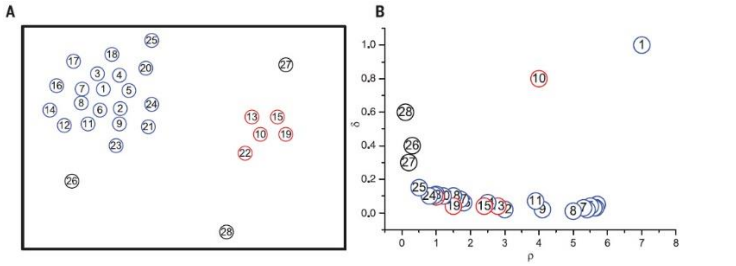
基于密度的聚类算法

1、对于每一个数据点i，计算局部密度ρ_i：d_c是截断距离，d_c的推荐值是使得平均每个点的邻居数为样本总数的 1%-2%, d_c的选择比较鲁邦；ρ_i相当于距离点i的距离小于d_c的点的个数；

$\rho\_i = \sum_j \chi(d\_ij - d\_c)$　$\delta\_i = \min(d\_ij)$　$(j: \rho\_j > \rho\_i)$

2、对于每一个数据点i，计算比i点密度高的点到i点的最小距离δ_i；

3、根据ρ和δ画出决策图, 找出聚类中心 ;划分剩余数据点到相应的簇。



△图 A 是所有点在二维空间的分布，所有点的密度值由高到低排列，1 表示密度最高的点；
△图 B 是以局部密度ρ为横坐标，以到高局部密度点的距离δ为纵坐标的决策图；
△1 和 10 两个点的ρ和δ都比较大，作为类簇的中心点；
△26、27、28 三个点的δ也比较大，但是ρ较小，所以是异常点；
△准则：当前点的类别标签，与高于当前点密度的最近点的标签一致；
△1 和 10 均为聚类中心，4 号点的类别标签应该和与其距离最近的、密度高于它的点一致，因此 4 号点属于聚类中心 1；
△由于 5 号点最近的密度比其高的点为 4 号点，因此其类别标签与 4 号相同，也为聚类中心 1。

k-means

```
#include<stdio.h>#include<stdlib.h>
#include<conio.h>#include<math.h>
struct point{  int lable;   double x;   double y;};
void main()
{  FILE *fp;  int i,j,k,c1,c2,n1,n2;
double dis1,dis2,prodis1,prodis2,disa,disb,sx1,sx2,sy1,sy2,dc1,dc2;
   point p[1000],center1,center2;
   fp=fopen("Test_data.txt","r");
   for(i=0;i<1000;i++)
   {  fscanf(fp,"%lf",&p[i].x);fscanf(fp,"%lf",&p[i].y);   }
   fclose(fp);   dis1=100000;   dis2=100000;
   for(i=0;i<1000;i++)
   {  for(j=i+1;j<1000;j++)
     {prodis1=0;prodis2=0;n1=0;n2=0;sx1=0;sx2=0;sy1=0;sy2=0;
     for(k=0;k<1000;k++){
disa=sqrt(((p[k].x-p[i].x)*(p[k].x-p[i].x))+((p[k].y-
p[i].y)*(p[k].y-p[i].y)));
disb=sqrt(((p[k].x-p[j].x)*(p[k].x-p[j].x))+((p[k].y-
p[j].y)*(p[k].y-p[j].y)));
 if(disa<disb){prodis1+=disa; n1++;sx1+=p[k].x;sy1+=p[k].y;}
 else{prodis2+=disb; n2++; sx2+=p[k].x; sy2+=p[k].y;}
     }
       if((prodis1+prodis2)<(dis1+dis2))
       {  c1=i; c2=j; dis1=prodis1; dis2=prodis2;
          center1.x=sx1/n1; center1.y=sy1/n1;
          center2.x=sx2/n2; center2.y=sy2/n2;
     dc1=sqrt(((p[c1].x-center1.x)*(p[c1].x-
center1.x))+((p[c1].y-center1.y)*(p[c1].y-center1.y)));
     dc2=sqrt(((p[c2].x-center2.x)*(p[c2].x-
center2.x))+((p[c2].y-center2.y)*(p[c2].y-center2.y)));
         if(dc1<0.01&&dc2<0.01) break;
       }
      }
     if(dc1<0.01&&dc2<0.01)    break;
   }
   for(k=0;k<1000;k++)
   {
disa=sqrt(((p[k].x-p[c1].x)*(p[k].x-p[c1].x))+((p[k].y-
p[c1].y)*(p[k].y-p[c1].y)));
disb=sqrt(((p[k].x-p[c2].x)*(p[k].x-p[c2].x))+((p[k].y-
p[c2].y)*(p[k].y-p[c2].y)));
       if(disa<disb)     p[k].lable=1;
       else     p[k].lable=2;
   }
   fp=fopen("data.txt","w+");
   for(k=0;k<1000;k++)
fprintf(fp,"%g     %g     %d\n",p[k].x,p[k].y,p[k].lable);
fprintf(fp,"%g     %g      3\n",center1.x,center1.y);
fprintf(fp,"%g     %g      4\n",center2.x,center2.y);
fclose(fp);
}
```

```
11
1.5 4.5, 4 2, 1 4, 5 2, 2 5, 4 3, 1 1.5, 1 5, 5 3, 1 1.5, 6 3,
Cluster 0 :Centroid: (1.25,3.58333)
Samples: (1.5,4.5)  (1,4)  (2,5)  (1,1.5)  (1,5)  (1,1.5)
Cluster 1 : Centroid: (4.8,2.6)
Samples: (4,2)  (5,2)  (4,3)  (5,3)  (6,3)
10 Cluster 0 :Centroid: (1.3,4)
Samples: (1.5,4.5)  (1,4)  (2,5)  (1,1.5)  (1,5)
Cluster 1 :Centroid: (4.8,2.6)
Samples: (4,2)  (5,2)  (4,3)  (5,3)  (6,3)
```

```
#include <iostream>  #include <cstdlib> #include <ctime>
#include <vector>  #include <cmath>
using namespace std;
class Cluster//聚类，每个聚类都包含两个属性，一个是簇心的属性（维数），另一个是距离本簇心最近的样本点
{ public:     vector <double> centroid;//存放簇心的属性（维数）
   vector <int> samples;//存放属于相同簇心样本的下标  };
double CalculateDistance(vector<double> a, vector<double> b)//计算两个向量之间的距离
{   int len1 = a.size();     int len2 = b.size();
   if(len1 != len2)  cerr<<"Dimensions must be same!!\n";
   double temp = 0;
   for(int i = 0; i  < len1; ++i)temp += pow(a[i]-b[i], 2);
   return sqrt(temp);
}
//max~iteration 表示最大的迭代次数，min~move~distance
vector<Cluster> KMeans(vector<vector<double> >data~set, int k, int max~iteration, double threshold)
{   int row~number = data~set.size();//数据的个数
   int col~number = data~set[0].size();//每个向量(属性)的维数
   //初始随机选取 k 个质心
   vector<Cluster> cluster(k);//存放 k 个簇心。vector<T> v(n,i) 形式，v 包含 n 个值为 i 的元素
   srand((int)time(0));
   for(int i = 0; i < k; ++i)
   {   int c = rand()%row~number;
      cluster[i].centroid = data~set[c];
         //把第 c 个作为簇心，并把它相应的属性赋值给 centroid
   }
   int iter = 0;  //iteration
   while(iter < max~iteration)
   {   iter++;
      for(int i = 0; i < k; ++i)
         cluster[i].samples.clear();
      //找出每个样本点所属的质心
      for(int i = 0; i < row~number; ++i)
      {   double min~distance = INT~MAX;
         int index = 0;
         for(int j = 0; j < k; ++j)  //计算离样本点 i 最近的质心
         {   double temp~distance = CalculateDistance(data~set[i], cluster[j].centroid);
            if(min~distance > temp~distance)
            {   min~distance = temp~distance;
               index = j;
            }
         }
         cluster[index].samples.push~back(i);
             //把第 i 个样本点放入，距离其最近的质心的 samples
      }
      double max~move~distance = INT~MIN;
      for(int i = 0; i < k; ++i)  //更新簇心
      {   vector<double> temp~value(col~number, 0.0);
         for(int num=0;num<cluster[i].samples.size();++num)
                                    //计算每个样本的属性之和
         {   int temp~same = cluster[i].samples[num];
            for(int j = 0; j < col~number; ++j)
               temp~value[j] += data~set[temp~same][j];
         }
        vector<double> temp~centroid = cluster[i].centroid;
        for(int j = 0; j < col~number; ++j)
cluster[i].centroid[j]=temp~value[j]/cluster[i].samples.size();
         //计算从上一个簇心移动到当前新的簇心的距离
         double temp~distance = CalculateDistance(temp~centroid, cluster[i].centroid);
         if(max~move~distance < temp~distance)
            max~move~distance = temp~distance;
      }
      if(max~move~distance < threshold)   break;
   }
   return cluster;
}
int main()
{   int threshold = 0.001;//当从上一个簇心移动到当前粗心的距离几乎不变时，可以结束。这里用 threshold 作为阈值
   int point~number;  cin>>point~number;
   vector <vector<double> >data~set(point~number, vector<double>(2, 0.0));
   for(int i = 0; i < point~number; ++i)
   {   for(int j=0;j<2;++j)  cin>>data~set[i][j];     }
   int col = data~set[0].size();
   vector<Cluster> cluster~res = KMeans(data~set, 2, 200, threshold);
   for(int i = 0; i < cluster~res.size(); ++i)
   {   cout<<"Cluster "<<i<<" : "<<endl;
      cout<<"\t"<<"Centroid: ";//<<endl;
      cout<<"(";
      for(int j=0;j<cluster~res[i].centroid.size()-1; ++j)
         cout<< cluster~res[i].centroid[j]<<",";
cout<<cluster~res[i].centroid[cluster~res[i].centroid.size()-1]<<")"<<endl;
      cout<<"\t"<<"Samples: ";
      for(int j = 0; j < cluster~res[i].samples.size(); ++j)
      {   int c=cluster~res[i].samples[j];   cout<<"(";
         for(int m = 0; m < col-1; ++m)
            cout<<data~set[c][m]<<",";
         cout<<data~set[c][col-1]<<")  ";
      }
      cout<<endl;
   }   return 0;
}    //~代表下划线
```