**目录**

1. **课题信息**············································3
2. **使用手册·**···········································3
3. **算法介绍·**···········································4
4. **程序设计****·**···········································4
5. **程序结构·**···········································8
6. **测试与调试·**·········································9
7. **思考与总结·**·········································10
8. **附录**················································11
9. **课题信息**

**实验平台：**Visual Studio 2013 Ultimate

题号：50

题目信息：

K-means（K均值）算法是一种聚类算法。实现K-means算法。要求输入一个班级同学3们以上课程的百分制成绩，通过K-means聚类输出A、B、C、D和E 5级制成绩。

1. **使用手册**
2. 将包含学生成绩的文件拷贝至该项目的路径下，数据文件的格式为：一个元组表示一名学生所有课程的成绩，每一列都表示一门课。
3. 运行该项目后，请根据提示来进行操作。第一步为：

Input file name contains datas we need:

即输入包含我们需要的数据的文件，使用的文件为：in.txt。如果输入的文件不在当前路径下，会给出错误信息：

error: XXX（file name） doesn't exist in current directory!

第二步为：

Input the number of courses:

即输入课程数，文件中最大课程数为4

第三步为：

Input the total datas:

即输入所有的数据量，如，假设有10个学生的4门课，则输入40

上述三步完成后，屏幕显示“starting clustering”，并显示迭代次数以及迭代结果。

以最后一次迭代结果作为聚类结果。

3.注：项目中的初始质心是手动生成的，如果要改为随机生成质心，请在void KMeans()函数体中将“/\*手动初始化质心\*/”下的语句注释掉，并去掉“/\*随机初始化质心\*/”下的注释。至于为什么会有两种初始化质心的方法，我会在之后的程序设计部分做解释。

1. **算法介绍**

K-means算法是机器学习中的一种非监督学习算法，该算法可以将一组数据分类。分类的具体步骤为：

1. 读取数据；
2. 将数据进行正规化；
3. 初始化质心K，此处的K值为5，即将数据分为5类；
4. 以欧式距离计算各个学生分别对五个中心点的相异度；

5.将每个学生分到距离当前中心点最近的簇，进行第一次聚类；

6.根据第一次聚类结果，将质心移动到它所在的簇的均值中心，重复4和5，直到聚类结果无变化。

1. **程序设计**

根据算法介绍中的步骤，可以将其用编程思想进一步细化。

**1.读取数据。**

在一开始的构想中，数据文本应该用表格的形式存储，其包含的信息有：学生姓名、课程名、以及对应的分数。成绩表如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 机器学习 | 计算机图形学 | 计算机网络 | 编译原理 |
| 迪迦 | 82 | 90 | 72 | 88 |
| 雷欧 | 81 | 76 | 66 | 87 |
| 怪兽 | 55 | 59 | 34 | 64 |
| 奥特之王 | 92 | 84 | 65 | 76 |
| 奥特之母 | 87 | 85 | 76 | 72 |
| 奥特之父 | 65 | 63 | 74 | 72 |
| 光之使者 | 34 | 36 | 55 | 58 |
| 梦比优斯 | 78 | 77 | 70 | 82 |
| 赛文 | 86 | 88 | 92 | 90 |
| 泰罗 | 95 | 83 | 92 | 84 |

这些数据在读入到程序后，可以存储在std::tuple中。由于我的codeblocks不支持C++11标准，因此便选择了Visual Studio 2013作为开发平台。

但是这种做法会出现一个问题：没有理想的存储元组的容器。一开始我使用std::vector<std::tuple>tuples来存储所有元组，但是这样无法遍历每一个元组的数据，因为std::tuple是不支持迭代器或者数组下标的访问方式的。

为此，我先用typedef std::vector<double> Tuple，再用std::vector<Tuple>tuples来存储元组。这是一种折中的方式，使用这种方法的影响是只能存储数字（即成绩），文件in.txt就只包含学生的成绩了：

82 90 72 88

81 76 66 87

55 59 34 64

92 84 65 76

87 85 76 72

65 63 74 72

34 36 55 58

78 77 70 82

86 88 92 90

95 83 92 84

该部分在主函数中完成。

**2.将数据进行正规化。**

进行正规化是为了降低数据值相差较大所带来的影响。如一个学生的成绩为1分，另一个学生的成绩为100分，那么100分的学生将影响相异度的计算，可能导致程序无法正确分类。

一般来说，只有当用欧氏距离来计算相异度时才需要将数据正规化。对每一个数据正规化的公式为：



其中，表示当前的数据，’表示的是正规化后的数据，max()和min()分别表示当前课程的最高分和最低分。

进行正规化后的数据为（此处仅保留两位有效数字，程序计算结果保存到小数点后六位）：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 机器学习 | 计算机图形学 | 计算机网络 | 编译原理 |
| 迪迦 | 0.83 | 1 | 0.66 | 0.94 |
| 雷欧 | 0.81 | 0.74 | 0.55 | 0.91 |
| 怪兽 | 0.36 | 0.43 | 0 | 0.17 |
| 奥特之王 | 1 | 0.89 | 0.53 | 0.56 |
| 奥特之母 | 0.91 | 0.91 | 0.72 | 0.44 |
| 奥特之父 | 0.54 | 0.5 | 0.69 | 0.44 |
| 光之使者 | 0 | 0 | 0.36 | 0 |
| 梦比优斯 | 0.76 | 0.76 | 0.62 | 0.75 |
| 赛文 | 0.90 | 0.97 | 1 | 1 |
| 泰罗 | 0.53 | 0.59 | 0.74 | 0.47 |

该部分由函数void normalize(vector<Tuple>&tuples)完成。

1. **初始化质心K。**

最初的方案是随机初始化质心。随机初始化的算法为：

1. 用rand()%tuples.size()随机生成[0,tuples.size()-1]的整数，
2. 判断该整数是否已生成，若已生成，则进入1，若未生成，则将该整数对应的元组的数据存入当前质心；该部分由函数bool selected(int n, vector<int>v, const vector<Tuple>tuples)完成；
3. 重复1和2，直到5个质心已全部被初始化。

在该方案下运行程序后，我发现了一个问题：学生不能被正确地评级，如，可能存在所有60分以下的同学被评为A的情况，这是因为该程序虽然能将学生的成绩分为5类，却不能根据分数的高低来进行评级。这是不符合实际情况的。 根据我在《机器学习基础》课程学到的知识，该课题应该被归类为监督学习下的分类问题：给定大量已经被分为“A-E”5个等级的学生的数据集合，训练一组新的学生成绩，将该组学生进行评为“A-E”5个等级。个人认为K-Means算法是不适用于此课题的。

上述问题可以通过“手动初始化质心”来解决，也就是初始化质心的第二个方案。该方案很简单：根据经验先选5个学生作为质心，以替代监督学习中的训练过程。

取赛文、迪迦、梦比优斯、奥特之父、光之使者作为5个簇的种子（正规化后的数据），即

A{0.90,0.97,1,1};

B:{0.83,1,0.66,0.94};

C:{0.76,0.76,0.62,0.75};

D:{0.54,0.5,0.69,0.44};

E:{0,0,0.36,0}

该部分在void KMeans(vector<Tuple>& tuples)函数中实现。

1. **以欧式距离计算各个学生分别对五个质心的相异度。**

相异度可简单理解为“两个属性间相异的程度的大小”。如人和大猩猩的相异度，人和狗的相异度，都可用向量或标量来表示，以通过计算机来计算。

相异度的计算方法有很多。常用的计算方法有曼哈顿距离、欧氏距离、闵可夫斯基距离。本课题用相对简单且常用的欧式距离来计算两个点之间的相异度，其他计算方法姑且不管。

欧式距离即两个坐标点之间的几何距离，具体到本课题，计算方法为：

循环遍历每一个质心，计算每一个质心和当前元组的距离。

该部分在double getDistance(const Tuple &t1, const Tuple &t2)函数中实现。

1. **将每个学生分到距离当前质心最近的簇，进行第一次聚类。**

该算法的实现过程为：用循环遍历每一个元组，调用聚类函数，将每一个元组都归类到距离它最近的质心。

1. **根据第一次聚类结果，将质心移动到它所在的簇的均值中心，重复4和5，直到聚类结果无变化。**

移动质心位置，移动质心的方法为：计算每一个簇的均值，将该均值作为新的质心。如，假设A簇包含的元组有{1,1,1,1}、{1 ,2,3,4}、{4,5,6,7}，则A簇的新的质心为{(1+1+4)/3,(1+2+5)/3,(1+3+6)/3,(1+4+7)/3}。

重复4和5，直到聚类过程收敛。

移动质心由Tuple moveCentroids(vector<Tuple> cluster)函数实现；

重复过程由一个while循环实现。

**五.程序结构**

在程序设计的基础上，进一步具体说明程序的结构及函数的作用：

**1.将数据进行正规化：**

调用double getMax(const vector<Tuple>& tuples,int col)函数得到每一列的最大值，

调用double getMin(const vector<Tuple>& tuples,int col)函数得到每一列的最小值，

调用void normalize(vector<Tuple>&tuples)函数根据公式对数据进行正规化。

**2.以欧式距离计算各个学生分别对五个质心的相异度：**

调用double getDistance(const Tuple &t1, const Tuple &t2)函数计算某个学生和五个质心的相异度。

**3.将每个学生分到距离当前中心点最近的簇，进行第一次聚类：**

调用char clusterOfTuple(Tuple means[],const Tuple &tuple)函数，返回值为每一个元组的标签，如第一个元组和第三个质心的距离最近，则返回值应为C（因为从第一到第五个质心的标签分别为A-E）。

**4.根据第一次聚类结果，将质心移动到它所在的簇的均值中心，重复4和5，直到聚类结果无变化：**

调用Tuple moveCentroids(vector<Tuple> cluster)函数以移动质心；

调用double getCentroids(vector<Tuple>clusters[], Tuple means[])函数以得到质心与它所在的簇的距离；

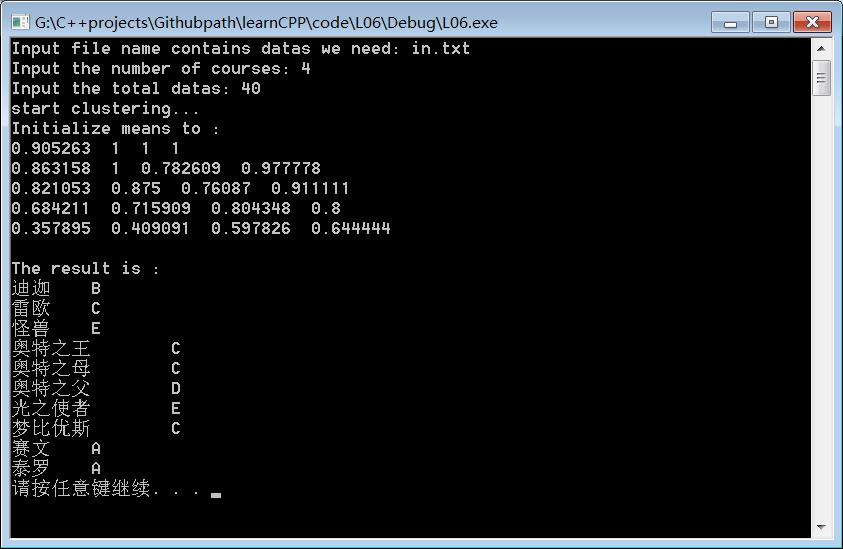
调用bool selected(int n, vector<int>v, const vector<Tuple>tuples)函数以判断新生成的随机数是否已经生成过。

注：2-4中包含的函数均在void KMeans(vector<Tuple>& tuples函数中调用；

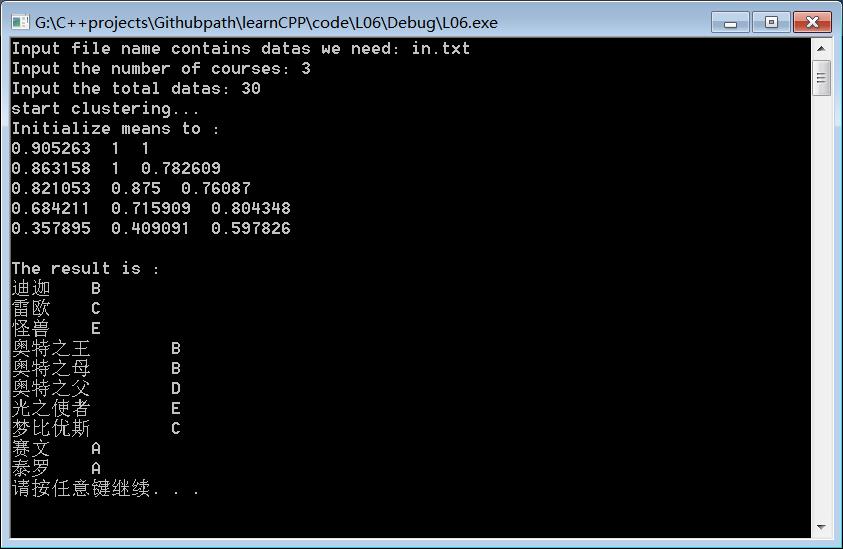
void print(Tuple means[], const vector<Tuple>& tuples)函数用于打印出最后的分类结果。

**六.测试与调试**

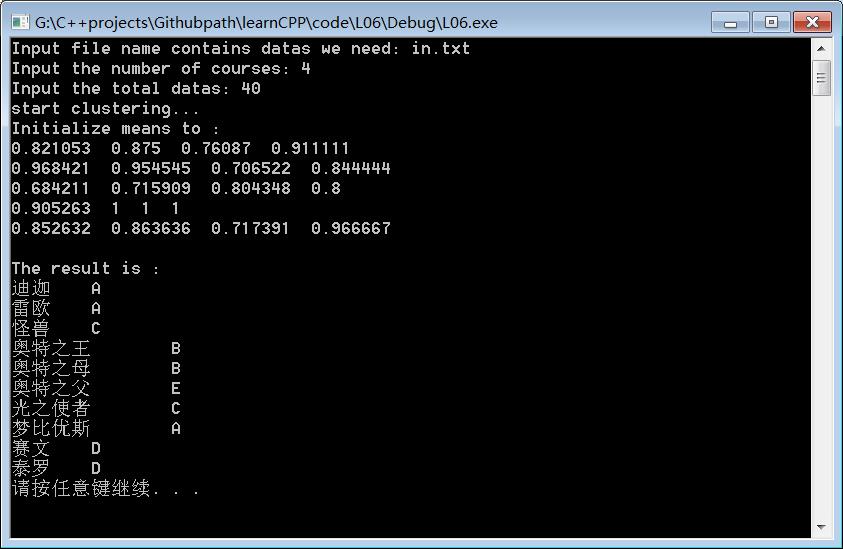
**1.手动生成质心的运行结果（有4门课参与比较）如图：**

****

1. **手动生成质心的运行结果（有3门课参与比较）如图：**

****

1. **随机生成质心的运行结果（有4门课参与比较）如图：**

****

**七.思考与总结**

1. 解决这个课题是基于“自上而下，逐步求精”的思想的，也就是说，从这个算法入手，每遇到一个问题就解决一个问题，但是这种方法给修改程序带来了一些困难：有时候解决了某个问题后，新出现的问题已经超出了我的能力范围了，这不得不使我修改原先的程序来寻求另一种解决途径。这也导致我花了一个星期的时间才勉强完成这个程序。

我觉得出现这种问题的原因是我没有设计好程序就开始动手，我应该先设计一个类，把所有可能需要的成员函数都写进去，以更好地掌控整个程序。

1. 应该加强对文件操作的训练。如读入或写入一个包含多种数据类型的文件。
2. 应该加强训练以熟练地使用STL。

## **附录**

**[参考资料]**

Andrew Ng 《Machine learning》 Stanford University

## 博客——《[算法杂货铺——k均值聚类(K-means)](http://www.cnblogs.com/leoo2sk/archive/2010/09/20/k-means.html)》

<http://www.cnblogs.com/leoo2sk/archive/2010/09/20/k-means.html>

# 博客——《深入浅出K-Means算法》

http://www.csdn.net/article/2012-07-03/2807073-k-means