Outlier Detection 实验报告

一、 实验内容

离群点是一个数据对象，它显著地不同于数据集中的其它对象，好像它是被不同的机制产生的一样。离群点检测，也称为异常检测，是找出其行为很不同于预期对象的过程。除欺诈检测外，离群点检测在很许多方面都有重要的应用，例如医疗处理，公共安全，工业损毁检测，图像处理，网络入侵检测和垃圾邮件识别等。

离群点检测现在有很多成熟的方法，可以按照不同方面进行分类。例如根据是否标记数据集以及标记的比例可以分为监督，半监督和无监督方法。根据对离群点与其余数据做出的假定，可以分为统计方法，基于邻近性的方法和基于聚类的方法。统计学方法对数据的正常性作出假定。它们假定正常的对象由一个统计模型产生，而不遵守该模型的数据是离群点。基于邻近性的方法假定一个对象是离群点，如果它在特征空间中的最近邻也原理它，即该对象与它的最近邻之间的邻近性显著地偏离数据集中其它对象与它们的近邻之间的邻近性。基于聚类的方法假定正常数据对象属于大的，稠密的簇，而离群点属于小或稀疏的簇，或者不属于任何簇。

这里主要是实现了基于距离的 CELL 离群点检测方法，并对其性能，正确性，可扩展性以及后续的优化等做了分析。

二、 算法简介

CELL 是一种基于距离的离群点检测的基于网格的方法。在这种方法中，数据空间被划分成多维网格，其中每个单元是一个其对角线长度为$\frac{r}{2}$的超立方体，其中 r 是一个距离阀值。如果有 L 维，则单元的每个边长为$\frac{r}{2\sqrt{l}}$。

考虑划分后的其中一个单元 C。单元 C 的近邻单元可以划分成两组。直接与 C 相邻的单元构成第 1 层单元，而在任意方向远离 C2$\sqrt{l}$个单元距离内的单元构成第 2 层单元，这两层单元具有如下性质：

- 第 1 层单元的性质：给定 C 的任意点 x 和第 1 层中的任意点 y，有 dist(x,y)<=r。
- 第 2 层单元的性质：给定 C 的任意点 x 和任意点 y，若 dist(x,y)>r，则 y 在一个第 2 层单元中。

设 a 是单元 C 的对象数，b1 是第一层单元中的对象数，b2 是第 2 层单元中的对象数，可以使用如下规则：

- 层 1 单元剪枝规则：根据第 1 层单元的性质，如果$a + b1 > \lceil \pi n \rceil$,则 C 中的每个对象 o 都不是$DB(r, \pi)$离群点，因为 C 和第 1 层单元中的所有对象都在 o 的 r 邻域中，并且至少有$\lceil \pi n \rceil$个这样的近邻。
- 层 2 单元剪枝规则：根据第 2 层单元的性质，如果$a + b1 + b2 < \lceil \pi n \rceil + 1$，则 C 中的所有对象都是$DB(r, \pi)$离群点，因为它们的 r 邻域中的其它对象都少于$\lceil \pi n \rceil$个。

使用以上两个规则，CELL 方法使用网格把主句分组-在一个单元中的所有对象形成一组。对于满足以上两个规则之一的组，可以确定单元中的所有对象都是离群点或者都不是离群点，因而不必逐个检查这些对象。此外，为了使用以上两个规则，只需要检查有限多个邻近的目标单元的单元，而不是整个数据集。使用以上两个规则，许多对象都可以确定为非离群点或离群点。只需要检查不能使用以上两个规则剪枝的那些对象。即使对于这样的对象 o，也只需要计算 o 与 o 的第 2 层中的对象的距

离。这是因为第一层中的所有对象到 o 的距离最多为 r，并且不在第 1 层或者第 2 层的对象到 o 的距离都超过 r，因而不可能在 o 的 r 邻域中。

三、 具体实现

1. 实现思想简介

首先按照输入格式随机生成数据集到一个文件中。然后读取文件的每一行作为一个数据点，计算这个数据点对应的块。为了节省内存，块中并不保存这个数据点的所有信息，而只保存该数据点的 ID 并增加块内点的计数，然后将这个数据点的具体信息写入块对应的文件中。如此处理输入文件中的所有点。

第二步是计算每个块 C 的数据点数目 a，层 1 邻近块中的数据点总数 b1 和层 2 邻近块中的对象总数 b2。a 在读入文件中数据点并分块的时候已经获得。为了获得 b1，首先需要计算块 C 的层 1 邻近块，然后累加这些邻近块中对象数目。同理可获得 b2。

第三步是对每个块 C 进行处理：

- 情况 1：如果$a+b1>\lceil \pi n \rceil$,则 C 中的每个对象 o 都不是$DB(r,\pi)$离群点.
- 情况 2：如果$a+b1+b2<\lceil \pi n \rceil+1$，则 C 中的所有对象都是$DB(r,\pi)$离群点。
- 情况 3：如果不属于以上两种情况，则需要计算块 C 内每个点到块 C 的层 2 邻近块中每个点的距离。

对于情况 3：初始块 C 内每个数据点的邻居数目都是 a+b1。每次处理块 C 层 2 的一个块 B，计算块 C 内每个数据点到块 B 每个数据点的距离，如果小于 r，则是一个 r 邻域内邻居，更新邻居数目，如果邻居数目达到阀值，则该数据点不是异常点，从块 C 中移除；如此处理每个层 2 块，都处理完之后仍在块 C 中的数据点都是异常点。

2. 程序简介

```
class DataNode{
private:
    int id;
    vector<int> attributes;                    //属性(坐标)，整型
public:
    DataNode(int id);
    DataNode(int id, vector<int> attr);
    DataNode(const DataNode &src);

    int getID()const;
    vector<int> getAttributes()const;
    void addAttributes(vector<int>::iterator being,vector<int>::iterator end);
    int getCellID(double width, int base);    //根据超立方体边长和坐标基数计算所在的块
    double distance(DataNode node);           //和另一个数据点的距离
};

class DataCell{
private:
    int id;
```

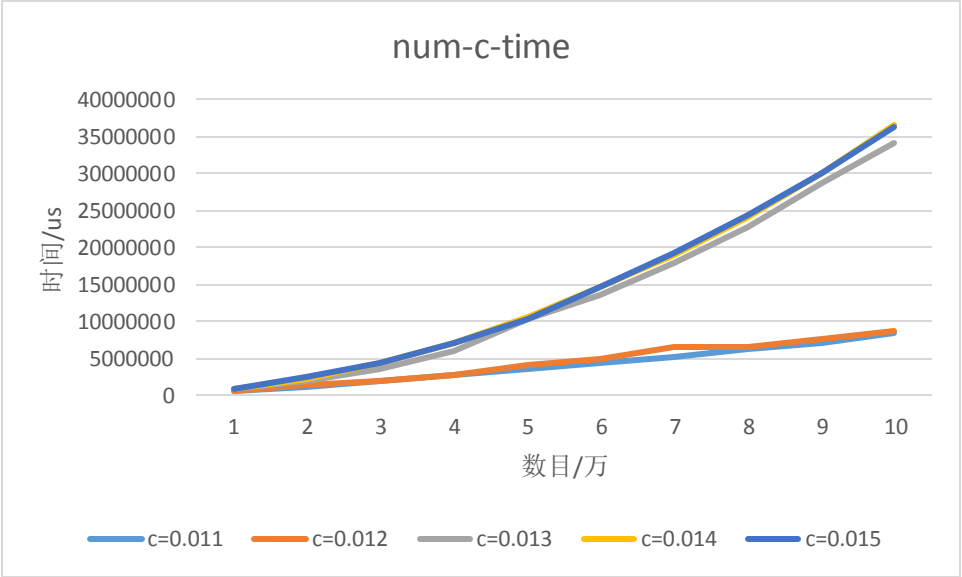
```
    int flag;                       //-1 表示块内的都是 outliers，0 表示不确定，1 表示都不是
    int cellNodeCount;              //块内数据点数目
    int layer1CellNodeCount;        //层 1 块的数据点数目
    int layer2CellNodeCount;        //层 2 块的数据点数目

    vector<DataNode> nodeList;      //块内数据点列表，为节省内存，只在需要计算数据点距离的时候从文件加载
    vector<int> nodeIdList;         //块内数据点 ID 列表
    vector<int> layer1CellIDList;   //层 1 块 ID 列表
    vector<int> layer2CellIDList;   //层 2 块 ID 列表

public:
    DataCell(int Id);
    DataCell(const DataCell &src);

    int getID()const;                       //获取块 ID
    int getFlag()const;                     //获取块标记
    int getCellNodeCount()const;            //获取块内数据点数目
    int getLayer1CellNodeCount()const;      //获取层 1 数据点数目
    int getLayer2CellNodeCount()const;      //获取层 2 数据点数目
    vector<DataNode> getNodeList()const;    //获取层内数据列表
    vector<int> getNodeIdList()const;       //获取层内数据点 ID 列表
    vector<int> getLayer1CellIDList()const; //获取层 1 块 ID 列表
    vector<int> getLayer2CellIDList()const; //获取层 2 块 ID 列表

    void setFlag(int f);
    void addDataNodeId(int id);             //增加一个数据点到块
    void removeDataNode(int id);            //移除块内的数据点
    void loadDataNodeFromFile(int dimension);//从文件中加载数据点坐标
    bool isLayer1Cell(int cellId);          //判断块是否属于层 1
    bool isLayer2Cell(int cellId);          //判断块是否属于层 2
    void addLayer1Cell(DataCell &cell);     //增加块到层 1
    void addLayer2Cell(DataCell &cell);     //增加块到层 2
    void calcLayer1CellList(vector<DataCell>& dataCellList, int base, int dimension);//计算层 1 块列表
    void calcLayer2CellList(vector<DataCell>& dataCellList, int base, int dimension);//计算层 2 块列表
};

class Outliers{
private:
    vector<int> outliers;               //outlier ID 列表
public:
```

```
    void addOutlier(int src);            //增加 outliers
    bool isOutlier(int index);           //判断是否为 outlier
    vector<int> getOutliers();           //获取 outliers ID 列表
};

class OutlierDetection{
private:
    vector<DataCell> dataCellList;       //数据块列表
    string inFilepath;                   //输入文件路径
    int numOfDataNode;                   //数据点总数
    int dimension;                       //坐标维度
    int max;                             //坐标上界
    int min;                             //坐标下界
    double percentage;                   //异常点百分数阀值
    double radius;                       //半径

public:
    OutlierDetection(string infile, int num, double percent, int dimen, double r, int maxm, int mini);
    void addCell(DataCell &src);             //增加数据块
    void addNodeToCell(int cellID, DataNode);//增加数据点到块
    void randomGenerate();                   //随机生成数据,坐标为整型
    void generateCell();                     //生成数据块
    void layerNodeCount();                   //计算每个块层 1 和层 2 数据点数目
    Outliers outlierDetection();             //异常点检测
};

int vectorToIntCellID(vector<int> id, int base);//将块 ID 转换为整型形式
vector<int> intToVectorCellID(int id, int base, int dimension);//将块 ID 转换为向量形式
vector<int> getLayerCellIdListByRecursive(vector<int> vId ,int base, int width);//通过递归计算邻近块的 ID 列表
int gaussRand(double e, double v);//高斯分布
void resultOutput(string path, vector<int> outliers,long cost_time);//输出结果到文件

主函数：
gettimeofday(&t_start, NULL);            //获取开始时间
//实例化异常检测类
OutlierDetection outlierDetection(inFile, numOfDatas, fracTotal, numOfDim, neighborRadius, 4, 0);
outlierDetection.randomGenerate();   //生成测试数据
outlierDetection.generateCell();         //根据测试数据生成块
outlierDetection.layerNodeCount();   //计算每个块层 1 和层 2 块数据点数目
outliers = outlierDetection.outlierDetection();//异常点检测
gettimeofday(&t_end, NULL);              //获取结束时间
cost_time = (t_end.tv_sec - t_start.tv_sec) * 1000000    + t_end.tv_usec - t_start.tv_usec;
resultOutput("out.txt",outliers.getOutliers(),cost_time);//将结果写入输出文件
```

四、 结果及分析

1. 运行时间与数据点数目以及阀值 c 的关系

数目/万	时间/us				
1	592371	631974	745120	832771	872929
2	1252184	1388116	1908203	2341488	2362799
3	1909577	2047630	3571271	4463743	4484139
4	2653796	2804525	6078828	7122113	7220706
5	3506632	4234814	10243689	10501775	10472693
6	4343657	4895695	13689231	14547886	14582734
7	5173298	6437681	17954605	19110918	19195334
8	6197175	6638845	22787546	24276277	24333831
9	7167364	7565054	28841312	30224084	30174465
10	8361840	8823329	34252996	36690439	36257209
	r=1	r=1	r=1	r=1	r=1
	dimen=2	dimen=2	dimen=2	dimen=2	dimen=2
	c=0.011	c=0.012	c=0.013	c=0.014	c=0.015

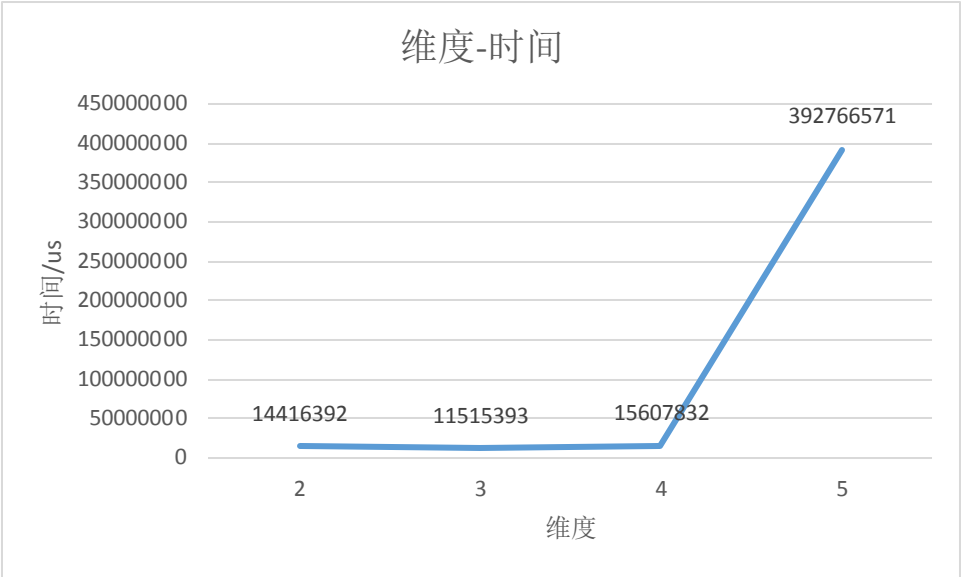


从图中可以看出，对于确定的 C，运行时间随着数据点数目的增加而增加，并且基本成线性关系。

对于确定数据点数目，C 越大运行时间越长。这是因为 C 越大，属于情况 1 的和情况 2 的数据点越少，属于情况 3 的数据点越多；而对于情况 3，由于需要从文件中读入数据点的具体信息，并对每个点计算距离，所需要花费的时间明显比较大。

2. 运行时间与维数的关系

维数	2	3	4	5
时间/us	14416392	11515393	15607832	392766571
num=100000, c=0.011, r=1				



从上图可以看出，随着维度的增长，运行时间会急速增加，基本成指数关系。这是因为随着维度的增加，块的数目成指数增加，基是坐标点最大值与最小值之差除以超立方体的边长，即$\frac{(max-min)*2\sqrt{l}}{r}$.

五、 后续优化

由于对于第三种情况的数据块需要从文件中加载然后计算每个点之间的距离，这里需要耗费巨大的时间，可以进行优化。

1 是针对文件加载，一次性加载所有的 2 层块，而不是每次处理一个，可以减少文件加载时间。2 是针对距离计算，将点与点之间的距离保存到内存中，如果之前计算过，直接从内存中读出即可，而不是重新加载文件进行计算。

六、 明细分工

罗远浩：设计类并完成 DataCell 和 OutlierDetection 类的编写
薛杰瑜：完成 DataNode 和 Outliers 类的编写并调试程序
刘振青：记录实验数据，撰写实验报告。
高枫：设计并完成了另一份程序。

七、 总结

罗远浩：通过理论学习，我加深了对异常点检测的理解，包括相关的概念，应用，以及实现方法，巩固了课堂上学到的知识；通过编写程序，提高了编程的能力，以及遇到问题时查找解决方法的能力；通过团队协作，我认识到合作的重要性，提高了和他人讨论问题，交流学习的能力。

刘振青：首先，认真学习了关于离群点检测的相关理论知识，对离群点检测有了比较系统的认识；然后，深入学习了本次大作业要用到的算法，加深了我对基于距离的离群点检测的认识；最后，通过撰写实验报告我对大作业完成的各个阶段进行了认真的总结和学习。通过这次的小组合作，我认识到自己的团队合作能力有待加强，此外，还认识到自己的编程能力和

小组中的其他成员存在较大的差距，因此，我需要积极向小组其他成员学习，提高自己的编程能力。
高枫：通过这次实验，了解了离群点检测的三种方法，并用 C++实现了基于单元的离群点检测。这种方法对于大规模数据能够有效减少时间复杂度，缺点是对于维度的增长比较敏感。
薛杰瑜：通过这次实验，将数据挖掘所学算法实现，更深入理解了离群点分析的原理。通过跟大家的分工合作，学习到了同学们的很多思考方法，让我获益匪浅。

小组成员：
薛杰瑜 2014E8013261174
罗远浩 2014E8013261184
高枫　2014E8013261148
刘振青 201428013229032