**南 京 师 范 大 学**

**《数据挖掘课程》**

**实验报告**



**题 目： K-means算法实现**

**专 业： 计算机科学与技术**

**姓 名： 蒋 泽**

**学 号： 19190248**

**仓库地址： https://gitee.com/jz19190248/data-mining.git**

**计算机与电子信息学院/人工智能学院**

**2022**  **年 4 月 19 日**

目录

[目录 0](#_Toc103000002)

[一、 实验内容 2](#_Toc103000003)

[1. 要求 2](#_Toc103000004)

[2. 输入格式 2](#_Toc103000005)

[3. 实验环境 2](#_Toc103000006)

[二、 算法实现 3](#_Toc103000007)

[1. 事物数据库的读取 3](#_Toc103000008)

[2. K-means算法 4](#_Toc103000009)

[算法流程 4](#_Toc103000010)

[3．可视化 9](#_Toc103000011)

[三、 实验结果 10](#_Toc103000012)

[测试文件生成 10](#_Toc103000013)

[测试1 12](#_Toc103000014)

[测试数据（老师提供） 12](#_Toc103000015)

[实验结果 12](#_Toc103000016)

[测试2 14](#_Toc103000017)

[测试数据（手动添加） 14](#_Toc103000018)

[实验结果 14](#_Toc103000019)

[测试3 15](#_Toc103000020)

[测试数据（k=3 100条数据） 15](#_Toc103000021)

[实验结果 16](#_Toc103000022)

[测试4（生成数据 k=5 1000条数据） 16](#_Toc103000023)

[数据 16](#_Toc103000024)

[结果 17](#_Toc103000025)

[附录：源码运行说明 18](#_Toc103000026)

# 实验内容

## 1. 要求

1. 使用Java语言实现

2. K-Means聚类程序的数据点为(x,y)，采用欧氏距离计算邻近性。

## 2. 输入格式

第一行：K值

第二行开始，每一行为一个数据点(x,y)

“--END--”为结束符号

数据文件名称为：test.txt

文件示例内容如下：  
2

1,2

3,4

10,9

22,3

--END--

## 3. 实验环境

IDE环境：IDEA2020

JDK：JDK13

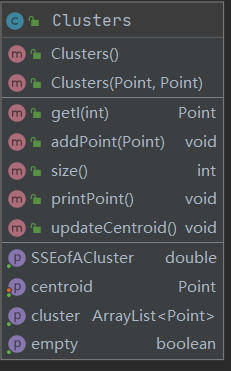
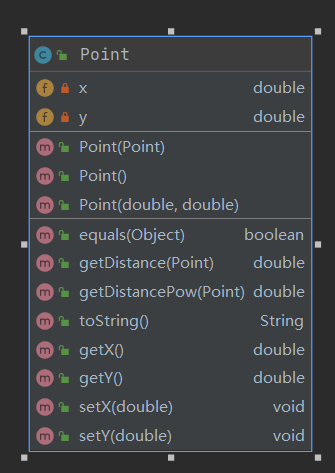
# 算法实现

本次实验算法实现可分为3个模块（1）事物数据库的读取；（2）Kmeans算法聚类；（3）可视化。

其中，kmeans算法为本次实验核心。

为了更便于体现面向对象编程思想，我首先对本次实验中的各个点进行了对象封装，主要有Point类，Cluster类两个实体类，以及其他的操作类

类的构建如下

## 事物数据库的读取

从文件中读取按行读取，第一行为K值；其余每行为一个点，通过逗号分隔为x与y坐标，直到遇到—END—结束，返回得到的所有点集与K值。

/\*\*

\* 读取文件

\*

\* @param input\_test\_file

\*/

static public ArrayList<Clusters> read\_file(String input\_test\_file) {

ArrayList<Clusters> allPoint = new ArrayList<Clusters>();

int i = 1;

Clusters clusters = new Clusters();

try {

BufferedReader br = new BufferedReader(new FileReader(new File(input\_test\_file)));

String line = "";

line = br.readLine();

K = Integer.parseInt(line);

while (!(line = br.readLine()).equals("--END--")) {

Point point = new Point();

point.setX(Double.valueOf(line.split(",")[0]));

point.setY(Double.valueOf(line.split(",")[1]));

if (i <= K - 1) {

allPoint.add(new Clusters(point, point));

i++;

} else {

clusters.addPoint(point);

}

}

clusters.setCentroid(clusters.getI(0));

allPoint.add(clusters);

br.close();

} catch (FileNotFoundException e) {

e.printStackTrace();

} catch (IOException e) {

e.printStackTrace();

}

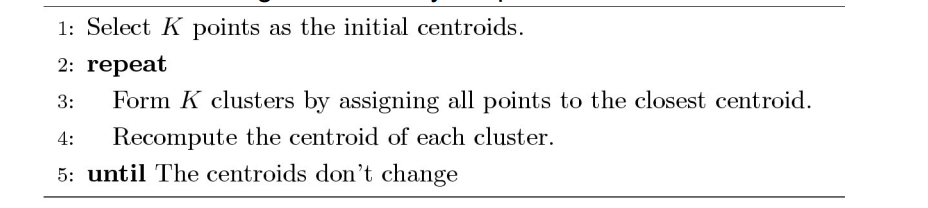
return allPoint;

}

## K-means算法

### 算法流程

如图，Kmeans算法过程较为简单。



程序的主程序如下：

private static void bootKmeans() {

init();

double preSSE = 0;

double currentSEE = getSSE();

while(preSSE!=currentSEE ){

SSEList.add(currentSEE);

// 更新每一轮的质心

updateCentroids();

// 根据计算的质心移动簇见的元素

updateClusters();

// 更新SSE

preSSE = currentSEE;

currentSEE=getSSE();

}

// 可视化展现

showAllpoints();

GUI.init(MyTool.getDataSet(allPoints),MyTool.getDataset(SSEList));

}

#### （1）初始化质心

选择K个点作为K个簇的质心，这k个点的选择是任意的，在我的实现过程中，将读入顺序的前K-1个点每个点分为一个簇，并且簇的质心就这个点，最后一个簇为剩余的所有点，质心为剩余点中的第一个点。

#### （2）更新簇

对于每个点，计算每个点到所有质心的位置，从而找出距离最小的质心，这里的距离采用欧几里得距离，即（）,找到距离最小的质心之后，将该点划到距离最近的质心所在的簇中。

private static void updateClusters() {

// 为了便于更新，定义一个map来理清楚簇与对应质心的关系

Map<Point, Clusters> clustersMap = new HashMap<>();

ArrayList<Point> cendroids = new ArrayList<>();

for (Clusters clusters : allPoints) {

cendroids.add(clusters.getCentroid());

}

for (Clusters clusters : allPoints) {

// clusters.printPoint();

int clusterSize = clusters.size();

for (int i = 0; i < clusterSize; i++) {

// 难点：对于每个簇中的每个点，计算点与质心的距离，并记录最小距离，

// 而后将点从本簇中移除，加入到另外一个最近质心所在的簇中；

// 如果加入到另外一个簇中不做标记的话，下次遍历这个簇的时候会被再次计算，形成死循环,

// 或者是将这些所有的簇移入到另外临时簇中，临时簇的个数不会大于K，占用的空间一直为点的个数。

int min\_flag = 0;

double minDistance = 999999999;

for (int j = 0; j < cendroids.size(); j++) {

double tempDistance = clusters.getI(i).getDistance(cendroids.get(j));

if (tempDistance < minDistance) {

min\_flag = j;

minDistance = tempDistance;

}

}

// 如果当前质心的簇没有形成，形成一个簇，添加进去

Point tempCentroid = cendroids.get(min\_flag);

if (!clustersMap.containsKey(tempCentroid)) {

clustersMap.put(tempCentroid, new Clusters(clusters.getI(i), tempCentroid));

} else {

Clusters temp = clustersMap.get(tempCentroid);

temp.addPoint(clusters.getI(i));

clustersMap.put(tempCentroid, temp);

}

}

}

allPoints.clear();

Iterator<Map.Entry<Point, Clusters>> it = clustersMap.entrySet().iterator();

while (it.hasNext()) {

Map.Entry entry = it.next();

allPoints.add((Clusters) entry.getValue());

}

}

需要注意的是，这里使用了Map结构，point作为Map的键，比较是否相同的依据不是对象地址相同，而是点坐标是否相同，所以需要重写Point的equal函数

#### （3）更新簇的质心

对于每个新更新完毕的簇，计算簇的新质心，即将簇中所有的点相加，请平均值，作为新的质心

//更新所有簇的质心

private static void updateCentroids() {

for (Clusters clusters : allPoints) {

clusters.updateCentroid();

}

}

/\*\*

\* 更新一个簇的质心

\*/

public void updateCentroid(){

double sumX = 0 ;

double sumY = 0;

for(Point point: cluster){

sumX += point.getX();

sumY+= point.getY();

}

double newx =(sumX/(double) cluster.size());

double newy = (sumY/(double) cluster.size());

setCentroid(new Point(newx,newy));

// System.out.println("aaa"+centroid);

// System.out.println("bbb"+cluster.get(0));

}

#### （4）判断聚类效果，决定是否继续聚类

判断聚类是否达到要求的指标采用的是SSE，即误差平方和，计算方式为，计算每个簇中所有的点到质心的距离的平方和，而后，将所有簇中的平方和相加即为SSE，若SSE收敛，则说明簇内点到质心的距离最小，达到聚类要求，停止循环；如果没有达到聚类要求，则返回第二步，继续迭代。

private static double getSSE() {

double SEE = 0;

for (Clusters clusters : allPoints) {

SEE += clusters.getSSEofACluster();

}

return SEE;

}

/\*\*

\* 获取一个cluster的SSE

\*

\* @param

\* @return

\*/

public double getSSEofACluster() {

double SSEofACluster = 0;

for (Point point : cluster) {

// System.out.print( "centroid"+centroid+"point "+point+ "point.getDistancePow()"+point.getDistancePow(this.centroid));

SSEofACluster += point.getDistancePow(this.centroid);

}

// System.out.println("centroid "+ centroid+" " +SSEofACluster);

return SSEofACluster;

}

## 3．可视化

为了更好的体现聚类效果，我学习了java中一个基于Swing的一个可视化图形库Jfreechart，能够快速绘制折线图、散点图等图表。

public static void init(XYSeriesCollection dataset1, CategoryDataset dataset2){

//创建一个主窗口来显示面板

JFrame frame = new JFrame("Kmeans");

frame.setLocation(10, 0);

frame.setSize(900, 900);

//实现简单的散点图，设置基本的数据

JFreeChart scatterChart = ChartFactory.createScatterPlot(

"聚类结果",// 图表标题

"X",

"Y",

dataset1,//数据集，即要显示在图表上的数据

PlotOrientation.VERTICAL,//设置方向

true,//是否显示图例

false,//是否显示提示

true//是否生成URL连接

);

//以面板显示

ChartPanel chartPanel = new ChartPanel(scatterChart);

chartPanel.setPreferredSize(new java.awt.Dimension(400, 350));

JFreeChart lineChart = ChartFactory.createLineChart(

"SEE迭代变化",//图名字

"迭代次数",//横坐标

"SSE值",//纵坐标

dataset2,//数据集

PlotOrientation.VERTICAL,

true, // 显示图例

true, // 采用标准生成器

false);// 是否生成超链接

ChartPanel lineChartPanel = new ChartPanel(lineChart);

lineChartPanel.setPreferredSize(new java.awt.Dimension(500, 350));

lineChartPanel.setLocation(30,400);

JPanel jPanel = new JPanel();

jPanel.add(chartPanel);

jPanel.add(lineChartPanel);

//将主窗口的内容面板设置为图表面板

frame.setContentPane(jPanel);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setVisible(true);

}

# 实验结果

运行说明，老师要运行代码时，此处考虑测试文件与源文件同一目录下，若不同环境下更改位置，请修改对应输入路径

## 测试文件生成

点数较少不方便观测聚类结果，所以可以写一个小程序批量生成点，所以编写了一个生成测试文件的Python小程序如下

*#createRamdomPoint.py*

*# 调用方法：python createRamdomPoint.py K*

import sys

import random

def main(argv):

K=argv[1]

x\_center = 50

y\_center = 50

length = 100

width = 100

file\_handle=open('test.txt','w')

file\_handle.write("%s\n"%K)

count = 0

while count < 100:

count += 1

x = x\_center + (random.random() - 0.5) \* length

y = y\_center + (random.random() - 0.5) \* width

print("%f,%f"%(x,y))

file\_handle.write("%f,%f\n"%(x,y))

file\_handle.write("--END--\n")

file\_handle.close()

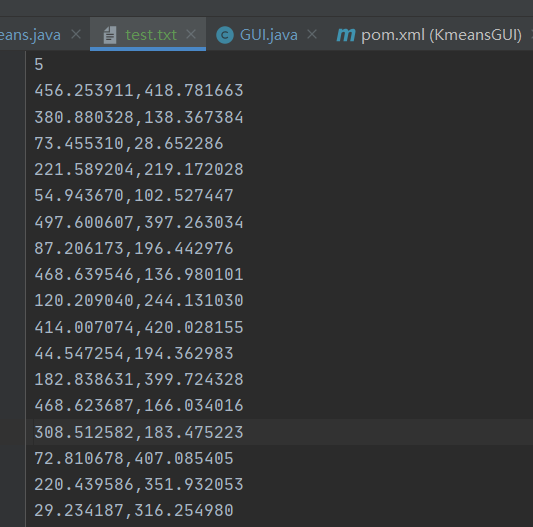
if \_\_name\_\_ == "\_\_main\_\_":

main(sys.argv)

使用是，只需要在调用 *python createRamdomPoint.py K ，*其中k为kmeans中的K，即可生成一个test.txt,其中包含100个在矩形（0,0）到（100，100）中的随机点，并符合老师提供的测试文件要求

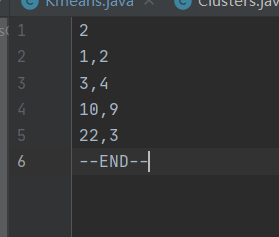
修改代码还能修改测试范围等，但是此处生成点的方式是随机数生成方式，不具备统计特性，生成的图形不一定够特殊。

生成数据示例



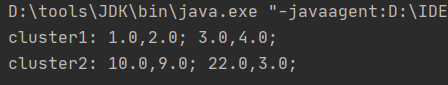
## 测试1

### 测试数据（老师提供）

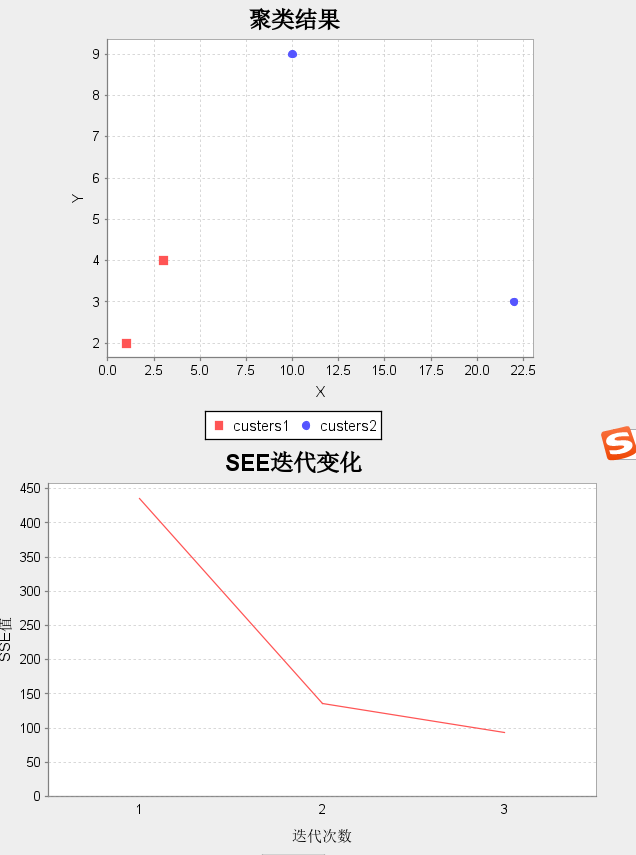


### 实验结果

控制台输出

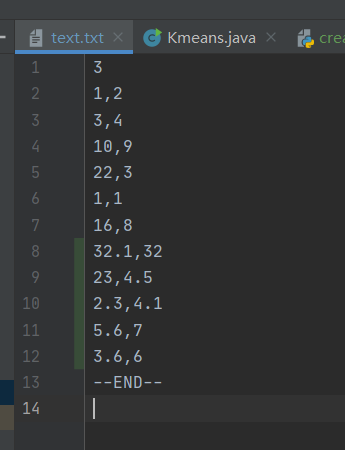


可视化结果



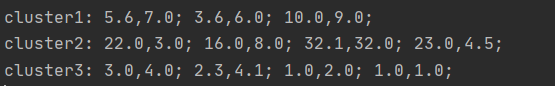
## 测试2

### 测试数据（手动添加）

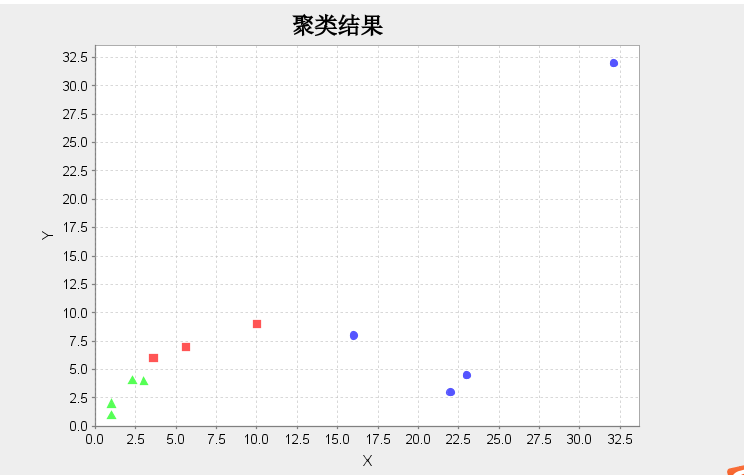


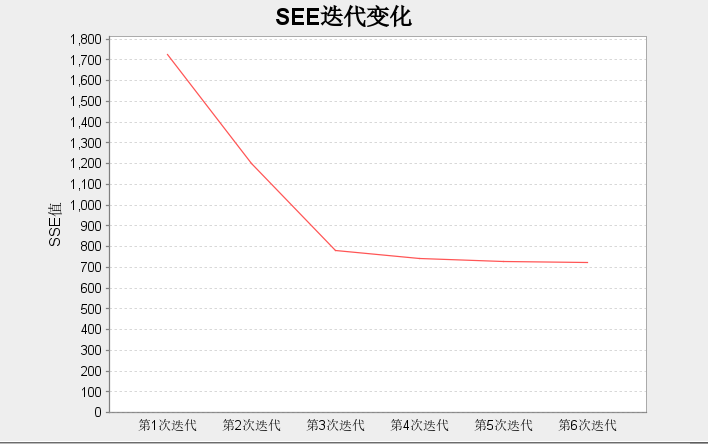
### 实验结果

控制台结果



可视化结果

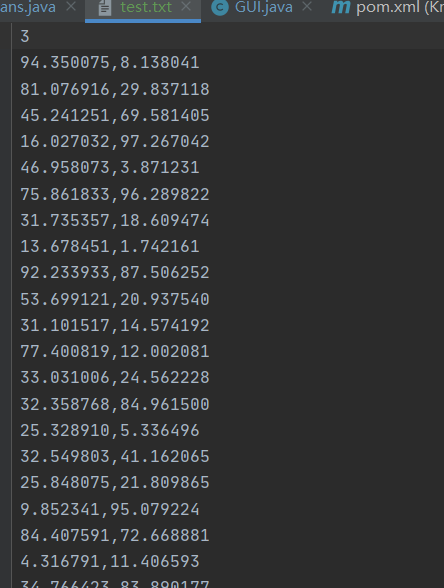




## 测试3

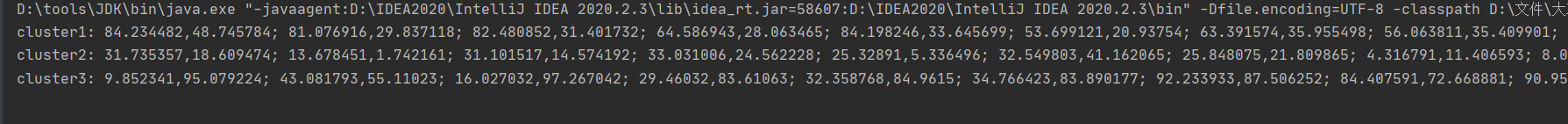
### 测试数据（k=3 100条数据）

生成数据测试

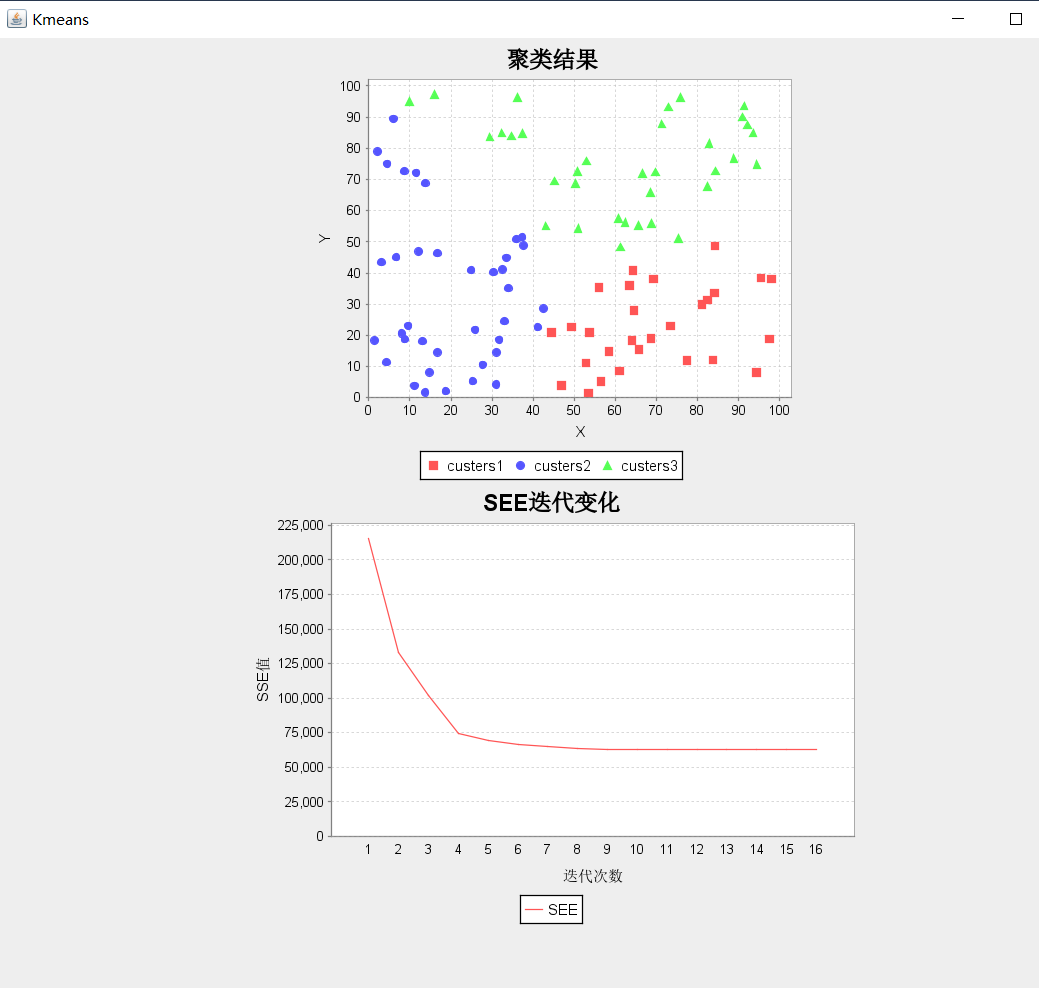


### 实验结果

控制台



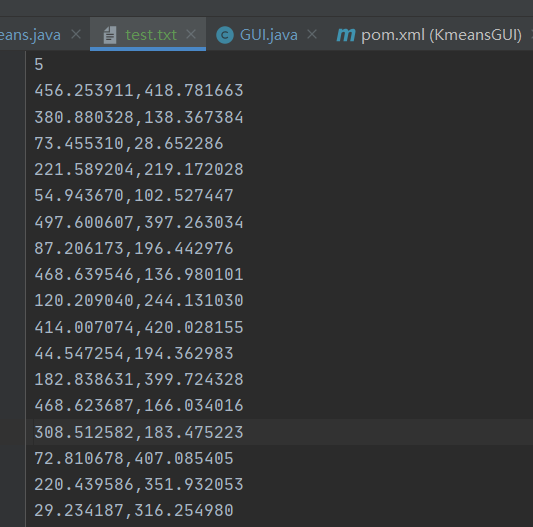
可视化结果



## 测试4（生成数据 k=5 1000条数据）

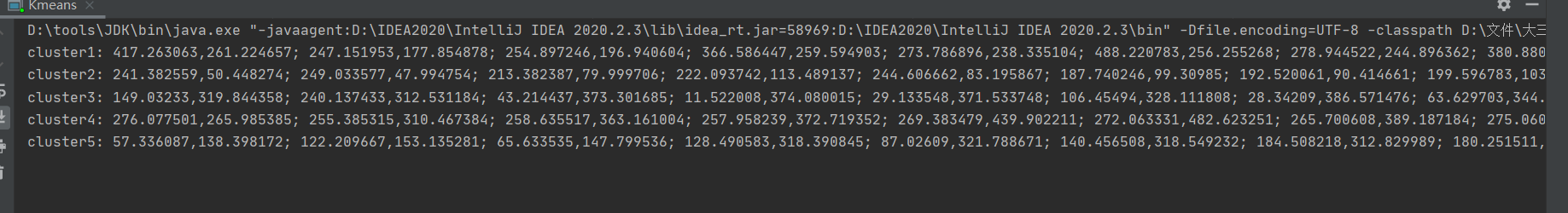
### 数据

一千条数据，k =5;数据点落在（0,0）到（500,500）的矩阵当中

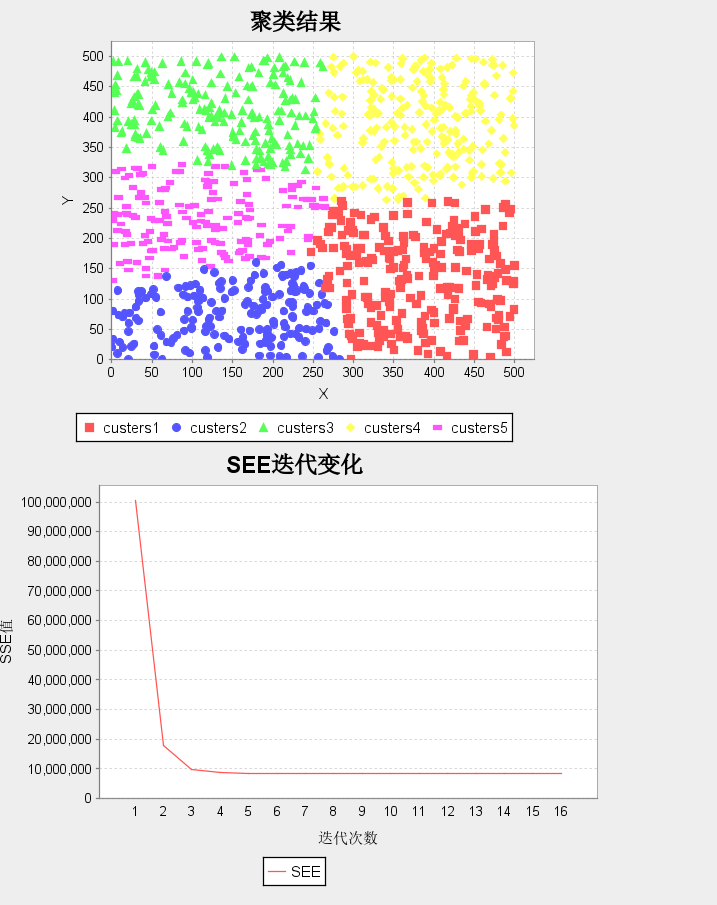


### 结果

控制台



可视化



附录：源码运行说明

项目是采用Maven构建，结构源码结构如下老师运行时需要下在maven里更新一下两个GUI使用到的包，测试文件在source下，如果变换位置，可以在Kmean.Java中修改对应的读取文件路径。

