DBSCAN

0. Introduction

- Lecture: 데이터사이언스(ITE4005)
- Student: 한양대학교 컴퓨터소프트웨어학부 2017030191 이현지
- Programming Assignment #3: Perform clustering on a given data set by using DBSCAN.

1. Main Concepts

1. Summary

1. Main Idea

> input file로부터 모든 점을 가져와 eps와 minPts를 기준으로 core points를 나누고 클러스터링을 실행한다. 클러스터링을 할 때에는 boarder points도 함께 추가하고 noise points(outlier)를 제거한다.

2. Epsilon: neighbor points와의 거리를 나타내는 최소 이웃 반경
   - 두 점 사이 거리 계산 공식

$$l = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

   - 두 점 사이의 거리가 Epsilon보다 작아야 neighbor라고 할 수 있음
3. Min Pts: 최소 이웃 수
   - 한 점으로부터 Epsilon 이내에 있는 neighbor들의 개수가 minPts 이상이면 core point라고 할 수 있음

2. Main.class

1. Get data and preprocess

```
String inputFile = args[0];
Integer n = Integer.parseInt(args[1]);
Double eps = Double.parseDouble(args[2]), minPts = Double.parseDouble(args[3]);

ArrayList<Pair<Double, Double>> points = getPoints(inputFile);
ArrayList<Integer> corePoints = getCorePoints(points, eps, minPts);
ArrayList<ArrayList<Integer>> neighbors = getNeighbors(points, eps);
```

   - inputFile: 소스 코드 실행 시 인자로 받는 input 파일의 이름
     - 경로를 명시하지 않으면 jar파일과 같은 경로에 존재
   - n: 클러스터 수
   - eps: neighborhood의 최대 반경
   - minPts: 최소 neighborhood 수
   - points: inputFile에 담긴 점들의 2차원 좌표를 담은 리스트
   - corePoints: points 중 core points의 index만을 담은 리스트

- neighbors: 각 점들의 neighbors의 index를 담은 리스트

2. Run BFS for core points

```
ArrayList<Boolean> isVisit = new ArrayList<>(Collections.nCopies(points.size()+5, false));
ArrayList<ArrayList<Integer>> clusters = new ArrayList<>();

for(int i = 0 ; i < corePoints.size() ; i++){
    Integer nowCorePointIndex = corePoints.get(i);
    if(!isVisit.get(nowCorePointIndex)){ // not visited
        Queue<Pair<Integer,Pair<Double, Double>>> qu = new LinkedList<>();
        qu.add(new Pair<>(nowCorePointIndex, points.get(nowCorePointIndex)));
        isVisit.set(nowCorePointIndex, true);

        ArrayList<Integer> nowCluster = new ArrayList<>();
        while(!qu.isEmpty()){
            Integer nowIndex = qu.peek().getFirst();
            Pair<Double, Double> nowPoint = qu.peek().getSecond();
            nowCluster.add(nowIndex);
            qu.poll();

            // add neighbors
            for(int j = 0 ; j < neighbors.get(nowIndex).size() ; j++){
                Integer nowNeighbor = neighbors.get(nowIndex).get(j);
                if(!isVisit.get(nowNeighbor) && !corePoints.contains(nowNeighbor)){
                    nowCluster.add(nowNeighbor);
                    isVisit.set(nowNeighbor, true);
                }
            }

            for(int j = 0 ; j < corePoints.size() ; j++){
                Integer nextCorePointIndex = corePoints.get(j);
                if(!nowCorePointIndex.equals(nextCorePointIndex) && !isVisit.get(nextCorePointIndex)){ // not visited
                    Pair<Double, Double> nextPoint = points.get(nextCorePointIndex);
                    if(isDensityReachable(nowPoint, nextPoint, eps)) {
                        isVisit.set(nextCorePointIndex, true);
                        qu.add(new Pair<>(nextCorePointIndex, nextPoint));
                    }
                }
            }
        }
        if(nowCluster.size() > 1){
            clusters.add(nowCluster);
        }
    }
}
```

- corePoints를 기준으로 BFS를 실행해서 클러스터를 형성한다. 각 corePoints를 클러스터에 추가할 땐 그 neighbor(boarder)도 함께 추가한다.
- nowCluster의 사이즈가 1이면 outlier로 판단하여 추가하지 않는다.
- isVisit: 반복적으로 접근하는 것을 막기위해 방문을 체크하는 리스트

3. Remove noise

```
ArrayList<ArrayList<Integer>> realClusters = new ArrayList<>();
for(ArrayList<Integer> cluster : clusters){
    Boolean hasCore = false;
    for(Integer point : cluster){
        if(corePoints.contains(point)) hasCore = true;
    }
    if(hasCore){
        realClusters.add(cluster)
    }
}
```

- 2번에서 얻은 clusters에서 noise를 제거한 realClusters를 얻는다.

4. Remove small cluster

```
if(realClusters.size() > n){
    ArrayList<Integer> sizeClusters = new ArrayList<>();
    for(ArrayList<Integer> cluster : realClusters){
        sizeClusters.add(cluster.size());
    }
    Collections.sort(sizeClusters);
    Integer realClusterSize = realClusters.size();
    for(int i = 0 ; i < realClusterSize-n ; i++){
        Integer nowSize = sizeClusters.get(i);
        for(int j = 0 ; j < realClusters.size() ; j++){
            if(realClusters.get(j).size() == nowSize){
                realClusters.remove(j);
                break;
            }
        }
    }
}
```

- 총 클러스터 수가 n보다 크다면 개수가 적은 클러스터는 삭제한다.
- sizeClusters: 3번에서 얻은 realClusters의 각 사이즈를 담은 리스트

5. Make output file

```
getResult(realClusters, inputFile, n);
```

- 정해진 형식에 맞추어서 output file을 만든다.

2. Details

1. getCorePoints()

1. Specification
    - Paramter: ArrayList<Pair<Double, Double>> points, Double eps, Double minPts
    - Return type: ArrayList<Integer>
2. Description

> 💡 points들 사이의 eps를 계산하여 neighbor 수가 minPts 이상인 core points를 찾는다.

3. Code

```
public static ArrayList<Integer> getCorePoints(ArrayList<Pair<Double, Double>> points, Double eps, Double minPts){
    ArrayList<Integer> corePoints = new ArrayList<>();
    for(int i = 0 ; i < points.size() ; i++){
        Pair<Double, Double> nowPoint = points.get(i);
        int neighbors = 0;
        for (Pair<Double, Double> nextPoint : points) {
            if (isDensityReachable(nowPoint, nextPoint, eps)) {
                neighbors++;
            }
        }
        if(neighbors >= minPts){
            corePoints.add(i);
        }
    }
    return corePoints;
}
```

2. getNeighbors()

1. Specification
    - Parameter: ArrayList<Pair<Double, Double>> points, Double eps
    - Return type: ArrayList<ArrayList<Integer>>
2. Description

> 💡 points들 중 eps를 계산하여 neighbor인 point를 체크한다.

3. Code

```
public static ArrayList<ArrayList<Integer>> getNeighbors(ArrayList<Pair<Double, Double>> points, Double eps){
    ArrayList<ArrayList<Integer>> neighbors = new ArrayList<>();
    for(int i = 0 ; i < points.size() ; i++){
        neighbors.add(new ArrayList<>());
    }

    for(int i = 0 ; i < points.size() ; i++){
        Pair<Double, Double> nowPoint = points.get(i);
        for(int j = 0 ; j < points.size() ; j++){
            Pair<Double, Double> nextPoint = points.get(j);
            if(isDensityReachable(nowPoint, nextPoint, eps)){
                neighbors.get(i).add(j);
                neighbors.get(j).add(i);
            }
        }
    }
    return neighbors;
}
```

3. getDistance()

1. Specification
    - Parameter: Pair<Double, Double> leftPoint, Pair<Double, Double> rightPoint
    - Return type: Double
2. Description

> 💡 두 점 사이의 거리를 구하는 공식(1-2번 참고)을 사용하여 leftPoint와 rightPoint 사이의 거리를 구한다.

3. Code

```
public static Double getDistance(Pair<Double, Double> leftPoint, Pair<Double, Double> rightPoint){
    Double leftX = leftPoint.getFirst(), leftY = leftPoint.getSecond();
    Double rightX = rightPoint.getFirst(), rightY = rightPoint.getSecond();

    return Math.sqrt(Math.pow((leftX-rightX),2) + Math.pow((leftY-rightY),2));
}
```

4. isDensityReachable()

1. Specification
    - Parameter: Pair<Double, Double> leftPoint, Pair<Double, Double> rightPoint, Double eps
    - Return type: Boolean
2. Description

> 💡 getDistance(2-3번 참고) 함수를 사용하여 left point로부터 right point가 density reachable한지 (neighbor인지) 구한다.

3. Code

```
public static Boolean isDensityReachable(Pair<Double, Double> leftPoint, Pair<Double, Double> rightPoint, Double eps){
    return getDistance(leftPoint, rightPoint) <= eps;
}
```

5. getPoints()

1. Specification
   - Parameter: String inputFile
   - Return type: ArrayList<Pair<Double, Double>>
2. Description

> 💡 inputFile로부터 모든 점의 2차원 좌표를 Pair 형태로 저장한다.

3. Code

```
public static ArrayList<Pair<Double, Double>> getPoints(String inputFile){
    ArrayList<Pair<Double, Double>> points = new ArrayList<>();
    try {
        BufferedReader reader = new BufferedReader(new FileReader(inputFile));
        String line = reader.readLine();

        while(line != null){
            StringTokenizer tokenizer = new StringTokenizer(line, "\t");

            Integer index = Integer.parseInt(tokenizer.nextToken());
            Double x = Double.parseDouble(tokenizer.nextToken());
            Double y = Double.parseDouble(tokenizer.nextToken());

            points.add(new Pair<>(x,y));
            line = reader.readLine();
        }
        reader.close();
    } catch (Exception e) {
        e.printStackTrace();
        System.exit(0);
    }
    return points;
}
```

6. getResult()

1. Specification
   - Parameter: ArrayList<ArrayList<Integer>> clusters, String inputFile
   - Return type: void
2. Description

> 💡 inputFile의 이름을 활용하여 outputFile을 정해진 형식에 맞춰 생성한다.

3. Code

```
public static void getResult(ArrayList<ArrayList<Integer>> clusters, String inputFile){
    String outputFile = inputFile.split("\\.")[0];
    for(int i = 0 ; i < clusters.size() ; i++){
        ArrayList<Integer> cluster = clusters.get(i);
        try {
            BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile + "_cluster_" + i + ".txt"));

            for(Integer point : cluster){
                writer.write(String.valueOf(point));
                writer.newLine();
            }
            writer.close();

        } catch (Exception e) {
            e.printStackTrace();
            System.exit(0);
        }
    }
}
```

3. How to Execute

1. Environment

- OS: Mac OS Catalina 10.15.7
- Runtime: JDK 15.0.1
- IDE: IntelliJ Ultimate 2020.03

2. Structure

execution - clustering.jar

ㄴ input1.txt

ㄴ input2.txt

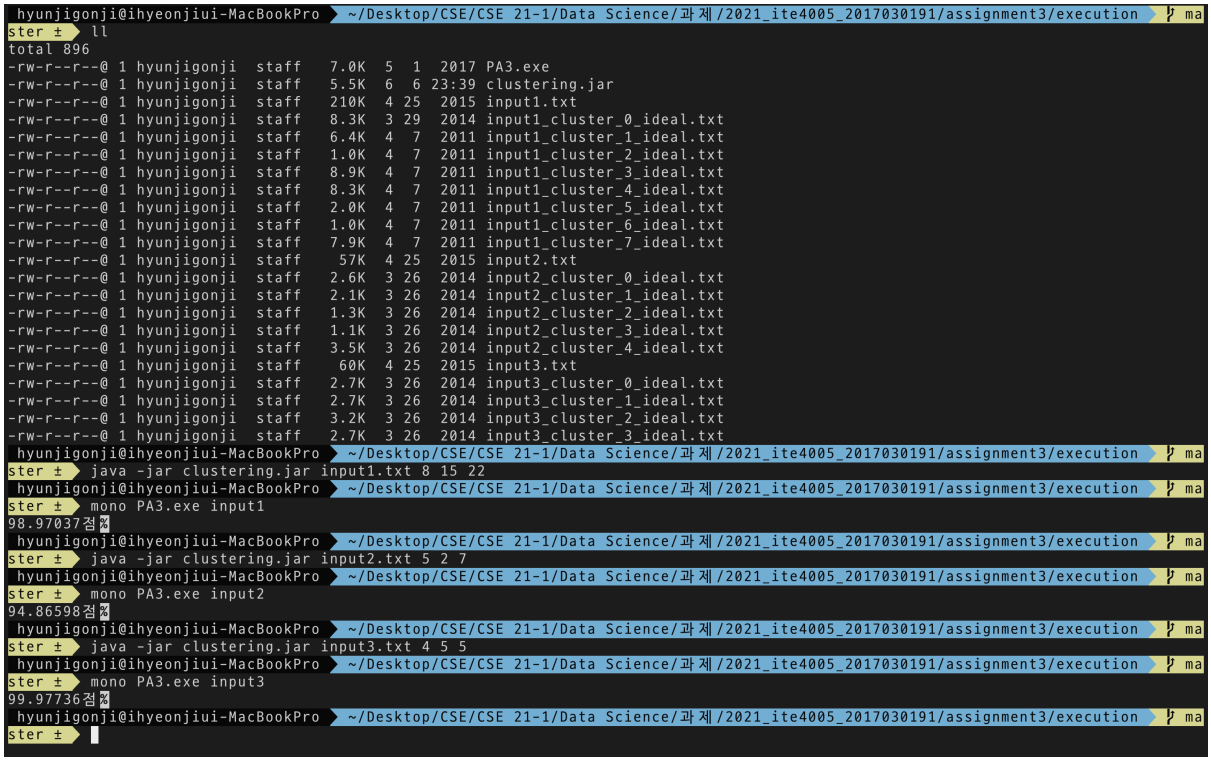
ㄴ input3.txt

project - src - cse.ds - Main.class

ㄴ Pair.class

ㄴ out - artifacts - clustering.jar

3. Usage



- execution 폴더의 clustering.jar 파일로 실행 가능

```
% java -jar clustering.jar {input_file} {n} {eps} {minPts}

ex) java -jar clustering.jar input1.txt 8 15 22
```