Progetto di Gestione Dell'Informazione Geospaziale - DBSCAN

Damiano Bianda

30 dicembre 2018

Sommario

In questo progetto si vuole implementare l'algoritmo di clustering DBSCAN tramite Java.
Dato un insieme di coordinate proiettate, queste vengono classificate secondo il cluster d'appartenenza o come outliers.
Infine i risultati ottenuti sono rappresentati attraverso una mappa utilizzando il software QGIS.

1 Algoritmo

1.1 Introduzione

DBSCAN è un algoritmo di clustering partitivo basato sulla densità. I principali vantaggi sono che è possibile scoprire clusters di forme arbitrarie (dimensioni e forme differenti) ed identificare gli outliers, ossia i punti all'infuori di un area densamente popolata e quindi non appartenenti a nessun cluster. I svantaggi sono invece che essendo un algoritmo parametrico è necessario definire dei parametri in base al tipo di dati da analizzare e che alcuni dataset presentano il problema della densità variabile, ossia sono presenti più cluster ma a densità diverse e quindi dati dei parametri non è possibile in un esecuzione identificarli tutti.

1.2 Definizioni

DBSCAN è parametrizzato tramite ε e MinPoints.
Il vicinato di un punto sono tutti i punti che ricadono nel cerchio con centro pari alla sua coordinata e raggio ε.
Un punto è detto:

- core point se il suo vicinato contiene almeno MinPoints elementi.
- border point se non è un core point, ma è nel vicinato di uno o più core point

- noise point se non è nè core point, nè border point

Le definizioni seguenti descrivono un rapporto di connessione tra i punti e servono per definire il concetto di cluster:

- directly density-reachable
  un punto p è detto directly density-reachable da un punto q se p è nel vicinato di q e se q è un core point

- density-reachable
  un punto p è detto density-reachable da un punto q se c'è una serie di punti, in cui il primo è q e l'ultimo è p, dove ogni elemento è directly density-reachable dal precedente

- density-connected
  un punto p è detto density-connected ad un punto q se c'è un punto r tale che p e q sono density-reachable da r

Quindi un cluster è un insieme massimo di punti density-connected.

1.3 Esecuzione di DBSCAN

DBSCAN itera su tutti i punti presenti nel dataset e per ognuno si determina se è un core point.
In caso positivo si crea un cluster che viene espanso coi punti presenti nel suo vicinato.
Il processo si ripete iterativamente controllando i punti appena aggiunti, se a loro volta sono core point viene aggiunto il loro vicinato e così via fin quando tutti i punti density-reachable sono stati inglobati nel cluster.
Il vicinato non viene aggiunto al cluster quando un punto è border point, poichè non esistono punti directly density-reachable da questo.
Una volta terminato l'algoritmo si ottiene un dataset con i suoi elementi etichettati secondo i differenti cluster o come noise.

2 Implementazione

2.1 Pseudocodice

```
NOISE := 0

void initDataSet(dataset):
        foreach point in points:
                point.cluster := NOISE

bool isCorePoint(neighborhood, minPoints):
        return neighborhood.size() >= minPoints

List<Point> neighborhood(points , point, epsilon):
        neighbors = [ ]
        foreach candidateNeighbor in points:
                if distance(candidateNeighbor, point) <= epsilon:
                        neighbors.add(candidateNeighbor)
        return neighbors

float distance(pointA, pointB):
        return sqrt((pointA.x − pointB.x)^2 + (pointA.y − pointB.y)^2))

void DBSCAN(points, epsilon, minPoints):
        initDataSet(points)
        clusterLabel := 1
        foreach point in points:
                if point.cluster = NOISE:
                neighbors := neighborhood(points, point, epsilon)
                if isCorePoint(neighborhood, minPoints):
                        foreach neighbor in neighbors:
                        neighbor.cluster := clusterLabel
                        neighbors.remove(point)
                        while neighbors.size() > 0:
                                currentPoint := neighbors.getFirst()
                                currentNeighborhood := neighborhood(points,
                                    ↪ currentPoint, epsilon)
                                if isCorePoint(currentNeighborhood, currentPoint):
                                        foreach currentNeighbor in
                                            ↪ currentNeighborhood:
                                            if currentNeighbor.cluster :=
                                                ↪ NOISE
                                                neighbor.add(
                                                    ↪ currentNeighbor)
                                                neighbor.cluster :=
                                                    ↪ clusterLabel
                        clusterLabel := clusterLabel + 1
```

Inizialmente ogni punto del dataset non appartiene a nessun cluster, quindi durante la lettura dei dati vengono classificati come NOISE.

Successivamente si itera su ogni punto, quelli che hanno ancora la label NOISE sono candidati a creare un cluster, quindi si controlla se rispettano le condizioni per essere identificati come border points, in caso negativo l'elemento rimane NOISE e potrà essere inglobato in un altro cluster durante l'esecuzione dell'algoritmo.
Se invece un punto risulta essere un core point, a questo punto è necessario espandere il cluster tramite un approccio breadth first.
Viene mantenuta una coda che inizialmente contiene solo i nodi directly density-reachable da quello di partenza, fino a quando la coda non è vuota si estrae il primo elemento ed il suo vicinato.
Se il nuovo elemento estratto risulta anch'esso essere un core point gli elementi del vicinato che non appartengono ancora a nessun cluster vengono assegnati al label corrente e inseriti nella coda.
Quando la coda è vuota significa che ogni elemento density-reachable a partire da quello iniziale è stato assegnato al cluster, quindi viene incrementato l'id globale che identifica il cluster corrente e si procede a iterare sui nodi ancora classificati come NOISE.

2.2 Codice

```
public static void DBSCAN(ArrayList<Coordinate> points, float eps, int minPoints
    ↪ ){
        int clusterLabel = Coordinate.NOISE + 1;
        for (Coordinate point: points){
                if(point.isNoise()){
                        final ArrayList<Coordinate> queue = neighborhood(points,
                            ↪ point, eps);
                        if(queue.size() < minPoints) {
                                continue;
                        }
                        for(Coordinate c: queue){
                                c.setLabel(clusterLabel);
                        }
                        queue.remove(point);
                        point.setLabel(clusterLabel); // credo che non serve
                        while (!queue.isEmpty()){
                                final Coordinate currentPoint = queue.remove(0);
                                final ArrayList<Coordinate> currentNeighborhood
                                    ↪ = neighborhood(points, currentPoint, eps);
                                if (currentNeighborhood.size() >= minPoints){
                                        for(Coordinate currentNeighbor:
                                            ↪ currentNeighborhood){
                                                if(currentNeighbor.isNoise()){
                                                        queue.add(currentNeighbor
                                                            ↪ );
                                                        currentNeighbor.setLabel(
                                                            ↪ clusterLabel);
```

```
                                        }
                                    }
                                }
                            }
                            clusterLabel++;
                        }
                    }
                }
```