# Progetto di Gestione Dell'Informazione Geospaziale - DBSCAN

Damiano Bianda

3 gennaio 2019

**Sommario**

Dato un insieme di punti nello spazio, la cui posizione è determinata da coordinate proiettate, si voglio identificare i clusters presenti ed assegnare ogni elemento ad ognuno di essi o etichettarlo come rumore.
L'algoritmo DBSCAN è un algoritmo partitivo basato sulla densità che permette di raggiungere l'obiettivo prefissato, quindi in questo progetto ne è stata implementata una versiona Java per analizzare un insieme di posizioni.
Infine i risultati ottenuti sono stati rappresentati su una mappa utilizzando il software QGIS.

# 1 Algoritmo

## 1.1 Introduzione

DBSCAN è un algoritmo di clustering partitivo basato sulla densità, questo significa che è in grado di assegnare ogni punto appartente ad un dataset ad un solo cluster, oppure identificarlo come rumore qualora si trovasse in una zona nello spazio dove la densità di elementi è bassa.
I principali vantaggi sono che è possibile scoprire clusters di forme arbitrarie (dimensioni e forme differenti) ed identificare i dati che rappresentano rumore, 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 di densità diverse e quindi dato un set di parametri non è possibile identificarli tutti in modo corretto.

## 1.2 Definizioni

DBSCAN è parametrizzato tramite $\varepsilon$ e MinPoints.
Il vicinato di un punto sono tutti i punti che ricadono nel cerchio con centro pari alla sua coordinata e raggio $\varepsilon$.
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.

Queste definizioni definiscono concetto di cluster, infatti quest'ultimo è un insieme massimo di punti density-connected, quindi di core e border points, che rispettivamente determinano l'espansione del cluster e il suo arresto.

## 1.3 Esecuzione di DBSCAN

DBSCAN itera una sola volta su tutti i punti presenti nel dataset e per ognuno che non appartiene ad un cluster si determina se è un core point.
Se non lo è viene etichettato come noise, questa situazione non è definitiva in quanto il punto potrebbe essere successivamente assegnato ad un altro cluster, durante l'espansione di quest'ultimo.
Se invece è un core point si creato un nuovo cluster contenente il punto ed il suo vicinato, ossia i punti directly density-reachable da esso.
Successivamente il cluster appena creato viene espanso, quindi per ogni vicino appena inserito si determina se è un border point oppure un core point, se si verifica la seconda situazione il nuovo vicinato viene aggiunto al cluster ed ogni elemento deve essere a sua volta controllato ricorsivamente tramite questo processo. L'espansione si ferma quando non ci sono più vicini da controllare e la dimensione del cluster è stata massimizzata.

# 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++;
                }
        }
}
```

L'implementazione tramite Java è composta da due classi:

- Coordinate
  Ogni istanza della classe Coordinate è un punto nello spazio.
  I campi consistono nelle coordinate x,y ed una label che identifica il cluster d'appartenenza o se si tratta di noise. Mette a disposizione il metodo distance che calcola la distanza euclidea tra se stesso ed un altra istanza di Coordinate

- DBSCAN
  Il metodo DBSCAN esegue l'algoritmo su un insieme di punti, l'implementazione è identica a quella descritta tramite pseudocodice. Per implementare il processo di espansione viene utilizzata una lista concatenata (LinkedList di Java) in modo che la rimozione e l'inserimento degli elementi all'inizio per simulare una coda FIFO.

L'implementazione tramite Java segue i passaggi descritti nel pseudocodice

## 2.3 Risultati ottenuti

L'algoritmo è stato eseguito con i parametri ε=300 e minPoints=20, sono stati identificati due clusters e cinque punti sono stati classificati come rumore.

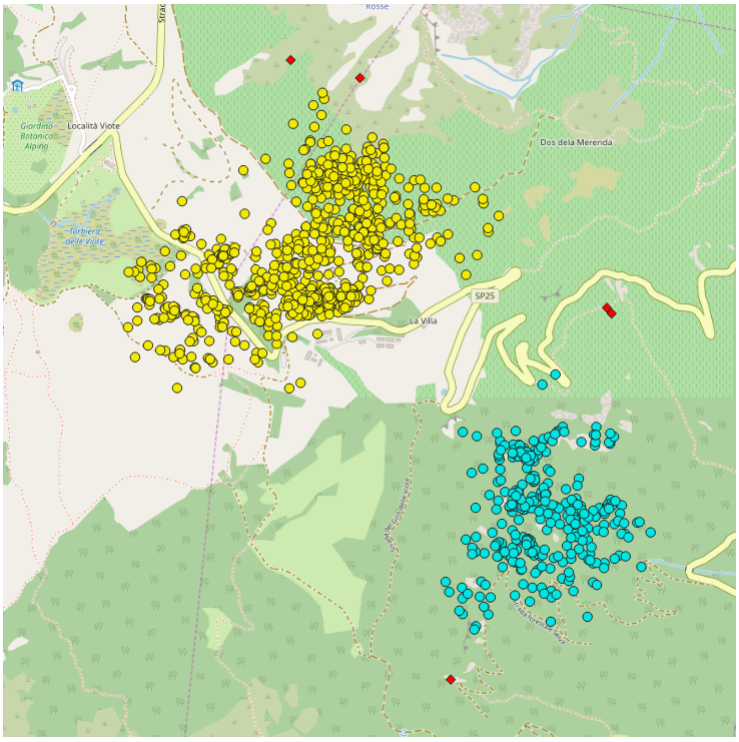


Figura 1: Risultati dell'esecuzione di DBSCAN(ε=300, minPoints=20)

In questa mappa si può avere un intuizione visiva di come l'algoritmo DBSCAN opera, le aree più bianche sono quelle dove c'è maggiore densità di punti. L'immagine è stata ottenuta applicando un buffer di raggio 300 e colore bianco trasparente attorno ad ogni punto.

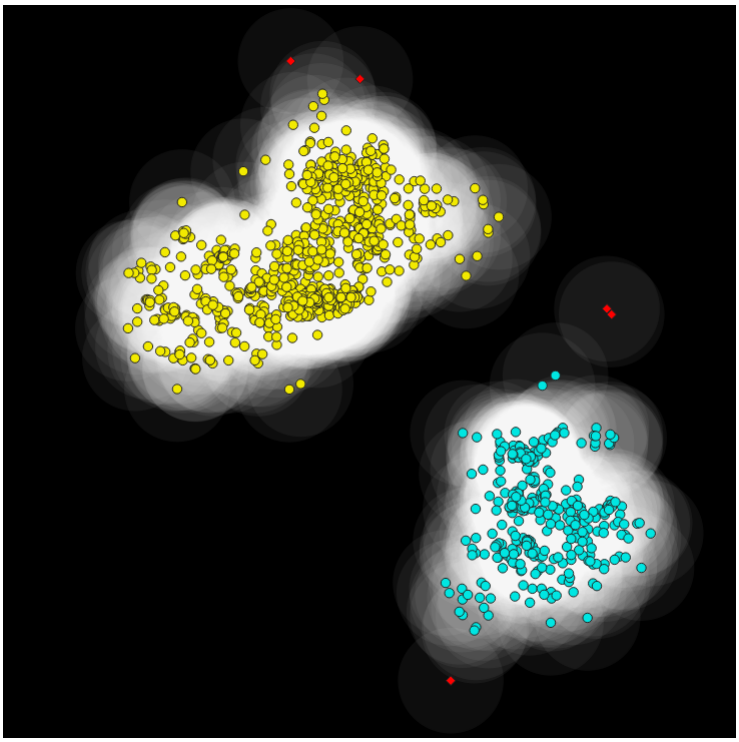

Figura 2: Rappresentazione della densità