Progetto di Gestione Dell'Informazione Geospaziale - DBSCAN

Damiano Bianda

7 gennaio 2019

Sommario

Dato un insieme di punti nello spazio, la cui posizione è determinata da coordinate proiettate, si vogliono identificare i clusters presenti ed assegnare ogni elemento ad ognuno di essi o eventualmente 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 versione in Java per analizzare un insieme di posizioni nello spazio bidimensionale.
I risultati ottenuti sono stati rappresentati su una mappa utilizzando un'applicazione GIS.

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 degli 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 alla conformazione dei 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 ε e MinPoints, necessari a definire la densità minima all'interno di un cluster.
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 points
- 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 crea 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 vicinato del che attualmente si sta considerando viene aggiunto al cluster ed ogni elemento deve essere a sua volta controllato

ricorsivamente tramite questo processo. Se uno dei vicini è invece un border point semplicemente il suo vicinato non viene aggiunto al cluster. L'espansione si ferma quando non ci sono più vicini da controllare, 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
```

L'algoritmo DBSCAN è composto da due iterazioni annidate. Il ciclo for esterno itera su tutti i punti presenti del dataset e considera solo quelli che non sono stati visitati (seed), ossia non sono mai stati etichettati come rumore o appartenenti ad un cluster. Il ciclo while interno invece determina se il punto di partenza ha le caratteristiche per espandersi e in caso positivo procede iterativamente a formare un cluster.
Una volta scelto un seed viene etichettato preventivamente come rumore ed inserito in una coda, inoltre viene creata una variabile booleana che riporterà l'informazione se un cluster è stato creato, il ciclo while itera finchè la coda è piena. La prima iterazione del ciclo while ha lo scopo di determinare se il seed è un core point. in caso affermativo parte del suo vicinato viene etichettata come appartenente al cluster, aggiunta alla coda e la variabile booleana viene impostata a true, in caso contrario nulla di tutto ciò avviene ed il ciclo while termina mantenendo inalterata l'etichetta rumore del seed e quella del cluster che deve essere creato.
Ogni iterazione successiva estrae un punto dalla coda, determinando nuovamente se è un core point attraverso il suo vicinato, in caso positivo parte dei vicini vengono assegnati al cluster corrente ed aggiunti alla coda, altrimenti è un border point e quindi la sua espansione termina.
Ogni volta che viene determinato un core point non tutti i vicini vengono etichettati ed aggiunti alla coda, questo perchè ognuno di essi può:

1. appartenere già ad un cluster: potrebbe essere un border point appartenente sia al cluster in espansione sia ad altri precedentemente creati, oppure nel caso più comune appartenente al cluster in espansione

2. essere rumore: nel caso in cui un seed che non era stato visitato non era un core-point

3. essere non visitato: un punto non ancora trattato

Nel caso 1 l'etichetta del punto non viene modificata e non viene inserito nella coda, questo perchè se il punto è un border point in comune è indifferente a quale cluster appartiene, se invece appartiene al cluster corrente in esecuzione l'inserimento in coda causerebbe un ciclo infinito.
Nel caso 2 si sa già che il nodo non sarebbe un core point bensì un border, quindi inserirlo nella coda non farebbe differenza ma causerebbe una chiamata ulteriore a neighborhood, di conseguenza viene solo etichettato come appartenente al cluster corrente.
Nel caso 3 non si può sapere se il punto è un core point o un border point, quindi viene etichettato ed inserito nella coda in attesa di essere controllato.

2.2 Codice

```
public static void DBSCAN(ArrayList<Point> points, float eps, int minPoints){
        int clusterId = Point.NOISE + 1;
        for (Point point: points){
                if (!point.isVisited()){
                        boolean clusterCreated = false;
                        point.setLabel(Point.NOISE);
                        LinkedList<Point> queue = new LinkedList<>();
                        queue.add(point);
                        while(!queue.isEmpty()){
                                final Point p = queue.remove(0);
                                final LinkedList<Point> neighborhood = neighborhood(points, p, eps);
                                if(neighborhood.size() >= minPoints){
                                        for (Point neighbor: neighborhood){
                                                if(!neighbor.belongToACluster()) {
                                                        if(!neighbor.isVisited()){
                                                                queue.add(neighbor);
                                                        }
                                                neighbor.setLabel(clusterId);
                                                }
                                        }
                                        clusterCreated = true;
                                }
                        }
                        if(clusterCreated){
                                clusterId++;
                        }
                }
        }
}
```

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.

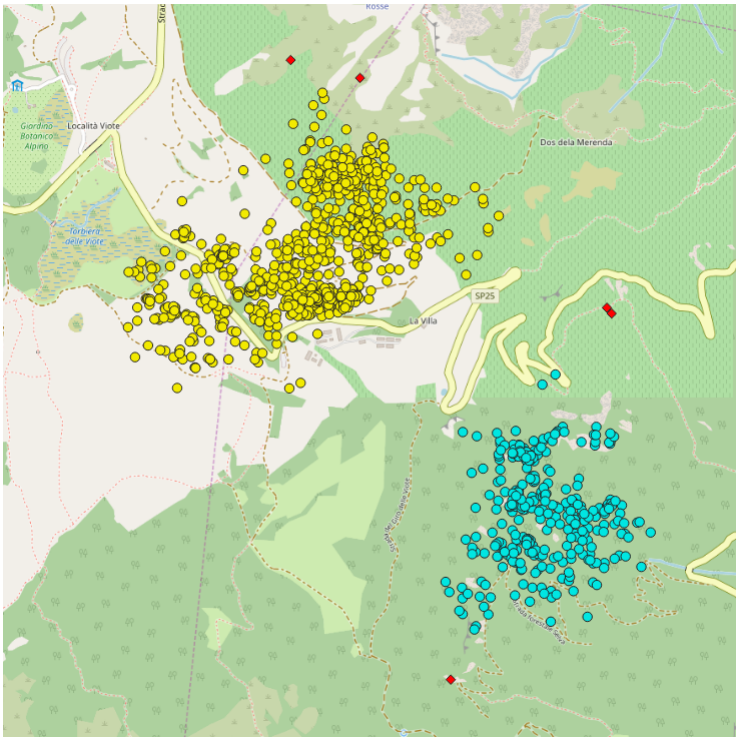

Figura 1: Risultati dell'esecuzione di DBSCAN(ε=300, minPoints=20)

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.

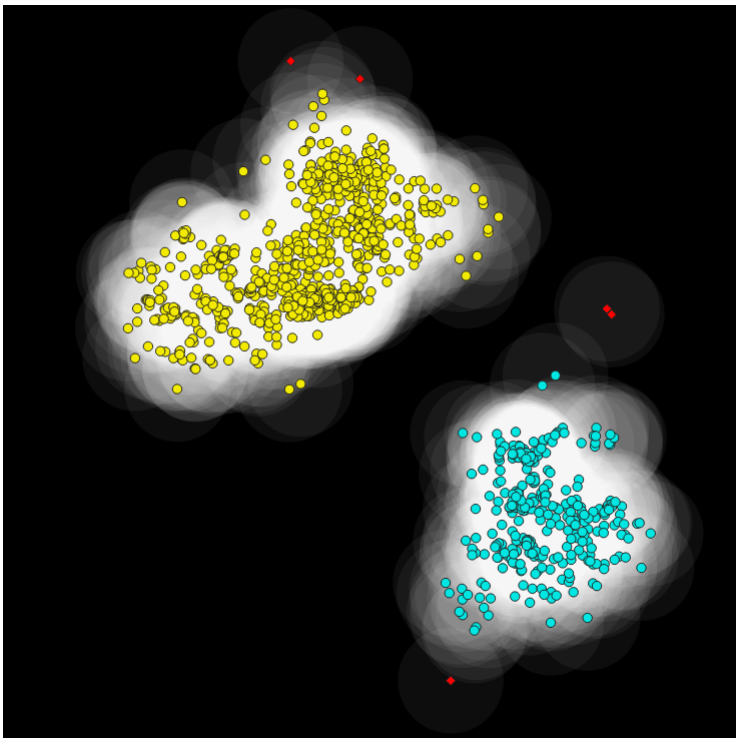

Figura 2: Rappresentazione della densità