Progetto di Gestione Dell'Informazione Geospaziale - DBSCAN

Damiano Bianda

9 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.
DBSCAN è un algoritmo di clustering partitivo basato sulla densità che permette di raggiungere l'obiettivo prefissato, per questo progetto ne è stata implementata una versione in Java per analizzare un dataset di posizioni nello spazio bidimensionale.
I risultati ottenuti sono stati rappresentati visivamente con 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 di 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) e discernere i dati che rappresentano rumore.
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.
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 un core point, nè un 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 una sola volta su tutti i punti presenti nel dataset e per ognuno non ancora visitato si determina se è un core point, se si viene aggiunto, assieme al suo vicinato, ad un nuovo cluster e quest'ultimo dovrà essere espanso, altrimenti si procede a controllare il punto successivo.
L'espansione è un processo iterativo in cui ad ogni iterazione viene controllato un punto appena aggiunto al cluster per determinare se a sua volta è un core point o un border point. Nel primo caso il vicinato viene aggiunto al cluster e necessita anch'esso di essere controllato in iterazioni successive, nel secondo caso il processo d'espansione a partire da quel punto s'interrompe. L'espansione termina quando tutti i core point di un cluster sono stati espansi fino a raggiungere i border point ed ottenendo quindi l'insieme massimo di punti density-connected.
Dopodichè l'iterazione principale continua cercando di espandere altri punti appartenenti al dataset non ancora visitati.
Al termine del processo, a dipendenza dei parametri specificati, si potranno ottenere un insieme di clusters e dei punti che a causa della loro posizione sono stati classificati come noise points.

2 Implementazione

2.1 Descrizione pseudocodice

DBSCAN si aspetta che ogni punto appartenente al dataset è inizialmente etichettato come non visitato, al termine dell'esecuzione ogni punto avrà un etichetta pari a 0 o superiore per indicare rispettivamente che è rumore o il cluster d'appartenenza.
L'implementazione è composta da due iterazioni annidate. Il ciclo for esterno itera su tutti i punti presenti nel dataset e considera solo quelli che non sono stati visitati, ossia non sono mai stati etichettati come rumore o appartenenti ad un cluster. Il ciclo while interno 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 punto non visitato viene etichettato preventivamente come rumore ed inserito in una coda, inoltre viene creata una variabile booleana inizialmente pari a false che indica se un cluster è stato creato. Il ciclo while estrae ogni ad ogni iterazione un elemento dalla coda e termina quando questa è vuota.
La prima iterazione del ciclo while ha lo scopo di determinare se il punto di partenza è un core point. In caso affermativo parte del suo vicinato viene etichettata come appartenente al cluster, aggiunto 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.
Durante ogni iterazione successiva del ciclo while si estrae un punto dalla coda, determinando a sua volta se è un core point, in caso positivo parte dei vicini vengono assegnati al cluster corrente ed aggiunti alla coda, se invece il suo vicinato non ha la dimensione necessaria, allora è un border point e quindi la sua espansione termina.
Ogni volta che viene determinato un core point non tutto il vicinato viene etichettato e/o aggiunto alla coda, questo perchè ogni vicino può:

1. appartenere già ad un cluster
   Nel caso più comune nel vicinato del core point corrente sono presenti punti che sono già stati etichettati come appartenenti al cluster corrente e già aggiunti alla coda, reinserirli un'altra volta causerebbe un ciclo infinito.
   in casi più rari un vicino potrebbe essere border point di più cluster, è indifferente a quale viene assegnato, ma lasciarlo inalterato è più conveniente.
   In entrambe queste situazioni il vicino non viene nè aggiunto ne etichettato.

2. essere rumore
   Se precedentemente un punto è stato etichettato come rumore, significa che era stato estratto dall'iterazione principale e durante la prima esecuzione del ciclo while è risultato non essere un core point.

In questo caso il punto diventa un border point del nuovo cluster e non c'è bisogno di controllare nuovamente la condizione di core point attraverso la dimensione del suo vicinato.
Quindi viene etichettato come appartenente al cluster corrente, ma non aggiunto alla coda.

3. essere non visitato
È un punto che non è mai stato controllato, di conseguenza non si sa se è un core point oppure un border point, in questo caso deve essere aggiunto alla coda ed etichettato come appartenente al cluster corrente. Durante una delle iterazioni successive verrà fatto il controllo.

Una volta che un cluster è stato completamente espanso o il punto di partenza non è risultato essere un core point viene controllata la variabile booleana, se questa è vera l'etichetta del cluster viene incrementata altrimenti la si lascia inalterata in quanto il cluster con quel identificatore non esiste ancora.
La parte più onerosa dell'algoritmo è il fatto di dovere determinare il vicinato di ogni punto tramite la funzione neighborhood, in quanto bisogna controllare se la distanza tra il punto passato come argomento ed ognuno presente nel dataset è minore od uguale a ε. Mantenendo l'informazione se il nodo è già stato visitato è possibile limitare i controlli ad uno per ogni punto.
Si è deciso di utilizzare una coda in quanto l'algoritmo ha molte similitudini con la ricerca in ampiezza su grafi/alberi e risulta essere un buon approccio anche per DBSCAN.

2.2 Implementazione tramite Java

L'implementazione tramite Java rispecchia il pseudocodice, la coda è stata implementata tramite una LinkedList ossia una lista concatenata che permette l'inserimento e la rimozione ai lati in tempi costanti. Le classi principali sono quella Clustering che contiene i metodi statici DBSCAN e neighborhood e la classe Point le cui istanze rappresentano i singoli punti del dataset e mette a disposizioni metodi per interrogarla sulla label corrente e calcolare la distanza con un altro oggetto Point.

Algorithm 1 Density-Based Spatial Clustering of Applications with Noise

```
procedure DBSCAN(points, eps, minPoints)
    unvisited ← −1
    noise ← 0
    clusterId ← 1
    for all point ∈ points do
        if point.label = unvisited then
            clusterCreated ← FALSE
            point.label ← noise
            queue ← []
            enqueue(queue, point)
            while |queue| > 0 do
                p ← removeHead(queue)
                neighborhood ← neighborhood(points, p, eps)
                if isCorePoint(neighborhood, minPoints) then
                    for all neighbor ∈ neighborhood do
                        if not pointBelongToACluster(neighbor) then
                            if point.label = unvisited then
                                enqueue(queue, neighbor)
                            end if
                            neighbor ← clusterId
                        end if
                    end for
                    clusterLabel ← TRUE
                end if
            end while
            if clusterLabel then
                clusterId ← clusterId + 1
            end if
        end if
    end for
end procedure

function neighborhood(points, point, eps)
    neighborhood ← []
    for all possibleNeighbor ∈ points do
        if distance(possibleNeighbor, point) <= eps then
            add(neighborhood, possibleNeighbor)
        end if
    end for
    return neighborhood
end function

function belongToACluster(point)
    return point.label > 0
end function

function distance(pointA, pointB)
    return sqrt((pointA.x - pointB.x) + (pointA.y - pointB.y))
end function

function isCorePoint(neighborhood, minPoints)
    return |neighborhood| ≥ minPoints
end function
```

```
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()){
            Point p = queue.remove(0);
            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++;
         }
      }
   }
}

private static LinkedList<Point> neighborhood(ArrayList<Point>
    points, Point point, float eps) {
   LinkedList<Point> neighborhood = new LinkedList<>();
   for (int i = 0; i < points.size(); i++) {
      Point possibleNeighbor = points.get(i);
      if (point.distance(possibleNeighbor) <= eps) {
         neighborhood.add(possibleNeighbor);
      }
   }
   return neighborhood;
}

private static boolean isCorePoint(int minPoints, LinkedList<Point>
     neighborhood) {
   return neighborhood.size() >= minPoints;
}

public boolean belongToACluster(){
   return this.label > 0;
}

public double distance(Point point){
   return Math.sqrt(Math.pow(point.x - this.x, 2) + Math.pow(point.
       y - this.y , 2));
```

```
52 }
```

3 Risultati ottenuti

L'algoritmo è stato eseguito con i parametri forniti ε=300 e minPoints=20, sono stati identificati due clusters (pallini gialli e azzurri) e cinque punti sono stati classificati come rumore (rombi rossi). I punti sono stati rappresentati con il sistema di coordinate proiettate EPSG 3857 e utilizzando un layer che mostra la posizione geografica tramite OpenStreetMap.

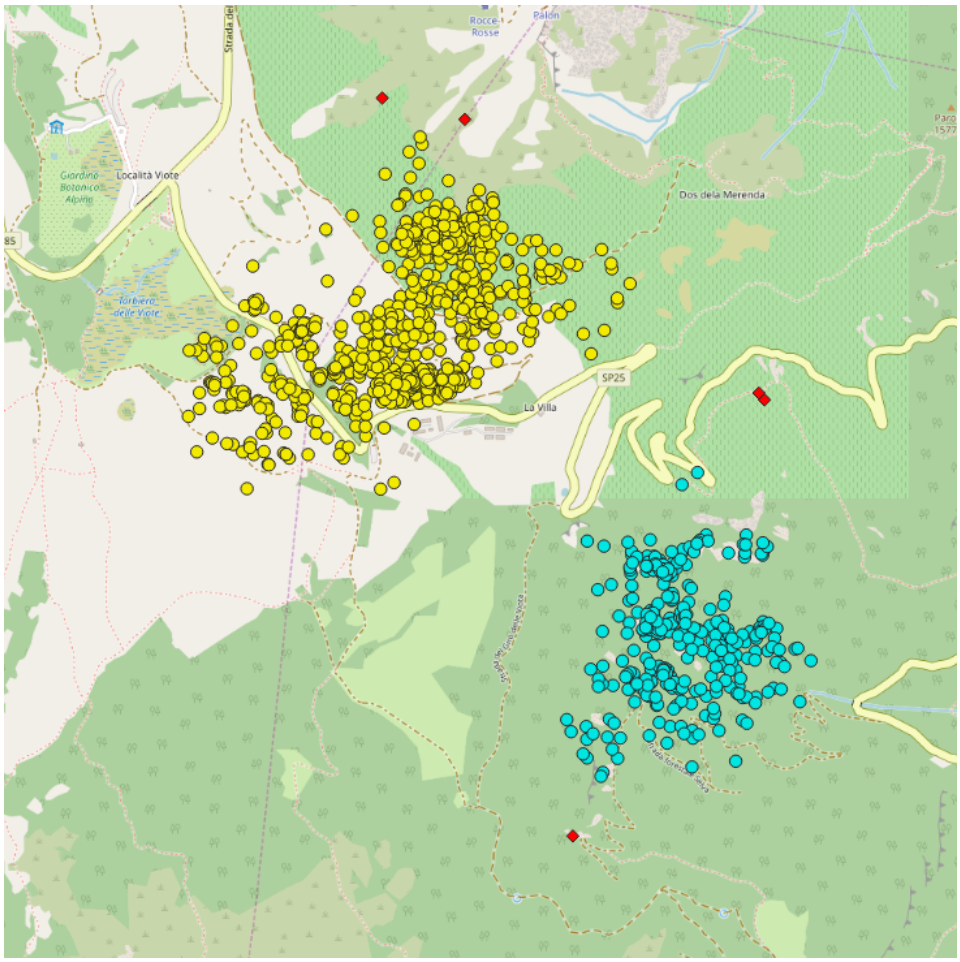

Figura 1: Risultati dell'esecuzione di DBSCAN(dataset, ε=300, minPoints=20)

In questa rappresentazione dei dati si possono visualizzare le zone dove la densità di punti è maggiore, è stato generato un buffer semi-trasparente di distanza pari a εattorno ad ogni punto. Le zone dove il bianco è più intenso sono anche quelle più dense. Da notare che l'informazione non è completa in quanto non tiene in considerazione il numero di punti presenti in ogni vicinato, tuttavia riesce a dare un intuizione visiva.

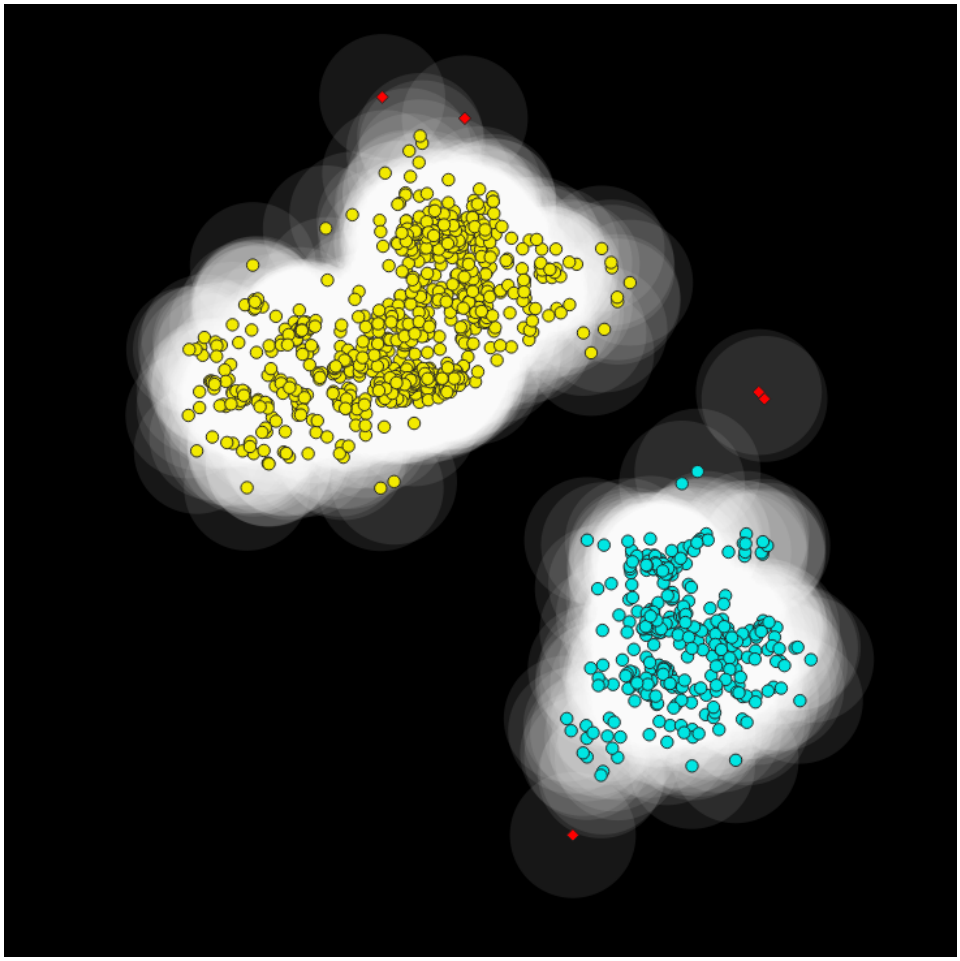

Figura 2: Rappresentazione della densità

Essendo un algoritmo parametrico non è semplice trovare un set di parametri che permetta di discernere i clusters in modo preciso, è possibile utilizzare delle euristiche come ad esempio quella descritta nell'articolo originale basata sulla distanza del k-nearest neighbor e/o sperimentare variando i parametri.
In questo caso avendo già un buon set di parametri da cui partire si è deciso di variarli e osservare i risultati in modo da definire maggiormente il secondo cluster (pallini azzurri). Come si vede dalla mappa, due ulteriori punti in precedenza appartenenti al cluster 2 ora sono classificati come rumore, tuttavia i risultati si ripercuotono anche nel cluster 1 dove più punti vengono esclusi da esso.

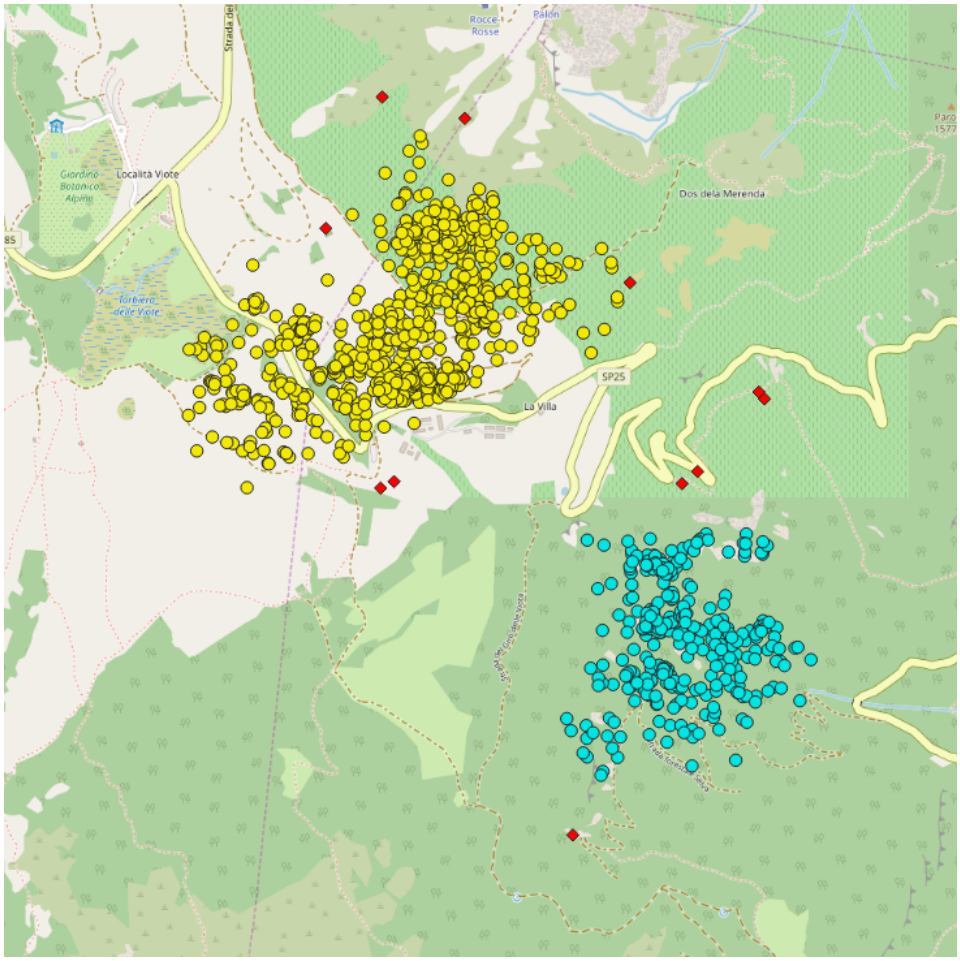

Figura 3: Risultati dell'esecuzione di DBSCAN(dataset, ε=245, minPoints=25)

Anche in questo caso è stata creata una mappa che dà un idea intuitiva della distribuzione di densità dei punti.

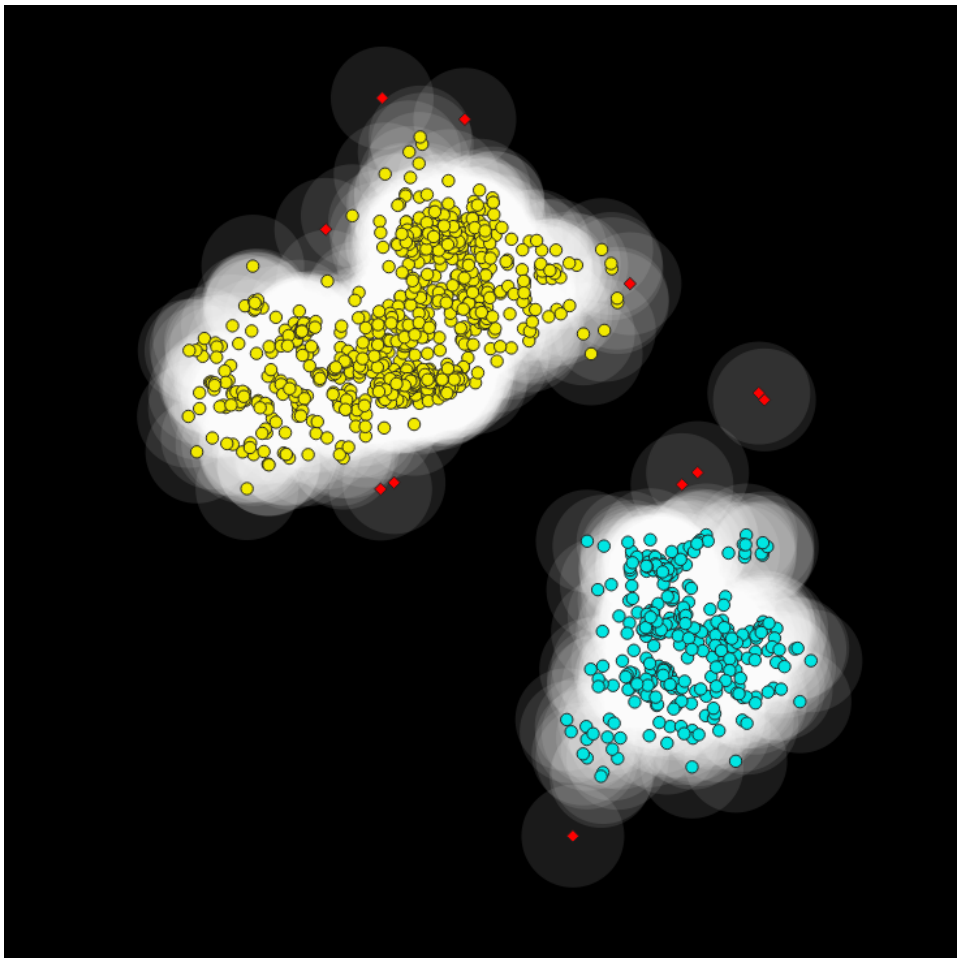

Figura 4: Rappresentazione della densità