A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise 리뷰

류영교

Abstract

공간적 특성이 있는 데이터에서 클러스터링 알고리즘은 유용하다. 이런 클러스터링 알고리즘은 input parameter에 대한 최소한의 요구사항을 갖게 하는 것(너무 많은 요구사항이 있으면 안됨)과, 임의의 모양의 클러스터라도 찾아내는 능력과, 큰 데이터베이스에 대해서 좋은 성능이 필요하다. 지금까지 잘 알려진 클러스터링 알고리즘중엔 이런걸 해결하기 쉽지 않았다. 따라서 여기선 DBSCAN을 소개한다. 이는 CLARANS 알고리즘과 비교되었고 훨씬 좋은 모습을 보였다.

1. Introduction

Spatial database system은 위성 이미지, X-ray 결정학 등등 기술이 발전하며 중요성이 높아짐, 여기서 이런 데이터의 의미있는 subclass를 골라내는 것이 중요하게 되었다. 이러한 클러스터링 알고리즘은 세 가지를 만족해야 한다.

(1) input parameter에 대해 최소한의 요구사항만 필요해야 한다. 왜냐면 거대한 데이터베이스에서는 이런걸 알기 어렵기 때문이다.
(2) 어떠한 모양의 클러스터도 발견할 수 있어야 한다. 왜냐면 공간적 특성이 있는 데이터베이스에서 클러스터는 공 모양일수도, 긴 모양일수도 있고 다양할 수 있기 때문이다.
(3)거대한 데이터베이스에 대해 효율적이여야 한다. 데이터가 아주 많을 수 있기 때문에...

지금까지는 이런걸 만족하는 알고리즘이 없었지만 DBSCAN은 단 하나의 파라미터만으로 이 위의것들을 만족하고 또 이 파라미터도 사용자가 결정할 수 있게 해준다.

2. Clustering Algorithms

클러스터링 방식에는 partitioning과 hierarchical 알고리즘 방식이 있다.
Partitioning 알고리즘은 n개의 object를 k개 클러스터로 나눈다. k가 여기선 input parameter가 된다. 하지만 이 k는 대부분의 경우 알 수가 없다... partitioning 알고리즘은 어쨌든 이 k개 partition에서 시작해서 반복적으로 목적함수를 돌리는 방식이다. k-means 알고리즘은 클러스터를 gravity center로 표현하고, k-medoid 알고리즘은 중심 주변에 위치한 한 object로 표현하는 식으로 차이가 있다. 알고리즘은 2단계로 나뉜다. 우선 목적 함수를 최소화하는 k개의 클러스터 representitves를 구한

다. 그 후 이 representitives에 가장 가깝다고 판단되는 object들을 클러스터에 assign하는 것이다. 이렇게 구해진 클러스터는 매우 restrictive하고 울퉁불퉁하다... CLARANS 방식은 k-medoid를 개선한 방식이다. 이는 더 효율적이고 속도도 빠르다. 또 이 방식에서는 클러스터의 개수를 지정하는 방식을 제안했는데, 군집의 수인 k를 2개부터 n개까지 전부 돌려보고 silhouette coefficient라는걸 계산해서 이게 maximum인 걸 natural clustering으로 판단했다고 한다. 하지만 당연히 시간이 매우 오래 걸리는 방식이다...

Hierarchical algorithm은 위 방식하고 다르게 데이터를 subset으로 쪼개가면서 one object만 남을때까지 tree를 만들면서 내려가는 방식이다. 이 dendrogram은 leaf에서 root로 갈 수도 있고(agglomeratie approach), root에서 leaf로 갈 수 도 있다.(divisive approach) Hierarchical 방식은 k를 입력으로 받을 필요가 없다! 하지만 종료 조건이 필요하다. 예를 들면 데이터의 거리가 일정 이하면 다른 군집으로 만들지 마라 같은 조건이다. 하지만 이것도 종료 조건을 정하는데 어려움이 있다. Jain이라는 사람이 k차원 point set에 대해서 density에 기반해서 이를 알아내는 방식을 제안했단다. 자기 주변에 point count가 높은 cell은 center point로 하고 아닌 애는 boundary point로 하는 방식이다. 이 아이디어는 아래 논문에서 제대로 설명이 나온다.

3. A Density Based Notion of Clusters

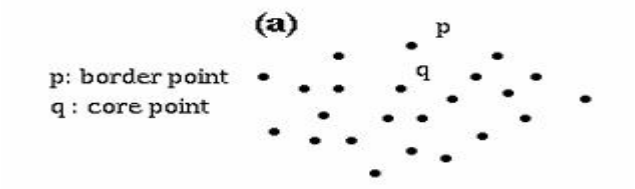
알고리즘에 들어가기 전에 정의해야 될 용어들이 논문에 아주 많이 나온다.

[정의 1]
점 q가 점 p에서 eps라는 특정 거리 안에 있다고 하자. 그러면 점 q는 점 p의 Eps-neighborhood중 하나이다.

$$N_{Eps}(p) = \{q \in D \mid dist(p,q) \leq Eps\}.$$

이렇게 정의된다.

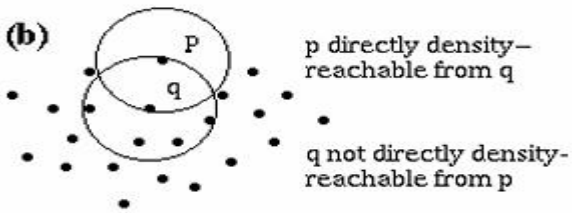
군집 내 모든 point는 Eps-neighborhood의 수가 어떤 값 MinPts보다 크다 라고 정의를 할 수 있을까? 아쉽게도 그렇지 않다. 군집의 중심 부분에 있는 point(core point)들의 Eps-neighborhood들의 수와 군집의 바깥쪽 부분에 있는 point(border point)들의 Eps-neighboorhood들의 수가 많이 차이 나기 때문이다.



이 그림에서 p에서 eps 길이의 원을 그리고 q에서 eps 길이의 원을 그렸을 때 그 안의 점들의 수는 core point인 q가 border point인 p보다 많을 것이다. 그러면 eps를 매우 작게 하면 되는 것 아닌가라고 할 수도 있는데 그러면 군집의 특성이 반영이 당연히 반영이 안될 것이다. 따라서 여기서는 다른 방식으로 이를 해결한다.

[정의 2]
p가 q의 eps 거리안에 있고 q의 eps-neighborhood 수가 MinPts보다 크면 point p는 point q에 directly density-reachable하다고 한다. 여기서 두 번째 조건은 core point를 만족하는 조건이다.

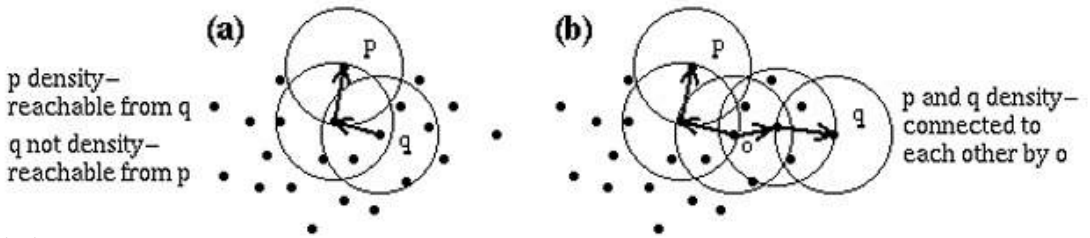
1) $p \in N_{Eps}(q)$ and
2) $|N_{Eps}(q)| \geq MinPts$ (core point condition).



여기서 p의 eps-neighborhood 수가 MinPts보다 작기 때문에 q는 p에 directly density reachable하지 않는다.(정확하게 명시는 안되있지만 MinPts는 2정도로 하면 됨)

[정의 3]
p1...pn이 있고 p1=q, pn=p라고 하자. 이때 모든 pi+1이 pi에 density reachable한 chain이 만들어 진다면 p는 q에 density reachable하다고 한다.
이 정의에 따르면 border point들은 서로 density reachable하지 않다(symmetric하지 않다.). 왜냐면 border point들은 core point 조건을 만족하지 않아서 그 각각이 붙어있는 core point에 대해 density reachable한지 알 수 없기 때문이다. 따라서 이것도 커버하는 새로운 정의를 만든다.

[정의 4]
p와 q가 어떤 point o에 density reachable 하다면 point p는 point q에 density-connected 되어있다고 한다. 이는 symmetric한 성질이다.
아래 그림을 보면 이해가 잘 될 것이다.



[정의 5]
클러스터 C를 아래 조건 2개를 만족하는 데이터 D에 속하는 non-empty subset으로 정의한다.
1. 클러스터 C에 속하는 p가 있고 q가 p에 density reachable하면 q는 C에 속한다 (Maximality)
2. p, q가 클러스터 C에 속하면, p, q는 density-connected되어 있다.(Connectivity)

1) ∀ p, q: if p ∈ C and q is density-reachable from p wrt. Eps and MinPts, then q ∈ C. (Maximality)
2) ∀ p, q ∈ C: p is density-connected to q wrt. EPS and MinPts. (Connectivity)

[정의 6]
어떠한 클러스터에도 속하지 않으면 그 point를 noise라고 한다.

정확성 검증에 관한 내용이 있는데 굳이 할 필요 없을 것 같아서 뺐다.

4. DBSCAN: Density Based Spatial Clustering of Applications with Noise
이 알고리즘을 쓰기 위해 Eps와 MinPts 값, 그리고 클러스터에서 최소한 한 점을 알아야 한다. 최적의 매개변수들을 구하는 쉬운 방법은 없다. 하지만 휴리스틱한 방식으로 이를 구할 수는 있다. 이것은 후에 설명한다.

4.1. The Algorithm(중요)
핵심은 한 점에서 eps neighborhood를 구하는데 이 수가 minPts보다 작으면 클러스터링 안하고 다음 점으로 넘어가고 크면 그 구한 eps neighborhood 들에서 클러스터링을 하는 것이다.
1. 임의로 point p를 하나 고른다.
2. p에서 density reachable한 모든 point q들을 찾는다.
3. p가 core point이면 클러스터를 형성한다(이 클러스터 형성 과정이 중요한데 코드에서 설명한다).
4. p가 border point이면 다른 p를 고른다.
5. 반복한다.

디테일은 아래 코드를 보면서 설명한다. 논문 코드는 너무 이해할 수 없게 되어있어서 위키의 코드를 가져왔다.

```
DBSCAN(DB, distFunc, eps, minPts) {
    C := 0                                                  /* Cluster counter */
    for each point P in database DB {
        if label(P) ≠ undefined then continue               /* Previously processed in inner loop */
        Neighbors N := RangeQuery(DB, distFunc, P, eps)     /* Find neighbors */
        if |N| < minPts then {                              /* Density check */
            label(P) := Noise                               /* Label as Noise */
            continue
        }
        C := C + 1                                          /* next cluster label */
        label(P) := C                                       /* Label initial point */
        SeedSet S := N ₩ {P}                                /* Neighbors to expand */
        for each point Q in S {                             /* Process every seed point Q */
            if label(Q) = Noise then label(Q) := C          /* Change Noise to border point */
            if label(Q) ≠ undefined then continue           /* Previously processed (e.g., border point) */
            label(Q) := C                                   /* Label neighbor */
            Neighbors N := RangeQuery(DB, distFunc, Q, eps) /* Find neighbors */
            if |N| ≥ minPts then {                          /* Density check (if Q is a core point) */
                S := S ∪ N                                  /* Add new neighbors to seed set */
            }
        }
    }
}
```

```
RangeQuery(DB, distFunc, Q, eps) {
    Neighbors N := empty list
    for each point P in database DB {                   /* Scan all points in the database */
        if distFunc(Q, P) ≤ eps then {                  /* Compute distance and check epsilon */
            N := N ∪ {P}                                /* Add to result */
        }
    }
    return N
}
```

C는 클러스터 번호를 뜻한다. 우선 데이터베이스에서 한 점 P를 고른다. label(P)는 P가 어떤 클러스터에 들어가있는가를 뜻한다. 이 값이 Noise이면 노이즈라는 것이다. 어쨌든 label(P)가 undefined가 아니면 이미 클러스터가 정해졌거나 노이즈라는 뜻이므로 넘어간다. RangeQuery함수는 넣은 점의 eps-neighbor들을 리턴해서 N에 넣는다. 따라서 다음 if문은 P의 eps-neighbor의 수가 minPts 이하면 P를 Noise로 지정하고 다음 데이터로 넘어간다.

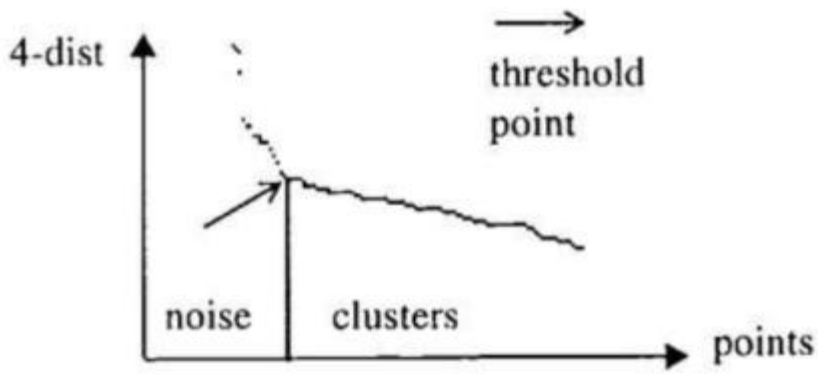
그렇지 않다면 그 점은 지금까지 할당되지 않은 새로운 클러스터 번호인 C + 1을 지정받고 거기서 클러스터링을 시작한다. S변수에는 N과 자기 자신의 집합이 들어간다. S안의 모든 point Q들에 대해 label(Q)가 Noise로 되어있으면 이 label(Q)를 C로 바꾼다. 그리고 이미 label(Q)가 다른 클러스터로 지정되있으면 넘어간다. 이 모든 것에 해당이 안되면 지금 클러스터에 지정하고 그 점에서 또 RangeQuery 함수를 호출해서 N에 넣는다. N의 개수가 minPts보다 크면 S에 이 요소들을 추가하고 계속 반

복문을 돌린다.

4.2. Determining the Parameters Eps and MinPts

휴리스틱한 방법으로 가장 'thinnest'한(밀도가 가장 낮으면서도 클러스터긴 한) 클러스터를 만드는 Eps와 MinPts를 구하고 싶다. 탐색 방법이 어떻게 나왔는지에 대한 설명이 나와있는 것 같은데 어렵고 필요 없을 것 같아서 탐색 방법만 설명하겠다.

1. 모든 데이터에 대해 k번째로 가까운 데이터와의 거리를 계산한다.
2. 그 거리에 대해 내림차순으로 포인트들을 정렬한다.
3. 분포도를 그려본다.
4. 그래프가 꺾이는 지점이 있을텐데 그곳의 k-dist가 eps, minpts = k가 된다.



sorted 4-dist graph