Unsupervised Anomaly Detection - Algorithm

* Clustering (K-means)
* Gaussian / Envelope
* Markov Chain
* Isolation Forest
* RNN

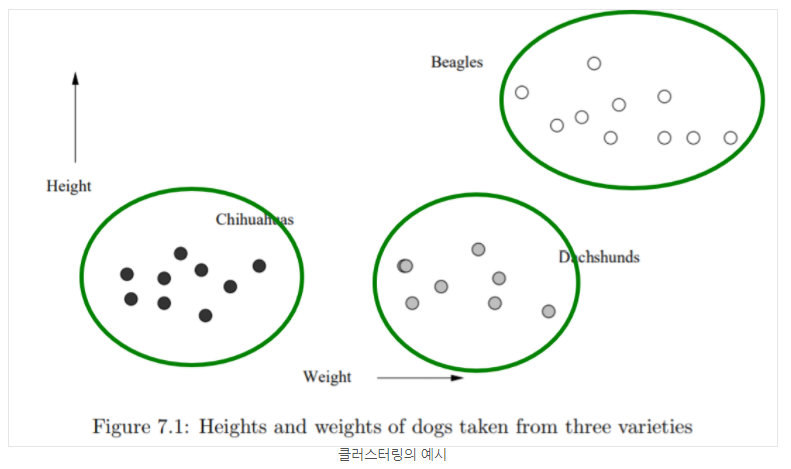
**[Clustering]**

(출처 : <https://www.secmem.org/blog/2019/05/17/clustering/> )

**클러스터링(군집화)**

개체들이 주어졌을 때, 개체들을 몇 개의 클러스터(부분 그룹)으로 나누는 과정.

개체들을 그룹으로 나누는 과정을 통해서, 클러스터 내부 멤버들 사이는 서로 가깝고 비슷하게, 서로 다른 클러스터 사이의 멤버 간에는 서로 멀고 비슷하지 않게 하는 것이 목표.

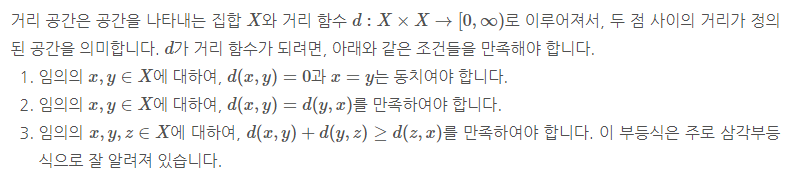


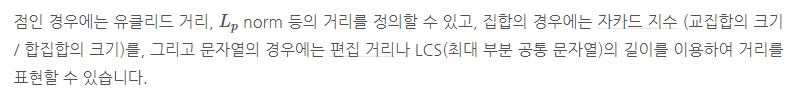
개체들을 거리 공간(Metric space)안에 나타낼 수 있다면, 개체-개체 사이에 거리(metric)을 정의할 수 있다. (예를 들어, 개체들을 유클리드 공간(Euclidean space)안에 나타낼 수 있다면, 유클리드 거리를 정의할 수 있다.)

**클러스터링의 목표**

* **같은** 클러스터 내의 두 멤버들 사이의 거리를 **최소화**
* **서로** **다른** 두 클러스터 사이의 멤버 간의 거리를 **최대화**

거리





자카드 지수 (Jaccard index)

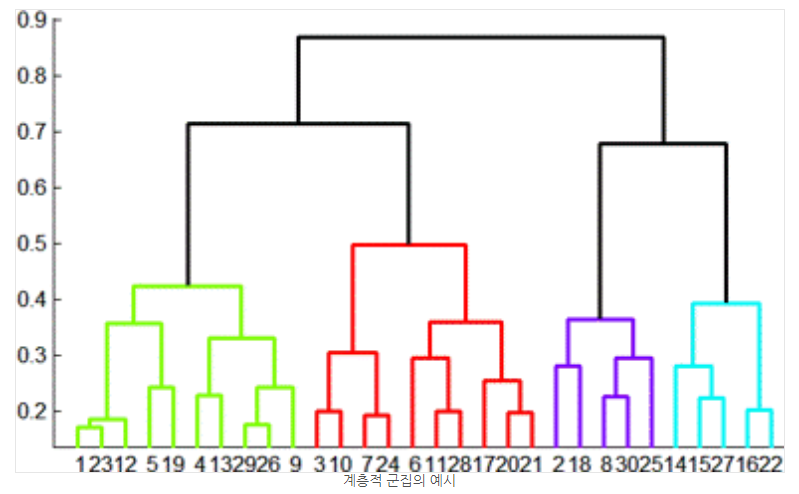
편집 거리(Edit distance)

**클러스터링 알고리즘**

1. 계층적 군집(Hierarchical clustering)
2. Point assignment clustering

<계층적 군집 방법>

1. **Divise(top-down)**한 방법 : 다시 큰 하나의 클러스터로부터 시작해서 모든 클러스터가 정확히 하나의 원소를 가질 때까지 계속 쪼개는 방법.
2. **Agglomerative(bottom-up)**한 방법 : 각각의 점을 원소로 가지는 클러스터들로부터 전체를 포함하는 클러스터 하나를 만들 때까지 반복적으로 두 개의 “가까운” 클러스터를 합치면서 진행하는 방법. (agglomerative : 응집성의)



**Hierarchical Clustering**

Agglomerative한 방법만 다룰 것. 가장 중요한 과정은 반복적으로 두 개의 “가까운” 클러스터를 찾는 것. 거리 공간에서는 거리를 통해서 두 개의 “가까운” 클러스터를 찾을 수 있다.

<알고리즘>

|  |
| --- |
| //V : 클러스터링을 할 Set  C: 클러스터를 관리하는 set, 처음에는 비어 있다.  For each v in V:  {v}를 C에 추가  While len(C)>1:  A, B가 C안에 있는 가장 가까운 두 클러스터라고 하자.  A, B를 C안에서 제거하고, A와 B를 합친 클러스터 S를 만든다.  S를 C에 추가한다. |

**두 개의 “가까운” 클러스터를 찾는 방법**

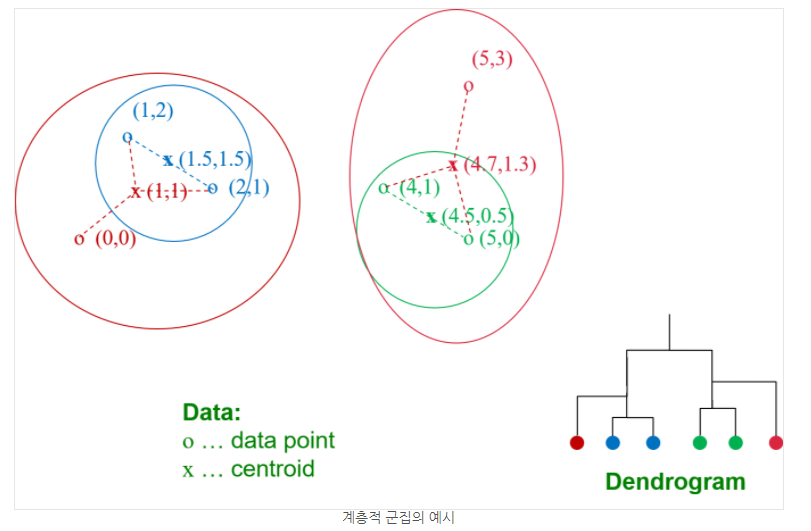
거리를 구할 때 – 두 점이 필요함. 만약 한 클러스터 안에 원소가 두 개 이상 존재한다면 어떻게 거리를 정의할 수 있는가? => centroid 개념 사용 (**유클리드 공간**)

(대표적으로 centroid를 구하는 방법 – 클러스터 내부에 있는 점의 좌표값의 평균을 이용하는 방법.)

* 유클리드 공간에서 두 클러스터 사이의 거리는 두 centroid 사이의 거리로 정의할 수 있다.

유클리드 공간이 아니라면, centroid를 명확히 정의할 수 없다.

**비유클리드 공간**에서는 클러스터 내에서 한 점을 **clustroid**로 정하고, 두 clustroid 사이의 거리를 이용한다. (좋은 clustroid를 찾기 위한 방법에는 여러가지 – 클러스터 내의 각 점으로부터 거리들의 최댓값을 최소화하는 점을 선택, 각 점으로부터 거리들의 평균을 최소화하는 점을 선택)



위 그림은 두 개의 “가까운” 클러스터를 반복적으로 찾는 과정을 표현.

1. 파란색 클러스터 만들고 centroid 지정
2. 녹색 클러스터 만들고 centroid 지정
3. (0, 0)을 가지는 클러스터와 파란색 클러스터를 합쳐서 왼쪽 빨간색 클러스터 만들기
4. (5, 3)을 가지는 클러스터와 녹색 클러스터 합쳐서 오른쪽 클러스터 만들기
5. 마지막으로 남아있는 두 클러스터를 합치면서 계층적 클러스터링의 과정이 끝난다.

(🡪 클러스터 구분은 어떻게 할까?)

**시간 복잡도 분석**

클러스터링을 진행할 집합의 원소의 개수를 N이라고 하자.

(가장 간단하지만 비효율적인 방법) – 클러스터를 합칠 때마다 매번 두 클러스터 사이의 거리를 모두 계산 (두 클러스터 합칠 때마다 O(N^2) 소요, 클러스터 합치는 횟수가 N-1=> O(N^3))

(조금 더 효율적인 방법) – 우선 순위 큐(또는 최대 힙)을 사용하는 방법.

* 모든 점과 점 사이의 거리를 우선 순위 큐에 넣고, 클러스터를 합칠 때, 클러스터를 합침으로써 영향을 받는 거리들만 다시 계산해준다.
* 우선순위 큐에 넣을 때 O(N^2 logN), 영향을 받는 거리의 수 O(N) => O(N^2logN)

앞의 방법보다는 속도가 개선되었지만 N의 크기가 커지면 속도가 매우 느려서 사용하기 힘들다.

\*\*거리 > 개수

**Point Assignment Clustering**

각 cluster가 어떤 개체를 가지는지 기록하는 방식으로 진행되는 클러스터링 방법. (k-means)

**K-means Algorithm**

처음에 클러스터의 개수 k를 정하고, 임의로 선택한 k개의 점을 이용해 초기의 클러스터 k개를 만들고, 클러스터를 계속 알맞게 변화시켜 나가면서 클러스터링을 완료하는 방법.

**초기에 k개의 점을 선택하는 방법**

* 완전히 랜덤하게 k개의 점을 선택하는 방법
* 처음에 한 점만 랜덤하게 선택하고, 두번째 점부터는 이전에 선택한 점들로부터 가장 멀리 떨어진 점을 순차적으로 k-1개 선택해서 k개의 점을 선택하는 방법.
  + - (가장 멀리 떨어진 점들을 선택하는 경우, 랜덤하게 선택하는 것보다 점들이 실제로 다른 클러스터에 포함될 가능성이 높다는 장점이 있다.)

**알고리즘**

1. 선택한 k개의 점을 centroid로 하는 초기의 클러스터 k개를 만든다.
2. 전체 점들에 대해서, 각 점마다 가장 가까운 centroid를 찾아서 찾은 centroid를 기준으로 클러스터를 만든다.
   * + **이렇게 만들어진 클러스터의 경우, 실제 최적의 centroid가 기존의 centroid와 달라지게 되므로 각 클러스터마다 새로 centroid를 찾는다.**
3. 새로 찾은 centroid를 바탕으로 또다시 각 지점마다 가장 가까운 centroid를 찾아서 클러스터를 만들고, centroid를 다시 조정한다.
4. 이 과정을 클러스터링한 결과가 달라지지 않을 때까지 반복한다.

* 가장 가까운 centroid를 찾는 과정과 centroid를 조정하는 과정에서 유클리드 거리를 사용하게 되므로, k-means알고리즘은 **유클리드 공간에서만** 사용 가능.

|  |
| --- |
| 초기에 k개의 점을 임의로 선택한다.  선택한 점들을 각 클러스터의 centroid로 설정한다.  While True:  For each p in V:  p로부터 가장 가까운 centroid가 포함된 클러스터에 p를 추가한다.  P를 추가한 클러스터에서 centroid의 좌표를 다시 찾는다.  만약 이전 loop에서 찾은 클러스터 집합과 현재 loop에서 찾은 클러스터 집합이 같다면, 알고리즘을 종료. |

**Example**

어떠한 직선 위에 있는 점 여덟 개에서 k-means 클러스터링을 한다고 생각해보자.

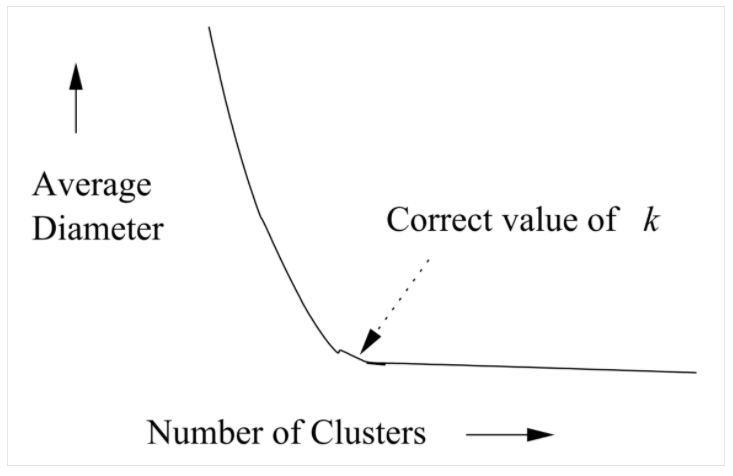
|  |  |
| --- | --- |
|  | 1. k=2로 설정, 가장 왼쪽에 있는 점과 오른쪽에서 네 번째에 있는 점을 centroid로 선택. 2. 그 후 각 점에서 가장 가까운 centroid를 찾고, 이를 바탕으로 클러스터를 구성한다. 3. 그 후 클러스터의 centroid를 다시 정한다. |
|  | 1. 새로 정한 centroid를 이용하여 다시 각 점에서 가장 가까운 centroid를 찾는다. 2. 다시 클러스터를 구성, 다시 centroid의 위치를 조정. 3. 다시 각 점에서 가장 가까운 centroid를 찾아 클러스터를 만들면 이전에 찾은 클러스터와 변함이 없으므로 알고리즘 종료. |

**최적의 k를 찾는 방법**

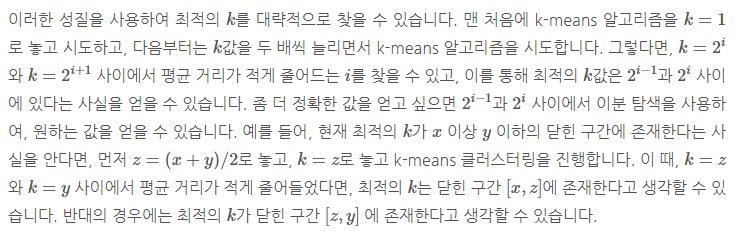
K-means알고리즘은 처음에 k를 선택, 따라서 좋은 품질의 클러스터링 결과를 얻기 위해서는 k를 잘 선택하는 것이 중요.

* K를 너무 작게 설정하면 : 실제로는 거리가 멀어서 서로 다른 두 클러스터에 위치해야 하는 점들이 한 클러스터에 묶이게 되는 경우가 생긴다 -> 클러스터링 품질 떨어짐
* K를 너무 크게 설정하면 : 거리가 충분히 가까워서 한 클러스터에 묶일 수 있는 두 점이 다른 클러스터에 속하게 됨 -> 클러스터링 비효율적

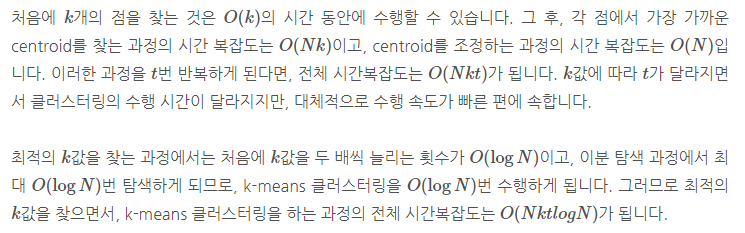
클러스터링의 품질을 한 클러스터 내의 centroid와 다른 점들 사이의 평균 거리를 이용해서 표현해보면.



* 최적의 k값 근처까지는 평균 거리가 급격히 작아지다가 그 이후부터는 큰 차이가 없음.



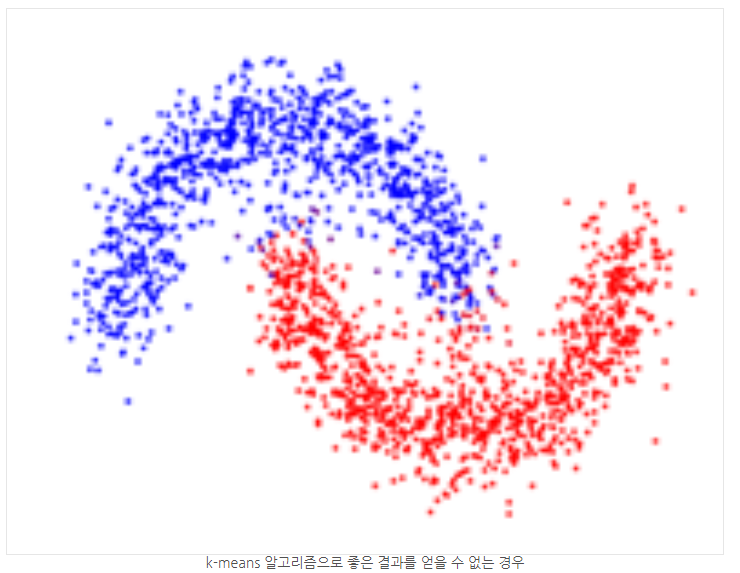
시간 복잡도



**문제점**

k-means 클러스터링은 유클리드 거리를 사용하기 때문에 클러스터의 모양이 주로 원 형태로 형성됨.

아래의 경우와 같이 **최적의 클러스터 모양이 원 모양이 아닌 경우**에는, k-means 클러스터링을 통해 최적의 결과를 얻을 수 없다. (-> CURE, DBSCAN 을 사용)

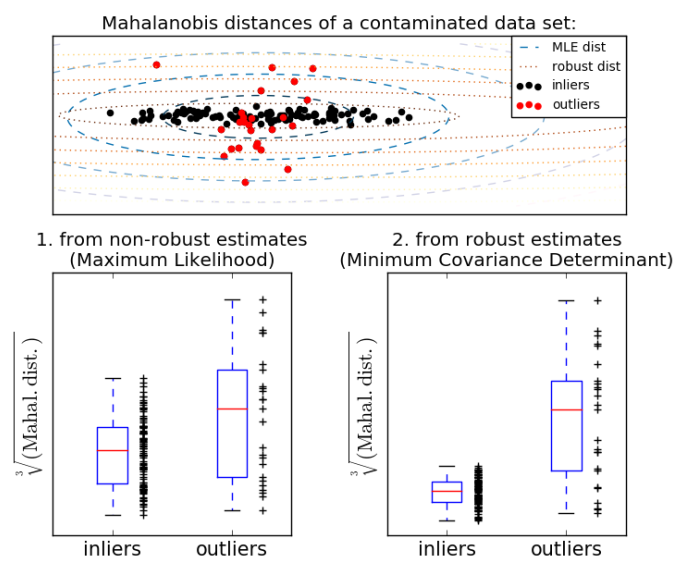


또, k-means 알고리즘은 초기에 어떤 점을 선택하는지에 따라 결과가 달라지는 경우가 발생할 수 있다는 문제점.

**[Elliptic envelope]**

(출처: <https://flonelin.wordpress.com/2017/03/29/novelty%EC%99%80-outlier-detection/>)

* 일반적으로 outlier detection은 알려진 일정한 분포(gaussian distributed)로부터 정상적인 데이터를 추정한다.
* 일반적으로 data의 shape을 정의하면, 이 shape을 벗어난 데이터를 사용해 추정할 수 있다.
* Covariance.EllipticEnvelope를 사용하여 공분산 추정을 하고, 중심 데이터들을 타원형에 fit한다. 그리고 central mode를 벗어난 데이터는 제외한다.
* 예를 들어 inliner data는 Gaussian distributed라고 가정, inlier location과 공분산을 outlier에 영향을 받지 않는 견고한 방법으로 추정한다. Mahalanobis distance가 이 추정 방법으로 사용된다.



**[Markov Chain]**

[강화학습] 마코프 프로세스(=마코프 체인) 제대로 이해하기

(출처: <https://bskyvision.com/573> )

Markov chain

<https://sites.google.com/site/machlearnwiki/RBM/markov-chain>

**마코프 체인**

마코프 체인(Markov Chain)은 마코프 성질(Markov Property)을 지닌 이산 확률 과정(Discrete-time Stochastic Process)을 의미한다.

마코프 성질