Unsupervised Anomaly Detection - Algorithm

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

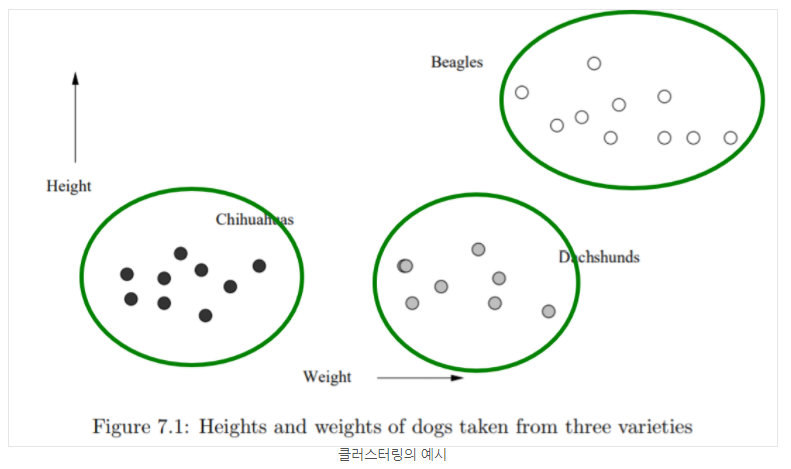
**[Clustering]**

(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.

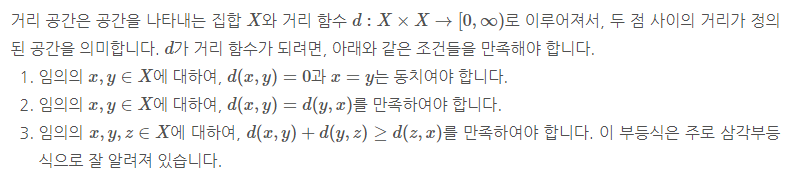


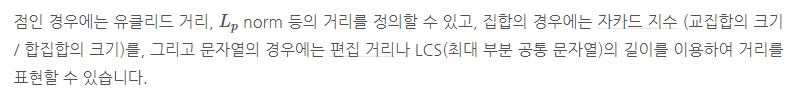
개체들을 거리 공간(Metric space)안에 나타낼 수 있다면, 개체-개체 사이에 거리(metric)을 정의할 수 있다. (예를 들어, 개체들을 유클리드 공간(Euclidean space)안에 나타낼 수 있다면, 유클리드 거리를 정의할 수 있다.)

**클러스터링의 목표**

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

거리





자카드 지수 (Jaccard index)

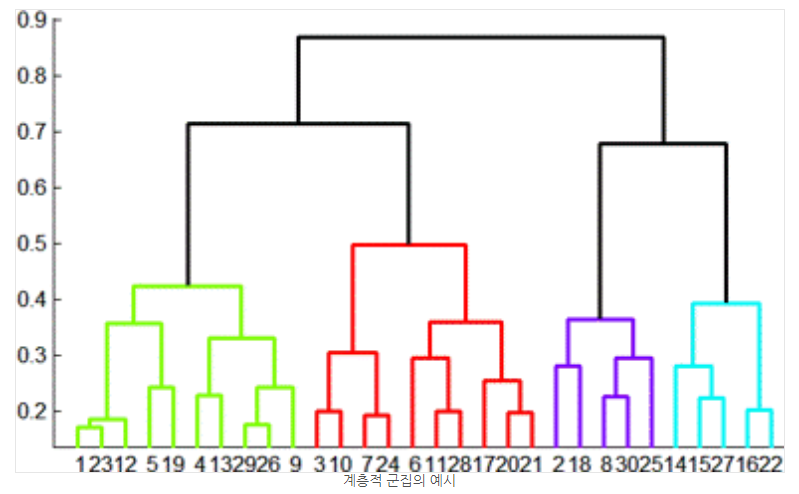
편집 거리(Edit distance)

**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**

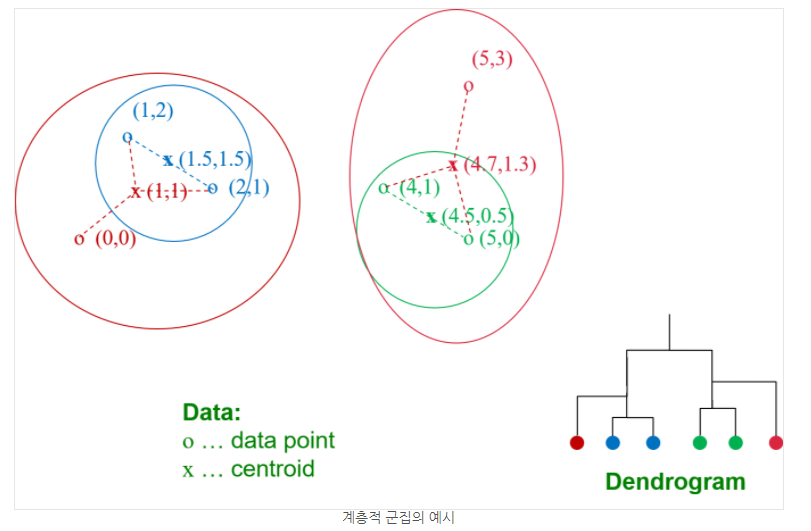
거리를 구할 때 – 두 점이 필요함. 만약 한 클러스터 안에 원소가 두 개 이상 존재한다면 어떻게 거리를 정의할 수 있는가? => centroid 개념 사용 (**유클리드 공간**)

(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.

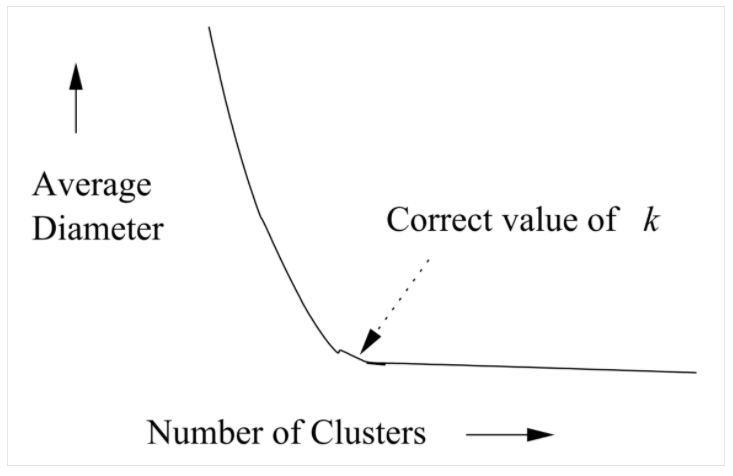
|  |  |
| --- | --- |
|  | 1. k=2로 설정, 가장 왼쪽에 있는 점과 오른쪽에서 네 번째에 있는 점을 centroid로 선택. 2. 그 후 각 점에서 가장 가까운 centroid를 찾고, 이를 바탕으로 클러스터를 구성한다. 3. 그 후 클러스터의 centroid를 다시 정한다. |
|  | 1. 새로 정한 centroid를 이용하여 다시 각 점에서 가장 가까운 centroid를 찾는다. 2. 다시 클러스터를 구성, 다시 centroid의 위치를 조정. 3. 다시 각 점에서 가장 가까운 centroid를 찾아 클러스터를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

**최적의 k를 찾는 방법**

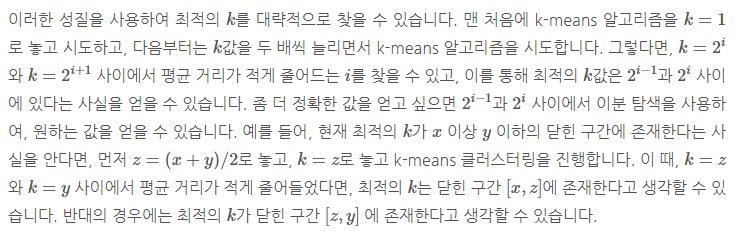
K-means알고리즘은 처음에 k를 선택, 따라서 좋은 품질의 클러스터링 결과를 얻기 위해서는 k를 잘 선택하는 것이 중요.

* 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

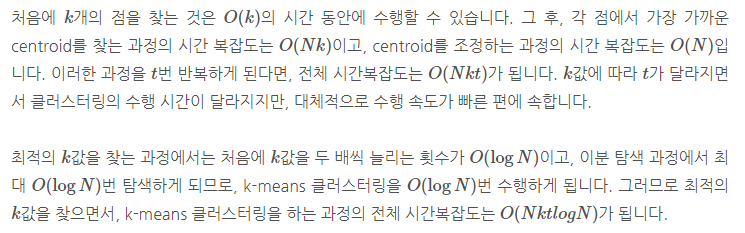
클러스터링의 품질을 한 클러스터 내의 centroid와 다른 점들 사이의 평균 거리를 이용해서 표현해보면.



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



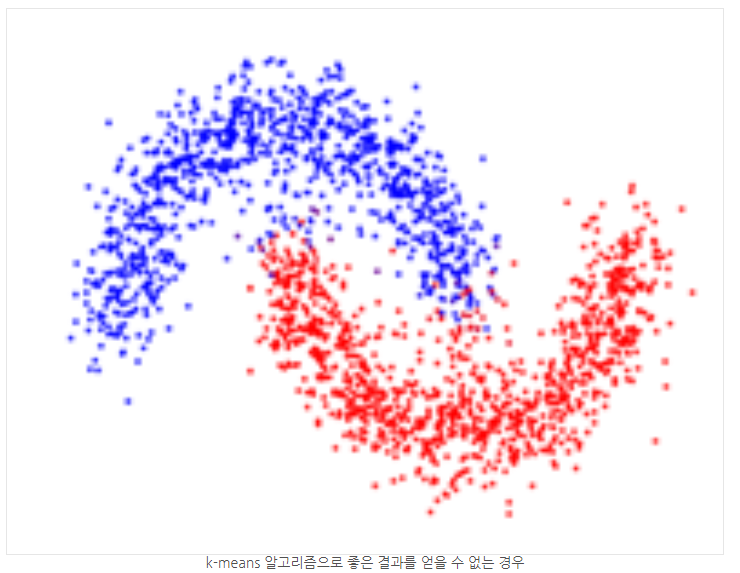
시간 복잡도



**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)

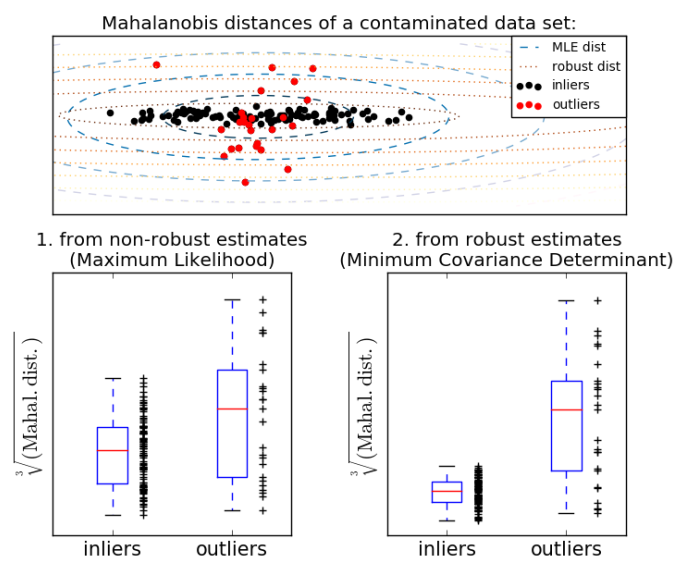


또, k-means 알고리즘은 초기에 어떤 점을 선택하는지에 따라 결과가 달라지는 경우가 발생할 수 있다는 문제점.

**[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>

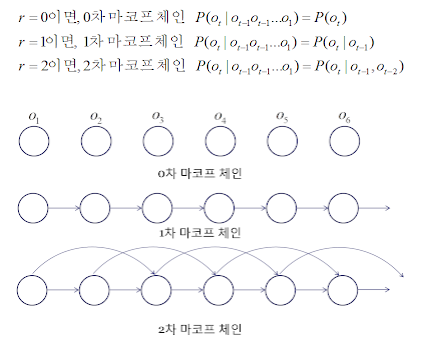
**마코프 체인**

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

**마코프 성질**

n+1회의 상태(state)는 오직 n회에서의 상태, 혹은 그 이전 일정 기간의 상태에만 영향을 받는 것을 의미함.

* 예를 들어 동전 던지기는 독립 시행이기 때문에 n번째의 상태가 앞/뒤 이던지 간에 n+1번째 상태에 영향을 주지 않는다.
* 하지만 1차 마코프 체인은 n번째 상태가 n+1번째 상태를 결정하는데 영향을 미친다.
* (시간 t에서의 관측은 단지 최근 r개의 관측에만 의존한다는 가정을 하고, 그 가정하에서 성립)



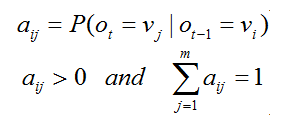
**마코프 모델**

마코프 모델은 위의 가정하에 확률적 모델을 만든 것으로써, 가장 먼저 각 상태를 정의하게 된다.

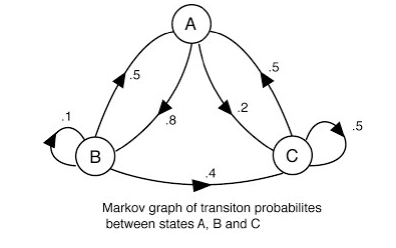
**상태(state)**는 V={v1, …, vm}으로 정의하고, m개의 상태가 존재하게 되는 것.

그 다음 **상태 전이 확률(State transition Probability)**을 정의할 수 있다. 상태 전이 확률이란 각 상태에서 각 상태로 이동할 확률을 말함. 상태 전이 확률 aij는 상태 vi에서 vj로 이동할 확률을 의미한다.

아래의 식은 상태 전이 확률을 식으로 나타낸 것, 그 아래는 확률의 기본 정의에 의한 상태 전이 확률의 조건



상태와 상태 전이 확률을 정리하여, 상태 전이도(state transition diagram)으로도 표현할 수 있다.



\*\*

**마코프 프로세스(마코프 체인)**

마코프 프로세스(Markov process, MP)는 마코프 특성(Markov property)을 지니는 이산시간(discrete time) 확률과정(stochastic process)이다.

확률 과정은 시간에 따라 어떤 사건이 발생할 확률이 변화하는 과정을 의미한다.

이산 시간은 연속적으로 변하지 않고 이산적으로 변함을 의미한다.

**마코프 특성은 과거 상태들(s1, s2, …, st-1)과 현재 상태(st)가 주어졌을 때, 미래 상태(st+1)는 과거 상태와는 독립적으로 현재 상태에 의해서만 결정된다는 것을 의미한다.**

즉, 과거와 현재 상태 모두를 고려했을 때, 미래 상태가 나타날 확률과 현재 상태만을 고려했을 때 미래 상태가 발생할 확률이 동일하다는 것이다.

이를 식으로 나타내면,

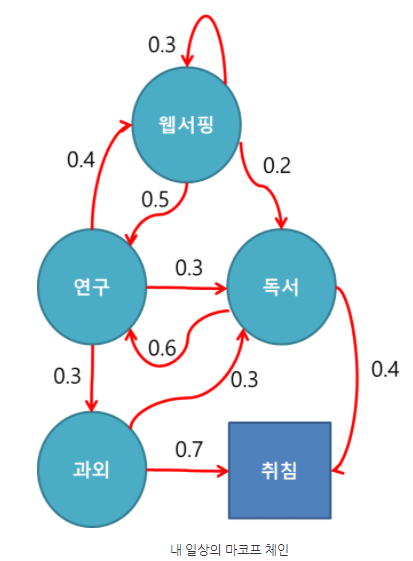


마코프 프로세스는 이처럼 과거 상태를 기억하지 않기 때문에 메모리리스(memoryless) 프로세스로 불린다. 또한 마코프 프로세스는 마코프 체인(Markov chain, MC)이라 불리기도 한다.

어떤 상태에서 다음 단계의 상태로 변화하는 것을 변이라고 하고, 그 확률을 상태변이확률(state transition probability)이라 한다. 시간 t에서의 상태를 s라고 하고, t+1에서의 상태를 s’이라고 할 때, 상태변이확률은 다음과 같이 나타낸다.



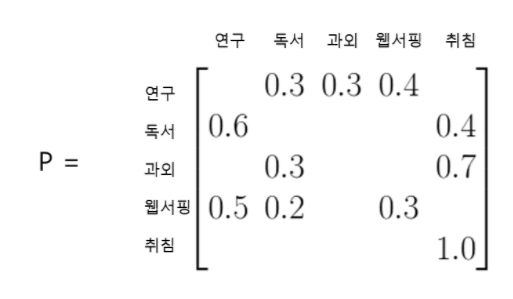
**<나의 하루를 마코프 프로세스로 나타내보자>**



한 상태에서 다른 상태들로 변이할 확률의 합은 1이다.

어떤 상태의 출력의 합은 1이지만, 입력의 합은 다양하다.

이 마코프 프로세스를 하나의 행렬로 나타낼 수 있다. -> **상태변이확률 행렬**



모든 요소가 0이상이고, 각 행의 요소를 더하면 모두 1이된다.

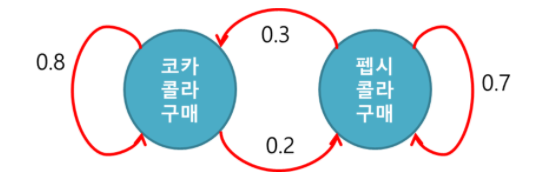
**<마코프 프로세스 예제>**

*이번 주에 코카콜라를 마신 사람의 80%가 다음주에도 역시 코카콜라를 마신다. 나머지 20%는 마음이 바뀌어 펩시콜라를 마신다.*

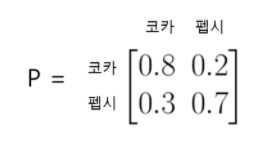
*이번 주에 펩시콜라를 마신 사람의 70%가 다음주에도 역시 펩시콜라를 마신다. 나머지 30%는 마음이 바뀌어 코카콜라를 마신다.*

*이번주에 10억명이 코카콜라를 마셨고, 8억명이 펩시콜라를 마셨다고 할 때, 10주 후와 50주(약 1년) 후에 코카콜라, 펩시콜라를 마시는 인구는 각각 어떻게 될까? (전세계에서 콜라를 마시는 인구는 18억명이라고 가정했고, 또한 비(非)콜라인 중에 새롭게 콜라를 마시기 시작하는 사람은 없다고 가정했다.)*

과거 상태가 미래 상태에 전혀 영향을 미치지 않고, 오로지 현재 상태만 미래 상태에 영향을 끼치기 때문에 현재 시장의 상황을 마코프 체인으로 나타낼 수 있다.



상태변이확률 행렬로 나타내면



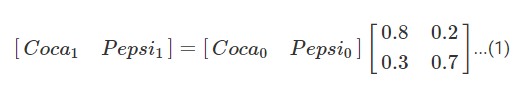
n주 후 코카콜라를 구매하는 인구를 Cocan, 펩시콜라를 구매하는 인구를 Pepsin 이라고 나타내자.

이번주에 코카콜라를 사마신 인구가 10억이고, 펩시콜라를 사마신 인구가 8억이므로 C=10, P=8이 된다. 그렇다면 1주 후 코카콜라를 사마시는 인구와 펩시콜라를 사마시는 인구는 각각 식으로 표현하면



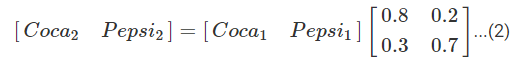


두 식을 모아서 행렬의 형태로 나타내면

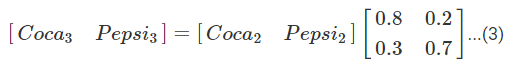


우변에 오른쪽 정방행렬이 상태변이확률 행렬과 같음에 주목하자.

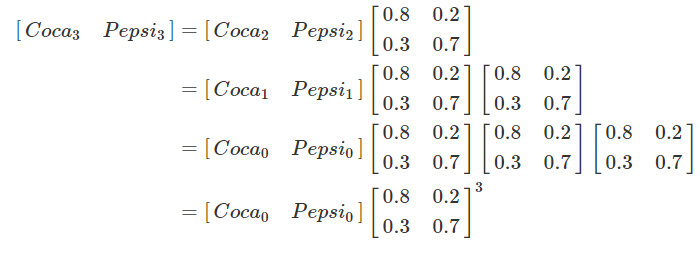
2주후의 상태… -> 1주 후의 상태에만 영향을 받으므로 다음과 같이 나타낼 수 있다.



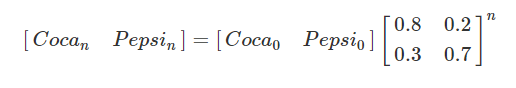
3주 후의 상태는 역시 2주 후의 상태에만 영향을 받는다.



식(3)에 식(2)를 대입시키고, 이어서 식(1)을 대입시키면 현재 상태와 상태변이확률 행렬만을 이용해서 3주 후의 상태를 나타낼 수 있다.

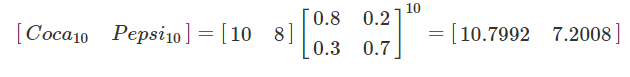


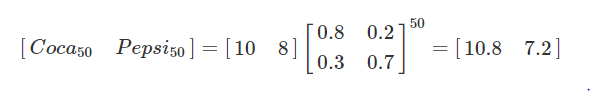
이것을 n주 후로 일반화하면

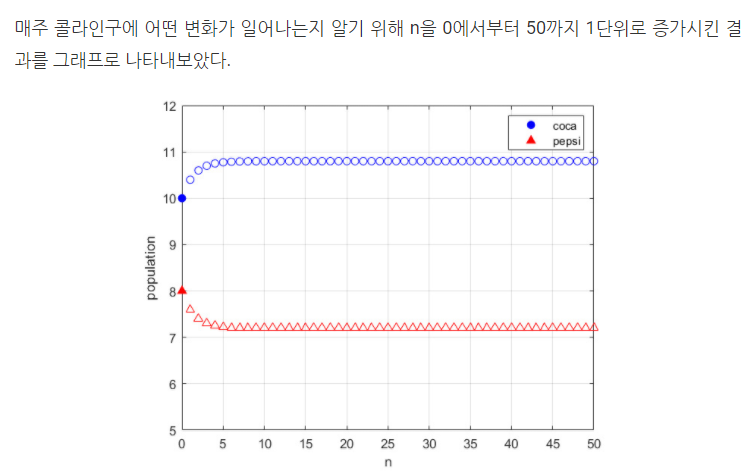


n대신에 10을 대입하면 10주 후의 코카콜라 인구와 펩시콜라 인구를 예측할 수 있고,

50을 대입하면 50주(약 1년) 후의 상태를 예측할 수 있다.







강화학습에 사용하는 이유?

<https://daeson.tistory.com/317>

**[Isolation Forest]**

\*Decision Tree

\*Random Forest : decision tree의 앙상블 형태, random하게 feature를 선택한다.

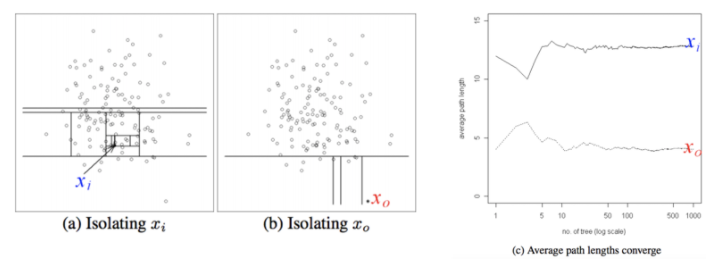
Isolation Forest

<https://donghwa-kim.github.io/iforest.html>

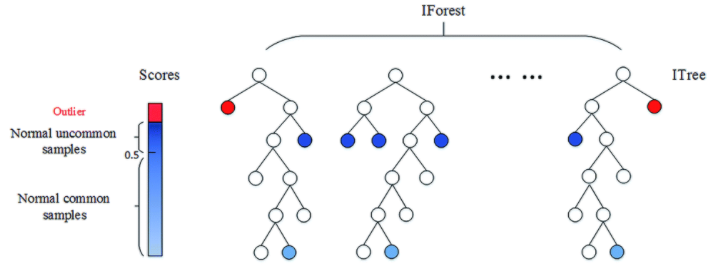
**<Isolation Forest>**

* Regression tree 기반의 split으로 모든 데이터 관측치를 고립시키는 방법
* 비정상 데이터가 고립되려면, root node와 가까운 depth를 가짐
* 정상 데이터가 고립되려면, tree의 말단노드에 가까운 depth를 가짐

(-> 비정상 데이터는 일찍 발견된다?)

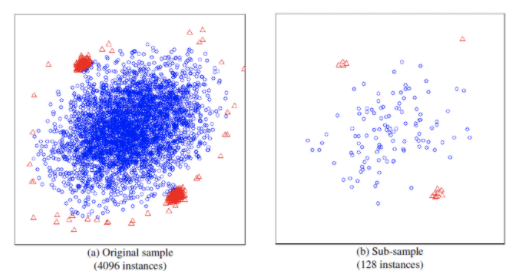


* 특정 한 개체가 isolation 되는 leaf 노드(terminal node)까지의 거리를 outlier score로 정의.
* 그 평균거리(depth)가 짧을수록 outlier score는 높아짐

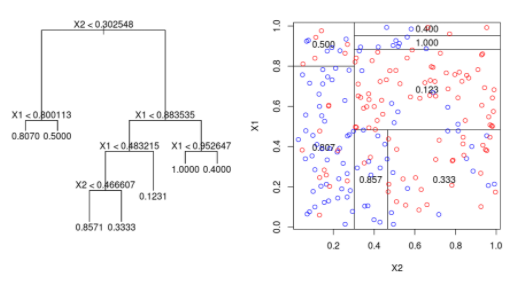


**<학습방법>**

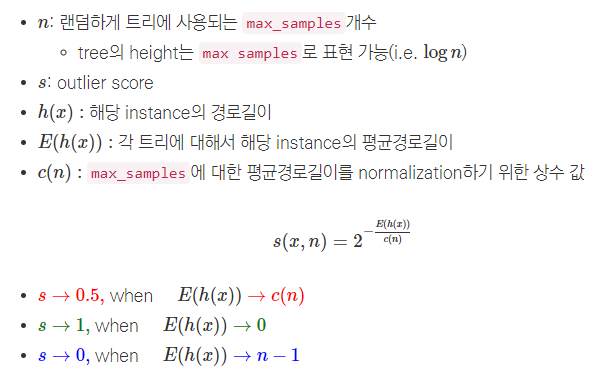
* 먼저 데이터를 부분 샘플링 함(max\_samples로 그 개수를 정의)



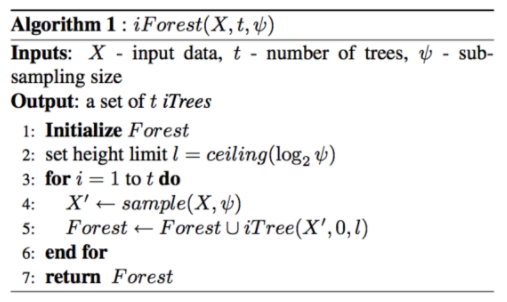
* 데이터 X의 **변수 q를 랜덤하게 선택** (그 개수는 max\_feature로 결정할 수 있음
* 선택된 변수들의 값을 **랜덤하게 p라는 값에서 split**하기 위해 label을 uniform 분포에서 샘플링
* 아래의 그림과 같이 regression tree를 사용



**<Novelty Score>**



**<Algorithm>**



1: Forest를 저장할 빈 객체 생성

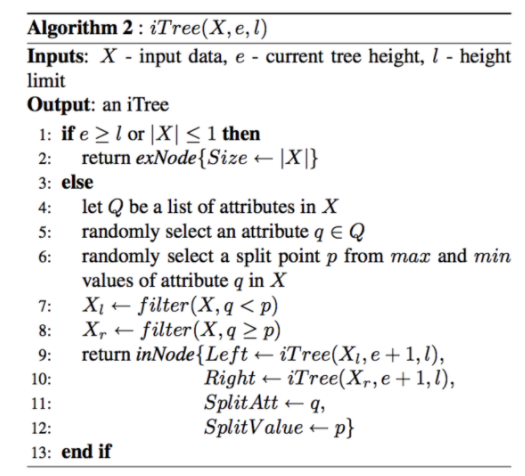
2: for loop

3: 데이터(X) 샘플링

4: 샘플링 X’으로 만들어진 iTree를 Forest 객체에 추가

5: 종료

6: Forest 객체 반환



1: 샘플링된 X’가 Isolation이 된다면 external node로 반환

4: X의 변수를 Q라는 list에 저장

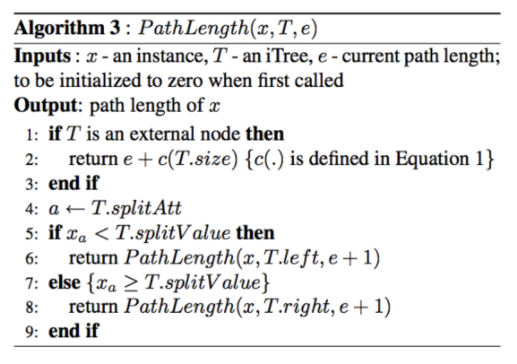
5: 랜덤하게 변수를 선택

6: 랜덤하게 split할 p를 해당 변수의 최댓값과 최솟값에서 선택

7: p값보다 작은 데이터는 Xl(left)로 할당

8: p값보다 큰 데이터는 Xr(right)로 할당

9: 재귀적으로 데이터 point가 고립될 때까지 iTree를 반복하게 되며, 분기되는 history들을 저장



1: 데이터의 수가 1보다 크고 external node이거나 hlim보다 클 때,

2: 분기하지 못한 것에 대한 평균 height(c(n))를 더해줌

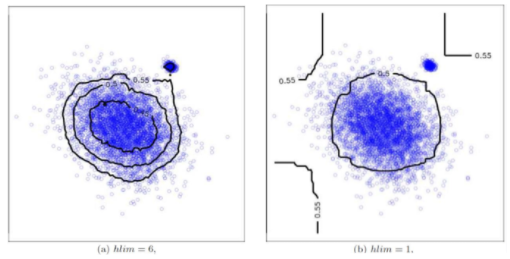
4: split하려는 변수를 a로 정의

5: 변수 a의 값을 가져와 split point보다 작으면 누적된 현재 pathLength를 반환

6: 변수 a의 값을 가져와 split point보다 크면 누적된 현재 pathLength를 반환

**<height limit에 따른 영향>**

* 트리 높이의 제한을 엄격하게 줄 때(hlim = 1) – 정상분포와 비정상분포의 스코어가 유사함
* 트리 높이의 제한을 덜 줄 때(hlim = 6) – 정상분포(0.45)와 비정상분포(0.55)의 스코어가 적절히 구분되어 표기됨.



**[One class SVM]**

collective anomalies (undordered)

novelty detection에 좋다. (train set에 어노멀리가 없을 때)

multimodal data에 잘 작동한다.

Introduction to One-class Support Vector Machines

<http://rvlasveld.github.io/blog/2013/07/12/introduction-to-one-class-support-vector-machines/>

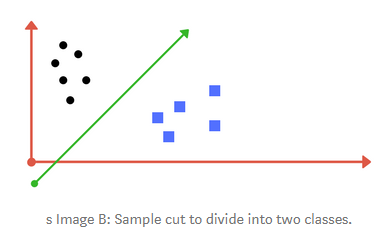
**[SVM]**

SVM

<https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72>

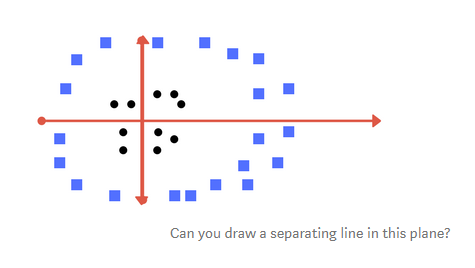
1. Introduction

*SVM은 hyperplane을 separating하여 formal하게 정의하는 discriminative classifier다. 즉, labeled된 트레이닝 데이터가 주어지고 (supervised learning), 알고리즘은 새로운 예를 카테고리화하는 optimal hyperplane을 output으로 낸다. 2차원 공간에서는 이러한 hyperplane이 plane을 두 부분으로 나누는 (한 클래스가 각각의 사이드에 있는) 선이다.*



클래스를 분리하는 것, - SVM이 하는 일.

1. Making it a Bit complex.

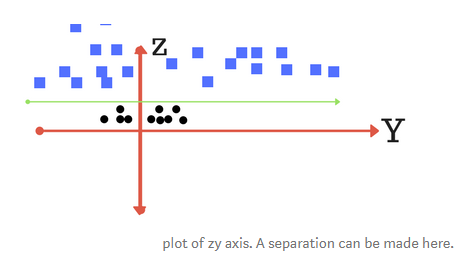


이러한 경우, 직선으로 두 클래스를 분리할 수는 없다.

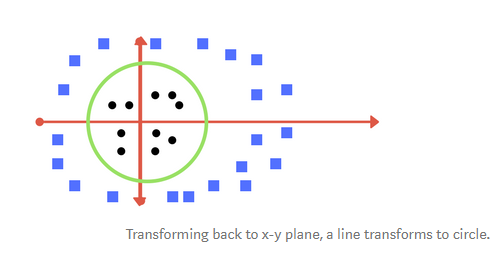
* transformation을 적용하고, 한 차원(z축)을 더 추가한다.

z plane을 w = x2 + y2 라고 하자.

이러한 경우 우리는 z원점으로부터의 거리를 다룰 수 있다. z축에 점을 찍게 되면, 명확한 구분이 눈에 보이고, 선을 그을 수 있다.



만약 우리가 original plane으로 이 선을 transform한다면, 아래처럼 원모양의 boundary로 mapping된다. 이러한 transformations을 kernels라 한다.



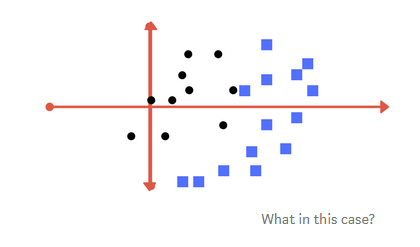
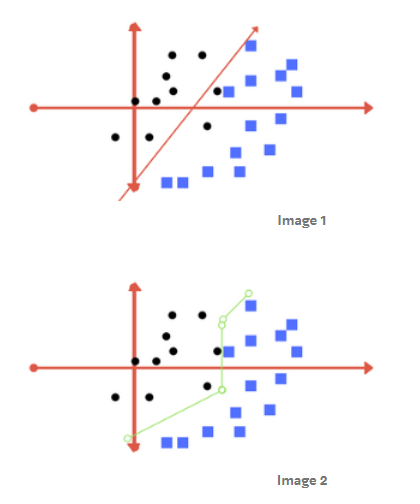
(우리는 transformation을 추측하거나 유도할 필요가 없다. sklearn 라이브러리에서 사용하면 된다.)

유튜브: <https://youtu.be/3liCbRZPrZA>

1. Making it a little more complex…

데이터 점들이 겹친다면(overlaps)? , 그리고 검은 점이 blue 점 안에 속하는 경우라면?

1? 2? 어떻게 선을 그려야 할까?

두 가지 경우 모두 정답!

하지만 첫번째의 경우 몇가지 outlier points들을 허용하고(tolerate), 두번째 경우는 완벽한 partitioni, 0 tolerance.

그러나 trade-off가 있다. 실제 세계에 적용하려면 완벽한 클래스를 나누는 것은 시간이 매우 오래 걸린다. – regularization parameter.

다음 섹션에서 – regularization parameter과 gamma를 정의한다.

이것들은 SVM classifier에서 tuning parameters이다. 이러한 파라미터들을 튜닝함으로써 SVM의 정확도를 더욱 높일 수 있다.

또 다른 파라미터는 kernel이다. 이것은 우리가 a linear of linear separation을 원하는지 아닌지 정의한다. (다음 섹션)

1. Tuning parameters: Kernel, Regularization, Gamma and Margin

<Kernel>

\*\*

초짜 대학원생의 입장에서 이해하는 Support Vector Machine(1)

<http://jaejunyoo.blogspot.com/2018/01/support-vector-machine-1.html>

링크된 영상 한번 볼 것! (MIT OCW)

**Support Vector Machine**

알고리즘의 실제 적용이 여러 모로 쉽고 성능이 강력하다 -> 실전적이다. (매력)

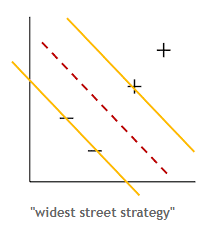
SVM에서 풀고자 하는 문제는 다음과 같다.

“How do we divide the space with decision boundaries?”



1. 우리가 ‘+’ 샘플과 ‘-‘ 샘플을 구별하고 싶다면 어떤 식으로 나눠야 하는가?
2. 만약 선을 그어 사이를 나눈다면 어떤 선이어야 할 것인가?

직관적으로 생각할 수 있는 답은 – ‘+’와 ‘-‘ 샘플 사이의 거리를 가장 넓게 쓰는 line



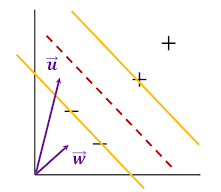
하지만 이 문제에 대한 답을 논리정연하게 이론으로 일반화하는 것은 쉬운 작업이 아니다.

* SVM은 이 문제에 대한 답을 찾기 위해 풀어나간 과정이다.

(SVM이기도 하지만, 어떤 문제를 풀기 위해 체계적으로 아이디어를 개발하고 논리를 전개하는 과정 자체이기도 하다. )

<Decision rule>

“Decision boundary를 정하기 위한 decision rule은 어떤 형태여야 할 것인가?



w벡터는 우리가 그릴 street의 중심선에 직교하는 벡터이다.

그리고 모르는 샘플 u벡터 하나가 있을 때, 우리가 궁금한 것은 street을 기준으로 이 벡터가 오른쪽에 속할지 왼쪽에 속할지 이다.

w와 u를 내적한 후 그 값이 어떤 상수 c보다 큰지를 확인해본다.



edwith – 문일철 교수님 강의 ch.5