단단한 머신러닝

ch.9,10,16

연세대학교 컴퓨터과학과 조영완

2023년 5월

과제명: IoT 환경을 위한 고성능 플래시 메모리 스토리지 기반 인메모리 분산 DBMS 연구개발

과제번호: 2017-0-00477

과학기술정보통신부
Ministry of Science and ICT

연세대학교
YONSEI UNIVERSITY

IITP 정보통신기술진흥센터
Institute for Information & communications Technology Promotion

Contents

클러스터링

- **클러스터링 (clustering)**
  - **Unsupervised learning**에서 사용하는 학습
  - 레이블이 없는 훈련샘플의 학습을 통해 데이터에 **특성과 규칙을 찾음**
  - 데이터세트의 샘플들을 교차하지 않는 **여러 개의 부분집합(cluster)으로 분할**

- **유효성 지표 (validity index)**

  성능 척도 = 유효성 지표
  - 결과가 좋은 clustering ➔ 클러스터 내 유사도(**intra**-cluster similarity) ↑
    클러스터 간 유사도(**inter**-cluster similarity) ↓

  1. 클러스터링 결과와 reference model을 비교 ➔ **external index**
  2. 다른 reference model을 사용하지 않는 것 ➔ **internal index**

- **유효성 지표 (validity index)**

1. external index

Dataset D = { $x_1, x_2, \dots, x_m$}, Cluster C = $\{C_1, C_2, \dots, C_k\}$, reference model Cluster $C^* = \{C_1^*, C_2^*, \dots, C_s^*\}$
λ, λ^*는 각각 C와 C^*에 대응하는 클러스터 레이블 벡터

> a = |SS|, SS = $\{(x_i, x_j) \mid \lambda_i = \lambda_j, \lambda_i^* = \lambda_j^*, i < j)\}$
> b = |SD|, SD = $\{(x_i, x_j) \mid \lambda_i = \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j)\}$
> c = |DS|, DS = $\{(x_i, x_j) \mid \lambda_i \neq \lambda_j, \lambda_i^* = \lambda_j^*, i < j)\}$
> d = |DD|, DD = $\{(x_i, x_j) \mid \lambda_i \neq \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j)\}$

- **Jaccard 계수**(jaccard Coefficient) : JC = $\frac{a}{a+b+c}$
- **FM 지수**(Fowlkes and Mallows Index, FMI) : FMI = $\sqrt{\frac{a}{a+b} \cdot \frac{a}{a+c}}$
- **Rand 지수**(Rand Index,RI) : RI = $\frac{2(a+d)}{m(m-1)}$

➢ 모두 [0,1]구간에 있고, **클수록 좋음**

- **유효성 지표 (validity index)**

2. internal index

avg(C) = $\frac{2}{|C|(|C|-1)}\sum_{1\le i<j\le |C|} dist(x_i, x_j)$

diam = $max_{1\le i<j\le |C|} dist\,(x_i, x_j)$
$d_{min}(C_i, C_j) = min_{x_i\in ci,\, x_j\in cj} dist(x_i, x_j)$
$d_{cen}(C_i, C_j) = dist(\mu_i, \mu_j)$

- **DB지수** (Davies-Bouldin Index, DBI) : DBI = $\frac{1}{k}\sum_{i=1}^{k} \underset{j\neq i}{max}(\frac{avg(C_i)+avg(C_j)}{d_{cen}(C_i,C_j)})$
- **Dunn지수** (Dunn Index, DI) : DI = $\underset{1\le i\le k}{min}\{\underset{j\neq i}{min}(\frac{d_{min}(C_i,C_j)}{max1\le l\le k\, diam(C_l)})\}$

DBI값은 **작을수록 좋고**, **DI**는 반대로 **클수록 좋음.**

- **거리 계산법**

계산척도(distance measure)

- **비음수성** (non-negativity) : dist $(x_i, x_j) \geq 0$
- **동일성** : $x_i = x_j$일때 만, dist $(x_i, x_j) = 0$
- **대칭성** : dist (x_i, x_j) = dist (x_j, x_i)
- **삼각부등식성질** : dist $(x_i, x_j) \leq$ dist (x_i, x_k) + dist (x_k, x_i)

➢ 함수 dist(·, ·)가 계산 척도라면 이러한 기본 성질들을 만족해야함.

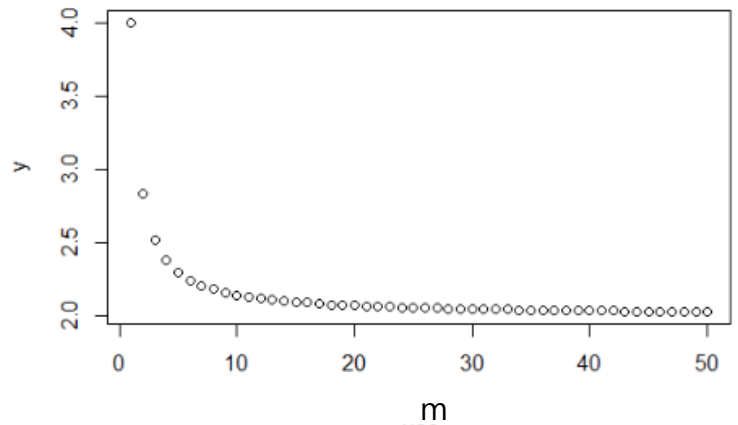
민코프스키 거리 (Minkowski distance)

$$dist(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^m \right)^{\frac{1}{m}}$$

민코프스키 거리는 **m차원 민코프스키 공간에서의 거리**.

m=1일 때 **맨하탄 거리**와 같고, m=2일 때 **유클리드 거리**와 같음.

m이 정수가 아니어도 되지만 **반드시 1보다 커야 됨.**



클러스터링

- **거리 계산법**

속성(attribute)

- 연속 속성, 이산속성, 순서형 속성(ordinal), 무순서형 속성(non-ordinal)

VDM(value Difference Metric) : 무순서형 속성일때 계산방법

- **민코프스키 거리**와 **VDM**을 결합하면 혼합속성 처리 가능
- **가중거리**(weighted distance) : 각 속성의 중요성이 서로 다를 때 사용
- **비척도 거리**(non-metric distance) : 각 속성에 값을 부여해서 계산 (x는 y와 비슷 = 2, x는 z와 다르다 = 8)
- **거리 척도 학습**(distance metric learning)
- **유사도측정**(similarity measure) : 특정 거리 계산으로 거리가 크면 유사도가 작고 거리가 작으면 유사도가 큼.

클러스터링

- **프로토타입 클러스터링**

  - 클러스터 구조가 프로토타입을 통해 형상화 될 수 있다고 가정
  - 프로토타입에 대한 초기화를 진행, 반복적으로 갱신하며 해를 구함

여러가지 알고리즘 : **k-means clustering**, **학습벡터 양자화**(learning vector quantization),
가우시안 혼합 클러스터링(mixture-of-Gaussian)

클러스터링

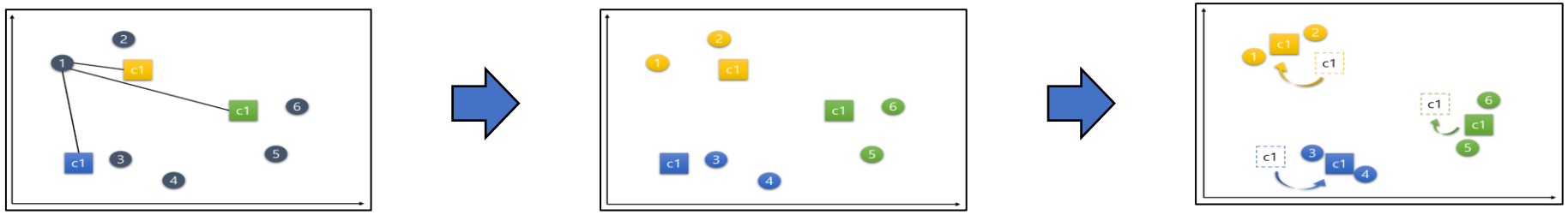
- **K-means clustering**

$E = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2$, $\mu_i = \sum_{x \in C_i}$ 이고, x는 클러스터 C_i의 평균벡터

목적 : E를 최소화하는 것

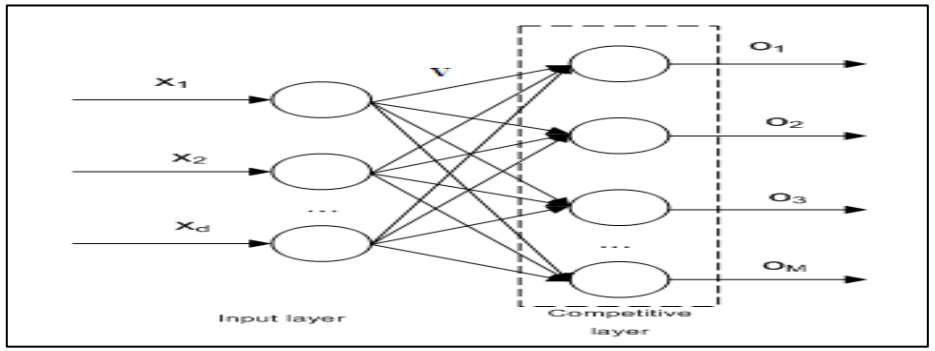
프로세스

1. 클러스터 개수(k) 설정
2. 초기 Centroid(각 그룹의 중심) 선택
   – 랜덤하게 설정
   – 수동으로 설정
   – k-means++방법
3. 모든 데이터를 순회하며 각 데이터마다 가장 가까운 Centroid가 속해 있는 클러스터로 assign
4. Centroid를 클러스터의 중심으로 이동
5. 클러스터에 assign 되는 데이터가 없을 때까지 스텝 3, 4를 반복



클러스터링

- **학습벡터 양자화(learning vector quantization, LVQ)**
  - K-means 알고리즘과 비슷하게 **원형벡터**를 찾아 **클러스터 구조를 형상화** 하는 방법
  - **LVQ**는 데이터 샘플이 클래스 레이블을 갖고 있다고 가정, 학습과정에서 샘플의 지도적 정보를 사용함.



1. 클러스터의 수 **M**과 **η**를 설정한다.
2. 클러스터의 **중심** v_{ij}를 **0과 1 사이의 임의의 값으로 초기화**한다, $i = 1,2,\dots,M, j = 1,2,\dots,N$.
   N : **입력 벡터의 크기**, **η** : **learning rate**
3. **K를 0으로 초기화**
4. x_k 를 인공신경망의 입력으로 설정
5. **경쟁학습**을 통해 **클러스터를 선택**
6. 선택된 클러스터의 중심으로 수정

7-1. 현재까지 학습한 결과가 종료 조건을 만족하면 알고리즘 종료
7-2. 종료 조건을 만족하지 않는다면 $k = k + 1$ 을 수행하고 과정 4부터 다시 알고리즘 실행

클러스터링

- **가우시안 혼합 클러스터링(mixture-of-Gaussian, GMM)**

  - **확률 모델을 이용**해 클러스터의 프로토타입 표현 (k-means, LVQ : 원형벡터 이용)
  - 데이터가 여러 다른 모양의 **가우시안 분포**(Gaussian Distribution)로 결합되어 있다는 가정 하에 개별 데이터를 동일한 **가우시안 분포별로 묶어주는 비지도 학습 알고리즘**

$$P(X_i) = \sum_{k=1}^{K} w_k N(X_i|\mu_k, \Sigma_k)$$

K : 클러스터 개수, 데이터 X_i, i = 1,...,n , p차원 벡터

이때 $N(X_i|\mu_k, \Sigma_k)$은 평균 벡터가 μ_k, 공분산 행렬이 Σ_k인 p 차원 정규분포이고

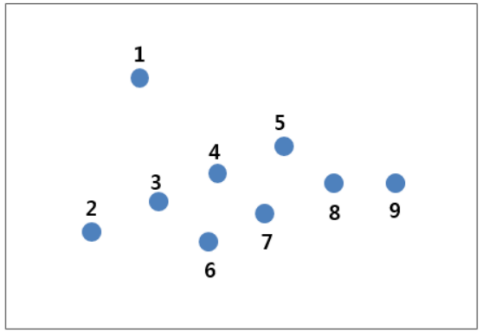
$\sum_{k=1}^{K} w_k = 1, w_k \geq 0, k = 1, \ldots, K$이다.

GMM 클러스터링은 데이터의 생성 확률 모형을 GMM이라는 가정을 사용한다.

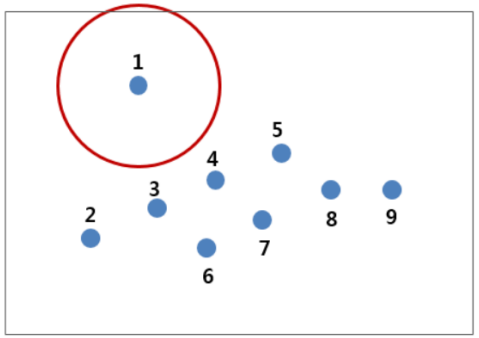
- μ_k: 각 가우스 분포의 중심을 나타냄
- Σk: 분포의 퍼짐을 나타내는 공분산 행렬
- πk: 각 가우스 분포의 크기의 비율을 나타내는 혼합 계수,
  이 혼합 계수는 0과 1 사이의 실수로 K로 합을 취했을 때 1임 $\sum_{k=0}^{k-1} \pi = 1$

클러스터링

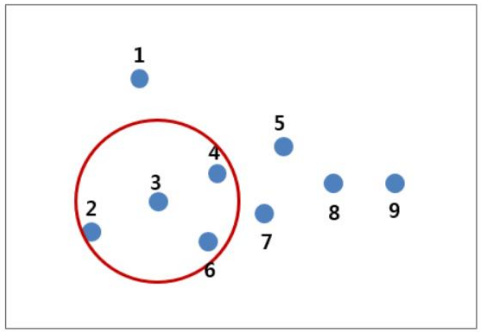
- **밀도 클러스터링(Density-based Spatial Clustering of Applications with Noise, DBSCAN)**
  - 클러스터의 구조가 샘플 분포의 밀집된 정도로 결정된다고 가정



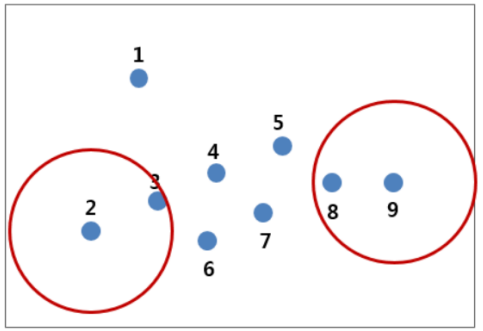
반지름이 ϵ 인 원이 있다고 하고 1번부터 원의 중심에 데이터를 둠.



원 안에 하나의 데이터만 있는경우 **noise point**라고 부르며 클러스터에서 제외.



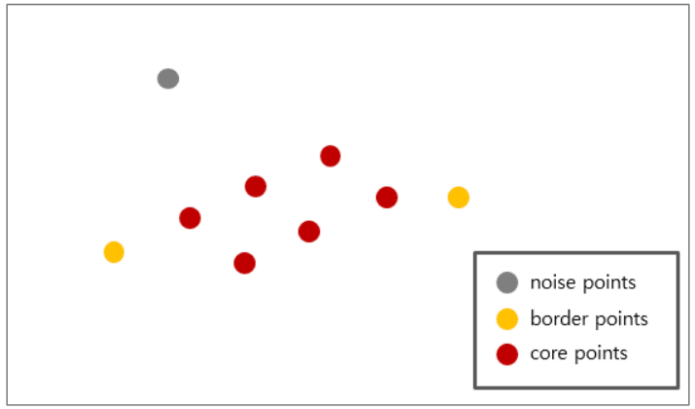
원 안에 4개의 데이터가 들어오면 **core point**라고 부르며 클러스터에 포함

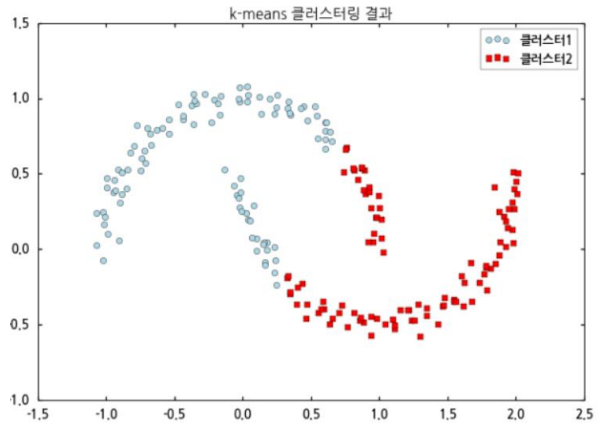


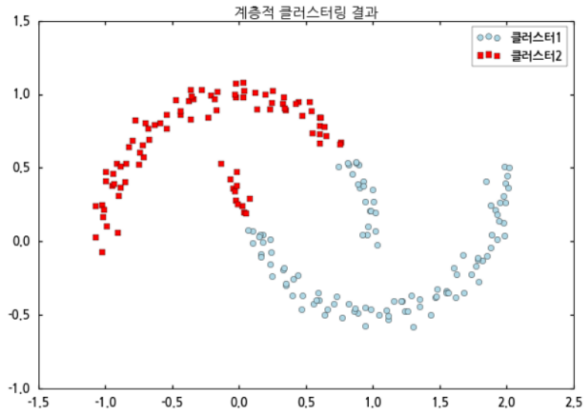
Core point에는 속하지 못하지만, core point를 포함하면 **border point**.

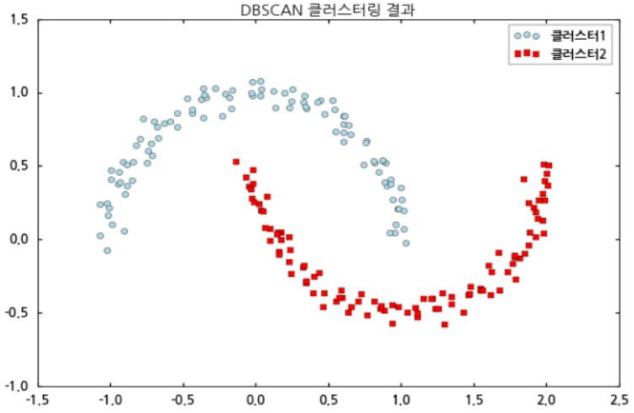
클러스터링

- **밀도 클러스터링(Density-based Spatial Clustering of Applications with Noise, DBSCAN)**



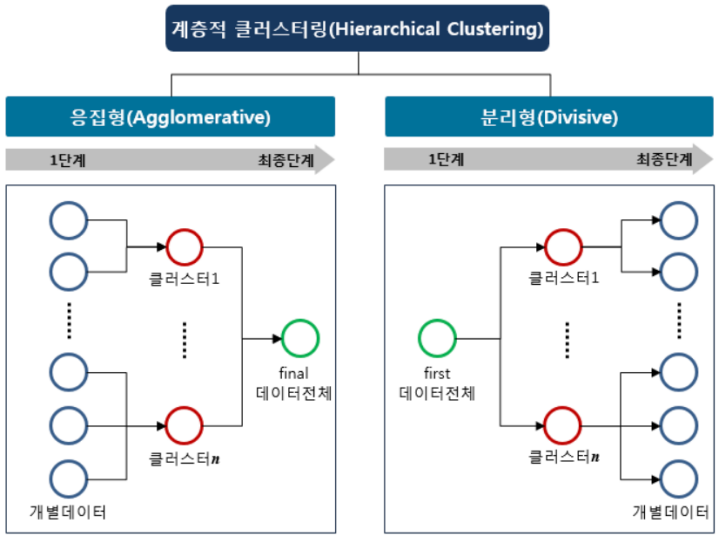






클러스터링

- **계층 클러스터링(hierarchical clustering)**
  - **단계별로 데이터를 분할**하여 **트리 형태의 클러스터링** 구조를 형성
  - 최초에 **클러스터의 개수를 가정할 필요 없음**
  - **Bottom-up**클러스터링과 **top-down** 분해전략 사용 가능



상위 단계의 클러스터를 구성하기 위한 대표적인 알고리즘 :

1. **단순연결(single linkage)**
2. **완전연결(complete linkage)**

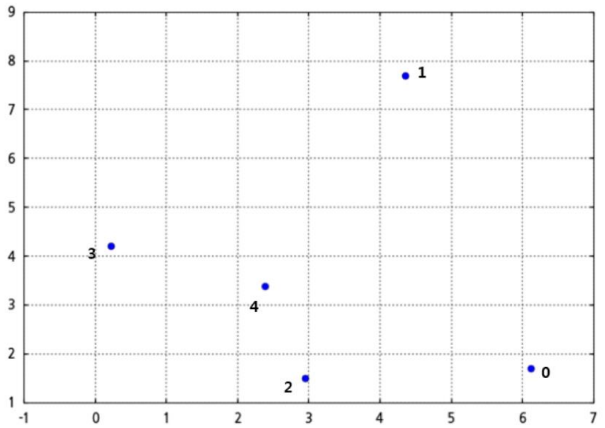
단순연결 : 2개의 클러스터에서 각 클러스터에 속하는 멤버 사이의 거리가 가장 **가까운 거리**를 모두 계산하고, 이 값들 중 가장 작은 값을 가지는 2개의 클러스터를 병합하여 상위 단계 클러스터를 구성하는 방법

완전연결 : 2개의 클러스터에서 각 클러스터에 속하는 멤버 사이의 거리가 가장 **먼 거리**를 모두 계산, 이 값들 중 가장 작은 값을 가지는 2개의 클러스터를 병합

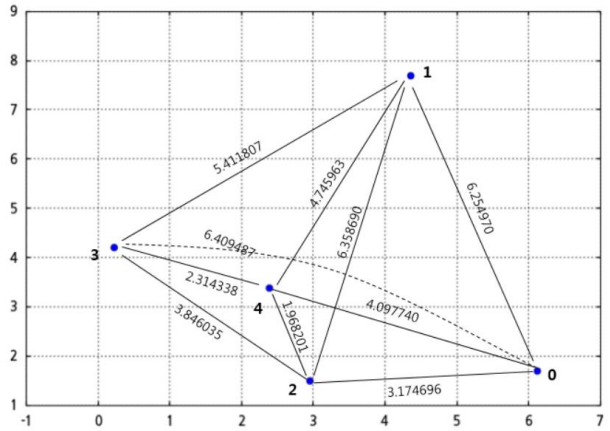
클러스터링

- **계층 클러스터링(hierarchical clustering)**

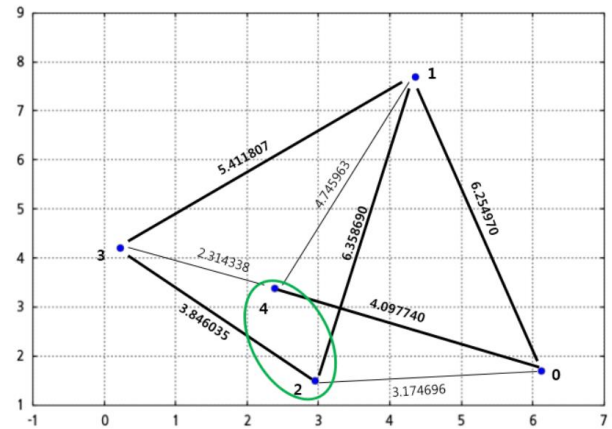
1



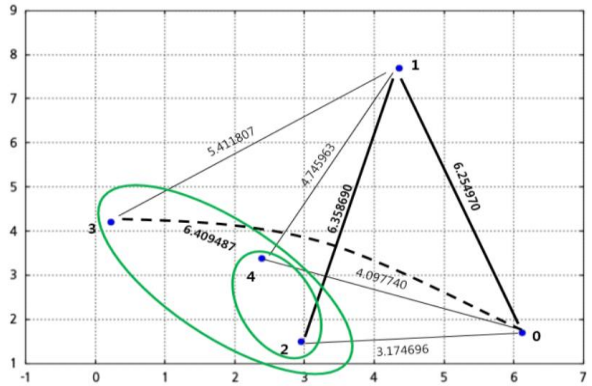
2



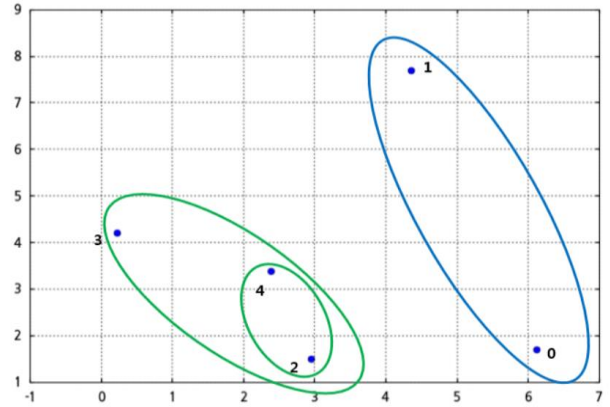
3



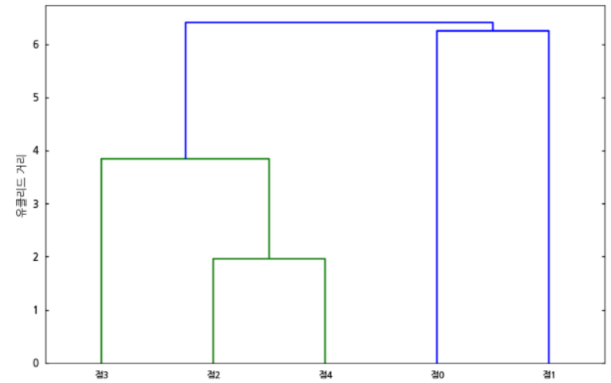
4



5

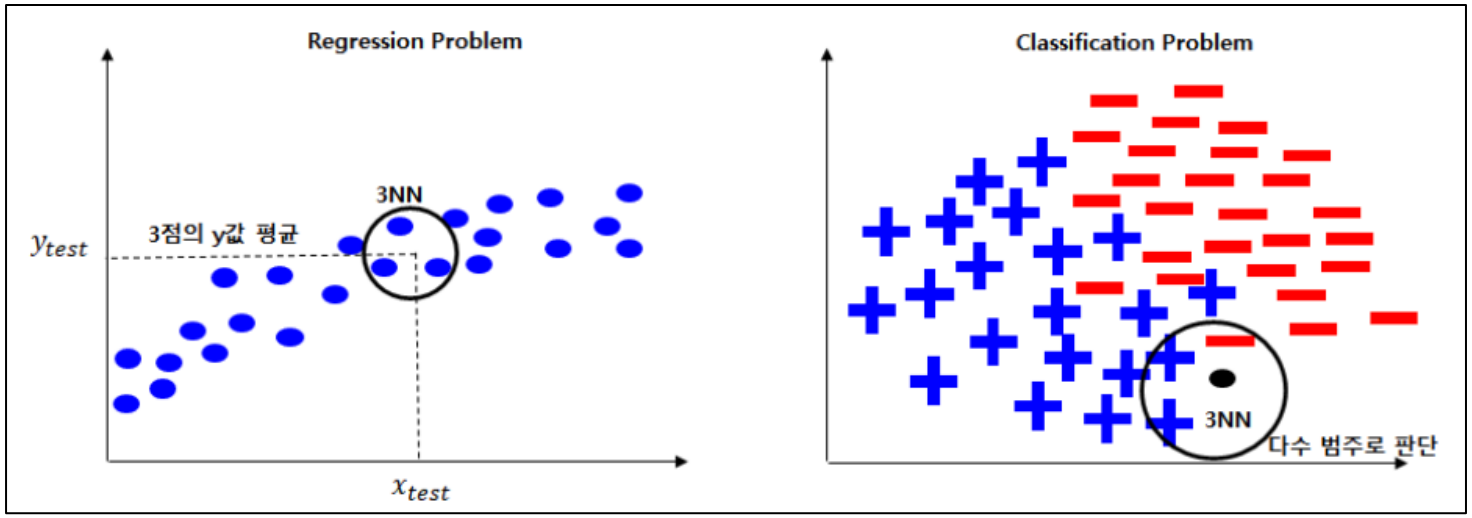


6



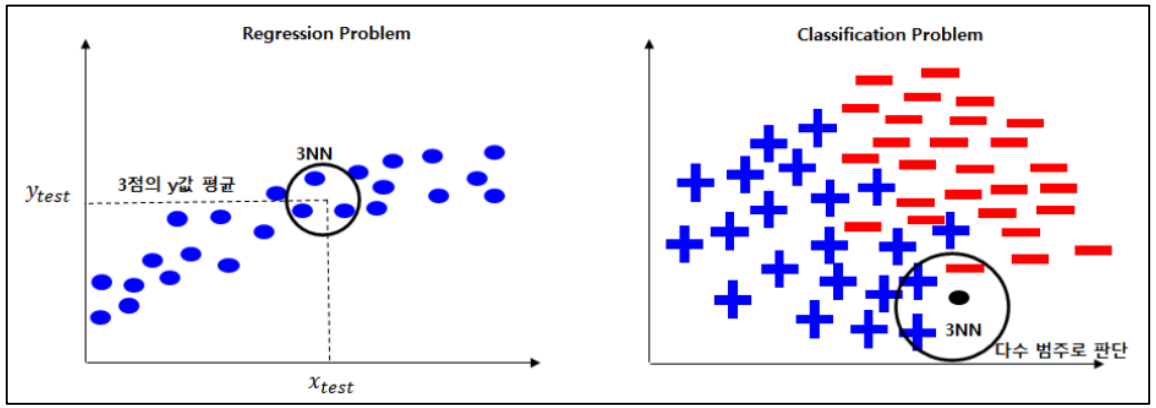
차원 축소와 척도 학습

- **K-최근접 이웃 기법(k-Nearest Neighbor, KNN)**
  - 새로운 데이터가 들어오면 데이터 베이스에 저장되어 있는 데이터(학습 데이터)와 비교하여 **가장 거리가 가까운** (즉, 가장 유사도가 높은) k의 데이터로 예측 또는 분류를 수행
  - 기존 데이터와 **단순 비교**이기 때문에 별도의 모델을 학습하지 않음. 때문에 **lazy learning**이라고 부름.



차원 축소와 척도 학습

- **K-최근접 이웃 기법(k-Nearest Neighbor, KNN)**



Regression : 가장 가까운 k개의 데이터의 **산술 평균**으로 예측
x_test와 가까운 x값을 갖는 k개의 점들의 y값 평균으로 y_test를 추정

Classification : 가장 가까운 k개의 데이터 중 다수 범주로 분류
두 범주의 학습 데이터가 있을 때, 새롭게 주어진 테스트 데이터가 어느 범주에 속하는지 결정하기 위해 가장 가까운 k개의 포인트를 찾고 다수에 해당하는 범주로 분류

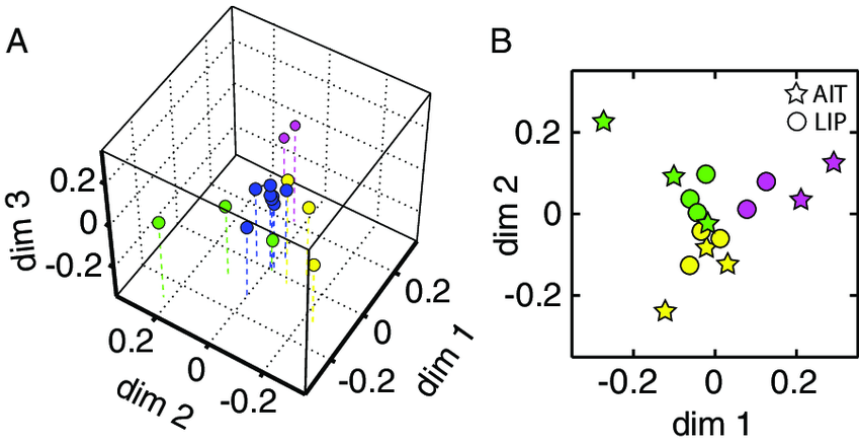
차원 축소와 척도 학습

- **임베딩(embedding)**

  - 사람이 쓰는 **자연어**를 기계가 이해할 수 있도록 **숫자형태인 vector로 바꾸는 과정** 혹은 일련의 전체 과정
  - 고차원 공간 속 하나의 저차원

  - 차원이 높으면 데이터 샘플들은 희소하게 되고, 거리 계산이 어려워 진다 ➔ 차원의 저주(curse of dimensionality)
  - 해결 방법 : **차원 축소**(dimension reduction)
    고차원의 공간을 **저차원**의 subspace로 변환

다차원 스케일링(Multiple Dimensional Scaling, MDS) : **샘플 간 거리를 유지하면서 차원 축소를 함**

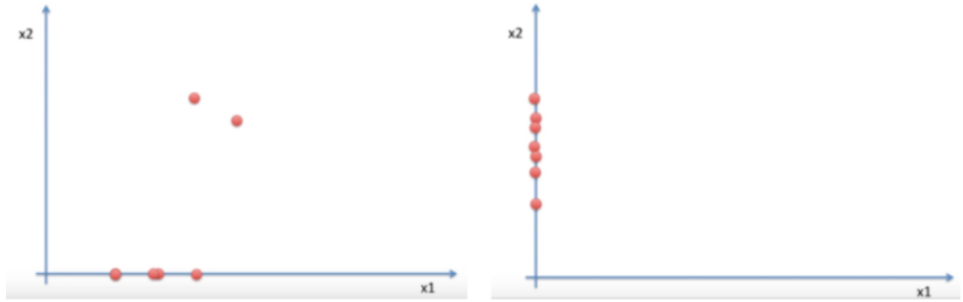


- 거리 **행렬 D**가 주어졌을 때, 임베딩 공간에서 Euclidean distance 인 **|yi – yj|**와 거리 행렬 **δi**의 차이가 최소가 되는 임베딩 y를 학습

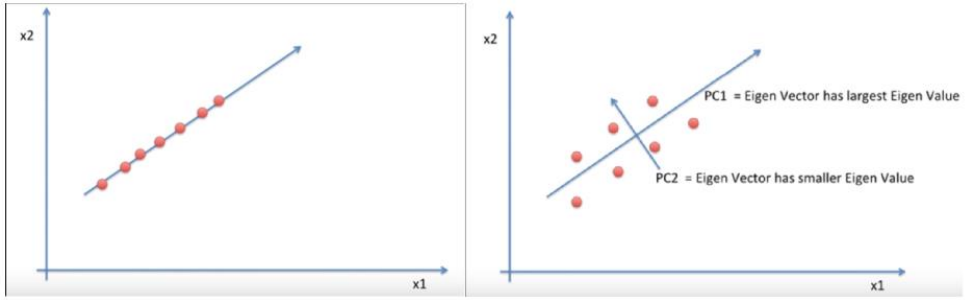
$$minimize \sum_{i<j} \left(|y_i - y_j| - \delta_{ij}\right)^2$$

차원 축소와 척도 학습

- **주성분분석(Principal Component Analysis, PCA)**
  - 가장 자주 사용하는 차원축소법



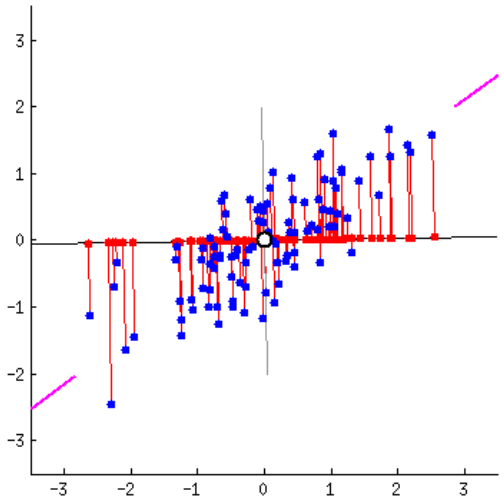
정사영(projection) 되면서 겹치게 되는 문제가 발생.
이 과정에서 **기존의 정보가 유실**된다고 볼 수 있다.



x1, x2 축이 아닌 **새로운 축**을 찾아야 한다.
축은 각 점들이 퍼져있는 정도인 **분산이 최대로 보존 될 수 있도록 해야** 한다.
그리고 이렇게 찾은 축이 **PC(**principal component**)** 이다.

차원 축소와 척도 학습

- **주성분분석(Principal Component Analysis, PCA)**
  - 분산의 최대화



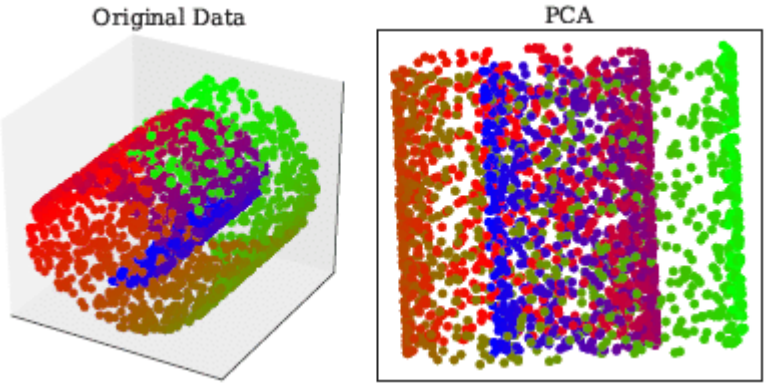
  - 분산이 커져야 데이터들 사이의 차이점이 명확해지고, 모델을 더욱 좋은 방향으로 만들 수 있을 것이기 때문

  - PCA와 MDS의 차이점

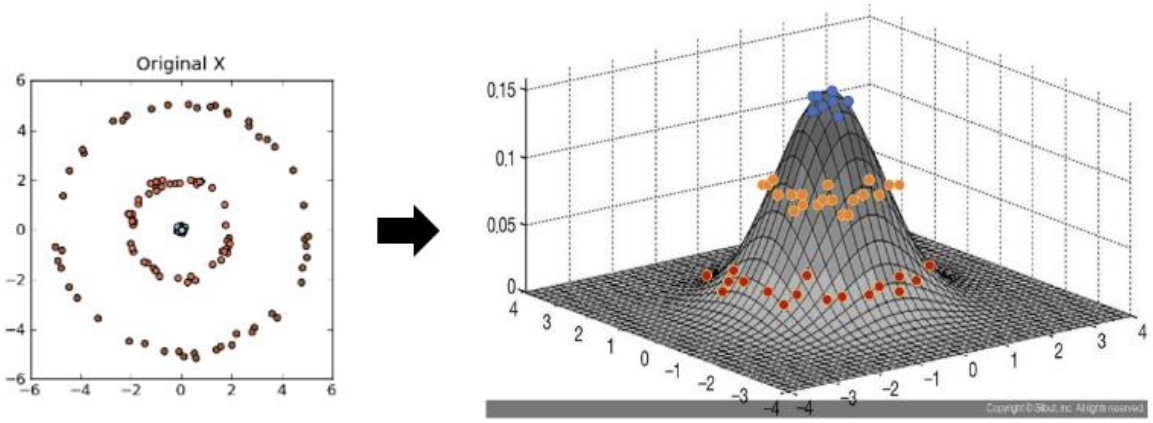
	주성분분석(PCA)	다차원스케일링(MDS)
데이터	d 차원 공간상에 있는 n 개의 인스턴스. {n×d 행렬로부터 시작 (X∈R^d)}	n 개의 인스턴스 간의 근접도(Proximity) 행렬. {n×n 행렬로부터 시작 (Dn×n)}
목적	원 데이터 분산을 보존하는 기저의 부분집합 찾기	인스턴스의 거리 정보를 보존하는 좌표계 찾기
출력값	1) d 개의 고유벡터(eigenvectors) 2) d 개의 고윳값(eigenvalues)	d 차원에 있는 각 인스턴스의 좌표값 (X∈R^d)

차원 축소와 척도 학습

- **커널 선형 차원 축소**
  - **선형 함수**로 **고차원**에서 **저차원**으로 **축소** ➔ 고유 **저차원공간** 구조를 **유실**하게 된다.
  - ➢ 따라서 **비선형 차원축소를 사용하게 됨.**



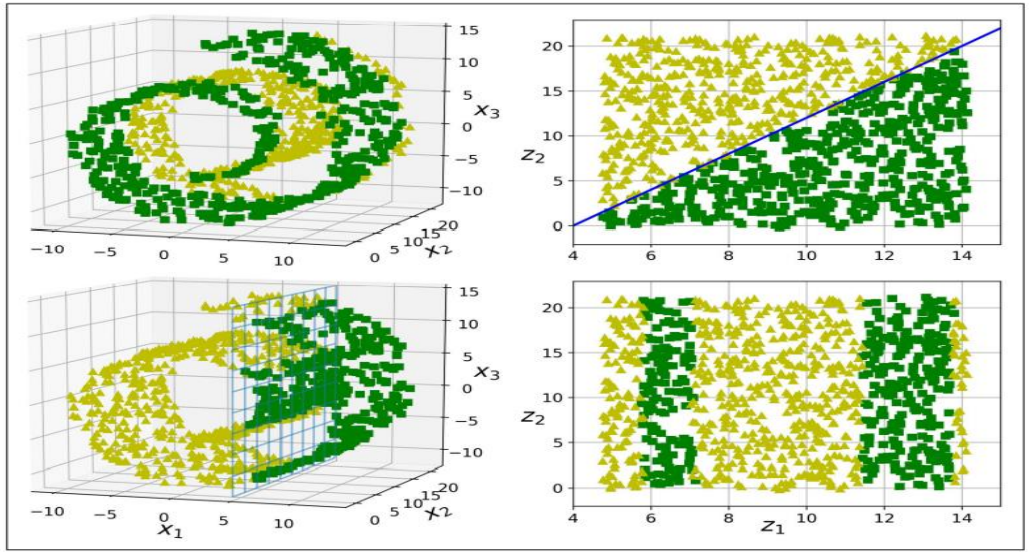
- **커널 비선형 차원 축소(Kernelized PCA)**
  - Kernel 기법은 비선형 함수인 커널함수를 사용하여 **비선형 데이터**를 **고차원 공간으로 매핑**하는 기술



❖ 가우시안 커널 비선형 PCA

차원 축소와 척도 학습

- **매니폴드 학습(manifold learning)**
  - 많은 차원 축소 알고리즘이 훈련 샘플이 놓여 있는 **매니폴드를 모델링**하는 식으로 작동하는 것
  - **매니폴드(manifold)** : 국소적으로 유클리드 공간과 동형인 공간, **고차원 공간에 내재한 저차원 공간**

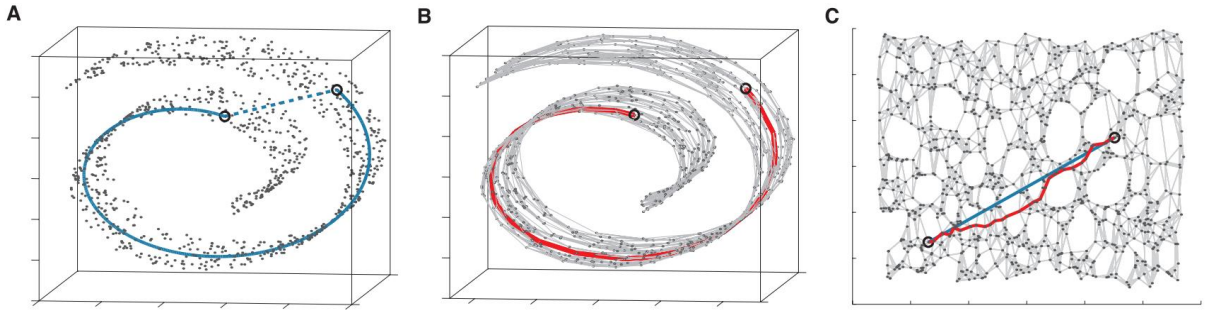


- 복잡한 3D 공간에서의 데이터가 2D 매니폴드 공간에서는 단순한 직선 경계로 나뉜 것을 볼 수 있다.

- 3D 공간에서 경계가 매우 단순하지만, 2D로 펼칠 경우에 더욱 복잡해지는 것을 볼 수 있다.

차원 축소와 척도 학습

- **IsoMap(ISOmetric feature MAPping)**
  - 다차원 척도법(MDS)과 거의 유사하지만 **인스턴스 사이의 거리**를 비선형 데이터 구조에 맞도록 계산함.

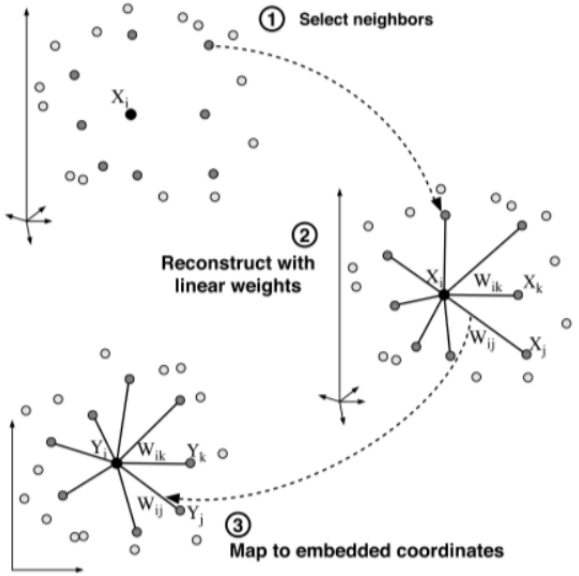


- 점선 : **MDS**
- 실선 : **IsoMap**

B : **A에 있는 모든 인스턴스**를 **그래프 구조**로 나타냄.

차원 축소와 척도 학습

- **LLE(Locally Linear Embedding)**
  - **서로 인접한 데이터 (Locally)** 들을 보존(neighborhood-preserving)하면서 **고차원인 데이터셋을 저차원으로 축소**하는 방법
  - **고유벡터**를 사용하는 방법으로 다루기가 쉽고 **지역 최적점(Local optimum)**에 빠지지 않을 수 있다는 장점



Step 1: Select Neighbors (user set parameter k)
모든 M개 각각의 데이터 기준으로 k개의 가까운 점을 찾는다.

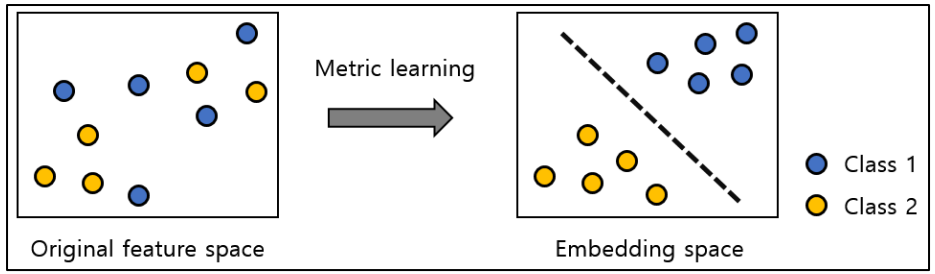
Step 2 : Reconstruct with linear weight
이웃을 찾게 되고, 찾은 이웃들로부터 자기 자신을 가장 잘 재구축 할 수 있는 가중치를 구함.

Step3 : Map to embedded coordinates
가중치가 최대한 보존되도록 데이터를 저차원인 공간으로 매핑

차원 축소와 척도 학습

- **척도 학습(metric learning)**

  - **Metric Learning** : 데이터로부터 **Metric function을 학습**하는 것
  - 비슷함의 정도를 나타내는 방법으로 Metric이 사용되며, 가장 흔히 알고 있는 Metric은 Euclidean Distance

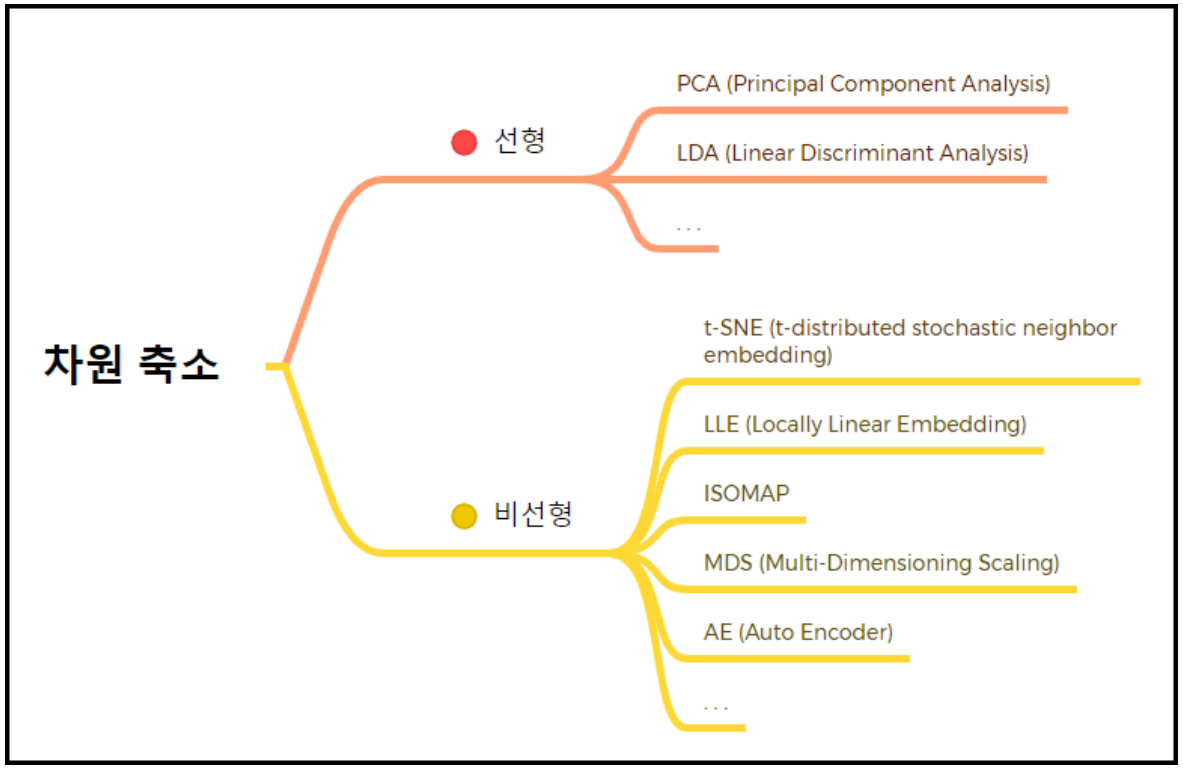


metric learning을 통해 학습된 거리 함수를 f(x;θ) 일때, $d_\theta(\mathbf{x}_1, \mathbf{x}_2) = ||f(\mathbf{x}_1) - f(\mathbf{x}_2)||_2^2$

➢ **metric learning 목적**은 데이터를 각 목표 값에 따라 **잘 구분되도록 변환하는 embedding 함수 f를 학습**하는 것

차원 축소와 척도 학습

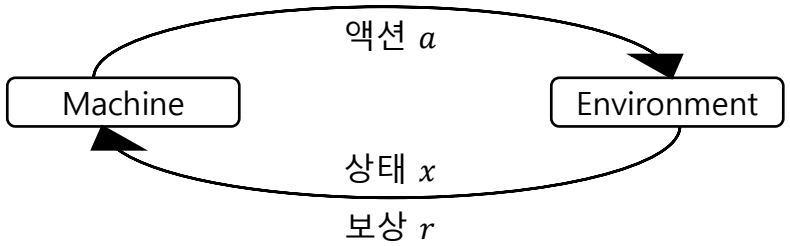
- **차원축소 방법**



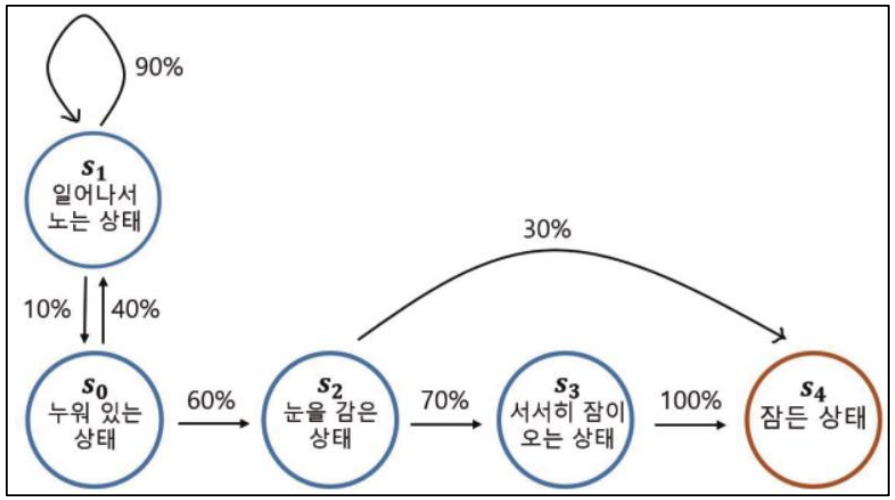
강화학습

- **과업과 보상**

  - **강화학습(reinforcement learning)**



  - **마르코프 프로세스(Markov Decision Process)**



- **마르코프 성질의 뜻**

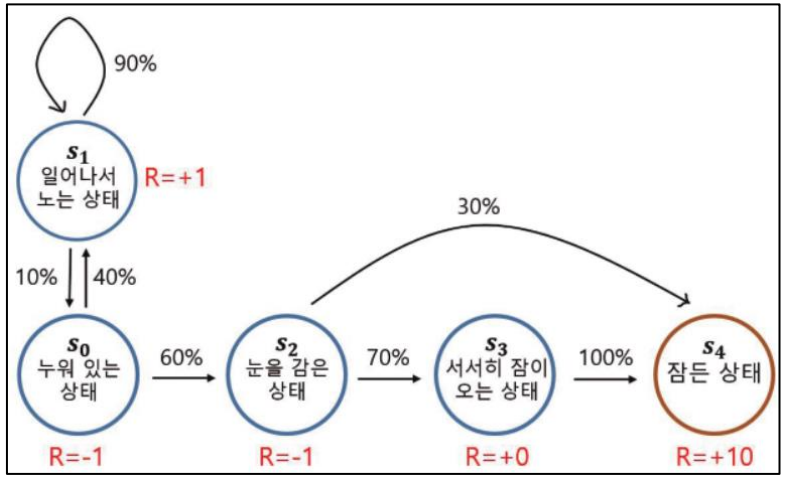
  - "미래는 오로지 현재에 의해 결정된다."
  - 상태(St)가 되기까지의 과정은 확률 계산에 영향 x

- **과업과 보상**
  - **마르코프 프로세스(Markov Decision Process)**

$$P[s_{t+1}|s_t] = P[s_{t+1}|s_1, s_2, \cdots s_t]$$

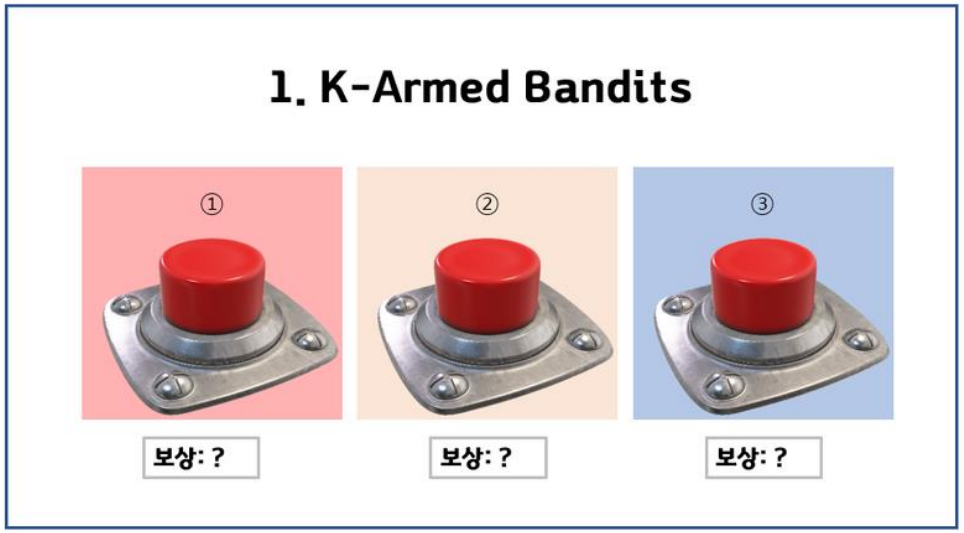
  - 상태(S_t)일때 상태(S_t+1)로 전이 될 확률

  - 마르코프 리워드 프로세스(Markov Reward Process)



아이가 잠이 드는 MP에 빨간색으로 보상 값이 추가된 것

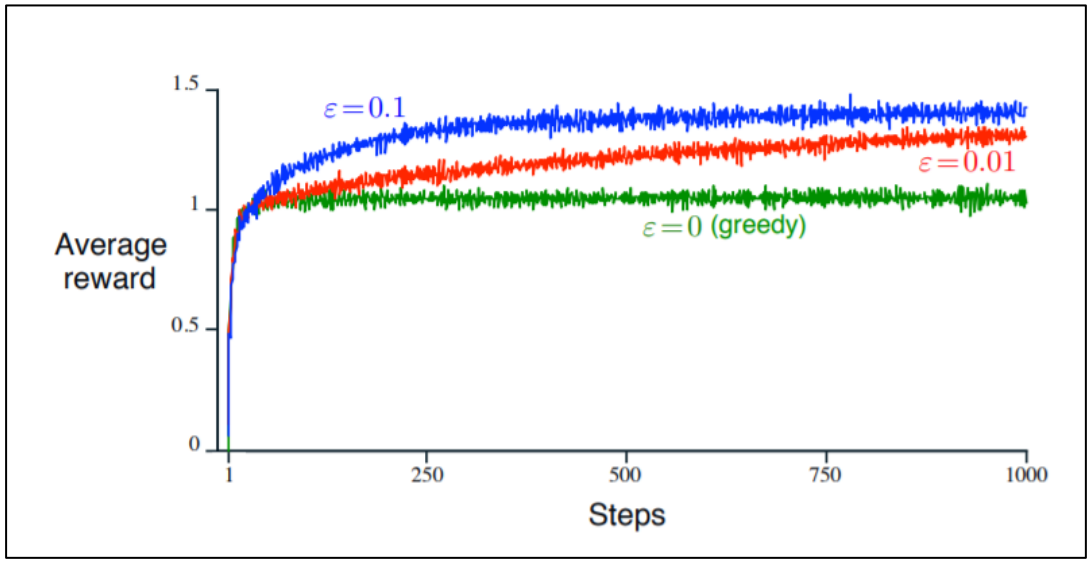
- **K-암드 밴딧 (Armed-bandit)**
  - **N개의 슬롯머신**이 있고, 각각의 슬롯머신은 **수익률이 다름**.
  - 어느 머신에 돈을 걸고 **암(슬롯머신의 손잡이)**을 눌러야할까?
  - 슬롯머신을 밴딧(강도), 손잡이를 암이라고 하고, **성공하기 위해서는 어느 슬롯머신을 노려야 하는 지** 구하는 알고리즘.



- **탐색법(exploration-only)** : 정보를 수집
- **이용법(exploitation-only)** : 주어진 정보로 최선의 선택

❖ Exploration-Exploitation dilemma (trade-off)

- **K-암드 밴딧 (Armed-bandit)**
  - $\boldsymbol{\varepsilon - \text{greedy}}$
  - **Greedy algorithm** : 탐욕 알고리즘, 선택의 순간에 가장 최적의 상황을 선택 ➔ 이 선택으로 이후 모든 선택을 결정
  - $\boldsymbol{\varepsilon - \text{greedy}}$ **:** 매번 동전을 던져서, 앞면이 나오면 새로운 버튼을 눌러보고, 뒷면이 나오면 지금까지의 최적 버튼으로 선택



❖ ε : 앞면이 나올 확률

- **K-암드 밴딧 (Armed-bandit)**
  - **소프트맥스 알고리즘(softmax)**
  - 입력 받은 값을 출력으로 **0~1사이의 값으로 모두 정규화** 하며 **출력 값들의 총합은 항상 1**이 되는 특성을 가진 함수
  - 분류하고 싶은 **클래스수의 수 만큼 출력**으로 구성

$$f(\vec{x})_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_k}} \quad \text{for } i = 1, \ldots, K$$

  - 실제 test data에 모델을 적용할 때는 잘 사용하지 않음. 지수(e) 함수를 사용하면 기하급수적으로 커지기 때문.

- **모델 기반 학습 (model-based learning)**

  - 데이터 셋에 적합한 모델을 만들고, 모델을 이용하여 새로운 데이터를 어떻게 분류할지 예측
  - 모델을 선택할 수 있지만 책에서는 모델을 이미 알고 있다고 가정.

- 상태가치(**State value function**) : 현재 상태가 얻을 **Return의 기댓값** -> 현재 State에 대한 가치를 내놓는 함수

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$$

- 행동가치(**Action value function**) : 지금 행동으로부터 기대되는 Return = 어떤 state s에서 행한 행동(action)에 대한 가치를 평가

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

❖ **Value(가치)** : 큰수의 법칙을 따라 예상되는 **모든 return의 평균**으로써 가치함수를 정의

- **모델 기반 학습 (model-based learning)**

  - **정책 반복(Policy iteration)** : 정책 평가와 정책 개선을 적용해 Bellman 방정식을 푸는 알고리즘
  - **가치 반복(Value iteration)** : 정책반복은 다음 상태의 가치를 정책 함수의 확률과 곱해 모두 더했지만, 가치 반복에서는 탐욕적으로 가장 큰 다음가치를 선택

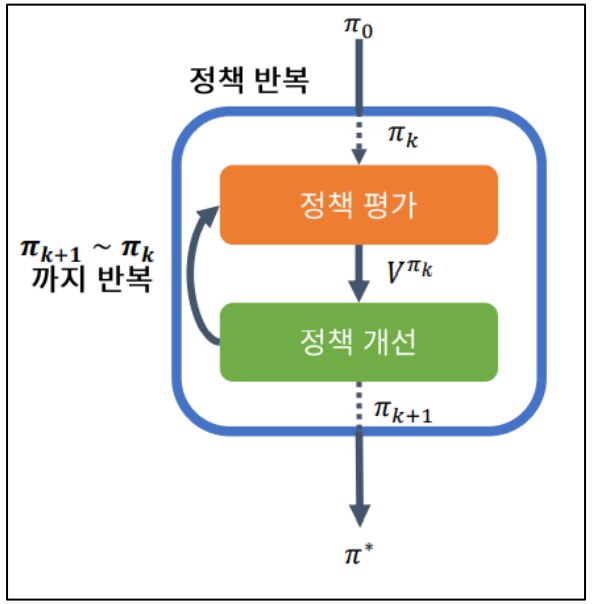
- **Bellman Equation**

  - 각각 상태에 대한 value function 기대값을 얻음으로 인해 이것들을 모두 계산하여 최적의 policy (behavior)를 찾는 과정
  - 현재 state와 다음 state` 간 관계를 나타내는 방정식이라 볼 수 있음

  1. **Bellman Expectation Equation** : 반복적으로 기댓값을 업데이트 하기 위해서, 현재와 다음의 state간 value function 사이 관계를 정의. 특정 policy 를 따를 때의 value function 사이의 관계를 의미하고, 모든 state 에 대한 value function 을 반복적으로 업데이트 하여 real value function값을 구함.

  2. **Bellman Optimality Equation** : 가장 큰 value가 높은 value function을 구하는 policy 를 optimal policy 라고 하는데, Bellman Expectation Equation에 맞추어 더 좋은 policy 를 계속 찾아내면 해당 정책이 최적의 정책. 이때 최적의 value function 값을 나타내는 것을 optimal value function 이라 하며 그때의 policy 가 바로 optimal policy.
  이때의 value function 사이의 관계식을 Bellman Optimality Equation 이라 함.

- **모델 기반 학습 (model-based learning)**
  - **정책 반복(Policy iteration)** : 정책 평가와 정책 개선을 적용해 Bellman 방정식을 푸는 알고리즘
  - **가치 반복(Value iteration)** : 정책반복은 다음 상태의 가치를 정책 함수의 확률과 곱해 모두 더했지만, 가치 반복에서는 탐욕적으로 가장 큰 다음가치를 선택



가치 반복 (Value iteration) 꼭 0 이 아니라 어떤 값으로 시작해도 하나의 값으로 알고리즘의 결과는 수렴함.

입력: 임의의 가치 함수 $V_0(s) \leftarrow 0$ 모든 $s \in S$

출력: 최적 가치 함수 $V^*(s)$

반복: $(k = 0, \ldots,)$:

<Bellman optimal backup>

- 모든 상태 s 에 대해서, $V_{k+1}(s) = \max_{a \in A}(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a V_k(s'))$ 적용
- 모든 상태 s 에 대해서, $V_{k+1}(s) \sim V_k(s)$ 이면, 반복문 탈출

반환: $V^*(s) \leftarrow V_{k+1}$

❖ **정책 반복**은 정책 평가, 정책 개선 **2개의 loop를 돌아야 하지만 가치 반복은 1개의 loop를 돌면** 되기 때문에 계산 복잡도가 정책 반복보다 낮아 수렴 속도가 더 빠름.

- **모델-프리 학습(model-free learning)**
  - **몬테카를로 강화학습(Monte Carlo)**

- Monte Carlo는 모집단의 확률 밀도 함수를 모를 때 Expectation을 추정하기 위해 사용되는 가장 대표적인 방법
- 최초 방문만 따질 것인가 모든 방문에서 따질 것인가에 따라 2가지로 나뉨

✓ **First-Visit Monte Carlo Policy Evaluation**
✓ **Every-Visit Monte Carlo Policy Evaluation**

하나의 episode를 마치면 episode 동안 진행해왔던 state들 마다 return값이 존재.

이때, 중복 방문에 대한 return값을 처리하는 방식에 따라 **first-visit MC와 every-visit MC**로 나누어짐.

first-visit MC는 state에 **첫 번째 방문했을 때의 return값만 취하고** 두 번째 이후는 고려하지 않음.

every-visit MC는 이와 반대로 **방문할 때마다 바뀐 return값으로 계산**.

- **모델-프리 학습(model-free learning)**
  - **시간차 학습(Temporal Difference, TD)**
  - 동적 프로그래밍과 몬테카를로 방법의 아이디어를 결합해 더 효율적인 model-free learning 만듦
  - Monte Carlo는 실시간이 아님 ➔ **타임스텝마다 가치함수를 업데이트함**
  - 시간차 예측에서는 업데이트의 목표 값이 실제 값이 아니라 예측 값임.

- **가치 함수 근사(value function approximation)**
  - 지금까지 살펴본 강화학습은 모두 **action value function**을 **Table(행렬)**로 만들어 푸는 **Tabular Methods 임**
  - **state**나 **action** (차원)이 작을 경우에만 가능하며, Table이 점점 커질수록 값들을 기억할 메모리는 물론이고 학습 자체에도 엄청난 시간이 소요
  - 다양한 실생활 문제에 **Generalization(일반화)**를 하려면, (실제 세상은 continuous state space로, 사실상 무한대의 state를 가지므로) 새로운 방법론을 가져올 필요

- **Linear function approximation**
  - 각 상태의 가치는 상태의 특성(feature)들과, 가중치들을 모두 곱해 더한 값으로 근사

    $\hat{v}(s, w)^{\cdot} = \sum wixi(s)$

    ❖ 더 정확한 상태의 근사는 주로 선형함수의 근사가 아닌, 인공신경망을 통한 비선형 근사로 이루어짐.

강화학습

- **이미테이션 러닝(imitation learning)**
  - 최상의 성능을 달성하기 위해 전문가의 행동을 모방하는 순차적 작업
  - 직접적 보상이 불필요함
  - 정책을 직접 설계하여 전문가가 원하는 행동을 보다 쉽게 발현시킬 수 있음

- **역강화학습(inverse reinforcement learning)**
  - 에이전트의 정책이나 행동이력을 통하여 해당 행동을 설명하는 보상함수를 구하는 알고리즘
  - RL의 역이 되는 학습, 에이전트가 최선의 행동을 선택했다는 가정하에 이 행동에 대한 보상함수를 추정하는 방식

정리

- **Clustering**

✓ validity index - internal, external
✓ Minkowski distance
✓ prototype algorithm- k-means, LVQ, GMM,
✓ Density-based, hierarchical

- **차원축소 및 척도학습**

✓ KNN
✓ MDS, PCA, kernelized PCA
✓ manifold, Isomap, LLE, metric learning

- **Reinforcement learning**

✓ Markov Process, K-armed-bandit(greedy, softmax)
✓ model-based learning - state value, action value, policy&value iteration
✓ model-free learning - Monte Carlo, TD
✓ value function approximation - linear function, imitation learning, inverse reinforcement learning