AI 특강

Prof. Jibum Kim

Department of Computer Science & Engineering

Incheon National University

- **Polynomial regression**

- 지금까지는 입력 변수들과 출력 변수의 관계가 선형 (linear)관계, 2차원 직선, 라고 가정하였다
- 하지만, 이러한 가정은 항상 성립하지 않는다
- 이번에는 비선형 관계인 경우의 회귀분석인 다항 회귀 (polynomial regression)에 대해서 배워본다

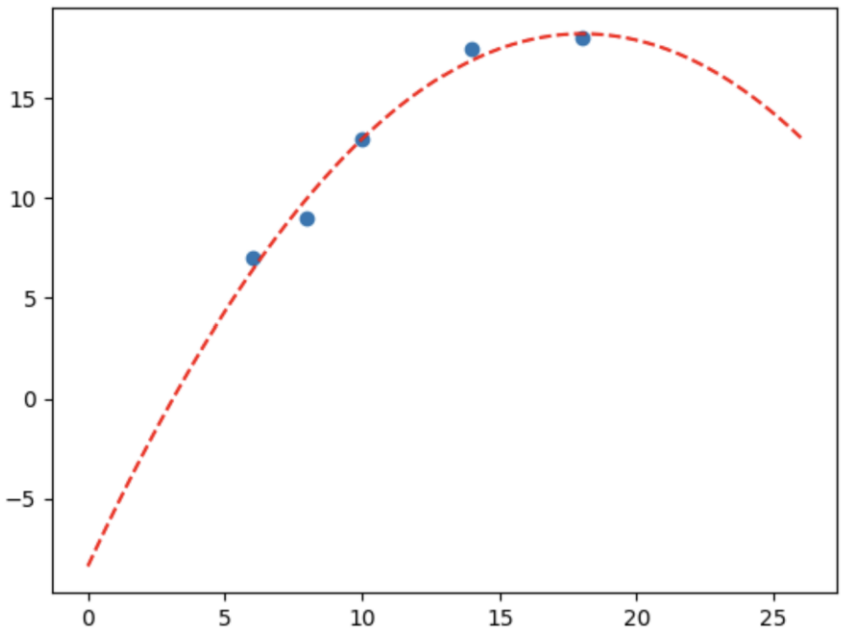
- 1. Quadratic regression (regression with a second order polynomial, 2차 다항식 회귀분석)
- 입력 변수: x, 출력 변수: y
- 수식: $y = \alpha + \beta_1 x + \beta_2 x^2$
- 이번에는 best fit을 통해서 알아내야 하는 계수가 3개로 늘어남

- Scikit-learn 의 Polynomial Regression 모델은 비선형 데이터를 학습하기 위해, 선형 회귀 모델을 사용
- $y = \alpha + \beta_1 x + \beta_2 x^2$
- 즉, 원래 입력 data x와 이를 제곱한 x^2을 먼저 만든 후에 다시 linear regression 수행

- 실습
- 아래와 같은 데이터를 2차 다항식으로 fitting
- $y = \alpha + \beta_1 x + \beta_2 x^2$, 목표: best fit하는 계수들을 polynomial regression으로 찾음

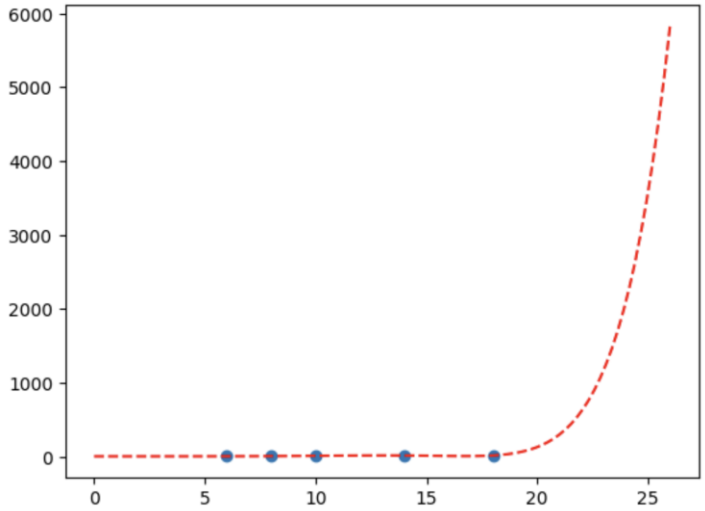
- $y = \alpha + \beta_1 x + \beta_2 x^2$
- **Intercept: $\alpha = -8.39$ …**
- $\beta_1 = 2.95$ …
- $\beta_2 = -0.08$ …

```
[-8.39765458]
[[ 2.95615672 -0.08202292]]
```

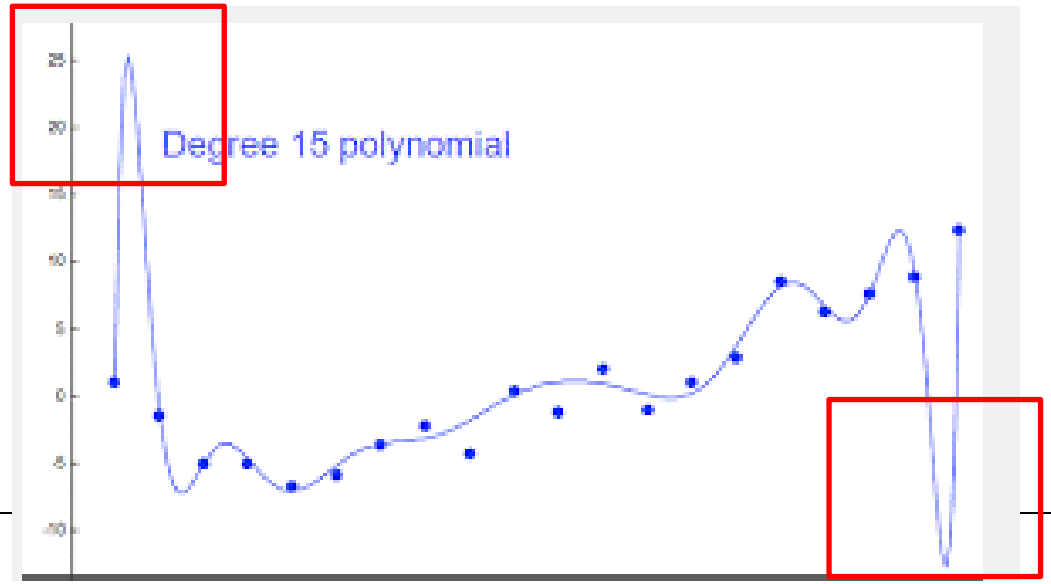


- **Overfitting-underfitting**

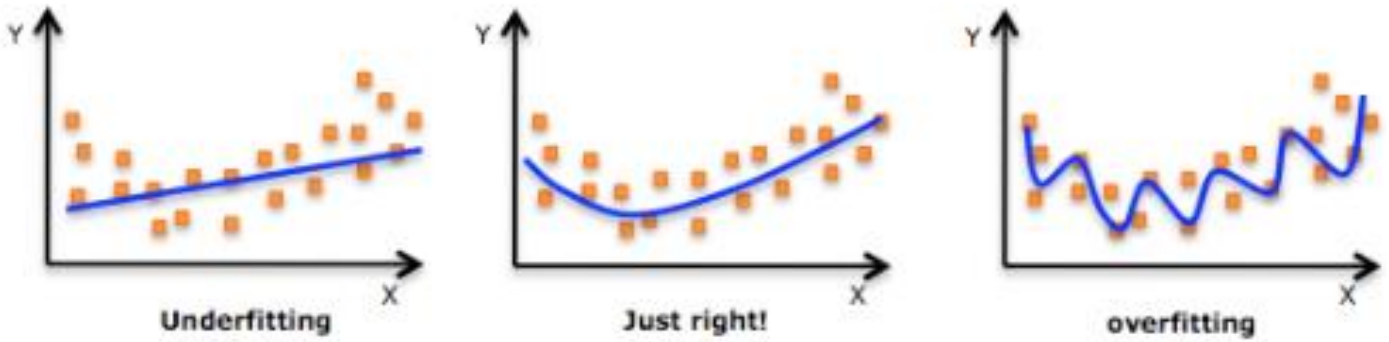
- 다항식의 차수를 올리다보면 거의 오차가 없게 만들 수 있다. 학습 데이터를 거의 완벽하게 지나게 (아래 그림)
- 아래의 경우에는 9차 다항식
- We created an extremely complex model that fits the training data exactly, but fails to approximate the real relationship
- This problem is called over-fitting



- **Overfitting: 과도하게 training data를 regression함으로써 미래의 데이터값을 예측하는데 실패할 확률이 높은 문제**
- **(현재 데이터는 잘 근사화 하겠지만)**



- 아래의 training 데이터들을 살펴보자

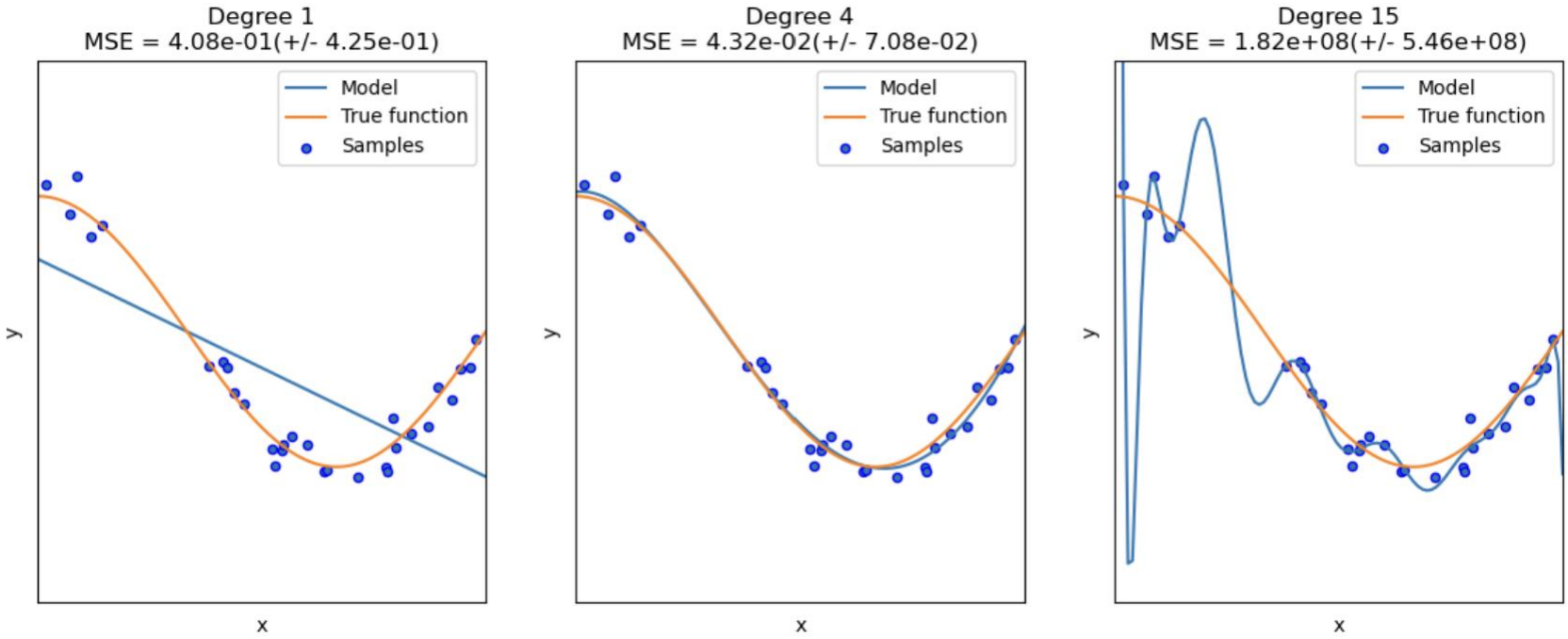


- Underfitting: 근사화한 것이 너무 간단해서 Fitting이 잘 안 된다
- Overfitting: training data에 대해서는 근사화가 잘 됨
  근사화한 것의 차수가 너무 높아서 새로운 데이터가 조금의 변화 (variance)가 생기면 부정확한 결과 생김

- **Scikit-learn의 over-fittiing vs underfitting**

- **http://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html**

- 왼쪽: under fitting, 오른쪽: over-fitting



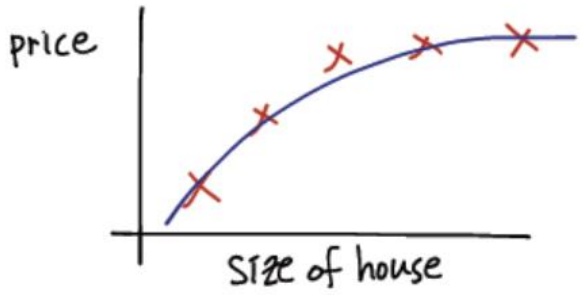
- **Adressing overfitting**

- **Regression시에 입력의 특징 (변수) 개수가 너무 많은 경우에도 overfitting문제가 생길 수 있다**
- **예: y:집값, x_1: 집의 크기, x_2: 침실수, x_3: 층수**
- **x_4: 집지은 연도, x_5: 이웃의 평균 수입, x_6: 부엌의 크기 등등…수십가지 가능**
- **$y=a_1x_1+ a_2x_2+ a_3x_3+ a_4x_4+\ldots..a_{100}x_{100}$**
- **목표: 오차의 제곱의 합 (MSE)을 최소화 하는계수 (a_1, a_2, …. a_{100} 을 찾음)**
- **이러한 경우 학습에 사용된 training data에 맞게 계수들을 찾게 되면 너무 세세하게 데이터에 맞추다 보면 overfitting이 생기기 쉽다**

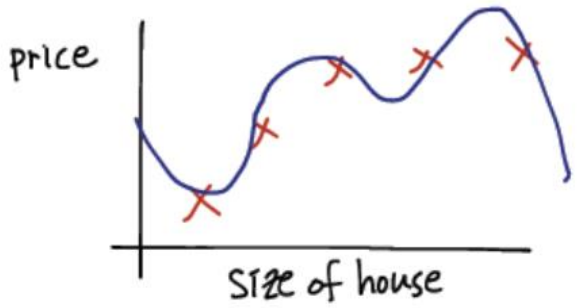
- 우리의 목적은 training set과 완벽하게 똑같은 모델을 만드는 게 아니라 training set에 없는 새로운 데이터에 대해서 값을 정확하게 예측하는 것이다.

- 그러나 training data에 지나치게 맞춰진 모델은 오히려 새로운 데이터를 예측하는 데에는 실패할 수 있다. 이와 같이 training data에 지나치게(over) fit 되어 일반적인 추세를 표현하지 못하는 문제를 overfitting이라 한다

- 예: 왼쪽 (적절한 fitting) vs 오른쪽 (overfitting)



$\theta_0 + \theta_1 x + \theta_2 x^2$



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

- 이 경우 θ_3와 θ_4를 규제하면 적절한 fitting이 되지 않을까? 어떻게 규제? **Idea: Penalty를 주자**

- Adressing overfitting
- 1. 입력의 특징 개수 (변수 개수)를 줄인다
- 수동적으로 어떤 변수를 사용할지 선택한다
- 2. Regularization
- 모든 입력 특징은 유지하대 계수의 크기를 규제 (작게 유지) 한다
- 대부분 입력 특징의 개수가 많은 경우에 사용

Regularization의 이해

- 데이터 (x_1,y_1), … (x_{100}, y_{100})이 있을때
- $y=a_1x_1+a_2x_2+\ldots.a_{100}x_{100}$ 를 찾고자 한다
- 원래 목적: $\min \sum_{i=1}^{n}(\hat{y}_i - y_i)^2$ ($\hat{y}_i$: 예측값)
  해주는 $a_1 \ldots a_{100}$ 찾음
- Regularization 후의 목적
- $\min\{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + 1000a_1^2 + 1000a_4^2\}$
- 해주는 $a_1 \ldots a_{100}$ 찾음…그렇다면 답은?

Regularization의 이해

- Regularization 후의 목적
- $\min\{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + 1000a_1^2 + 1000a_4^2\}$
- 해주는 $a_1 \ldots a_m$ 찾음…그렇다면 답은?

- $a_1=0$, $a_4=0$으로 놓아야 한다. 혹은 a_1,a_4를 아주 작은 값으로 놓아야 한다. 이를 regularization이라 한다

- Regularization을 하고 나면 대부분 계수
- $a_1 \ldots a_{100}$ 들이 작아지게 된다! => overfitting 해소
- 실제로는 이렇게 하지 않고 계수들 앞에
- λ_i를 붙인다 (어떤 값이 최적인지 모르므로)
- $\min\{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + \lambda_1 {a_1}^2 + \cdots \lambda_n {a_n}^2\}$
- λ_i 를 regularization (shrinkage) parameter라고 한다
- (λ_i = 보통 λ로 많이 놓는다)
- 즉, $\min\{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 + \lambda {a_1}^2 + \cdots \lambda {a_n}^2\}$
- λ=0이면 선형 회기분석과 동일 !

Scikit-learn에서의 ridge regression

- # alpha 값 = 앞의 λ 값
- from sklearn import linear_model
- import pylab as pl
- X=[[0,0], [0,0], [1,1]]
- y=[[0], [0.1], [1]]
- clf=linear_model.Ridge(alpha=0.5)
- clf.fit(X,y)
- print clf.coef_
- print clf.intercept_

- 주어진 λ값에서 cross validation test를 통한 최적의 λ값 찾기

1.1.2.2. Setting the regularization parameter: generalized Cross-Validation

`RidgeCV` implements ridge regression with built-in cross-validation of the alpha parameter. The object works in the same way as GridSearchCV except that it defaults to Generalized Cross-Validation (GCV), an efficient form of leave-one-out cross-validation:

```
>>> from sklearn import linear_model
>>> clf = linear_model.RidgeCV(alphas=[0.1, 1.0, 10.0])
>>> clf.fit([[0, 0], [0, 0], [1, 1]], [0, .1, 1])
RidgeCV(alphas=[0.1, 1.0, 10.0], cv=None, fit_intercept=True, scoring=None,
    normalize=False)
>>> clf.alpha_
0.1
```

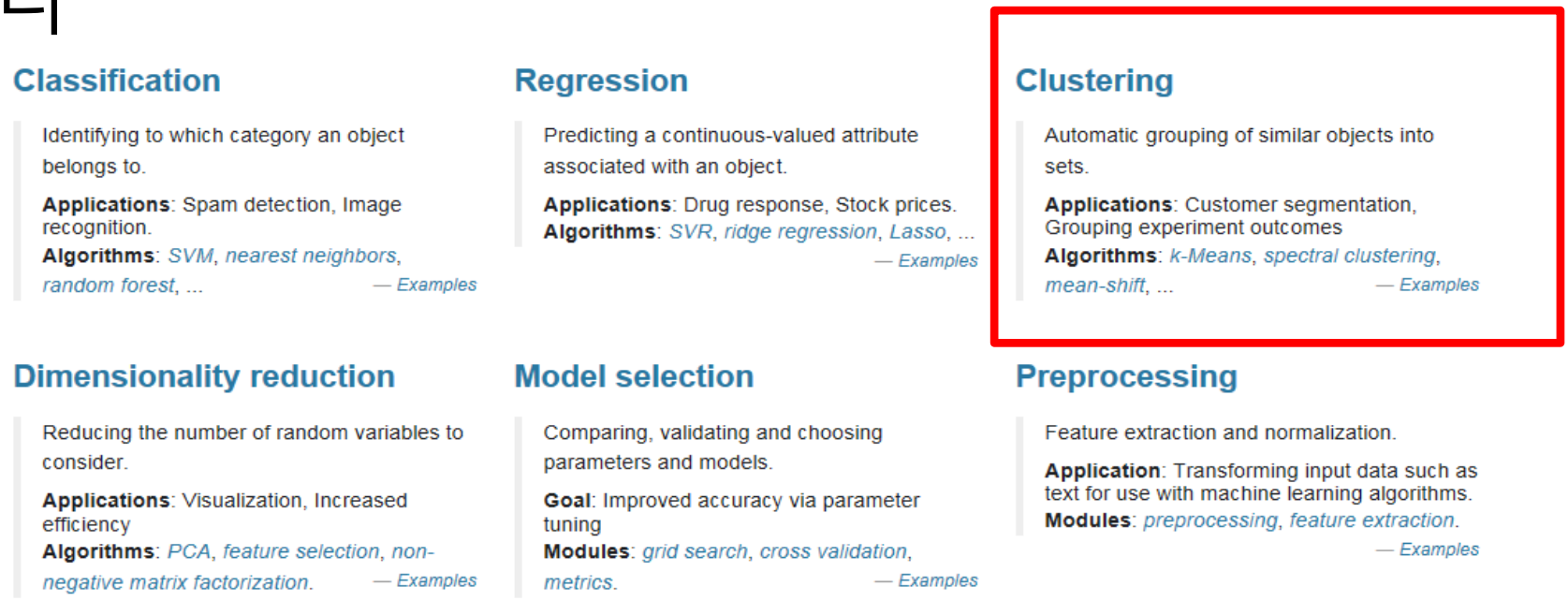
- **군집화 (clustering)**

- 비지도학습이란?
- The goal of unsupervised learning is to discover hidden structure or patterns in unlabeled training data
- 비지도학습의 대표적인 예는 군집화 (clustering)
- 군집화: automatic grouping of similar objects into sets

- Scikit-learn의 machine-learning 방법 분류

- 군집화 (clustering)

- 군집화는 매우 다양한 응용 분야를 가지고 있다



Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: *SVM*, *nearest neighbors*, *random forest*, ... — *Examples*

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: *SVR*, *ridge regression*, *Lasso*, ... — *Examples*

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: *k-Means*, *spectral clustering*, *mean-shift*, ... — *Examples*

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: *PCA*, *feature selection*, *non-negative matrix factorization*. — *Examples*

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: *grid search*, *cross validation*, *metrics*. — *Examples*

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: *preprocessing*, *feature extraction*. — *Examples*

- 군집화란?
- Clustering is the task of grouping a set of objects in such a way that objects in the same group (cluster) are more similar to each other than to those in other groups
- 대표적인 군집화 알고리즘: K-means 알고리즘

- 군집화 사용 예시

- 어떤 사람이 몇년간 쇼핑몰을 성공적으로 운영해 왔다 (수백만명의 고객을 가지고 있다). 그 동안 한 종류의 홍보 판플렛을 만들어 발송했는데 이제부터는 고객의 취향을 분석하여 4-6종류의 판플렛을 만들어 맞춤 홍보를 하고자 한다. 일종의 개인화 (personalization) 홍보 전략이다. 고객에 대한 각종 정보는 DB에 저장되어 있어 이것을 기초 자료로 활용 가능하다

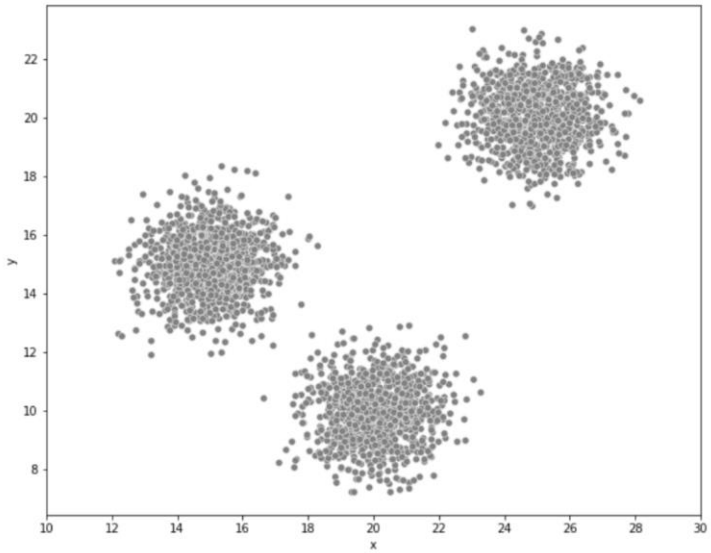
- 고객 정보는 월평균 구매액, 선호하는 물품의 종류와 수준, 결제 방법, 반품 성향, 직업, 성별, 나이, 거주 지역등 다양하다.

- Q) 수백만명의 고객이 있을때 이를 어떻게 4-6개의 그룹으로 나눌수 있을까?

- 해결 방법: 유사한 샘플(데이터)들끼리 모아서 4-5개의 그룹으로 만든다
- 여기서 유사하다는 것은 데이터와의 거리가 가깝다는 것을 의미 한다
- 이 각각의 그룹을 군집 (cluster)라고 하고 이 경우에는 4-5개의 군집이 만들어 진다
- 여태까지 배운 Supervised learning과 다르게 오늘 배울 군집화 (clustering)은 unsupervised learning으로써 데이터 정보만 있을 뿐 이 데이터의 분류는 주어져 있지 않다

K-means 알고리즘

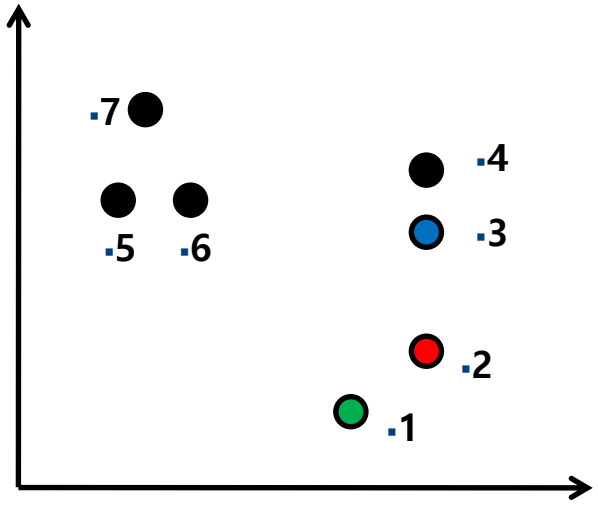
- K-means 군집화 알고리즘은 주어진 데이터를 K개의 클러스터 (군집)으로 묶는 알고리즘이다
- 아래와 같은 데이터의 갱우 K=3이라고 생각할 수 있다
- K-means 군집화 알고리즘의 경우 기본적으로 입력으로 사용자가 K를 정하지만 뒤에서 배울 elbow 방법과 같은 방법으로 K를 정할수도 있다



- 다음 예제는 총 7개의 데이터에 대해서 K-means 군집화 알고리즘을 실행한 예제이다

- 이 경우에 K=3으로 사용자가 정했다

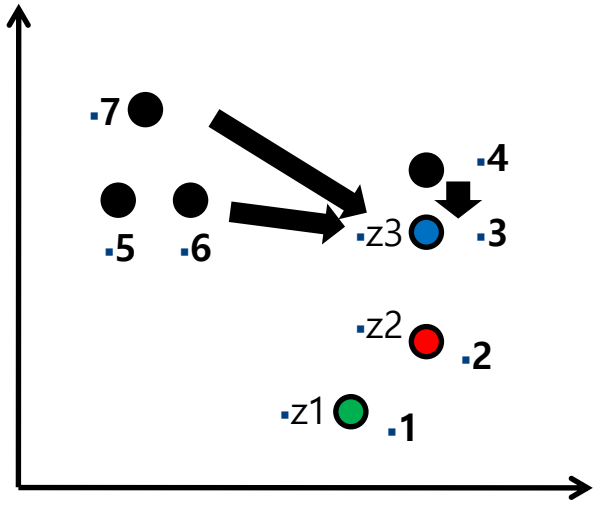
- k (군집개수)=3이고 7개의 데이터가 있는 경우를 생각해 보자



1. x1, x2, x3를 초기 군집 중심으로 삼고
- 이를 z1, z2, z3라고 놓는다
- z1=(18,5)
- z2=(20,9)
- z3=(20,14)

- x1=(18,5), x2=(20,9), x3=(20,14), x4=(20,17), x5=(5,15), x6=(9,15), x7=(6,20

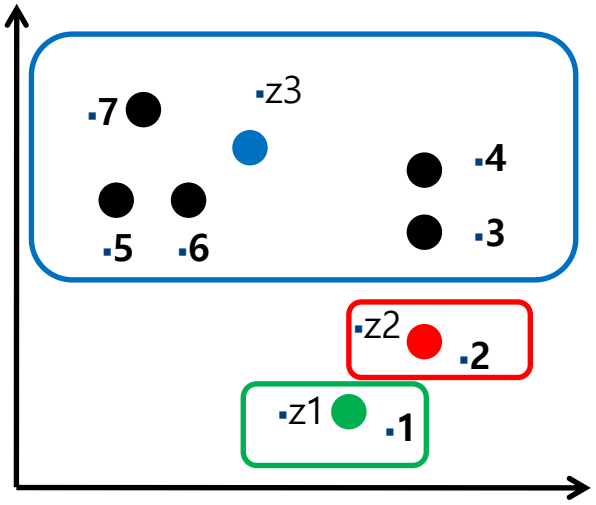
- k (군집개수)=3이고 7개의 데이터가 있는 경우를 생각해 보자



- 2. 각 데이터들은 z1,z2,z3중
- 가장 가까운 군집 중심에 각각 배정된다

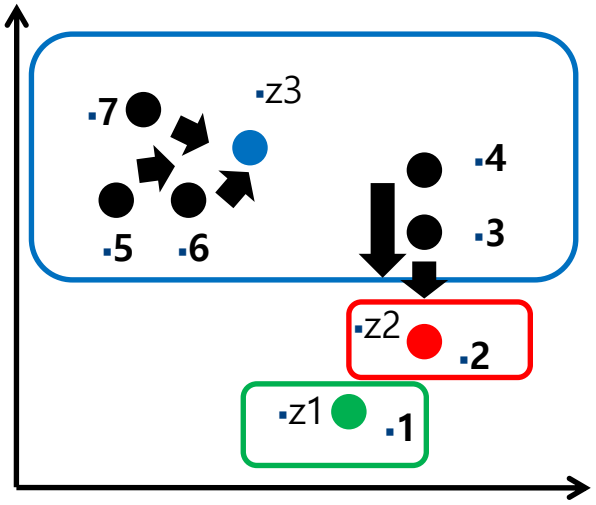
- {x1}은 z1
- {x2}은 z2
- {x3,x4,x5,x6,x7}은 z3 에 배정된다

- k (군집개수)=3이고 7개의 데이터가 있는 경우를 생각해 보자



- 2. {x1}은 z1
- {x2}은 z2
- {x3,x4,x5,x6,x7}은 z3 에 배정된다
- 3. 2에서 배정된 데이터들을 이용해 새로운
- 군집 중심을 찾고 (위치 평균)
- 이를 새로운 z1,z2,z3라 놓는다
- z1=(18,5)
- z2=(20,9)
- z3=(x3+x4+x5+x6+x7)/5=(12,16.2)

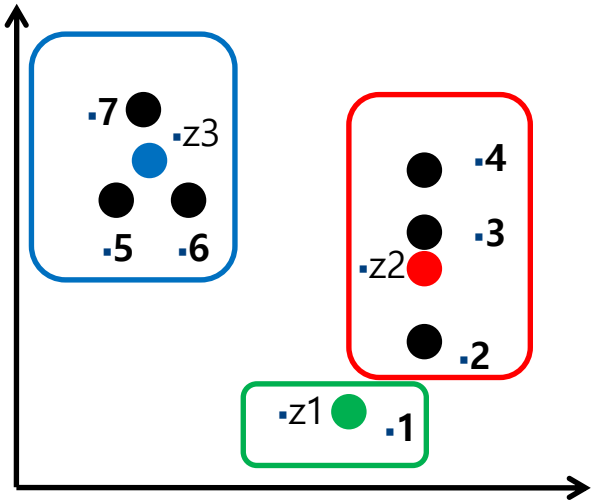
- k (군집개수)=3이고 7개의 데이터가 있는 경우를 생각해 보자



- 앞의 step 2를 반복한다. 즉
- 각 데이터들은 가장 가까운 군집 중심에
- 각각 배정된다

- {x1}은 z1
- {x2,x3,x4}는 z2
- {x5,x6,x7}은 z3 에 배정된다

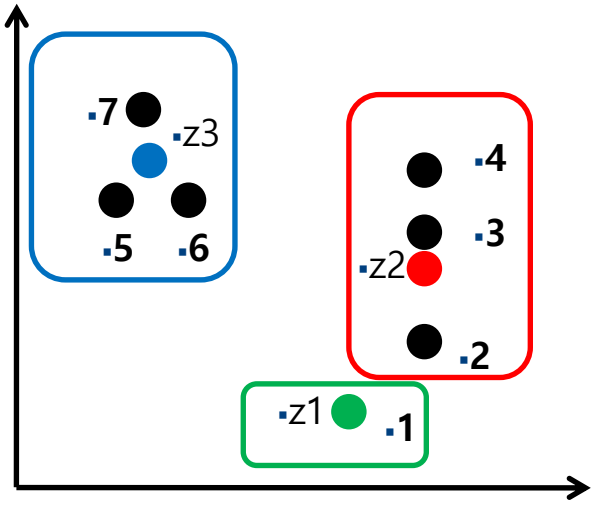
- k (군집개수)=3이고 7개의 데이터가 있는 경우를 생각해 보자



- {x1}은 z1
- {x2,x3,x4}는 z2
- {x5,x6,x7}은 z3 에 배정된다
- 앞의 step 3를 반복한다. 즉
- 배정된 데이터들의 군집 중심을 찾고
- 이를 새로운 z1, z2, z3라 놓는다
- z1=(18,5), z2=(x2+x3+x4)/3=(20,13.333)

- z3=(x5+x6+x7)/3=(6.667,16.667)

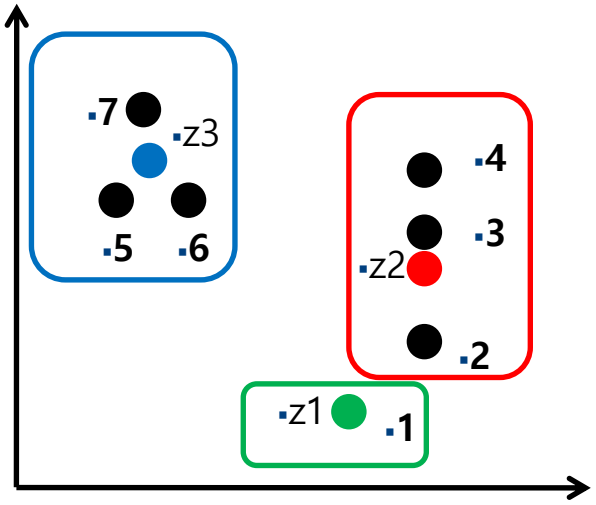
- k (군집개수)=3이고 7개의 데이터가 있는 경우를 생각해 보자



- 그 다음에 step 2와 3를 반복한다
- Step 2: 각 데이터들은 가장 가까운
- 군집 중심에 각각 배정된다
- 변화가 있는가? 없음
- Step 3: 배정된 데이터들의 각각의
- 군집 중심을 찾고 (위치 평균)
- 이를 새로운 z1,z2,z3라 놓는다

- 따라서 step3의 변화도 없음

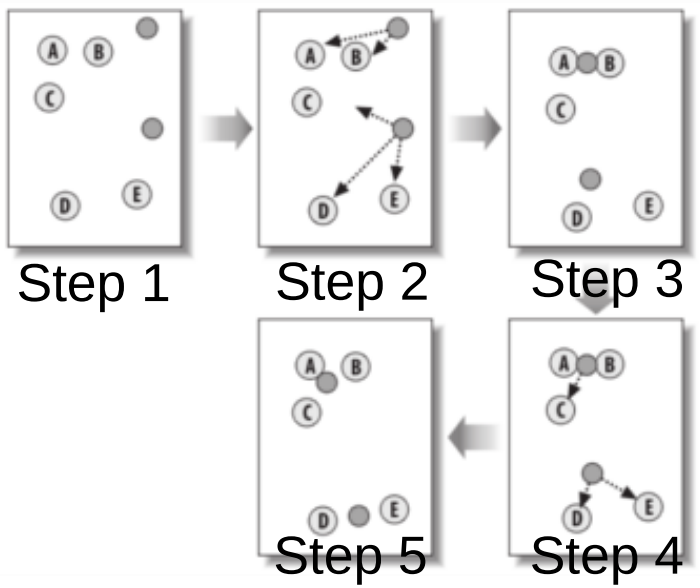
- k (군집개수)=3이고 7개의 데이터가 있는 경우를 생각해 보자



- 즉 최종적으로 왼쪽의 데이터들을
- K=3 (군집개수=3)으로 군집화 한
- 경우
- {x1}, {x2,x3,x4}, {x5,x6,x7}과 같이
- 세 개의 군집으로 군집화 할 수 있다

K-means 알고리즘

- K=2이고 5개의 데이터 (A, B, C, D, E)가 있는 경우
- 1. 임의의 위치에 군집 중심 2개가 위치
- 2. 각각의 데이터가 가장 가까운 군집 중심을 찾음. 이 경우에는 A, B는 위쪽 군집 중심과 가깝고, C, D, E는 아래쪽 군집 중심과 가까움
- 3. A, B의 평균 위치가 새로운 군집 중심의 위치, C,D, E의 평균 위치가 새로운 군집 중심의 위치
- 4. 다시 2 반복. 이 경우 A, B, C는 위쪽 군집 중심과 가까움. D, E는 아래쪽 군집 중심과 가까움
- 5. 다시 3 반복.
- 언제까지 2,3 반복?

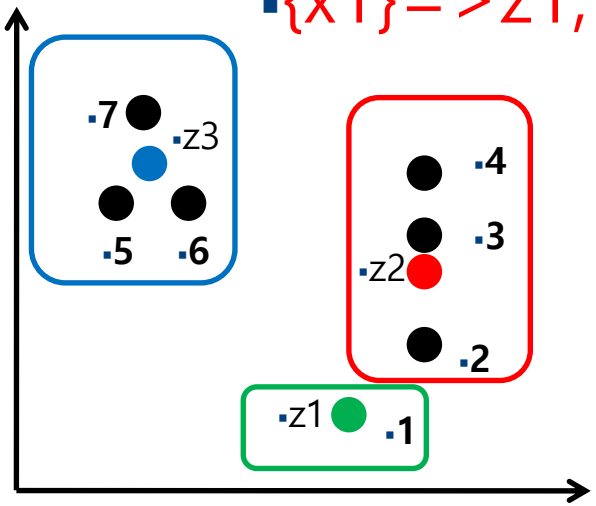


- SSE와 Elbow method

- 군집화가 잘 되었다라는것은 어떻게 판단하나?
- K-means 에서 군집화가 잘 되어 있다 라는건 데이터들이 군집 중심 근처에 모여 있는 것이다. 데이터 들이 군집내에서 퍼져 있는건 군집화가 잘되지 않았다라고 생각할 수 있다
- 군집화가 잘되었나 판단하는 척도 (metric) 중 하나는
- 각각의 데이터가 그것에 가장 가까운 군집 중심 사이의 거리 (혹은 거리의 제곱)을 모두 더해 보는 것이다
- 이를 Sum of squared error (SSE)라고도 한다

- 예를 들어 마지막 최종 군집화 결과의 경우

- {x1}=>z1, {x2,x3,x4}=>z2, {x5,x6,x7}=>z3



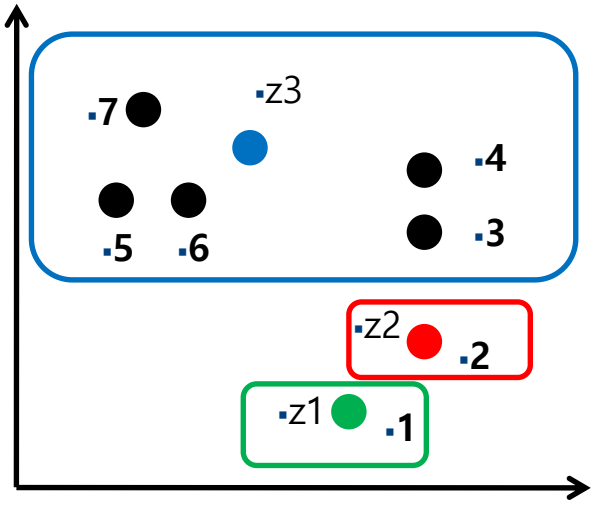
- 예를 들어 마지막 최종 군집화 결과의 경우

- {x1}=>z1, {x2,x3,x4}=>z2, {x5,x6,x7}=>z3

- SSE: 제곱 오류

- SSE=58

- 최종 군집화 바로 이전의 경우



- {x1}=>z1, {x2}=>z2, {x3,x4,x5,x6,x7}=> z3

- SSE=244.8 => 즉 최종 군집화된 결과가 제곱 오류가 더 적다!

- Iris dataset을 이용한 군집화 예제

- Iris dataset

총 150개의 데이터

데이터설명 : 아이리스(붓꽃) 데이터에 대한 데이터이다. 꽃잎의 각 부분의 너비와 길이등을 측정한 데이터이며 150개의 레코드로 구성되어 있다. 아이리스 꽃은 아래의 그림과 같다. 프랑스의 국화라고 한다.

필드의 이해 :

데이터의 이해를 돕기 위해 포함된 6개의 변수에 대하여 간략하게 설명한다.
총 6개의 필드로 구성되어있다. caseno는 단지 순서를 표시하므로 분석에서 당연히 제외한다.
2번째부터 5번째의 4개의 필드는 입력 변수로 사용되고, 맨 아래의 Species 속성이 목표(종속) 변수로 사용된다.

caseno	일련번호이다. (1부터 150까지 입력된다.)
Sepal Length	꽃받침의 길이 정보이다.
Sepal Width	꽃받침의 너비 정보이다.
Petal Length	꽃잎의 길이 정보이다.
Petal Width	꽃잎의 너비 정보이다.
Species	꽃의 종류 정보이다. setosa / versicolor / virginica 의 3종류로 구분된다.

- UCI Machine Learning Repository: Iris Data Set

```
from sklearn.datasets import load_iris

iris = load_iris()
iris
```

This code gives:

```
{'data': array([[5.1, 3.5, 1.4, 0.2],
                [4.9, 3. , 1.4, 0.2],
                [4.7, 3.2, 1.3, 0.2],
                [4.6, 3.1, 1.5, 0.2],...
'target': array([0, 0, 0, ... 1, 1, 1, ... 2, 2, 2, ...
'target_names': array(['setosa', 'versicolor', 'virginica'], dtype='<U10'),
...}
```

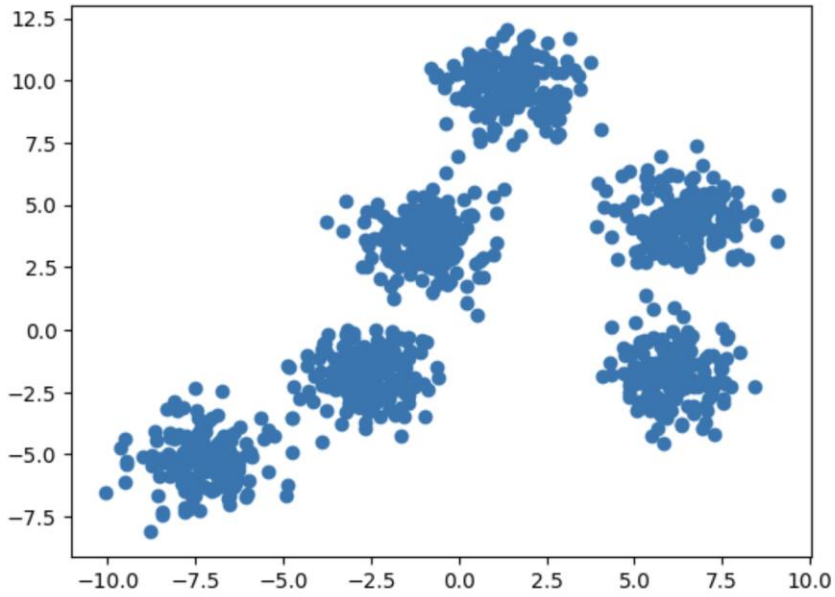
Iris data 군집화 실습

가상의 데이터를 생성하여

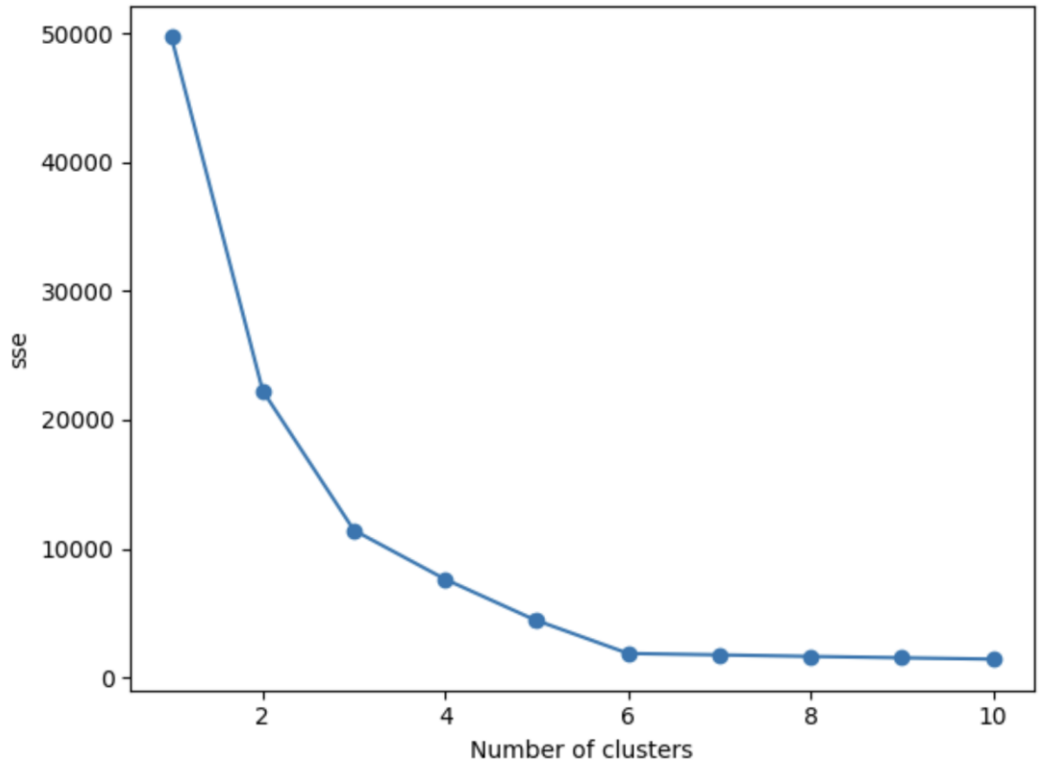
군집화 실험

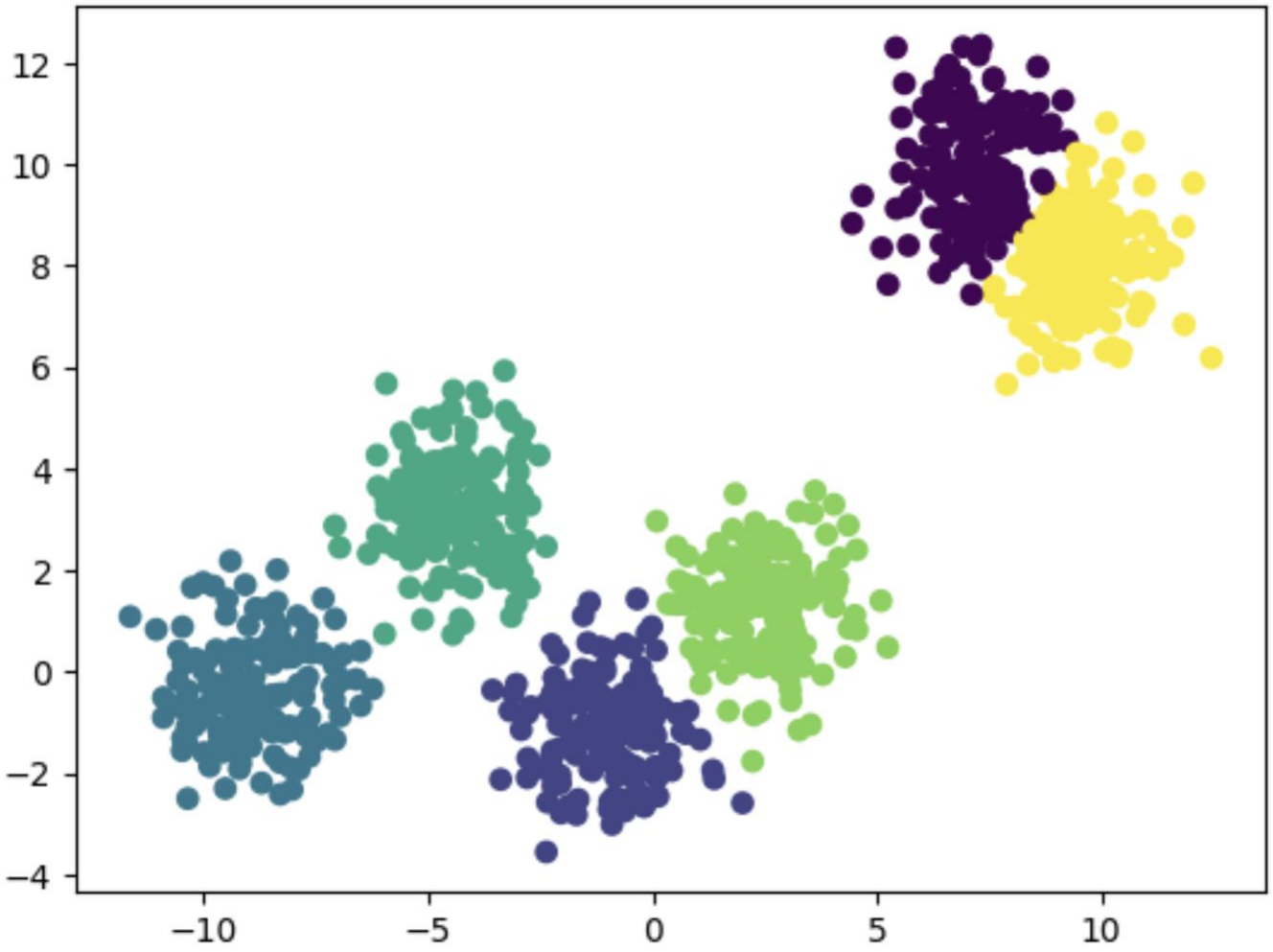
- Scikit-learn의 ‘make_blobs’함수를 사용하여 가상의 데이터 생성

- 예: 2차원 데이터, 군집 개수 6개, 1000개 데이터 생성



- Elbow method로 K (cluster 개수) 추정 K=6?




12
10
8
6
4
2
0
−2
−4
−10
−5
0
5
10

- 퍼셉트론 (Perceptron)

- 신경망은 machine learning (기계 학습) 역사에서 가장 오래된 기계 학습 모델

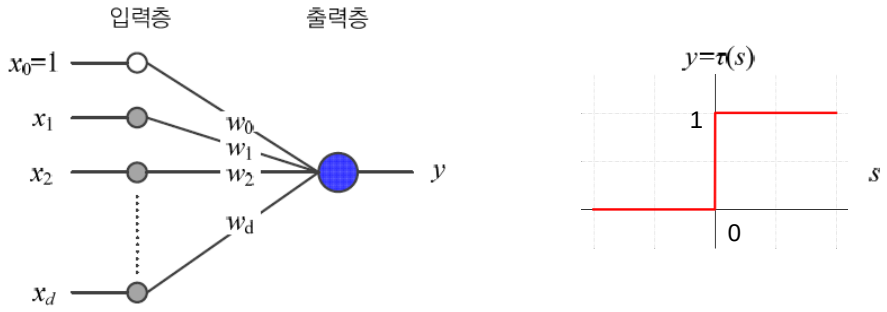
- 1950년대 퍼셉트론 -> 1980년대 다층 퍼셉트론
- 최근 딥러닝의 가장 기초

- 2000년대 딥러닝 등장

- 퍼셉트론 (perceptron)은 노드, 가중치, 층과 같은 새로운 개념을 도입하고 학습 알고리즘을 창안함

- 퍼셉트론은 원시적 신경망이지만, 딥러닝을 포함한 현대 신경망은 퍼셉트론을 병렬과 순차 구조로 결합하여 만듬 → 현대 신경망의 중요한 구성 요소

- 퍼셉트론이란 가장 간단한 형태의 인공 신경망으로 입력과 출력은 모두 숫자이며 입력은 가중치 (weight)라는 것과 연결되어 있다
- 여기서 젤 위에 있는 것은 바이어스 노드라 하고 출력은 한 개의 노드 이다
- 퍼셉트론에서는 다음 2가지 과정을 순차적으로 수행한다
- 1. 입력과 각각의 가중치를 모두 곱해서 더한 가중치합을 계산한다
- $s = w_0 + w_1 x_1 + \cdots w_d x_d = w_0 + \sum_{i=1}^{d} w_i x_i$
- 2. 이 가중치 합에 활성함수라고 하는 계단 함수를 통과시킨다.
- 1에서 계산한 s가 0보다 크거나 같으면 1을 출력, s가 0보다 작으면 0을 출력



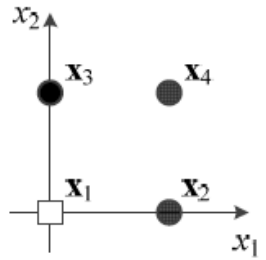
(a) 퍼셉트론의 구조 (b) 계단함수를 활성함수 $\tau(s)$로 이용함

그림 3-3 퍼셉트론의 구조와 동작

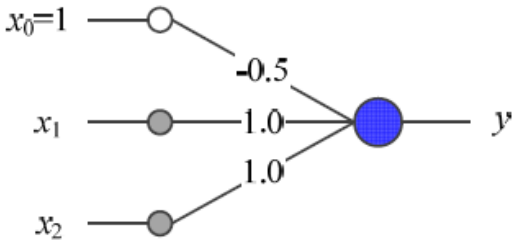
예제 3-1 퍼셉트론의 동작

2차원 특징 벡터로 표현되는 샘플을 4개 가진 훈련집합 $\mathbb{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$, $\mathbb{Y} = \{y_1, y_2, y_3, y_4\}$를 생각하자. [그림 3-4(a)]는 이 데이터를 보여준다.

$$\mathbf{x}_1 = \begin{pmatrix}0\\0\end{pmatrix},\ y_1 = -1,\ \ \mathbf{x}_2 = \begin{pmatrix}1\\0\end{pmatrix},\ y_2 = 1,\ \ \mathbf{x}_3 = \begin{pmatrix}0\\1\end{pmatrix},\ y_3 = 1,\ \ \mathbf{x}_4 = \begin{pmatrix}1\\1\end{pmatrix},\ y_4 = 0$$



(a) 훈련집합



(b) 퍼셉트론

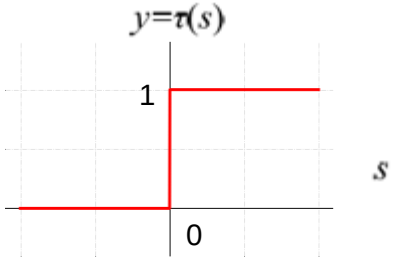
그림 3-4 OR 논리 게이트를 이용한 퍼셉트론의 동작 예시

▪OR Gate

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

샘플 4개를 하나씩 입력하여 제대로 분류하는지 확인해 보자.

$$\begin{aligned}
&\mathbf{x}_1\text{: } s = -0.5 + 0 * 1.0 + 0 * 1.0 = -0.5, && \tau(-0.5) = 0\\
&\mathbf{x}_2\text{: } s = -0.5 + 1 * 1.0 + 0 * 1.0 = 0.5, && \tau(0.5) = 1\\
&\mathbf{x}_3\text{: } s = -0.5 + 0 * 1.0 + 1 * 1.0 = 0.5, && \tau(0.5) = 1\\
&\mathbf{x}_4\text{: } s = -0.5 + 1 * 1.0 + 1 * 1.0 = 1.5, && \tau(1.5) = 1
\end{aligned}$$



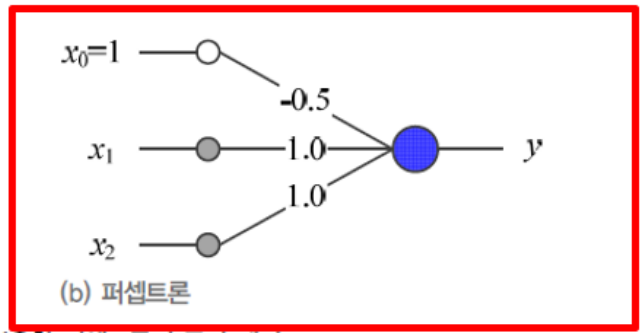
결국 [그림 3-4(b)]의 퍼셉트론은 샘플 4개를 모두 맞추었다. 이 퍼셉트론은 훈련집합을 100% 성능으로 분류한다고 말할 수 있다.

- Python으로 구현한 'OR' 퍼셉트론 예

```
def OR(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5,0.5])
    b = -0.4
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else :
        return 1

import numpy as np
print(OR(0,0))
print(OR(0,1))
print(OR(1,0))
print(OR(1,1))
```

- 이 퍼셉트론을 기하학적으로 설명하면
  - 결정 직선 $d(\mathbf{x}) = d(x_1, x_2) = w_1 x_1 + w_2 x_2 + w_0 = 0$ → $x_1 + x_2 - 0.5 = 0$
    - w_1과 w_2는 직선의 방향, w_0은 절편을 결정
    - 결정 직선은 전체 공간을 1과 0의 두 부분공간으로 분할하는 분류기 역할
- 즉, 퍼셉트론은 선형 분류기라고 생각할 수 있다



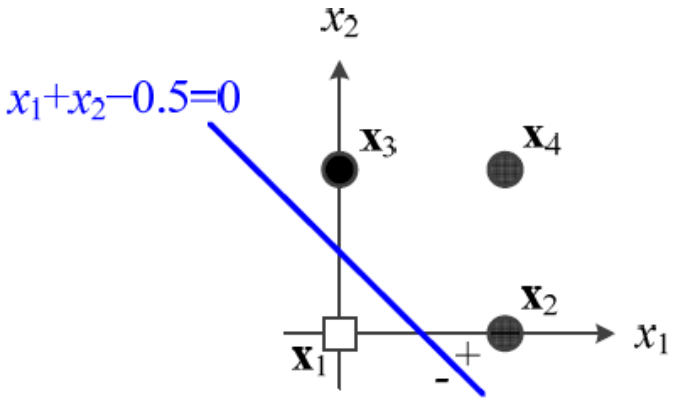
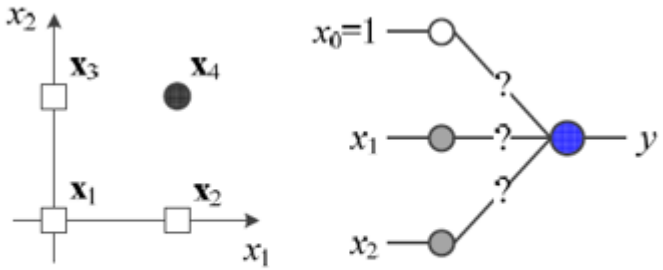


그림 3-5 [그림 3-4(b)]의 퍼셉트론에 해당하는 결정 직선

- 실습: 다음과 같은 AND Gate를 선형 분류할 수 있는 퍼셉트론을 직접 구현해보자

▪AND Gate

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



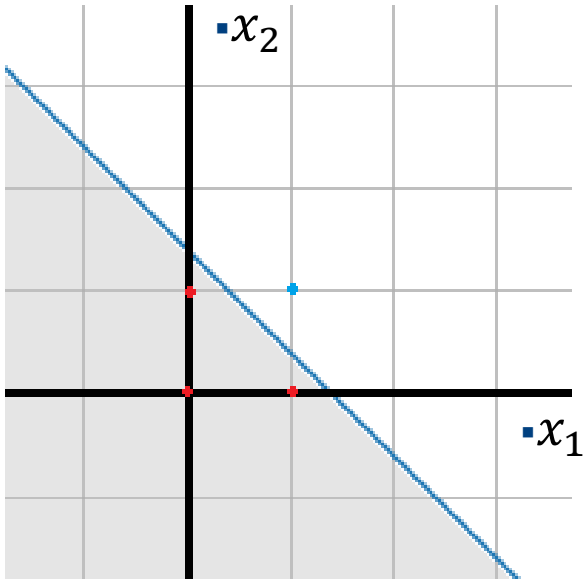
(a) AND 분류 문제

- **퍼셉트론으로 나타낸 AND Gate** $(x_1, x_2, \theta) = (0.5, 0.5, 0.7)$

- $$y = \begin{cases} 0 & (0.5x_1 + 0.5x_2 \leq 0.7) \\ 1 & (0.5x_1 + 0.5x_2 > 0.7) \end{cases}$$

▪AND Gate

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1



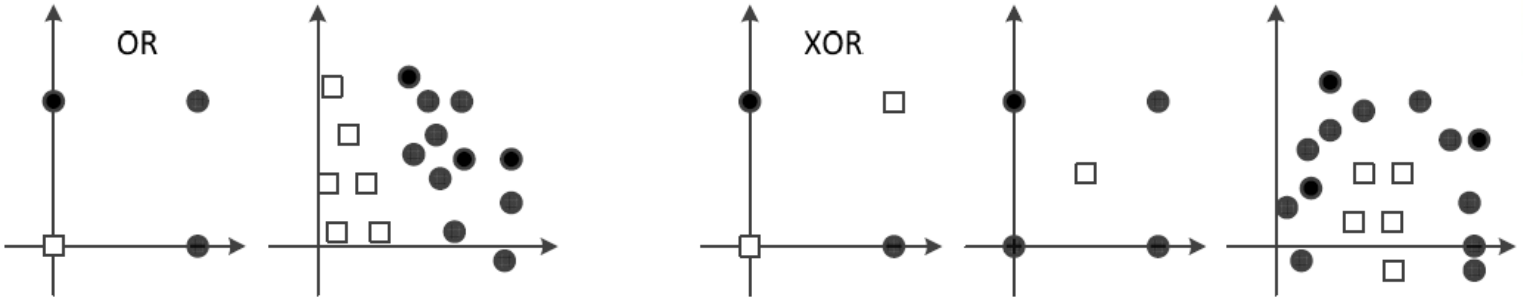
```
def AND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5,0.5])
    b = -0.7
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else :
        return 1

import numpy as np
print(AND(0,0))
print(AND(0,1))
print(AND(1,0))
print(AND(1,1))
```

- **단층 퍼셉트론의 한계**
- **지금까지 배운 내용을 단층 퍼셉트론이라고 한다**
- **단층 퍼셉트론은 단순한 선형 분류기 이기 때문에 XOR에 대한 선형 분류가 불가능하다**

XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0



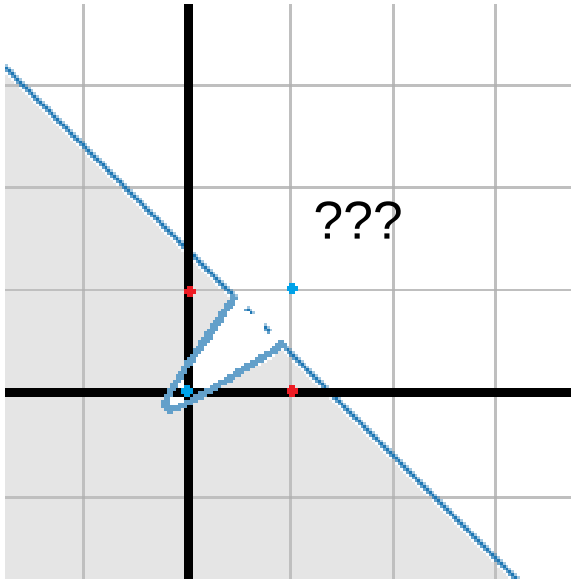
(a) 선형분리 가능 (b) 선형분리 불가능

그림 3-7 선형분리가 가능한 상황과 불가능한 상황

- **다층 퍼셉트론 (MLP)**

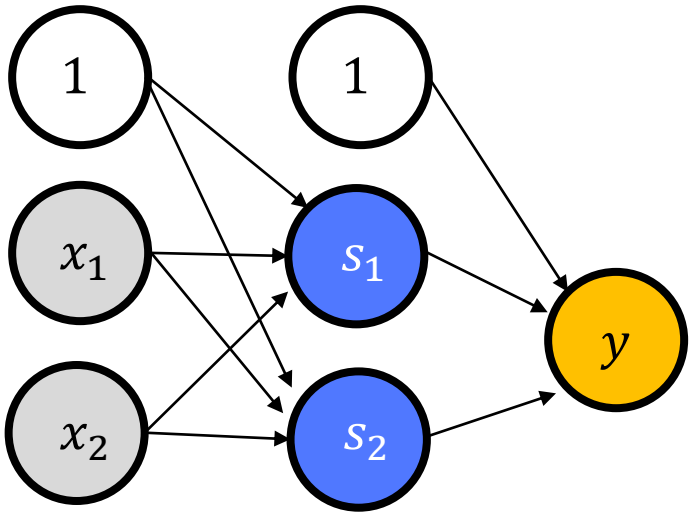
- 다층 퍼셉트론 (Multi-Layer Perceptron, MLP) 개요
- XOR과 같이 직선 하나로는 표현 불가능한 영역도 존재한다.

XOR		
x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

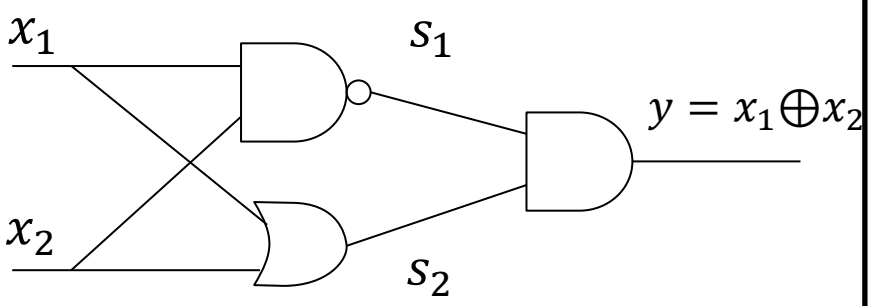


- 다층 퍼셉트론은 말 그대로, 퍼셉트론을 여러층 쌓는 것이다.
- i층의 퍼셉트론의 출력들이 i+1층의 퍼셉트론 입력이 된다.

- 입력데이터의 층을 입력층,
- 출력 신호가 나오는 퍼셉트론층을 출력층,
- 그 외 층들은 보이지 않는다 하여 은닉층이라 한다.



- 다층 퍼셉트론으로 XOR을 나타내보자.
- XOR은 NAND (Not-AND), OR, AND Gate로 나타낼 수 있다.



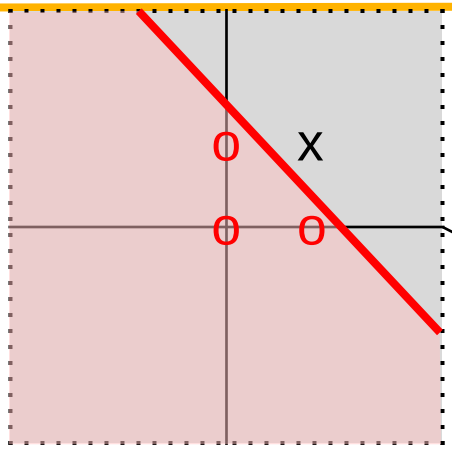

```
def XOR(x1, x2):
    s1 = NAND(x1, x2)
    s2 = OR(x1, x2)
    y = AND(s1, s2)
    return y
```

x1	x2	s1	s2	y
0	0	1	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	1	0

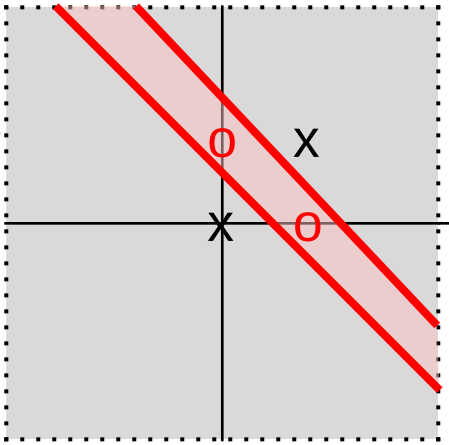
XOR

x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

s1=NAND(x1,x2)

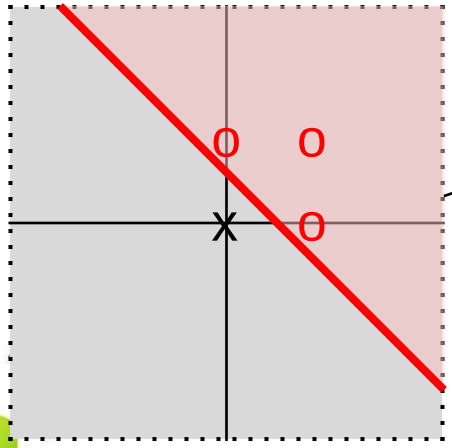


y=AND(s1,s2)



XOR(x1,x2)

s2=OR(x1,x2)



- **실습**