비선형 REGRESSION

비선형 REGRESSION

◆ LOGISTIC REGRESSION

- **입력 데이터와 타겟값의 관계**를 **구체적인 함수**로 나타내어 향후 예측 모델에 사용하는 회귀 분석과 목표 동일
- Linear Regression과 달리 **타켓값**이 **범주형 데이터**
- 입력 데이터에 대한 결과가 **특정 분류로 나뉘어 짐**
- 기본 규제 : **L2 적용**
- 비선형적 데이터에도 적용 가능 ➔ 커널 로지스틱 함수

비선형 REGRESSION

◆ LOGISTIC REGRESSION

- 종류
  - **이진 분류** : 2가지(0과 1) 분류
  - **다중 분류** : 3가지 이상으로 분류

각 **범주로 분류될 확률**로 **결과 반환**

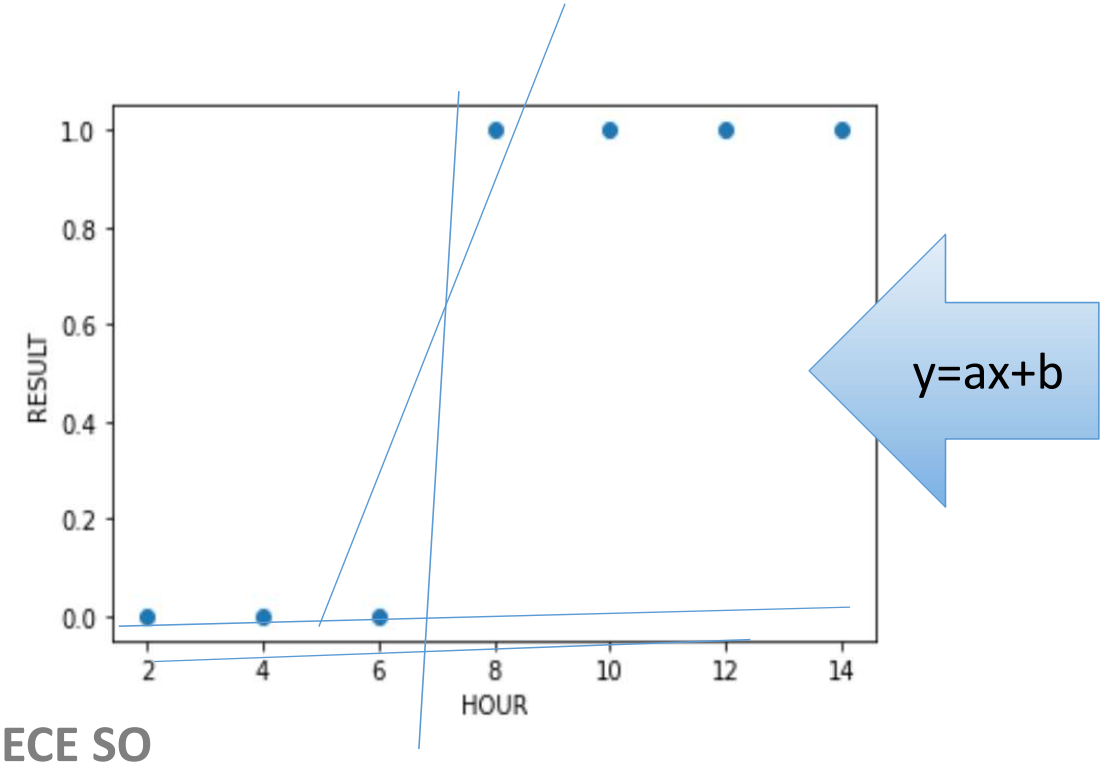
모든 범주의 **분류 확률 합 = 1**

비선형 REGRESSION

◆ LOGISTIC REGRESSION

➢ 예시 – 공부 시간과 성적

공부시간	2시간	4시간	6시간	8시간	10시간	12시간	14시간
성 적	40	48	54	68	72	89	91



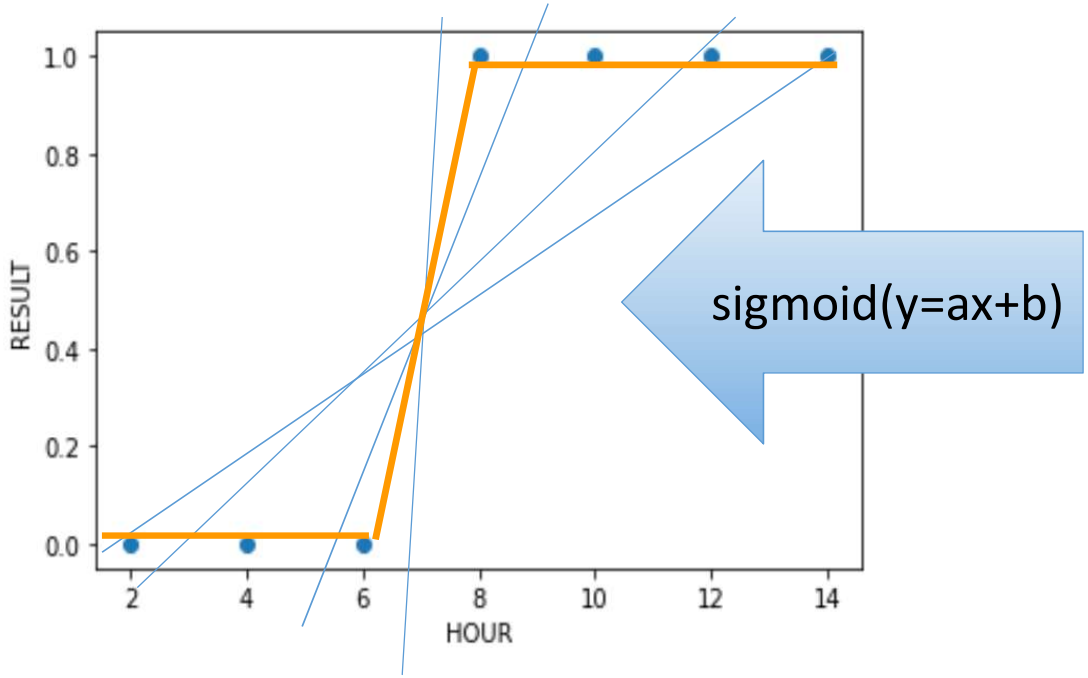
40=2a+b
48=4a+b
54=6a+b
68=8a+b
72=10a+b
89=12a+b
91=14a+b

비선형 REGRESSION

◆ LOGISTIC REGRESSION

➢ 예시 – 공부 시간과 합격 여부

공부시간	2시간	4시간	6시간	8시간	10시간	12시간	14시간
합격 여부	불합격	불합격	불합격	합격	합격	합격	합격



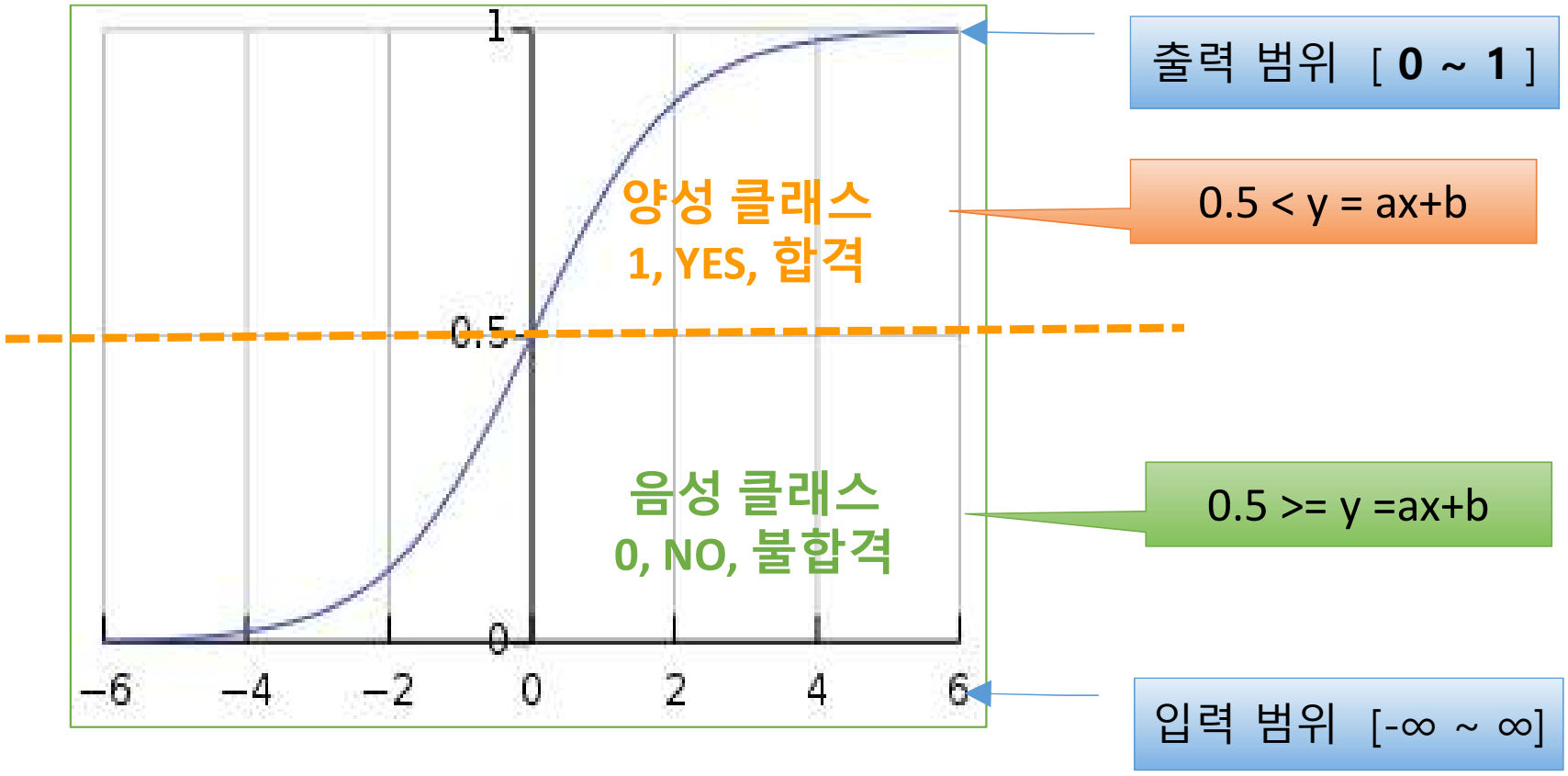
0=2a+b
0=4a+b
0=6a+b
1=8a+b
1=10a+b
1=12a+b
1=14a+b

비선형 REGRESSION

◆ LOGISTIC REGRESSION

➢ 시그모이드(Sigmoid) 함수

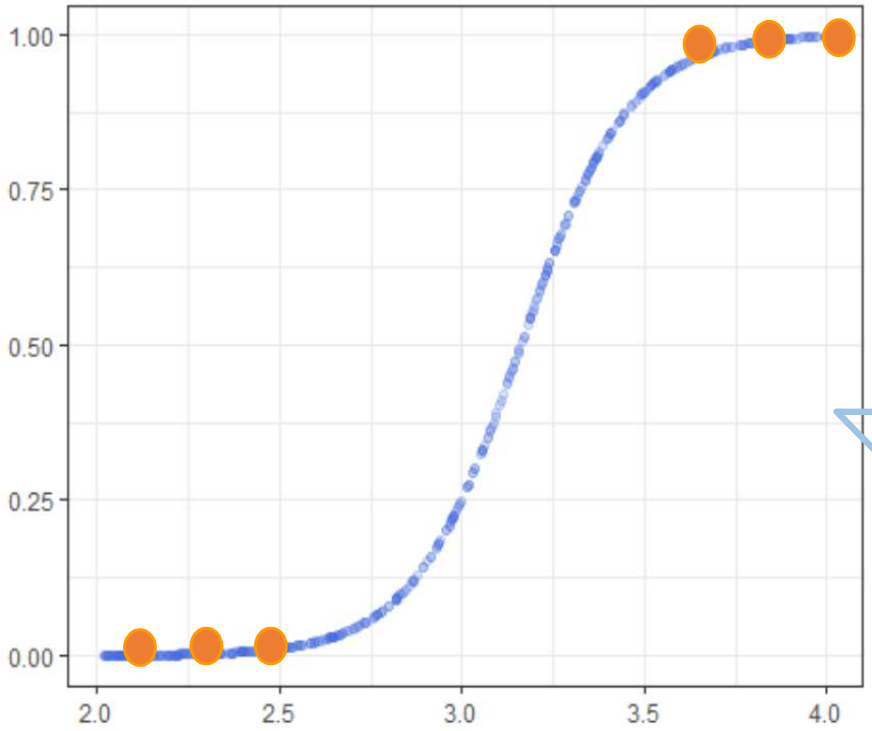
: S자형 곡선 또는 시그모이드 곡선을 갖는 수학 함수



비선형 REGRESSION

◆ LOGISTIC REGRESSION

➢ **S자 형태** 데이터 분포



많은 데이터 만족 S형 곡선

z=ax+b

Sigmoid(z)

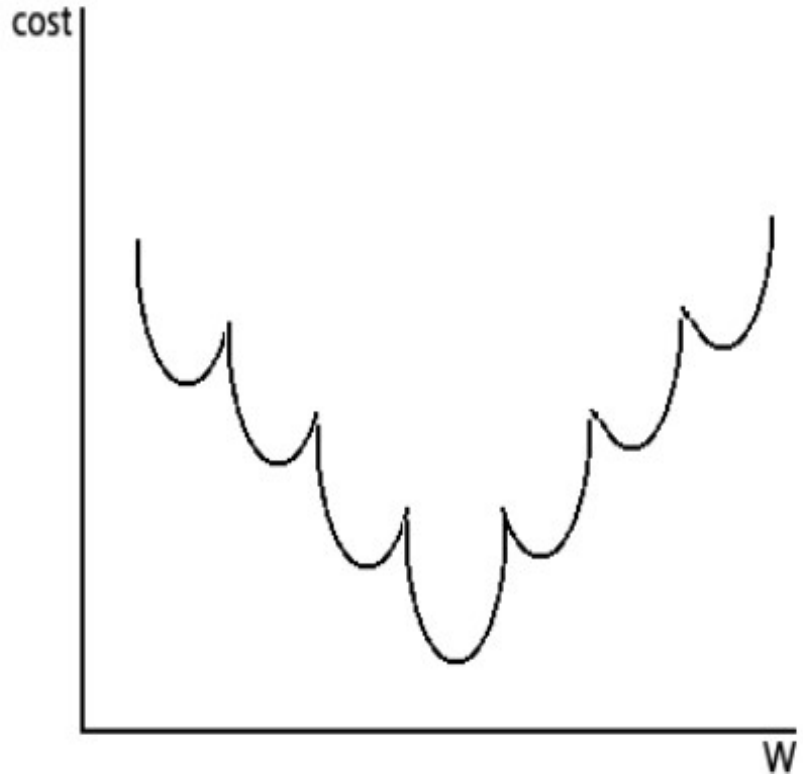
$$p = \frac{1}{1+e^{-z}}$$

p : 확률값

비선형 REGRESSION

◆ LOGISTIC REGRESSION

➢ 선형회귀 적용 손실함수



선형회귀 cost function을
로지스틱 회귀 가설 적용 하면
W의 값에 대한 cost function

비선형 REGRESSION

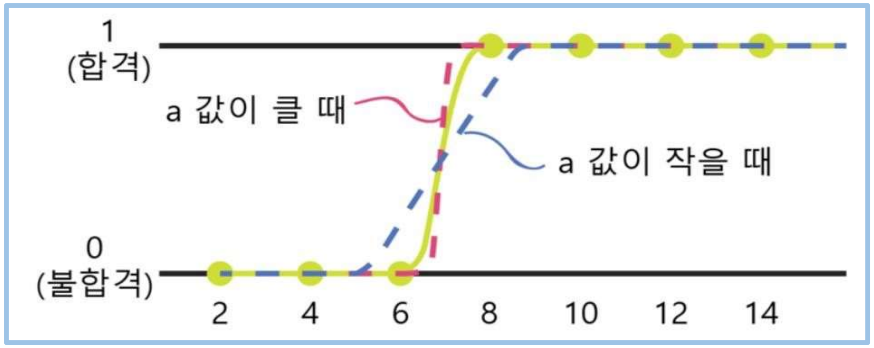
◆ LOGISTIC REGRESSION

➢ 선형회귀 적용 손실함수

비선형 REGRESSION

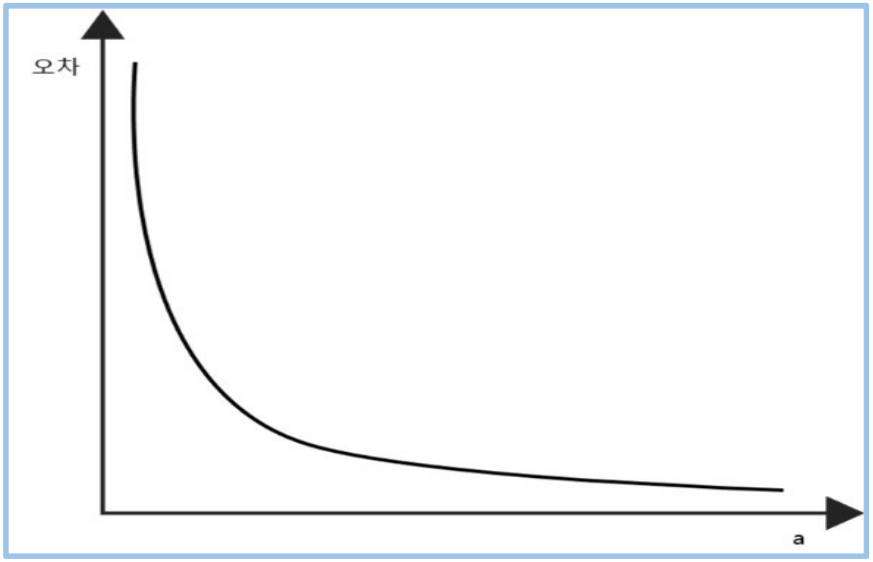
◆ LOGISTIC REGRESSION

➢ 실제값과 오차 관계



➔ a가 클수로 경사 커짐

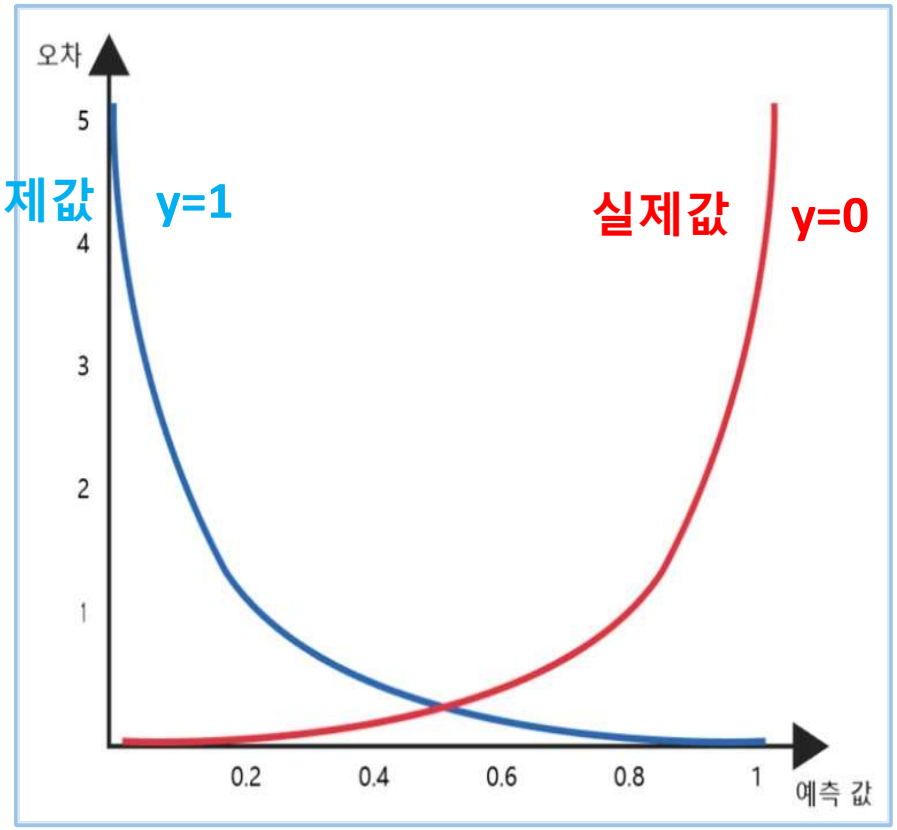
➔ a가 작을 수록 경사 작아짐
➔ 수평이 됨



비선형 REGRESSION

◆ LOGISTIC REGRESSION

➢ 실제값과 오차 관계



파랑색 선 ➔ **실제 값 1**

: -ylog(H(x))

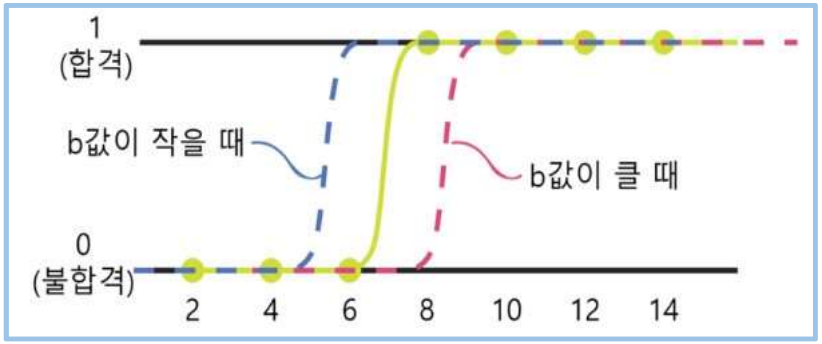
빨강색 선 ➔ **실제 값 0**

: (1-y)log(1-H(x))

비선형 REGRESSION

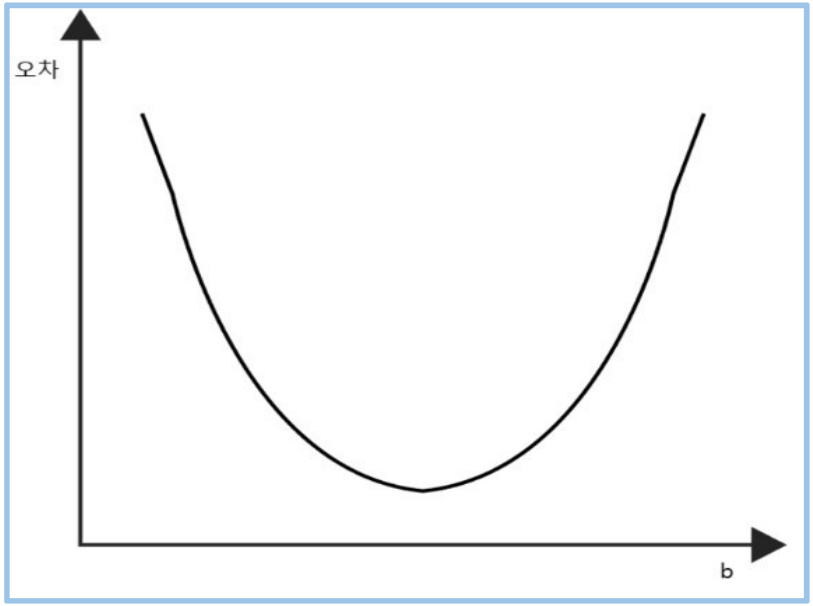
◆ LOGISTIC REGRESSION

➢ 절편과 오차 관계



➔ b가 클수로 오른쪽 이동

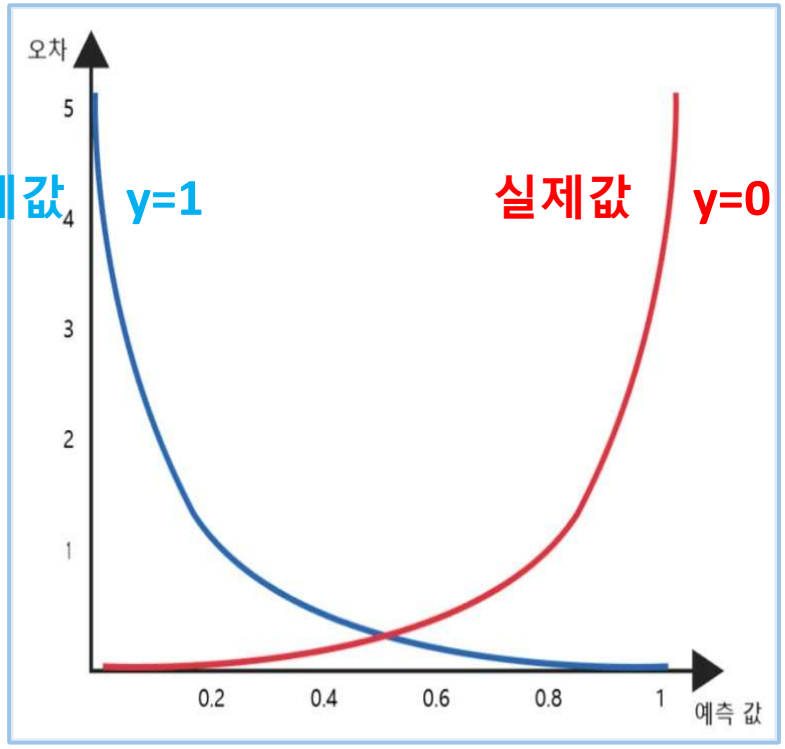
➔ b가 작을 수록 왼쪽 이동



비선형 REGRESSION

◆ LOGISTIC REGRESSION

➢ 손실함수 ➔ 로그 함수



실제 값이 1인 경우 Log

실제 값이 0인 경우 Log

$$cost(H(x),y) = -y\log(H(x)) - (1-y)\log(1-H(x))$$

손실(오차) =

-평균(ylog(H(x)) + (1-y) log(1-H(x))

비선형 REGRESSION

◆ LOGISTIC REGRESSION

➢ 시그모이드(Sigmoid) 함수

- **numpy 모듈**
  - numpy.**exp(1개)** ➔ **확률값**
- **scipy 모듈**
  - scipy.special.**expit(1개)** ➔ **확률값**

비선형 REGRESSION

◆ LOGISTIC REGRESSION

➢ 소프트맥스(Softmax) 함수

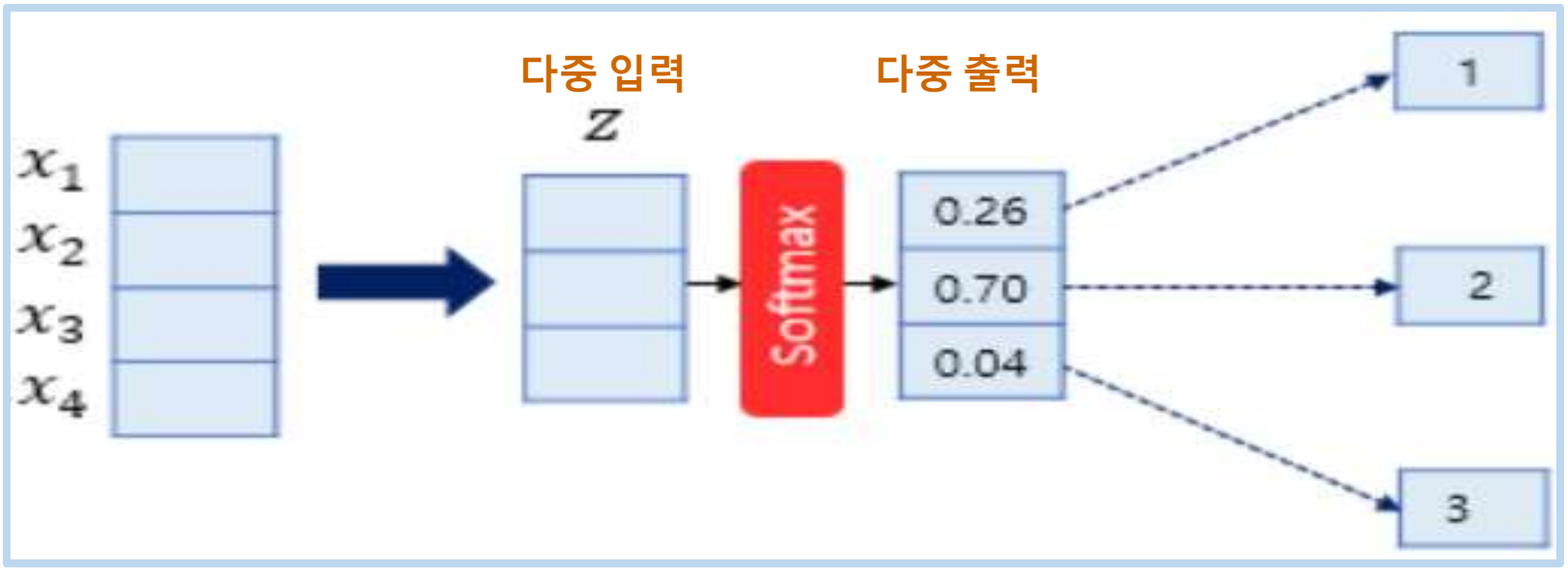
- **다중 분류 시**에 각 라벨(클래스/타겟)에 대한 **확률 추정 함수**
- 시그모드 함수에서 유래
  - 입력값 : 여러개
  - 출력값 : 0~1 사이의 실수 ➔ '확률'로 해석 가능
  - 출력 총합 : 1

$$y_k = \frac{e^{a_k}}{\sum_{i=1}^{n} e^{a_i}}$$

비선형 REGRESSION

◆ LOGISTIC REGRESSION

➢ 소프트맥스(Softmax) 함수



비선형 REGRESSION

◆ LOGISTIC REGRESSION

➢ 소프트맥스(Softmax) 함수

- **scipy 모듈**
  - ➢ scipy.special.**softmax(여러개) ➔ 확률값**

비선형 REGRESSION

◆ LOGISTIC REGRESSION

➢ Scikit-Learn Lib 사용 - **학습객체**

```
from sklearn.linear_model import LogisticRegression
( penalty = L2          규제 사용 기준 지정 (L2 = MSE+가중치 제곱합)
  dual                  이중 또는 초기 공식
  tol                   정밀도
  C=1.0                 규제 강도 ( Cost Funciton ) , 큰값(약) - 작은값(강)
  fit_intercept=True    절편 존재 여부 설정
  intercept_scaling=1   정규화 효과 정도
  class_weight =1       클래스 가중치
  random_state          난수 seed 설정
```

비선형 REGRESSION

◆ LOGISTIC REGRESSION

➢ Scikit-Learn Lib 사용 - **학습객체**

```
from sklearn.linear_model import LogisticRegression
(
solver='lbfgs'      # 최적화 문제 사용 알고리즘
max_iter            # 학습 횟수
multi_class         # 다중 분류 시 (ovr, multinomial, auto)로 설정
verbose             # 동작 과정에 대한 출력 메시지
warm_start          # 이전 모델을 초기화로 적합하게 사용할 것인지 여부
n_jobs              # 병렬 처리 할 때 사용되는 CPU 코어 수
l1_ratio            # L1 규제의 비율(Elastic-Net에만 사용)  )
```

ML OPTIMIZATION & MODEL

ML OPTIMIZATION

◆ 최적화

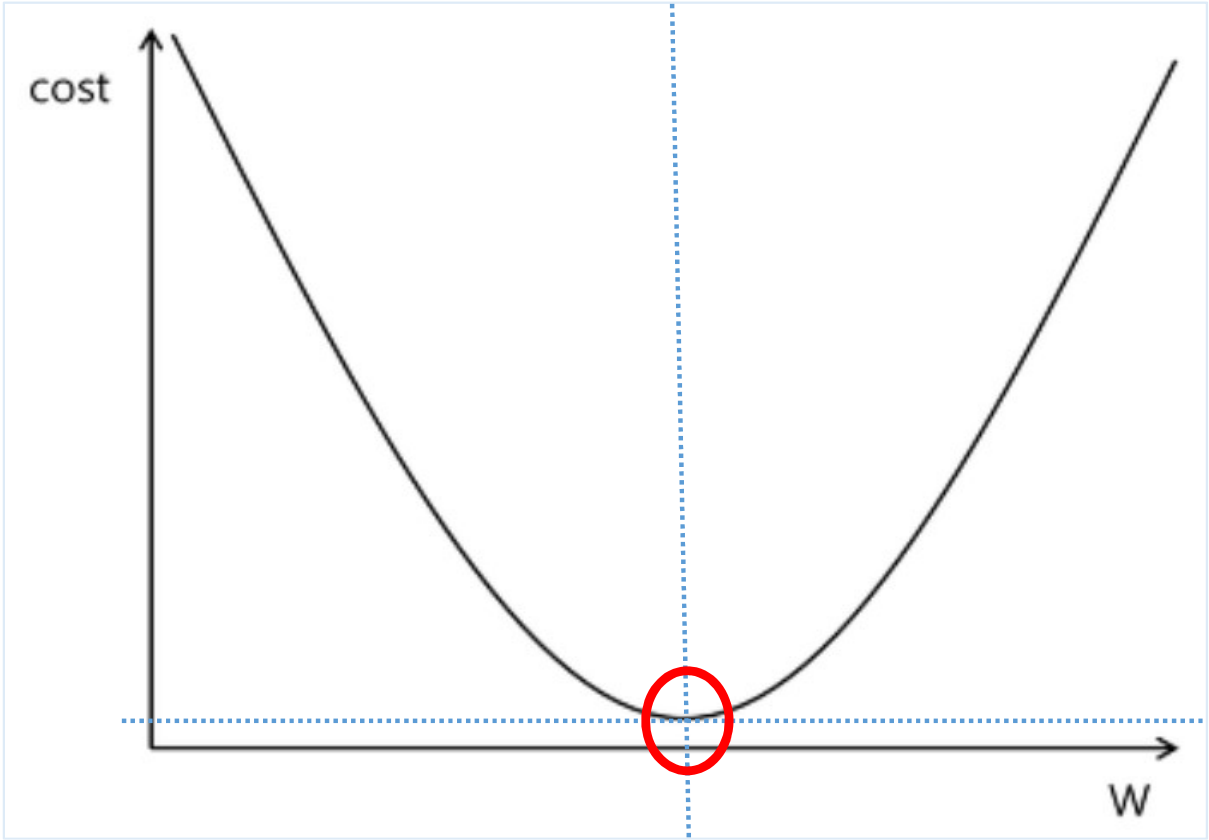
- **목적함수(Objective Function)**를 최대한, 혹은 최소화하는 파라미터 조합을 찾는 과정
  - 경사하강법 (Gradient Decent) : 비용(cost)/손실함수(loss), 오차(error)
    경사상승법 (Gradient Ascent) : 이익(profit), 점수(score)

- 모델 평가 시 **손실/비용함수 값이 최소가 될때**가 **최적의 모델**
- **손실/비용함수 값이 최소가 되는 모델 파라미터**를 찾는 것

ML OPTIMIZATION

◆ 최적화

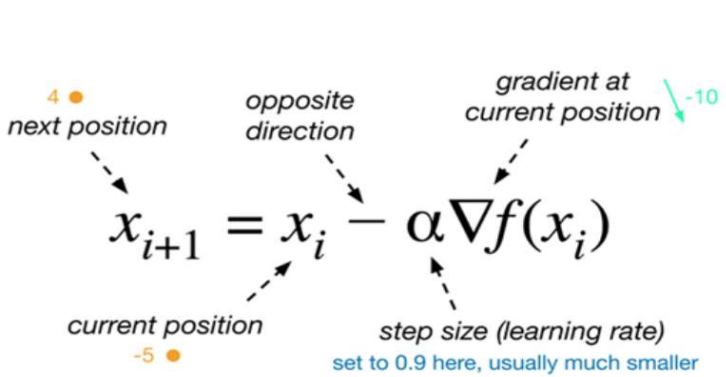
- ML/DL 오차와 기울기 관계

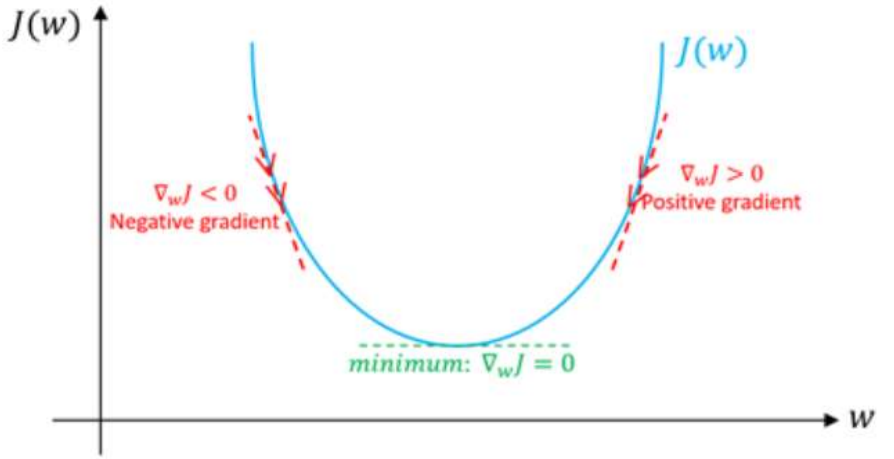


ML OPTIMIZATION

◆ 경사하강법 (Gradient Descent)

- 등산 후 하산 해야 하는 상황
- 현재 내 위치로부터 **경사가 가장 가파른 방향으로 이동** 하는 것
- **최소점을 만족하는 파라미터 값을 찾는 것**
- **일차 미분 이용**한 최적화 기법
- 미분 통해 gradient를 구한 후 **반대 방향(음수)으로 이동**

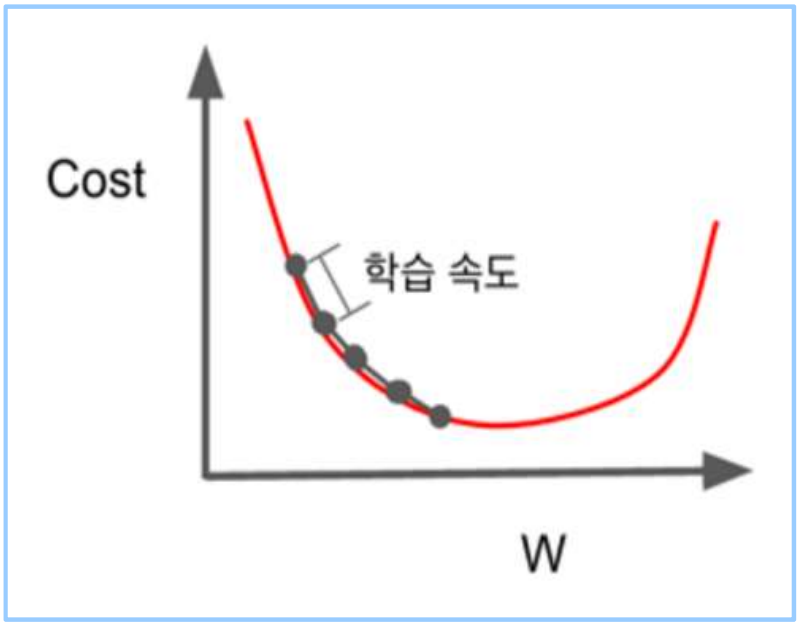




ML OPTIMIZATION

◆ 경사하강법 (Gradient Descent)

- 학습속도(Learning Rate)
  - 경사 하강법에서 **학습 단계별로 움직이는 학습 속도 정의**
  - W값 조정해 가면서 Cost 값이 최소가 되는 값을 찾기 위한 것
  - W값이 다음 W값이 되는 속도



오버슈팅(Over shooting)
학습속도가 큰 경우 발생
최소값으로 내려가지 않고 반대편으로 넘어가
무한대

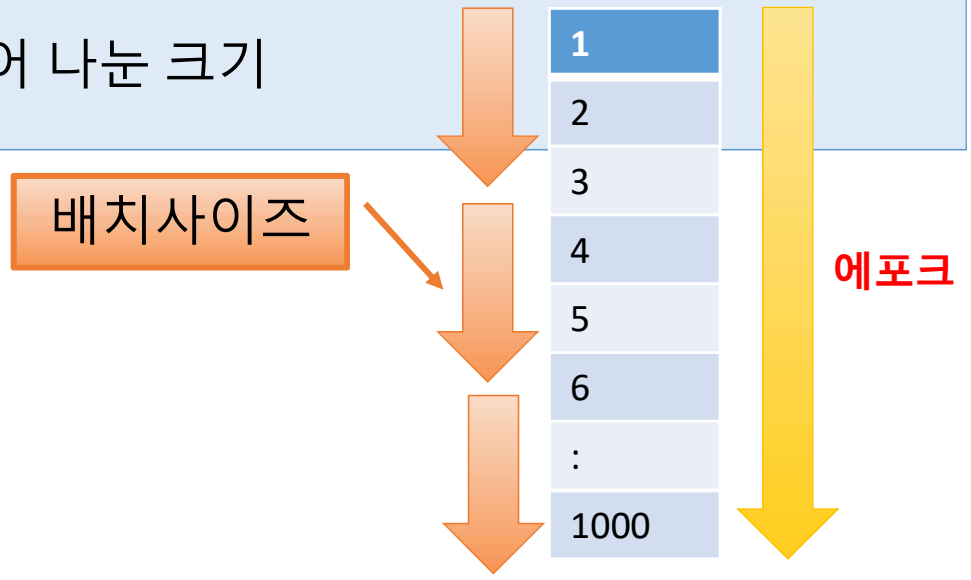
스몰 러닝 레이트(Small Learning Rate)
학습속도가 매우 작은 경우 발생
예) 0.0001
최소값 가기전에 학습 종료

ML OPTIMIZATION

◆ 경사하강법 (Gradient Descent)

- **에포크(Epoch) <- scikit-learn에서 max_iter 파라미터**
  - 전체 샘플을 모두 사용하는 한 번 반복 의미

- **배치 사이즈(Batch-size)**
  - 전체 샘플을 쪼개어 나눈 크기



ML OPTIMIZATION

◆ 경사하강법 (Gradient Descent)

- **배치 학습 / 오프라인 학습**
  - 모든 데이터를 **한꺼번에 학습**
  - **시간과 자원이 많이 소모**되어 오프라인에서 수행
  - 새로운 데이터 학습 위해 전체 데이터를 처음부터 다시 학습

- **점진적 학습 / 온라인 학습**
  - 데이터를 순차적으로 **한 개씩 또는 작은 묶음으로 학습 진행**
  - 연속적으로 데이터를 받고 변화에 빠르게 적용 가능
  - 기존 모델에 **새로운 데이터 추가 학습 진행**

ML OPTIMIZATION

◆ 경사하강법 (Gradient Descent)

BGD (Batch Gradient Descent) 배치경사하강법	전체 학습 데이터 Gradient 계산 많은 시간 및 계산량 소요되지만 정확도 높음
SGD (Stochastic Gradien Descent) 확률적경사하강법	한 번에 한 개 데이터만 랜덤 샘플링 통해 추출 후 Gradient 계산 BGD에 비해 다소 **부정확**할 수 있지만 속도가 훨씬 빠름
MSGD (mini-batch Gradient Descent) 미니배치경사하강법	BGD와 SGD를 절충하여 일부 학습 데이터(mini-batch)를 Gradient 계산 **SGD에 비해 정확도 높고 BGD에 비해 효율적**
Momentum	기울기 방향으로 힘을 받아 물체가 가속되어 공이 구르는 듯한 움직임 이전 값과 비교 후 같은 방향으로 업데이트 진행 -> 경사하강 + 관성, **SGD보다 빠름**
AdaGrad (Adaptive Gradient)	변수들을 update할 때 각각의 변수마다 step size를 다르게 설정해서 이동하는 방식, **기존 기울기 값을 제곱한 값을 더하여 학습률을 조정**
RMSProp / AdaDelta	AdaGrad의 갱신 속도 느려지는 단점을 해결한 방법 먼 과거의 기울기는 조금 반영하고 최신의 기울기를 많이 반영
Adam (Adaptive Moment Estimation)	과거 미분값 계속 가중평균 내면서 효율적 업데이트 AdaGrd + Momentum 방식 결합

ML OPTIMIZATION

◆ 경사하강법 (Gradient Descent)