GNU

Lab Seminar: 2024. 01.12

Supervised Machine Learning

(Machine Learning & Regression)

IDEALAB
Improving lives through learning

Su-Gyeong Ban
School of Computer Science/Department of AI Convergence Engineering
Gyeongsang National University (GNU)

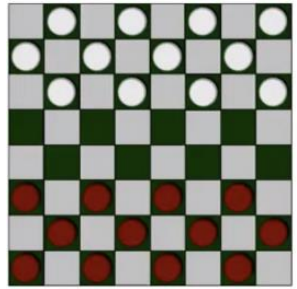
GNU 경상국립대학교 Gyeongsang National University

Contents

- Machine Learning
- Learning Algorithm types
- Linear Regression Model
- Cost Function
- Gradient descent Algorithm

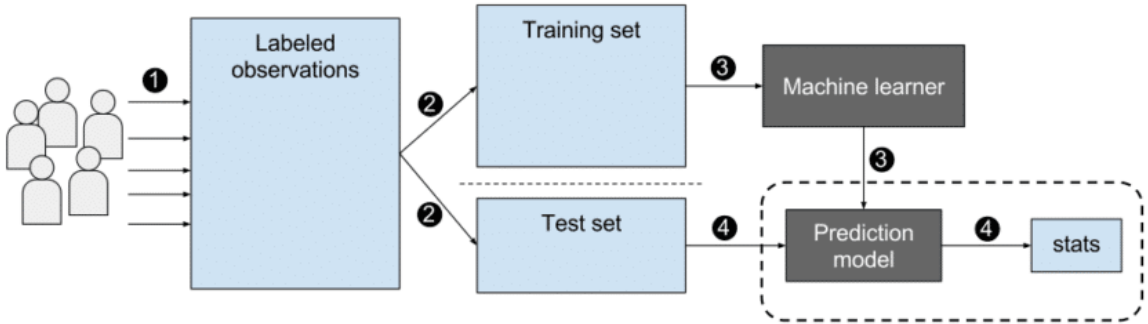
Machine Learning

- **What is Machine Learning?**
  - **인공지능 하위 분야**
  - 컴퓨터가 명시적인 프로그램이 없이도 스스로 학습할 수 있는 능력을 연구하는 학문 - Arthur Samuel
    - 체커 게임 프로그램
      - 컴퓨터는 수 만 번 게임을 반복할 끈기가 있음
      - 수많은 체커 게임 경험을 축적하여 인간을 이길 수 있는 체커 플레이어가 됨



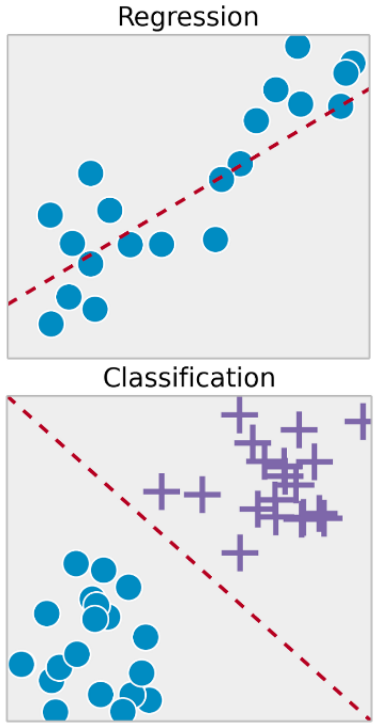
Learning Algorithm types (1/4)

- **Supervised Learning**
  - 인공지능이 **정답을 가지고 있는** 데이터셋을 통해 학습
    - ex) 고양이와 개 사진을 구분하는 프로그램 : 고양이와 개의 사진 수만장을 학습하고 실제로 사진을 보면서 구분



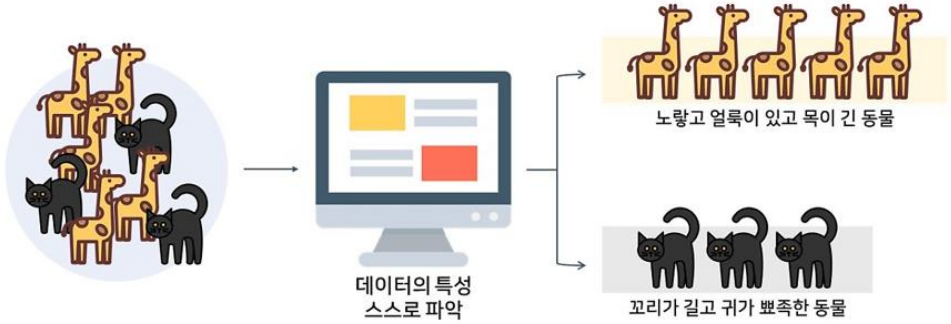
Learning Algorithm types (2/4)

- **Supervised Learning**
  - **Regression(회귀)**
    - 가능한 무한한 숫자로 숫자를 예측하는 것
      - ex) 집 가격 예측
  - **Classification(분류)**
    - 출력 범주를 예측하는 것
      - ex) 유방암 진단



Learning Algorithm types (3/4)

- **Unsupervised Learning**
  - 인공지능이 **정답이 없는** 데이터셋을 통해 **스스로 학습**
  - 입력 값만 있고 출력 값은 제공되지 않음



Learning Algorithm types (4/4)

- **Unsupervised Learning**
  - **Clustering**
    - 레이블이 지정되지 않은 데이터를 여러 클러스터에 배치
      - ex) 구글 뉴스

  - **Dimension Reduction (차원축소)**
    - 큰 데이터 집합을 가져와서 정보를 최대한 적게 손실하면서 더 작은 데이터 집합으로 압축
  - **Anomaly Detection (이상탐지)**
    - 비정상적인 이벤트를 탐지하는 데 사용

Linear Regression Model (1/2)

- **What is Linear Regression?**
  - 알려진 관련 데이터 값을 사용하여 알 수 없는 데이터 값을 예측하는 기법
  - 훈련 세트를 학습 알고리즘에 공급 -> 지도학습 알고리즘이 함수 생성 -> 예측값 출력하고 추정
  - 함수가 모델을 의미함

$$f_{w,b}(x) = wx + b$$
$$f(x) = wx + b$$

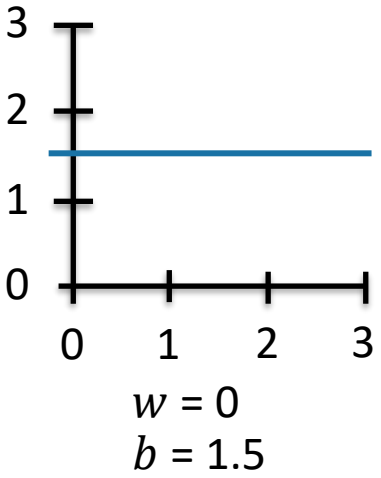
Code of Linear Regression

```
def compute_model_output(x, w, b):
    m = x.shape[0]
    f_wb = np.zeros(m)
    for i in range(m):
        f_wb[i] = w * x[i] + b

    return f_wb
```

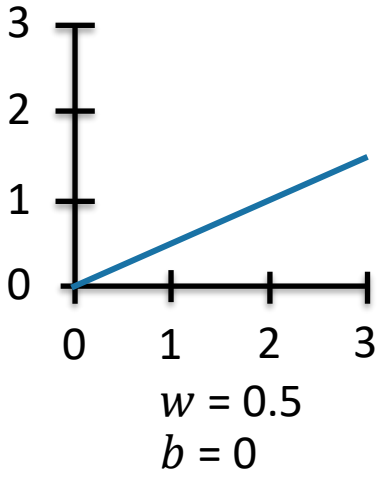
Linear Regression Model (2/2)

- **Simple Linear Regression**
  - 입력 변수가 하나인 선형 모델

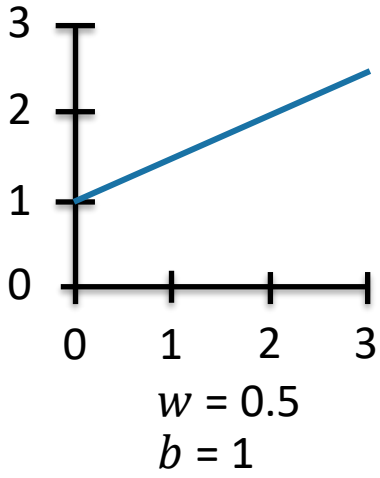
$$f(x) = wx + b$$



w = 0
b = 1.5



w = 0.5
b = 0



w = 0.5
b = 1

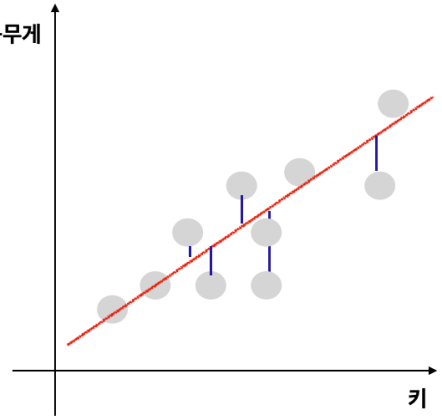
Cost Function (1/4)

- **What is Cost Function?**
  - 직선이 훈련 데이터에 얼마나 잘 맞는지 측정하는 방법을 위해 사용하는 함수
  - **실제값과 모델이 예측한 값 간의 차이** 즉 Error를 의미함
    - 비용함수 값이 클수록 모델의 정확도가 낮음

Cost function : Squared error cost function

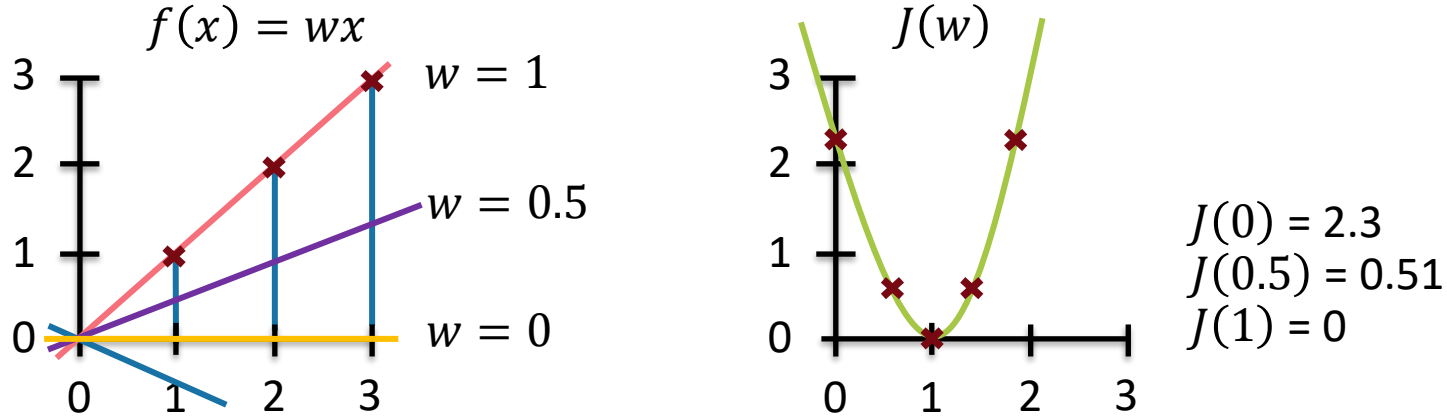
$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

m = number of training examples



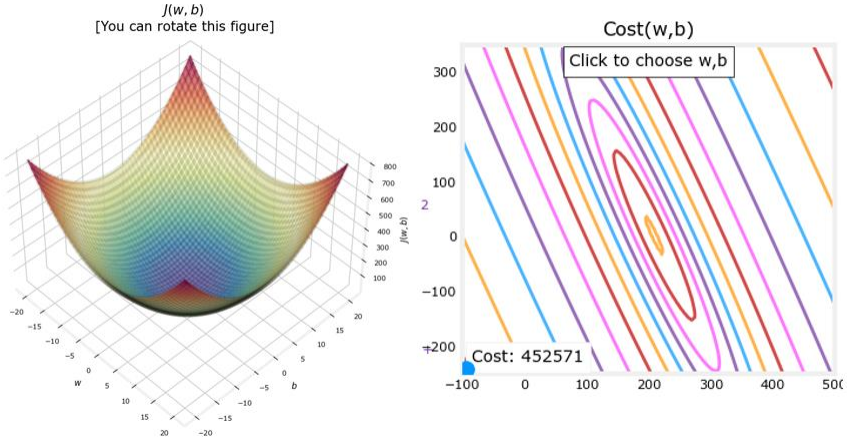
Cost Function (2/4)

- **Minimize cost function with parameters (parameter = 1)**
  - b=0 으로하여 매개변수를 w 하나로 설정
  - 단순화된 모델의 목표 : J를 최소화하는 w값을 찾는 것



Cost Function (3/4)

- **Minimize cost function with parameters (parameter = 2)**
  - 매개변수가 2개일 때 3차원 그래프가 나타남
  - 3차원 그래프를 등고선도를 이용해 2차원 그래프로 표현
  - **Gradient descent Algorithm 사용**



Cost Function (4/4)

- **Code of Cost Function**
  - 오차 제곱 식을 훈련 예제 수 만큼 반복하여 cost_sum 에 합산

Cost function : Squared error cost function

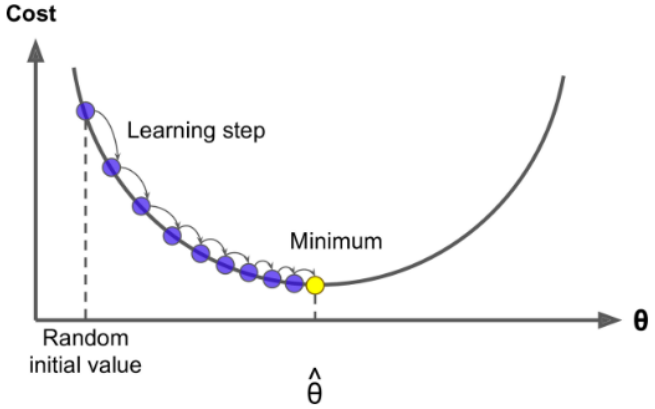
$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}\left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

m = number of training examples

```python
def compute_cost(x, y, w, b):
    # number of training examples
    m = x.shape[0]

    cost_sum = 0
    for i in range(m):
        f_wb = w * x[i] + b
        cost = (f_wb - y[i]) ** 2
        cost_sum = cost_sum + cost
    total_cost = (1 / (2 * m)) * cost_sum

    return total_cost
```

Gradient descent Algorithm (1/6)

- **What is Gradient descent Algorithm?**
  - 1차 근삿값 발견용 최적화 알고리즘
  - 함수의 기울기(Gradient)를 구하고 해당 경사의 반대방향으로 계속 이동시켜 **극값에 이를 때까지 반복시키는 것**
  - 두 매개변수를 동시에 업데이트 해야 함



Gradient descent Algorithm (2/6)

- **Implementation of Gradient descent (1/2)**
  - w, b의 도함수(기울기)를 계산하여 식에 다시 삽입
    - 도함수 즉, **기울기가 0에 가까워질수록 최소**
    - α = Learning rate

      - Learning rate : 모델의 파라미터를 업데이트할 때, 얼마나 많은 단계를 거쳐야 하는지 결정

Gradient descent algorithm

repeat until convergence {

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b) \rightarrow \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})x^{(i)}$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b) \rightarrow \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})$$

}

repeat until convergence {

$$w = w - \alpha \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})\ x^{(i)}$$

$$b = b - \alpha \frac{1}{m}\sum_{i=1}^{m}(f_{w,b}(x^{(i)}) - y^{(i)})$$

}

Gradient descent Algorithm (3/6)

- **Implementation of Gradient descent (2/2)**
  - w, b의 도함수(기울기) 계산 식 code

repeat until convergence {

$$w = w - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)}) \; x^{(i)}$$

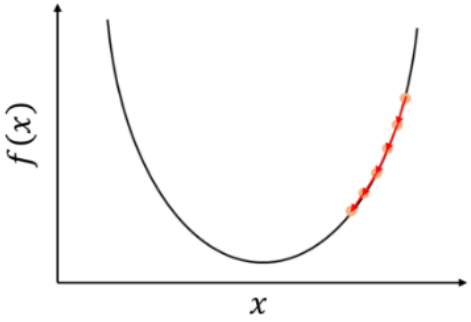
$$b = b - \alpha \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})$$

}

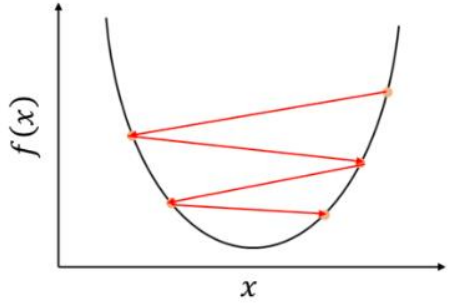
```python
def compute_gradient(x, y, w, b):

    # Number of training examples
    m = x.shape[0]
    dj_dw = 0
    dj_db = 0

    for i in range(m):
        f_wb = w * x[i] + b
        dj_dw_i = (f_wb - y[i]) * x[i]
        dj_db_i = f_wb - y[i]
        dj_db += dj_db_i
        dj_dw += dj_dw_i
    dj_dw = dj_dw / m
    dj_db = dj_db / m

    return dj_dw, dj_db
```

Gradient descent Algorithm (4/6)

- **Learning rate of Gradient descent**
  - 학습률이 너무 **낮을** 경우
    - J는 줄어들지만 속도 매우 느림
  - 학습률이 너무 **높을** 경우
    - 큰 교차점이 local minimum에 도달하지 못하고 갈라질 수 있음



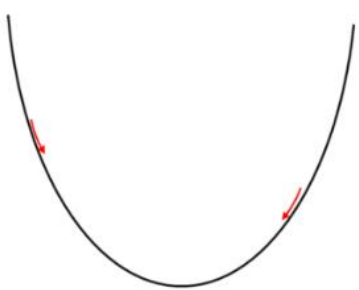
α가 너무 작은 경우 (수렴이 늦음)

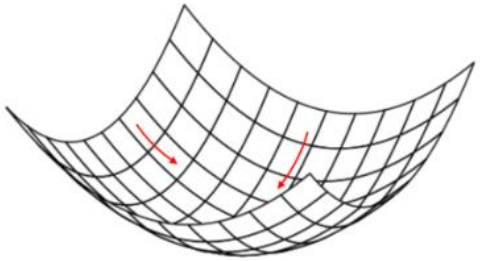


α가 너무 큰 경우 (진동)

Gradient descent Algorithm (5/6)

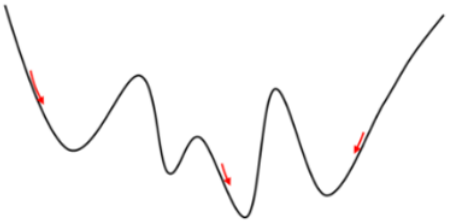
- **Convex Function of Gradient descent**
  - 경사하강법 구현에 **유용한 특성**을 가짐
    - 학습률을 적절하게 선택하면 항상 **global minimum**으로 수렴
    - 선형 회귀 모델은 **볼록함수**를 가짐

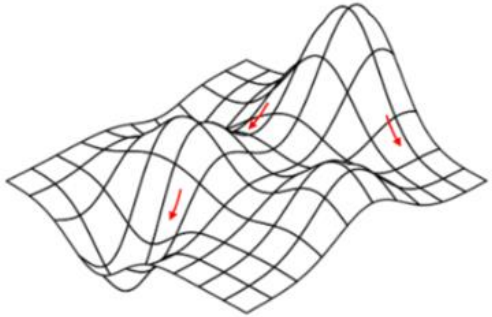


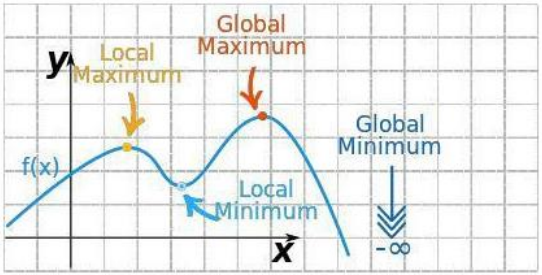


Gradient descent Algorithm (6/6)

- **Non - Convex Function of Gradient descent**
  - 비볼록함수는 시작지점에 따라 다른 global minimum에 도달
    - **초기값의 위치에 따라 전역최소값이 될 수 있고 지역최소값이 될 수 있음**
    - 단순 경사하강법의 한계
      - **Deep Learning**을 이용해 심화 경사하강법 실행







경상국립대학교

Gyeongsang National University

Improving lives through learning

IDEALAB