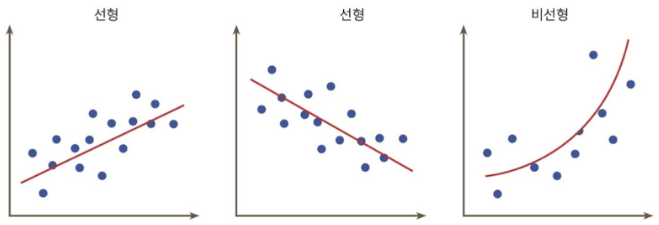
# 과제 1106

2315028 김성현

▼ 선형회귀



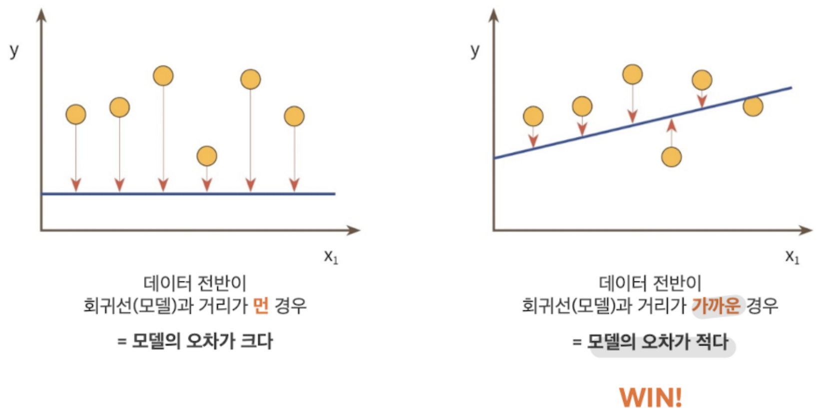

- 회귀(regression) : 특정 데이터를 가장 잘 설명(특징과 목푯값의 관련성을 규명)하는 방법
  y = f(x)에서 y, x가 실수일때 함수 f(x)를 예측하는 것
- ▼ 선형회귀(linear regression) : 입력데이터를 가장 잘 설명하는 기울기와 절편값을 찾는 문제
  - 단순 선형회귀 : 특징 x가 하나인 선형회귀. 회귀 선을 찾음

    ```
    f(x) = Wx + b (W:가중치, b:편향)
    ```

  - 다중 선형회귀 : 특징이 여럿인 복잡한 선형회귀. 회귀 공간을 찾음

    ```
    f(x) = w1 ×1 + w2 ×2 + w3 ×3 ... + wd xd + b
    ```

  - 회귀 선 : 데이터의 가장 적합한 직선으로, 데이터와 회귀 선 간의 간격을 통해 찾음




데이터와 회귀선(모델)의 거리를 ‘오차’라고 함

오차가 가장 적은 것이 좋음! ( 오차들의 합을 구해 비교함 )

▼ 손실함수(loss function) : 회귀선을 찾기위한 함수 (=비용함수, cost function)

예측값 실제값

손실함수 최소화하는 W와 b는?

$$Loss(W,b) = \frac{1}{n}\sum_{i=1}^{n}(f(x_i)-y_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(\underbrace{(Wx_i+b)}_{f(x)}-y_i)^2 \quad \Rightarrow \quad \underset{W,b}{argmin}\ Loss(W,b)$$

데이터와 회귀선의 간격을 제곱해 합한 값 ⇒ 오차제곱합

손실함수는 모델의 성능을 평가하고, 개선방향을 제시함

---

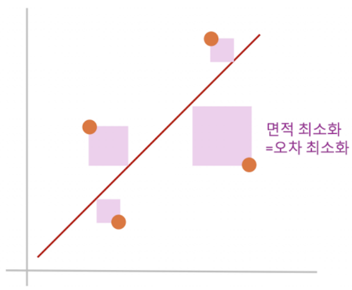
- 손실함수 최소화 기법

▼ 최소제곱법(OLS, ordinary least squares)

: 손실함수를 수학적으로 풀어 최적의 해를 직접 계산하는 방식

$$w = \frac{\sum_{i=1}^{n}(x_i - \overset{\text{평균}}{\overline{x}})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2} \qquad b = \overline{y} - w\overline{x}$$

( x - x평균 ) ( y - y평균) / ( x - x평균)^2



```
Function LinearRegression(X, Y):
    // 단계 1: X와 Y의 평균 계산
    mean_X = mean(X)
    mean_Y = mean(Y)

    // 단계 2: (X - mean_X)의 제곱합을 계산
    sum_X_squared = sum((X - mean_X) ** 2)

    // 단계 3: (X - mean_X)와 (Y - mean_Y)의 곱의 합을 계산
    sum_XY_product = sum((X - mean_X) * (Y - mean_Y))

    // 단계 4: 회귀선의 기울기(weight)와 절편(bias) 계산
    weight = sum_XY_product / sum_X_squared
    bias = mean_Y - (weight * mean_X)

    // 최종적으로 계산된 weight와 bias 반환
    return weight, bias
```

데이터가 적고 차원이 낮을때, 정확하고 빠른 계산 가능하지만...

데이터가 많고 차원이 높을때, 계산 복잡도가 올라가 비효율적임

▼ 경사하강법(Gradient decent)

: 손실함수의 기울기를 따라 조금씩 내려가는(가장 경사가 급한 아래방향) 반복적 최적화 기법

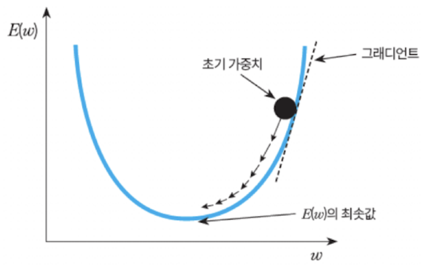
점진적으로 값을 찾아나감. 정확한 해를 찾는 대신 근사적인 해를 찾는 과정

→ 수렴속도와 학습률 조정이 중요

$$W \leftarrow W - \alpha \frac{\partial Loss(W, b)}{\partial W}$$

매개변수 w와 b를 찾음 .

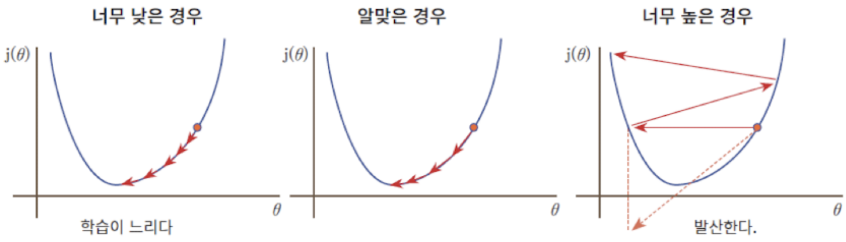
편미분 - 손실을 줄이기위한 방향을 찾음 (양수 - 감소방향, 음수 - 증가방향)




```
Function LinearRegression(X, Y, learning_rate, epochs):
    // 단계 1: 기울기(W)와 절편(b)을 0으로 초기화
    weight = 0
    bias = 0
    n = len(X)  // n = 데이터 개수

    // 단계 2: 지정된 에폭 수만큼 경사하강법 수행
    for i = 1 to epochs
        // 단계 2.1: 현재 W와 b에 기반한 예측값 계산
        Y_pred = weight * X + bias

        // 단계 2.2: W와 b에 대한 각각의 경사(gradient) 계산
        gradient_weight = (2 / n) * sum(X * (Y_pred - Y))
        gradient_bias = (2 / n) * sum(Y_pred - Y)

        // 단계 2.3: 경사와 학습률을 사용해 W와 b 업데이트
        weight = weight - learning_rate * gradient_weight
        bias = bias - learning_rate * gradient_bias

    // 단계 3: 최적화된 W와 b 반환
    return weight, bias
```

▼ 학습률 a(알파) (learning rate) : 한번에 매개변수를 얼마나 변경하는지에 관한 비율

초매개변수로(hyperparameter)임




너무 크면 최적해를 지나칠 수 있음. 너무 작으면 오랜시간이 걸리며 국소최적해에 갇힐 수 있음

▼ 학습 과정

편미분 : 함수가 여러개의 변수를 지닐때, 특정변수만 변화시키고 나머지변수는 상수로 고정한 상태에서 미분하는 방법

$$Loss(W,b) = \frac{1}{n}\sum_{i=1}^{n}((Wx_i + b) - y_i)^2$$

$$\frac{\partial Loss(W,b)}{\partial W} = \frac{1}{n}\sum_{i=1}^{n}2((Wx_i + b) - y_i)(x_i) = \underline{\frac{2}{n}\sum_{i=1}^{n}x_i((Wx_i + b) - y_i)}$$

$$\frac{\partial Loss(W,b)}{\partial b} = \underline{\frac{2}{n}\sum_{i=1}^{n}((Wx_i + b) - y_i)}$$

$$W = W - \overset{\alpha}{0.01} * \frac{\partial Loss}{\partial W}$$

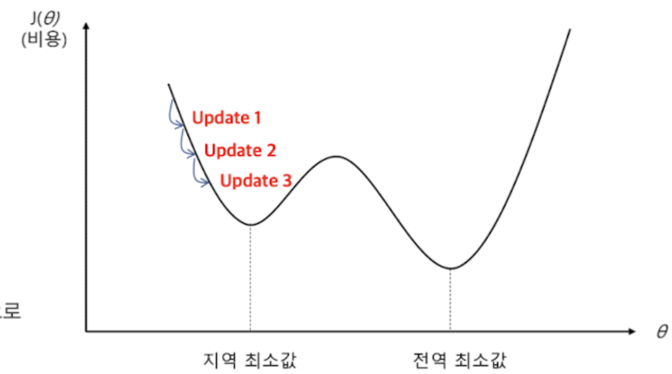
$$b = b - 0.01 * \frac{\partial Loss}{\partial b}$$

ative)은 함수가 여러 개의 변수를 가질
나머지 변수는 상수로 고정한 상태에서
들 중 특정 변수를 제외하고 나머지는

▼ 에폭 (epoch) : 모델이 학습을 위해 **전체 데이터셋**을 반복해서 학습하는 횟수

초매개변수로(hyperparameter)임

로




에폭이 높으면, 여러번 학습해 정교한 학습이 가능하지만 학습시간이 길어지고 과적합 발생

에폭이 적으면, 과적합 위험은 낮지만 충분히 학습하지못해 성능이 낮아지는 과소적합 발생

---

배치경사하강법 - 한 에폭을 기준으로 매개변수를 업데이트

다른 경사하강법 - 배치크기를 조절해 각 배치에 따라 매개변수를 업데이트하기도 함

▼ 평가 및 성능

- 평가

▼ MSE(mean squared error) : 해당 모델의 예측 정확도(예측과 실제값의 차이)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2$$

실제값에서 예측값을 뺀 값의 제곱 평균

특징 )

제곱값이기에 예측값과 단위가 다르고, 1미만의 오차는 더 작게 1이상의 오차는 더 크게 측정됨

이상치가 많이 존재하면, 성능을 과대평가할 수 있음.

---

- RMSE : MSE를 루트 씌움

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i)^2}$$

▼ R^2(mean squared error) : 해당 모델의 설명력(데이터의 변동성 설명)

전체 변동 중 회귀식이 해당 x, y관계를 설명

$$R^2 = \frac{\sum_{i=1}^{n}(f(x_i) - \bar{y})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

예측값의 분산을 실제값의 분산으로 나눔

특징 )

1에 가까워질수록 성능이 좋은 모델

특징(x) 수가 많아지면 값이 늘어남

---

- Adjusted R^2

$$Adjusted\ R^2 = 1 - \frac{(1-R^2)(n-1)}{(n-k-1)}$$

* n = 데이터 개수, k = 특징 개수

특징 수가 많을 때, 사용함

- 성능개선
  - 추가적 데이터수집 : 외부 데이터를 더 보강하거나, 이전에 고려하지않았던 데이터를 추가 등 더 많은 데이터를 활용
  - 추가적 데이터 전처리 : 결측치 처리, 범주형 데이터 인코딩 등 각종 데이터전처리 기술을 적용해 특징을 잘 추출할 추가적 전처리 진행
  - 데이터에서 다른 특징 선택 : 다른 특징을 선택해 탐색적 분석을 함. 적합한 특징을 선택하고 부적합한 특징은 제거
  - 다른 알고리즘으로 모델 훈련 : 해결하고자하는 문제나 데이터의 특성에 맞는 다른 알고리즘을 선택해 모델을 훈련

---

▼ numpy, matplotlib

> Numpy

: 수학 및 과학 분야의 수치 연산을 위한 파이썬 패키지
- 특히, 벡터, 행렬 등 계산할 때 빠른 고성능 계산이 가능하여 대량의 데이터를 처리하는데 유리함

▼ ndarray : 다차원 array형태를 ndarray객체를 제공함

- 필요성 )

행렬 및 벡터연산을 위해 다차원 array를 사용해야함

- 속성 )

  - ndarray.ndim: 배열의 차원 수
  - ndarray.shape: 각 차원의 크기를 나타내는 튜플
  - ndarray.size: 배열에 포함된 전체 요소 개수
  - ndarray.dtype: 배열에 저장된 요소의 데이터 타입

- 형 변환 )

  -ndarray.astype(자료형) : 배열을 특정 자료형으로 변환

- int8, int16, int32, int64
- float16, float32, float64, float128
- complex64, ...
- bool

---

- 넘파이 배열 생성 )

  - np.arange(): 원하는 숫자 범위 내 특정 간격에 따른 배열 생성
  - np.ones(): 1로 가득찬 배열 생성
  - np.zeros(): 0으로 가득찬 배열 생성
  - np.full(): 특정 값으로 가득찬 배열 생성
  - np.linspace(): 원하는 숫자 범위 내 원하는 개수의 요소를 가진 배열 생성

- 랜덤값 배열 생성 )

  - np.random.rand(): 0과 1 사이의 무작위 값이 들어간 배열 생성 (균등분포, uniform dist.)
  - np.random.randn(): -1과 1 사이의 무작위 값이 들어간 배열 생성 (정규분포, normal dist.)
  - np.random.randint(): 특정 범위 내 무작위 정수값 들어간 배열 생성

---

- 넘파이 배열구조의 재배열 )

  - np.reshape(변경할 배열, 차원)
  - ndarray.reshape(차원)

> 넘파이 배열간의 연산은 반복문 없이도, 내부적으로 벡터 내 성분 간 연산처리 가능함(벡터화 계산)

- 벡터의 내적구하기(dot product)

내적은 벡터의 같은 성분끼리 각각 곱해 합한 값, 스칼라 값으로 반환

- np.dot(벡터1, 벡터2)
- 벡터1 @ 벡터2

---

- 넘파이 배열 응용연산(통계량, 고급연산)

  - np.sum(): 배열 요소 전체 합산
  - np.mean(): 배열 요소 전체 평균
  - np.median(): 배열 요소 중앙값
  - np.var(): 배열 요소 분산
  - np.std(): 배열 요소 표준편차

- 그외 수치계산

- np.exp(): 자연상수 e의 지수함수
- np.log(): 자연상수 e의 로그함수
- np.sqrt(): 제곱근

### Matplotlib

: 시각화 하는데 사용하는 라이브러리

▼ 선 (line) 그래프

- plt.plot(): 기본적으로 매개변수로 하나의 데이터 (x) 혹은 두개의 데이터 (x,y)을 넣음

- 두 개 이상의 선을 그리고 싶을 때는 plt.plot()에 데이터를 추가 (x1, y1, x2, y2, ...)하거나,
- 여러번 plt.plot(x1, y1); plt.plot(x2, y2); .. 을 반복하면 됨

```
- plt.legend(): 범례
- plt.title(): 차트 제목
- plt.xlabel(): x축 라벨
- plt.ylabel(): y축 라벨
```

▼ 산점도 그리기

- plt.scatter()

▼ 히스토그램

- plt.hist()

  bins : 히스토그램 구간 개수 지정

▼ 파이 차트

- plt.pie()

  labels : 파이 별 라벨 설정

  autopct : 파이 문자열 출력형식 설정

▼ 히트맵 그리기

- plt.matshow()
- plt.colorbar() : 컬러바 추가

▼ 회귀분석

: 특징(x)과 실수인 목표값(y) 사이의 관계를 살피는 지도학습 모형

▼ 학습 및 평가 데이터

특징 : 다이아몬드의 캐럿 값

목푯값 : 다이아몬드의 가격

## ▼ 최소제곱법 기반 회귀분석

$$W = \frac{\sum_{(} x_i - \bar{x})(y_i - \bar{y})}{\sum_{(} x_i - \bar{x})^2}$$

$$b = \bar{y} - W\bar{x}$$

```
Function LinearRegression(X, Y):
    // 단계 1: X와 Y의 평균 계산
    mean_X = mean(X)
    mean_Y = mean(Y)

    // 단계 2: (X - mean_X)의 제곱합을 계산
    sum_X_squared = sum((X - mean_X) ** 2)

    // 단계 3: (X - mean_X)와 (Y - mean_Y)의 곱의 합을 계산
    sum_XY_product = sum((X - mean_X) * (Y - mean_Y))

    // 단계 4: 회귀선의 기울기(weight)와 절편(bias) 계산
    weight = sum_XY_product / sum_X_squared
    bias = mean_Y - (weight * mean_X)

    // 최종적으로 계산된 weight와 bias 반환
    return weight, bias
```

## ▼ 경사하강법 기반 회귀분석

$$\frac{\partial LOSS(W,b)}{\partial W} = \frac{2}{n}\sum_{i=1}^{n} x_i((Wx_i + b) - y_i)$$

$$\frac{\partial LOSS(W,b)}{\partial b} = \frac{2}{n}\sum_{i=1}^{n}((Wx_i + b) - y_i)$$

$$W = W - \alpha\frac{\partial LOSS}{\partial W}$$

$$b = b - \alpha\frac{\partial LOSS}{\partial b}$$

```
Function LinearRegression(X, Y, learning_rate, epochs):
    // 단계 1: 기울기(W)와 절편(b)을 0으로 초기화
    weight = 0
    bias = 0
    n = len(X)  // n = 데이터 개수

    // 단계 2: 지정된 에폭 수만큼 경사하강법 수행
    for i = 1 to epochs
        // 단계 2.1: 현재 W와 b에 기반한 예측값 계산
        Y_pred = weight * X + bias

        // 단계 2.2: W와 b에 대한 각각의 경사(gradient) 계산
        gradient_weight = (2 / n) * sum(X * (Y_pred - Y))
        gradient_bias = (2 / n) * sum(Y_pred - Y)

        // 단계 2.3: 경사와 학습률을 사용해 W와 b 업데이트
        weight = weight - learning_rate * gradient_weight
        bias = bias - learning_rate * gradient_bias

    // 단계 3: 최적화된 W와 b 반환
    return weight, bias
```