

# Machine Learning - Homework1

## Linear Regression

32184801 Ha Seung Won

### 0 Abstract

이번 과제에서는 Linear Regression 모델을 직접 구현하여 Linear Regression의 프로세스를 이해하는 것을 목표로 한다. 이론적인 부분을 기반으로 하여 직접 코드를 작성함으로써, Linear Regression의 기본 원리와 구현 방법을 더 깊이 이해할 수 있다. 이 과제를 통해 선형 회귀에 대한 이론과 코드 작성 능력을 향상시키고, 데이터 분석 및 예측 모델 구축에 필요한 기초를 다지는 데 도움이 될 것이다. 기본적인 이론을 설명하는 Relate Work, 코드 설명과 구현 결과는 Implementation에서 정리하였고 마지막으로 느낀 점으로 마무리하였다.

### 1 Relate Work

#### 1-1 Linear Regression

Linear Regression에서 기본적인 프로세스는 아래와 같다.

##### 1 Hypothesis Function 지정

##### 2 Model train

##### 3 Predict

Hypothesis Function을 지정하면 처음에 훈련할 모델에 대한 알고리즘을 선택을 하고 파라미터를 초기화한다. 그 이후에는 그 모델을 학습을 해 나가면서 최적의 파라미터 값을 찾아나간다. 그리고 마지막으로 Predict로 모델을 활용하여 train 데이터가 아닌 새로운 데이터로 값을 예측한다. 이런 프로세스를 통해서 Linear Regression과 전반적인 머신러닝 모델의 학습 과정이 진행된다.

- Error

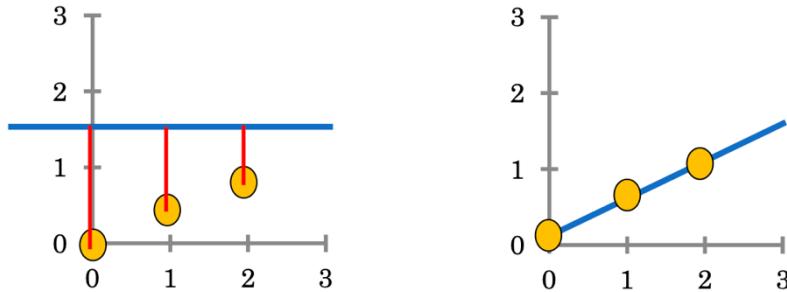


Figure 1 데이터 시각화

Error란 모델이 예측한 값과 실제 값의 차 이를 Error 혹은 Cost라고 한다. 그래서 이런 값을 최소화하는 게 모델 훈련의 목적이라고 이야기를 할 수 있다. 그렇다면 이런 Error를 구하는 방식은 Cost Function( $J$ )에 의하여 정해지는데, Cost Function은 input은 모델의 파라미터 값을 통해서 구해지며  $J(\theta_0, \theta_1)$ 라고 표시한다. 이런 함수들의 예시로 MSE(Mean Squared Error), Cross-Entropy 등등이 있다.

## 1.2 Mean Squared Error

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

Figure 2 MSE Formula

MSE는 Cost Function의 대표적인 예시로서 말 그대로 Error의 제곱의 평균이다. 식을 보자면 먼저 Hypothesis Function( $h$ )를 이야기한다. 기존에 정했던 예측 모델 함수를 input 값  $x$ 를 통해서 현재 모델의 예측값이 나오고 실제 target 데이터와의 차이를 구한다. 그리고 이것을 음수 양수 관계없이 사용하기 위해서 제곱을 하며 이것을 모든 input 데이터에서 구하고 이것의 평균을 구하는 것이 MSE의 방식이다. 여기서 2로 나눠주는 이유는 나중에 계산을 편하게 하기 위해서인데 변화량을 구하려고 미분을 추가적으로 수행하기 때문이다.

$$\underset{\theta_0, \theta_1}{\text{minimize}} J(\theta_0, \theta_1)$$

Figure 3 Cost Function

그래서 결국에는 이런 **Cost Function**의 값을 최소화 하는 파라미터 값을 찾는 게 모델 훈련의 과정이 된다. 이것을 **Recall**이라고 한다.

## 1.3 Learning Algorithms

- **Least Square**

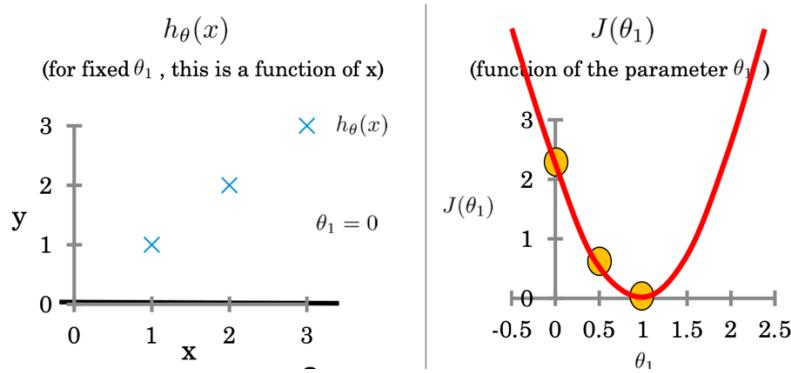


Figure 4 Hypothesis & Cost Function

이제 Error 값을 계산하는 함수인 Cost Function에서 최솟값이 되도록 하는 파라미터를 구하는 방식에 대하여 알아보겠다. 먼저 일반적으로 생각하면 Cost Function을 표현한 Figure 4 오른쪽 그림을 보면 그래프의 모양이 아래로 볼록한 모양이다. 이건 Convex 한 모양이라고 할 수 있다. (반대는 Concave 하다고 한다.) 그렇다면 이런 함수적 특성을 본다면 값이 최소가 되려면 기울기가 0 인 즉 미분하여 값이 0 이 되는 부분이 최소가 된다고 생각을 할 수 있다. 그래서 나온 첫 번째 방식이 편미분을 통해서 각각의 함수가 최소가 되는 파라미터의 값을 찾는 방법이다.

- **Partial Derivative**

- $\underset{\theta_0, \theta_1}{\operatorname{argmin}} J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$
- Taking the derivative of  $J$  w.r.t each parameter
  - $\frac{\partial J}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \frac{\partial h_\theta(x^{(i)})}{\partial \theta_0} = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})$
  - $\frac{\partial J}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \frac{\partial h_\theta(x^{(i)})}{\partial \theta_1} = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$

Figure 5 MSE Partial derivative

먼저 Cost Function이 있다고 가정하면 각각 파라미터 **P0**, **P1**에 대하여 각각 편미분을 통해서 식을 전개한다. 그래서 이 계산 과정에서 생기는 식을 통해서 Figure 6 방식으로 전개한다. 이 부분에서 앞서 이야기했던 계산을 편하게 하기 위해 2를 나눠준 이유가 된다.

( $\operatorname{argmin}$ 은 어떤 함수를 최소로 만드는 변수를 말한다.)

### Least Square

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

• For  $\theta_0$

$$\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) = \frac{1}{m} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)}) = 0$$

$$\theta_0 = \frac{\sum_{i=1}^m (y^{(i)} - \theta_1 x^{(i)})}{m}$$

• For  $\theta_1$

$$\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x^{(i)} = \frac{1}{m} \sum_{i=1}^m (\theta_0 x^{(i)} + \theta_1 x^{(i)} - y^{(i)} x^{(i)}) = 0$$

$$\theta_1 = \frac{\sum_{i=1}^m (y^{(i)} - \theta_0) x^{(i)}}{\sum_{i=1}^m x^{(i)} 2}$$

Figure 6 MSE Partial derivative calculate

편미분을 각각 파라미터를 기준으로 진행하고 그 식에서 convex 함수의 최솟값을 구하기 위해서 미분 값의 0 이 되는 미지수의 값을 구해야 한다. 그래서 이 값이 Error 가 최소가 되는 파라미터 값이 된다.

### 1.3.1 Normal Equation

$$\begin{aligned} & \cdot \begin{bmatrix} 1 & a^{(0)} \\ 1 & a^{(1)} \\ 1 & a^{(2)} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} b^{(0)} \\ b^{(1)} \\ b^{(2)} \end{bmatrix} \\ & \cdot Ax = b \end{aligned}$$

| i | a | b |
|---|---|---|
| 0 | 1 | 2 |
| 1 | 2 | 3 |
| 2 | 3 | 4 |

Figure 7 Represent data & Parameters in matrix

Normal Equation 은 선형대수를 사용하여 최적 x에 대한 공식을 유도하는 방식입니다. Hypothesis Function 을 Matrix 형식으로 계수를 표현하면 위와 같이 표현이 가능하다. Figure 8 예시는 Hypothesis Function 에서 단순히 1 차 함수고 y 절편(bias)이 없는 상태이니 이것을 그냥 1로 채우고 matrix 로 식을 쓰면 위의 행렬과 같다.

$$A^T A \hat{x} = A^T b \rightarrow \hat{x} = (A^T A)^{-1} A^T b$$

Figure 8 Normal Equation formula

$Ax=b$  인 상황에서  $A^T$  즉 A 행렬의 transpose 한 값을 양변에 곱하고 그걸 다시 묶어서 invertible 를 또 양변에 해주면 구하고자 했던 파라미터 값을 구하는 식이 완성된다.

- 증명

Figure 9 Error calculate in matrix

Matrix 방식으로 기존 Hypothesis 와 input 값을 포함하여 같이 표시하면 왼쪽 matrix 형식으로 나온다. 그 이후에 예측값과 target 값을 뺀 error 값을 표시를 하면 오른쪽 그림과 같다. 이제 error 를 구하는 방식을 알았다면 이제 error 를 통해서 Least Squared Method 를 적용하는 게 가능하다. 이전에 matrix 가 아닌 다향 함수에서의 방식에서는 모든 데이터에서 error 값을 제곱을 더한 값을 구하였다. Matrix 에서는  $\|Ax-b\|_2^2$ 로 그 값을 구하는 게 가능하다.

$\|Ax-b\|_2^2$  를 먼저 설명하자만  $Ax$  는 기존의 Hypothesis 함수를 말하고  $b$  는 실제 데이터를 말한다. 그래서 이것을 뺀 것은 기존에 말했던 error 를 표시한 것이다.  $\|\cdot\|$  는 norm 을 의미하며 2 를 표시한 것은 아래의 2 는 각각의 요소들을 전부 제곱을 해준다는 의미이고 위에 있는 2 는 이것을 제곱을 해준다는 의미이다. 그래서 이것을 기존에서 MSE 방식에서 error 의 제곱을 전부 더하는 부분과 같다.

하지만, Matrix 에서는 단순히  $\|x\|_2^2 = x^T * x$  로 표시하는 게 가능하다. 이렇게 MSE 를 Matrix 구성에서 구하는 방식은 아래와 같아.

$$\|Ax - b\|_2^2 = (Ax - b)^T (Ax - b) = x^T A^T A x - 2b^T A x + b^T b$$

Figure 10 Square Error Calculate

- **Minimize Cost Function**

$$\begin{aligned} \nabla_x (x^T A^T A x - 2b^T A x + b^T b) &= \nabla_x x^T A^T A x - \nabla_x 2b^T A x + \nabla_x b^T b \\ &= 2A^T A x - 2A^T b \end{aligned}$$

Figure 11 Partial Derivative in Square Error

$$x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \quad b = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \nabla_x b^T x = b$$

$$\nabla_x b^T x = \begin{bmatrix} 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = x_0 + 2x_1 \quad \nabla_x^2 b^T x = 0$$

$$\nabla_x b^T x = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \Rightarrow \frac{\partial b^T x}{\partial x_0} = 1 \quad \nabla_x x^T A x = 2Ax \text{ (if } A \text{ symmetric)}$$

$$\nabla_x b^T x = \begin{bmatrix} 1 \\ 2 \end{bmatrix} = b \quad \nabla_x^2 x^T A x = 2A \text{ (if } A \text{ symmetric)}$$

Figure 12 Matrix derivative formula

LSM 방식으로 Cost Function 을 Figure 11 과같이 만들었다. 이제 값이 최소가 되는 파라미터 값을 찾아야 한다. 그래서 Figure 12에서 파라미터 값을 기준으로 미분을 진행하였고 이후에는 Figure 13의 Matrix의 미분에 대한 특성을 활용하여 식을 정리하면 결국에는 Figure 12에서 나오는 최종의 식이 도출된다. 그렇게 미분을 통해서 최적의 파라미터 값을 찾으려면 결국 미분 값이 0 이 되어야 하므로 식을 간단히 하게 되면 결국에는 Figure 9에서 나왔던 식이 결국에는 도출된다.

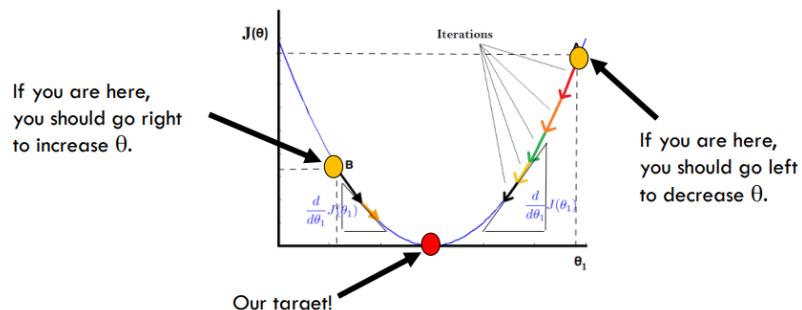
- 한계

Normal Equation에서 10,000 개의 데이터 보다 작은 잡업을 수행하는 데 여기서 많은 수행 시간이 필요하지 않다. 하지만 그 이상은 너무 많은 연산 시간을 필요로 한다.

Normal Equation을 이용하다 보면 non-invertible 문제가 발생하기도 한다. 이런 문제는 보통 중복된 feature 가 존재하거나 데이터의 총 양보다 feature의 양이 너무 많을 때 발생한다. 이런 문제를 해결하기 위해서는 feature 를 중복된 feature 를 제거하거나 regularization 을 통해서 문제를 해결한다.

### 1.3.2 Gradient Descent

기본적인 프로세스는 cost Function에서 일정 파라미터에 대하여 특정 시작점을 지정하고 점점 이동하면서 목표로 하는 지점까지 이동하는 방식이다. 한국어로 해석하면 경사하강법이라고 이야기를 한다.



- go right:  $\theta += (-) \text{gradient} * \alpha$
- go left:  $\theta += (-) \text{gradient} * \alpha$

Figure 13 Gradient Descent visualization

- Go right, Go left :  $\Theta += - \text{gradient} * \alpha$

식을 통해서 알 수 있듯이 파라미터의 값을 일정 gradient 값과 알파 값을 곱한 값을 빼서 파라미터를 업데이트한다. 여기서 생기는 의문은 왼쪽이던 오른쪽이던 값은 방식으로 적용이 된다는 것인데 그림을 보면 왼쪽에서 값이 업데이트된다고 하면 gradient 값이 음수이므로 결국 더해지게 되고 오른쪽에서 시작하면 gradient 값은 양수이므로 값이 줄어든다. 결국 왼쪽이던 오른쪽이던 같은 방식으로 적용이 가능하다.

#### • Gradient Descent Process

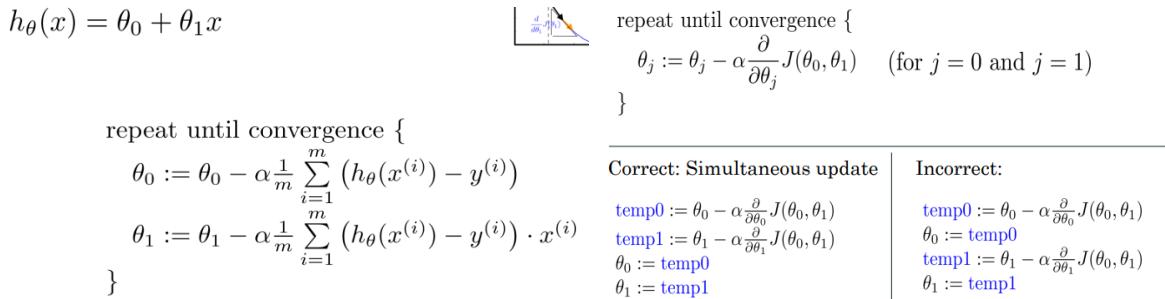


Figure 14 Gradient Descent Formula

**Iteration** : Iteration 은 파라미터가 업데이트되는 횟수를 말한다.

$\alpha$  : 알파값은 learning rate 라고도 이야기하고 학습되는 정도를 이야기하며 파라미터가 조정되는 정도를 지정된다.

이제 파라미터를 업데이트할 때는 Iteration 을 반복하여 0에 가까워질 때까지 반복한다. 여기서 파라미터값이 여러 개 있다면 각각의 파라미터에 대하여 Partial derivative 편미분을 통해서 각각의 파라미터를 업데이트 해준다. 이때 업데이트는 각각 따로 하는 게 아니라 동시에 업데이트시켜 준다.

#### 1.3.2.1 Gradient Descent Types

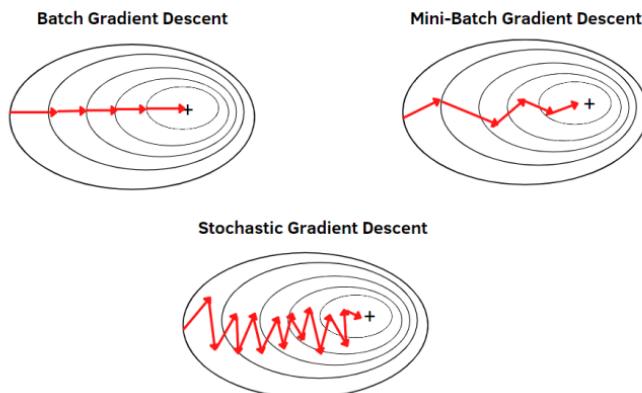


Figure 15 Different Gradient Descent

#### -Batch Gradient Descent

데이터셋의 전부를 탐색하는 방식으로 정확한 방향을 찾는 게 가능하지만 조금 느리고 Local Minimum 문제가 발생하기 쉽다.

#### -Stochastic Gradient Descent

한 개 한 개 탐색하는 방식으로 데이터를 한 개씩 탐색을 하기 때문에 정확하게 탐색하기 힘들다

#### -Mini-Batch Gradient Descent

가장 많이 쓰이는 방식으로 일정량의 데이터로 분할하여 탐색하는 방식이다. 용어를 정리하자면 **Batch** 는 분할한 데이터 샘플을 말하고 **Iteration** 은 그 데이터를 통해서 파라미터를 학습한 횟수를 말한다. 그리고 **Epoch** 는 이런 한 프로세스를 반복한 횟수를 말한다.

## 1.4 Local Minimum

#### •Local minimum

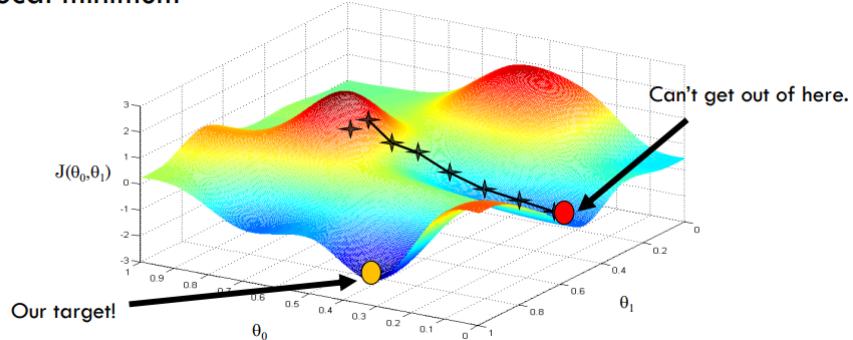


Figure 16 Local minimum visualization

Local Minimum 이란 Gradient Descent 방식을 사용하면 발생하는 문제로 Cost Function에 대한 값이 최소가 되는 부분을 찾아 나가는 과정에서 전체적인 최소가 아닌 부분적인 최소에 빠지는 부분이 발생한다. 이것을 미분값을 사용하여 0에 가까운 값이 될 때까지 파라미터를 업데이트를 하는데 이 부분이 전체적으로 보았을 때 최소가 아닐 수 있다는 것이다.

## 2 Implementation

### 2-1 Code Comment

#### Task 1

##### - Data & Initial Parameter setting

```
df=pd.read_csv("data_hw1.csv")
x = df['x'].values
y = df['y'].values

# Initialize parameters with small random values close to zero
a = random.uniform(-0.1, 0.1)
b = random.uniform(-0.1, 0.1)

print(f'initial parameters are {a,b}')

lr = 0.01 # Initialize the learning rate
          0.0s
```

Python

Figure 17 Data & Initial Parameter setting code

Pandas 라이브러리를 활용하여 데이터를 DataFrame 형식으로 파일을 읽고 input data 'x' 와 target data 'y' 변수에 각각 저장하였다. 그리고 초기 파라미터 값은 랜덤하게 초기 설정을 해주었다.

##### - Model Definition

```
class Model():
    def __init__(self, parameter1, parameter2, learning_rate):
        self.parameter1 = parameter1
        self.parameter2 = parameter2
        self.lr = learning_rate

    def function(self, x):
        return self.parameter1 * x + self.parameter2
```

1

```
def calculate_gradient(self,input,target,temp1,temp2,error):
    for index in range(len(input)):
        temp1 += (self.function(input[index]) - target[index]) * input[index]
        temp2 += (self.function(input[index]) - target[index])
        error += (self.function(input[index]) - target[index]) ** 2 # Calculate MSE

    gradient1 = temp1 / len(input)
    gradient2 = temp2 / len(input)
    mse_error = error / (2 * len(input))

    return gradient1,gradient2,mse_error
```

2

```

def train(self, input, target, iterates):
    flag = 0
    first_zero_iterate = 0

    for iterate in range(iterates):
        temp1 = 0
        temp2 = 0
        error = 0
        gradient1,gradient2,mse_error=self.calculate_gradient(input,target,temp1,temp2,error)

        #parameter update
        self.parameter1 -= self.lr * gradient1
        self.parameter2 -= self.lr * gradient2

        print(f"iterate) parameter1: {self.parameter1:.2f}, parameter2: {self.parameter2:.2f}, MSE_error: {mse_error:.2f}, Gradient1: {gradient1}, Gradient2: {gradient2}")

        #Findout first iterate about gradients are similar with 0
        if gradient1 < 1 and gradient2 < 1 and flag == 0:
            flag = 1
            first_zero_iterate = iterate
    return self.parameter1, self.parameter2, mse_error, first_zero_iterate

def mini_batch_train(self, input, target, epochs, batch_size, first_zero_iterate):
    iterate = 0
    flag = 0

    for epoch in range(epochs):
        for _ in range(len(input) // batch_size):
            # Randomly select a batch from the input and target data
            indices = np.random.choice(len(input), size=batch_size, replace=False)
            batch_x = input[indices]
            batch_y = target[indices]

            temp1 = 0
            temp2 = 0
            error = 0

            gradient1,gradient2,mse_error=self.calculate_gradient(batch_x,batch_y,temp1,temp2,error)

            #Parameter update
            self.parameter1 -= self.lr * gradient1
            self.parameter2 -= self.lr * gradient2

            print(f"iterate) parameter1: {self.parameter1:.2f}, parameter2: {self.parameter2:.2f}, MSE_error: {mse_error:.2f}, Gradient1: {gradient1}, Gradient2: {gradient2}")

            if (gradient1 < 1 and gradient2 < 1) and flag == 0:
                flag = 1
                first_zero_iterate = iterate
            iterate += 1
    return self.parameter1, self.parameter2, mse_error, first_zero_iterate

```

3

4

Figure 18 Task1 model class define

Figure 17 코드는 Task 1  $y=ax+b$  모델을 구현한 코드이다. 선언 당시 초기값은 파라미터 두개와 Gradient descent learning 에 필요한 learning rate 로 class 선언이 가능하다.

### 1. Function

Hypothesis Function 을 구현한 코드로 클래스의 파라미터를 기준으로 input 값 x 을 기준으로 현재 모델의 예측값을 계산하기 위한 함수이다.

### 2. Calculate\_gradient

Figure 5 그림에서 나왔던 식을 기반으로 각 파라미터의 Gradient 값을 계산하기 위해 필요한 함수이다. 또한 input 데이터를 기반으로 error<sup>2</sup> 의 평균을 구하는 MSE 도 이 부분에서 계산이 가능하게 하였다.

### 3. Train

Gradient descent 방식을 활용하여 최적을 파라미터를 찾아나가는 모델 훈련의 과정을 구현한 부분이다. Figure 13 번에 나왔던 식에 따라 gradient 값에서 초기 설정한 learning rate 값을 곱하여 기존의 파라미터 값에서 빼준다.

여기서 조건문을 추가한 이유는 Gradient descent 방식이 Gradient 값이 0에 가까워 질때까지 반복을 이여나간다는 이론적 부분을 직접 확인하기 위해서 처음으로 0에 가까워지는 연산횟수인 iteration 을 확인하기 위하여 추가를 해주었다.

#### 4. Mini-batch train

Mini-batch 훈련 방식은 1.5.1 Gradient Descent Types 에서 설명을 했듯이 batch size 로 데이터를 나눠서 학습하는 코드를 추가로 구현을 해보았다.

- Train code & output

- Batch model train

```
model = Model(a, b, lr)
iterates=1000

parameter1,parameter2,error,first_zero_iterate=model.train(x, y,iterates)

print()
print('-----train finish -----')
print(f'first zero iterate: {first_zero_iterate}, final parameter a : {model.parameter1}, final parameter b : {model.parameter2}'')
```

Python

Figure 19 Batch model train code

```
23 parameter1: -0.47, parameter2: 0.39, MSE_error: 76.99, Gradient1: 11.035502818573241, Gradient2: 0.5545413081619495
24 parameter1: -3.58, parameter2: 0.36, MSE_error: 76.99, Gradient1: 10.798243631926258, Gradient2: 0.6528314573232229
25 parameter1: -3.68, parameter2: 0.35, MSE_error: 75.78, Gradient1: 10.798243631926258, Gradient2: 0.6528314573232229
26 parameter1: -3.79, parameter2: 0.34, MSE_error: 74.62, Gradient1: 10.567313866751299, Gradient2: 0.7479081206278686
27 parameter1: -3.89, parameter2: 0.33, MSE_error: 73.51, Gradient1: 10.342348716289683, Gradient2: 0.8398611071039875
28 parameter1: -4.00, parameter2: 0.32, MSE_error: 72.44, Gradient1: 10.123187837119415, Gradient2: 0.9287777769566897
29 parameter1: -4.09, parameter2: 0.31, MSE_error: 71.42, Gradient1: 9.909675228121927, Gradient2: 1.0147431808504301
30 parameter1: -4.19, parameter2: 0.30, MSE_error: 70.44, Gradient1: 9.70165911273167, Gradient2: 1.097839766088481
31 parameter1: -4.29, parameter2: 0.29, MSE_error: 69.49, Gradient1: 9.498991824379305, Gradient2: 1.1781481401054124
32 parameter1: -4.38, parameter2: 0.28, MSE_error: 68.59, Gradient1: 9.30152969504238, Gradient2: 1.25574645851964
33 parameter1: -4.47, parameter2: 0.27, MSE_error: 67.71, Gradient1: 9.109132946818622, Gradient2: 1.3307107938779927
34 parameter1: -4.56, parameter2: 0.25, MSE_error: 66.87, Gradient1: 8.92166558644094, Gradient2: 1.4031151482006965
35 parameter1: -4.65, parameter2: 0.24, MSE_error: 66.07, Gradient1: 8.73099530264934, Gradient2: 1.4730315038729038
36 parameter1: -4.73, parameter2: 0.22, MSE_error: 65.29, Gradient1: 8.560993366356866, Gradient2: 1.540529878486566
37 parameter1: -4.82, parameter2: 0.21, MSE_error: 64.54, Gradient1: 8.387534533509413, Gradient2: 1.6056783781944055
38 parameter1: -4.90, parameter2: 0.19, MSE_error: 63.82, Gradient1: 8.218496958590923, Gradient2: 1.6685432496164119
39 parameter1: -4.98, parameter2: 0.17, MSE_error: 63.12, Gradient1: 8.05376262687081, Gradient2: 1.7291889303380783
40 parameter1: -5.06, parameter2: 0.15, MSE_error: 62.45, Gradient1: 7.893214524043548, Gradient2: 1.787678088038597
41 parameter1: -5.14, parameter2: 0.14, MSE_error: 61.80, Gradient1: 7.736742111050115, Gradient2: 1.844071718286206
42 parameter1: -5.21, parameter2: 0.12, MSE_error: 61.17, Gradient1: 7.584235637584841, Gradient2: 1.8984290910367752
43 parameter1: -5.29, parameter2: 0.10, MSE_error: 60.56, Gradient1: 7.435588727654509, Gradient2: 1.9508078958708945
44 parameter1: -5.36, parameter2: 0.08, MSE_error: 59.98, Gradient1: 7.290698466268734, Gradient2: 2.0012642360035744
45 parameter1: -5.43, parameter2: 0.06, MSE_error: 59.41, Gradient1: 7.1494638414874645, Gradient2: 2.0498526810999964
46 parameter1: -5.50, parameter2: 0.04, MSE_error: 58.86, Gradient1: 7.011787178582653, Gradient2: 2.0966263089295163
47 parameter1: -5.57, parameter2: 0.01, MSE_error: 58.33, Gradient1: 6.8775732812570265, Gradient2: 2.141636745889654
48 parameter1: -5.64, parameter2: -0.01, MSE_error: 57.82, Gradient1: 6.746729534863839, Gradient2: 2.1849342064304826
49 parameter1: -5.70, parameter2: -0.03, MSE_error: 57.32, Gradient1: 6.61916583057363, Gradient2: 2.2265675314094673
50 parameter1: -5.77, parameter2: -0.05, MSE_error: 56.83, Gradient1: 6.494794497435011, Gradient2: 2.266584225405596
51 parameter1: -5.83, parameter2: -0.08, MSE_error: 56.36, Gradient1: 6.373538236278101, Gradient2: 2.3050304930211616
52 parameter1: -5.89, parameter2: -0.10, MSE_error: 55.91, Gradient1: 6.25529065410746, Gradient2: 2.341951274198544
53 parameter1: -5.96, parameter2: -0.12, MSE_error: 55.46, Gradient1: 6.13999320885873, Gradient2: 2.37739027857879
54 parameter1: -6.02, parameter2: -0.15, MSE_error: 55.03, Gradient1: 6.027561131502894, Gradient2: 2.4113900189279516
...
999 parameter1: -12.24, parameter2: -8.40, MSE_error: 25.86, Gradient1: 0.024886468605874693, Gradient2: 0.04529260737145249
-----train finish -----
first zero iterate: 360, final parameter a : -12.236792221345064, final parameter b : -8.399886886005278
Output was truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Figure 20 Batch model train output

- Mini-Batch model train

```
batch_model = Model(a, b, lr)
epoch=32
batch_size=32

parameter1,parameter2,error,first_zero_iterate=batch_model.mini_batch_train(x, y,epoch,batch_size,first_zero_iterate)

print()
print('----- train finish -----')
print(f'first zero iterate: {first_zero_iterate}, final parameter a : {batch_model.parameter1}, final parameter b : {batch_model.parameter2}'')
```

Python

**Figure 21 Mini-Batch model train code**

```

24 parameter1: -4.18, parameter2: 0.41, MSE_error: 63.56, Gradient1: 10.026087930800946, Gradient2: 0.4671305975829786
25 parameter1: -4.26, parameter2: 0.39, MSE_error: 60.44, Gradient1: 7.787382724437495, Gradient2: 1.9475901234638757
26 parameter1: -4.26, parameter2: 0.34, MSE_error: 44.54, Gradient1: 0.26082844775821488, Gradient2: 5.565190611838515
27 parameter1: -4.30, parameter2: 0.30, MSE_error: 47.14, Gradient1: 3.37347943162706, Gradient2: 4.228040464372269
28 parameter1: -4.43, parameter2: 0.30, MSE_error: 95.54, Gradient1: 13.54442670261552, Gradient2: -0.2558529965799471
29 parameter1: -4.56, parameter2: 0.29, MSE_error: 89.00, Gradient1: 12.740317268511447, Gradient2: 0.5598664762766876
30 parameter1: -4.56, parameter2: 0.29, MSE_error: 39.40, Gradient1: 0.23534600232136296, Gradient2: 4.672749015373762
31 parameter1: -4.70, parameter2: 0.25, MSE_error: 88.79, Gradient1: 13.75744621223231, Gradient2: -0.560741207506542
32 parameter1: -4.82, parameter2: 0.25, MSE_error: 86.45, Gradient1: 12.498077959774008, Gradient2: -0.24914147290539984
33 parameter1: -4.85, parameter2: 0.21, MSE_error: 40.97, Gradient1: 2.4015519212599283, Gradient2: 4.680527941095323
34 parameter1: -4.94, parameter2: 0.19, MSE_error: 88.78, Gradient1: 9.515258739269164, Gradient2: 1.5829395199503093
35 parameter1: -5.00, parameter2: 0.17, MSE_error: 59.03, Gradient1: 5.828069372720023, Gradient2: 2.3929336976527695
36 parameter1: -5.04, parameter2: 0.13, MSE_error: 42.54, Gradient1: 3.6761797471449036, Gradient2: 3.409364033849886
37 parameter1: -5.11, parameter2: 0.11, MSE_error: 57.81, Gradient1: 7.021889938546449, Gradient2: 1.9721595819966895
38 parameter1: -5.13, parameter2: 0.07, MSE_error: 50.67, Gradient1: 2.5276195328279383, Gradient2: 4.7516656675415465
39 parameter1: -5.22, parameter2: 0.06, MSE_error: 70.53, Gradient1: 8.63861291816348, Gradient2: 1.0863558162753784
40 parameter1: -5.33, parameter2: 0.06, MSE_error: 78.53, Gradient1: 11.478248612148665, Gradient2: -0.10163417380153283
41 parameter1: -5.34, parameter2: 0.00, MSE_error: 29.58, Gradient1: 0.19280527778510592, Gradient2: 5.187414300174648
42 parameter1: -5.47, parameter2: 0.01, MSE_error: 86.96, Gradient1: 12.992209729916334, Gradient2: -0.7537249540286456
43 parameter1: -5.52, parameter2: -0.01, MSE_error: 41.19, Gradient1: 5.54670203212092, Gradient2: 2.496784637044953
44 parameter1: -5.68, parameter2: 0.01, MSE_error: 97.20, Gradient1: 15.87941872727051, Gradient2: -1.9618297037798769
45 parameter1: -5.72, parameter2: -0.02, MSE_error: 40.13, Gradient1: 4.128858667750795, Gradient2: 2.691831679873757
46 parameter1: -5.88, parameter2: 0.00, MSE_error: 87.02, Gradient1: 15.917197541382198, Gradient2: -2.483877684327416
47 parameter1: -5.99, parameter2: 0.00, MSE_error: 78.31, Gradient1: 11.102364731914411, Gradient2: 0.23857600679859496
48 parameter1: -6.03, parameter2: -0.04, MSE_error: 44.06, Gradient1: 4.1908328278568705, Gradient2: 3.9338689079087725
49 parameter1: -6.14, parameter2: -0.04, MSE_error: 74.30, Gradient1: 10.218529582450719, Gradient2: 0.2041618610211044
50 parameter1: -6.20, parameter2: -0.05, MSE_error: 52.04, Gradient1: 6.816831325857646, Gradient2: 0.8303955787903909
51 parameter1: -6.26, parameter2: -0.07, MSE_error: 41.55, Gradient1: 5.9831404473439227, Gradient2: 1.9812765437012438
52 parameter1: -6.30, parameter2: -0.10, MSE_error: 46.68, Gradient1: 4.155141909767398, Gradient2: 2.824820980542819
53 parameter1: -6.40, parameter2: -0.11, MSE_error: 81.92, Gradient1: 9.79620561638097, Gradient2: 1.262381198022648
54 parameter1: -6.47, parameter2: -0.13, MSE_error: 42.06, Gradient1: 6.298502033508166, Gradient2: 1.8123223492205995
...
991 parameter1: -12.38, parameter2: -8.37, MSE_error: 29.85, Gradient1: 2.0670082576186956, Gradient2: -1.0678990820995153

----- train finish -----
first zero iterate: 294, final parameter a : -12.382852461838016, final parameter b : -8.367270949339666
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

**Figure 22 Mini-Batch model train output**

- Library model

```

from sklearn.linear_model import LinearRegression

library_model=LinearRegression(n_jobs=1000)# similar iteration numbers with batch, mini-batch model
library_model.fit(x.reshape(-1, 1),y.reshape(-1, 1))
coefficients = library_model.coef_
intercept = library_model.intercept_

print()
print('----- train finish -----')
print(f'final parameter a : {coefficients[0][0]}, final parameter b : {intercept[0]}')
```

Python

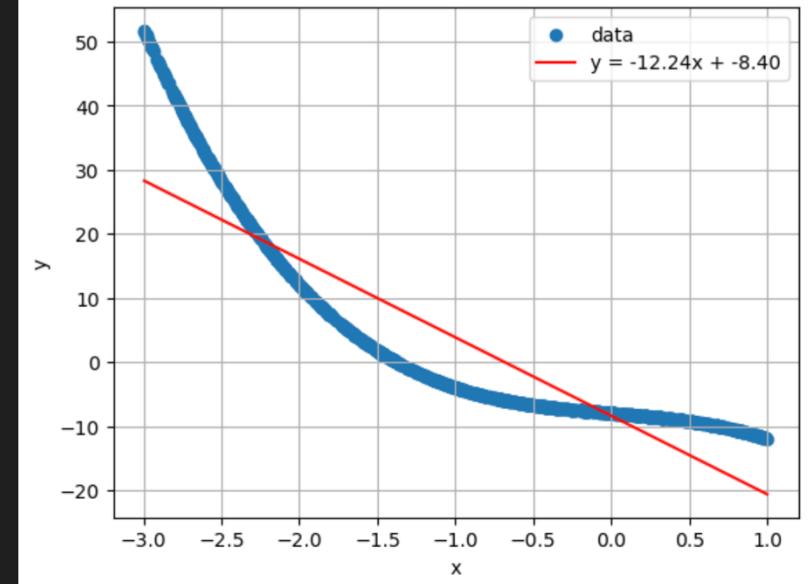
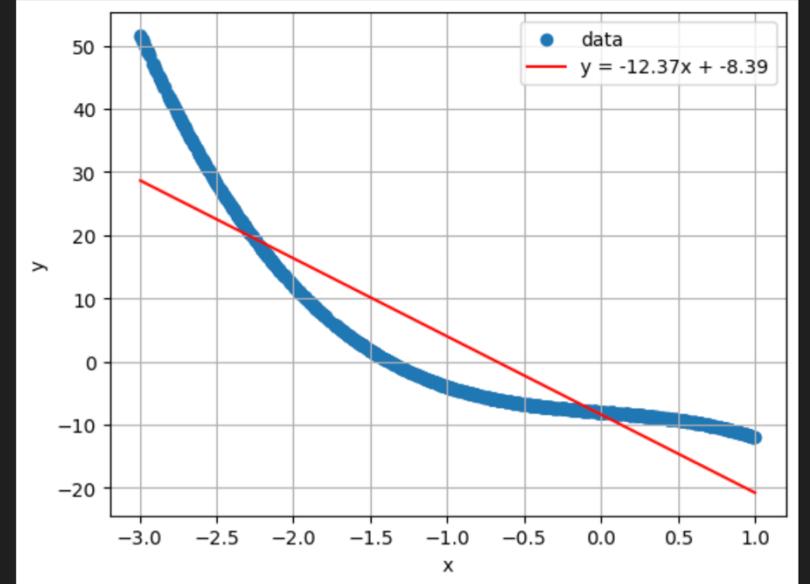
**Figure 23 Scikit-learn model train code**

```

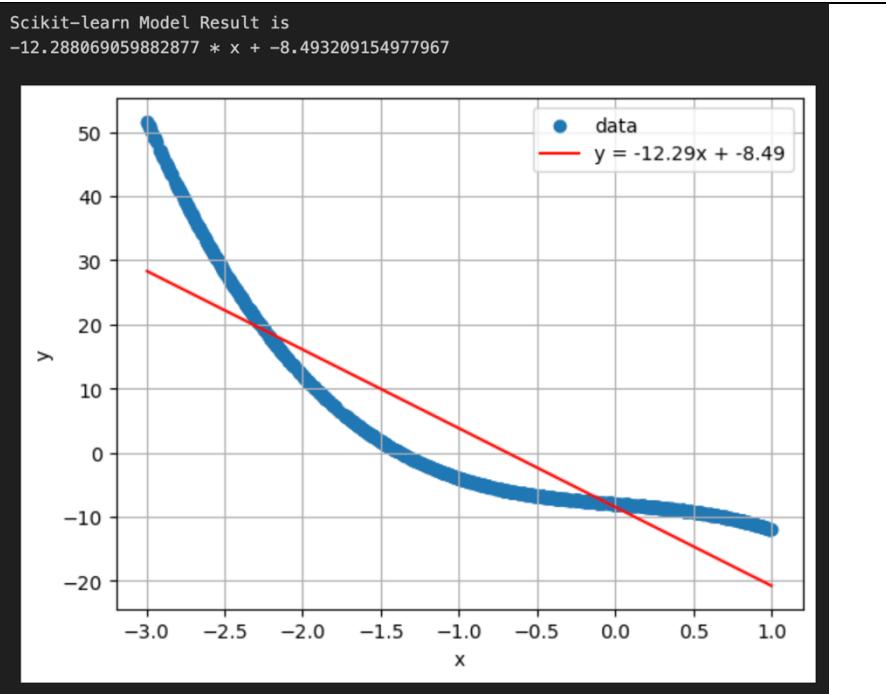
----- train finish -----
final parameter a : -12.288069059882877, final parameter b : -8.493209154977967
```

**Figure 24 Sciki-learn model train output**

## Result

| Model      | Plot  |
|------------|---|
| Batch      | <p>Batch model RESULT IS<br/><math>-12.236792221345064 * x + -8.399886886005278</math></p>  <p>A scatter plot showing data points (blue dots) and a linear regression line (red line). The x-axis ranges from -3.0 to 1.0, and the y-axis ranges from -20 to 50. The data points show a strong negative linear correlation. The red line represents the model's prediction, which closely fits the data points.</p> <p>First zero iterate : 360</p>     |
| Mini-Batch | <p>Mini-Batch Model Result is<br/><math>-12.372947684079 * x + -8.392574673229872</math></p>  <p>A scatter plot showing data points (blue dots) and a linear regression line (red line). The x-axis ranges from -3.0 to 1.0, and the y-axis ranges from -20 to 50. The data points show a strong negative linear correlation. The red line represents the model's prediction, which closely fits the data points.</p> <p>First zero iterate : 294</p> |

Scikit-  
learn



결과적으로 Batch, Mini-Batch 모델 모두 파라미터 학습이 잘 이루어진다는 사실을 확인할 수 있다. 하지만 Batch Gradient Descent 와 Mini-Batch Gradient Descent 의 First zero iterate 횟수를 비교를 해보면 Mini-Batch 모델이 더욱 빨리 수렴한다는 것을 알 수 있다. 이는 batch 사이즈를 적절하게 나누면 모델의 파라미터를 더욱 빠르게 수렴하게 하는 게 가능하다는 것을 보여준다. 그래서 데이터 셋이 큰 경우에는 보통 Mini-Batch 방식을 더 많이 사용하는 이유가 된다.

## Task 2

- Normal Equation

```

import numpy as np
1
x = df['x'].values
y = df['y'].values

X = np.array([]) # initialize empty NumPy array
for input in x:
    X = np.append(X, [1, input, input**2]) # Append [1, input, input**2] array in X
# Reshape X numpy array to 2D matrix
X = X.reshape(len(x), 3)

theta = np.linalg.inv(X.T @ X) @ X.T @ y
2

print(theta)
class Model2():
    def __init__(self, parameters):
        self.parameters = parameters

    def function(self, x):
        c, b, a = self.parameters
        return a * x**2 + b * x + c

model2 = Model2(theta)

print('Normal Equation model parameters')
print(f'parameter a : {theta[2]}')
print(f'parameter b : {theta[1]}')
print(f'parameter c : {theta[0]}')

```

Figure 25 Normal Equation Model

Normal Equation 방식을 통해서 파라미터 값을 구하는 방식을 구현한 코드이다. 먼저 데이터를 Matrix 구조로 변경하여서 데이터를 처리하고 그 이후에는 Matrix 의 연산을 통해서 진행하였다.

### 1. Convert to Matrix Form

먼저 구하고자 하는 polynomial regression 함수의 차수가 2 차수인 함수를 이야기함으로 matrix 로 값을 표현하면  $[1, x, x^2]$  값으로 표현을 해야한다 그래서 모듈 input 값을 기준으로 matrix 형식으로 input 값을 조정하여 2 차원 배열로 구성하였다.

### 2. Normal Equation Calculation

$$A^T A \hat{x} = A^T b \rightarrow \hat{x} = (A^T A)^{-1} A^T b$$

Normal Equation 공식의 식을 그대로 구현한 방식으로 Matrix 의 곱 연산을 통해서 계산된 파라미터 Matrix X 값을 theta 변수에 저장하는 형식으로 코드를 구현하였다.

- Library model

```

import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

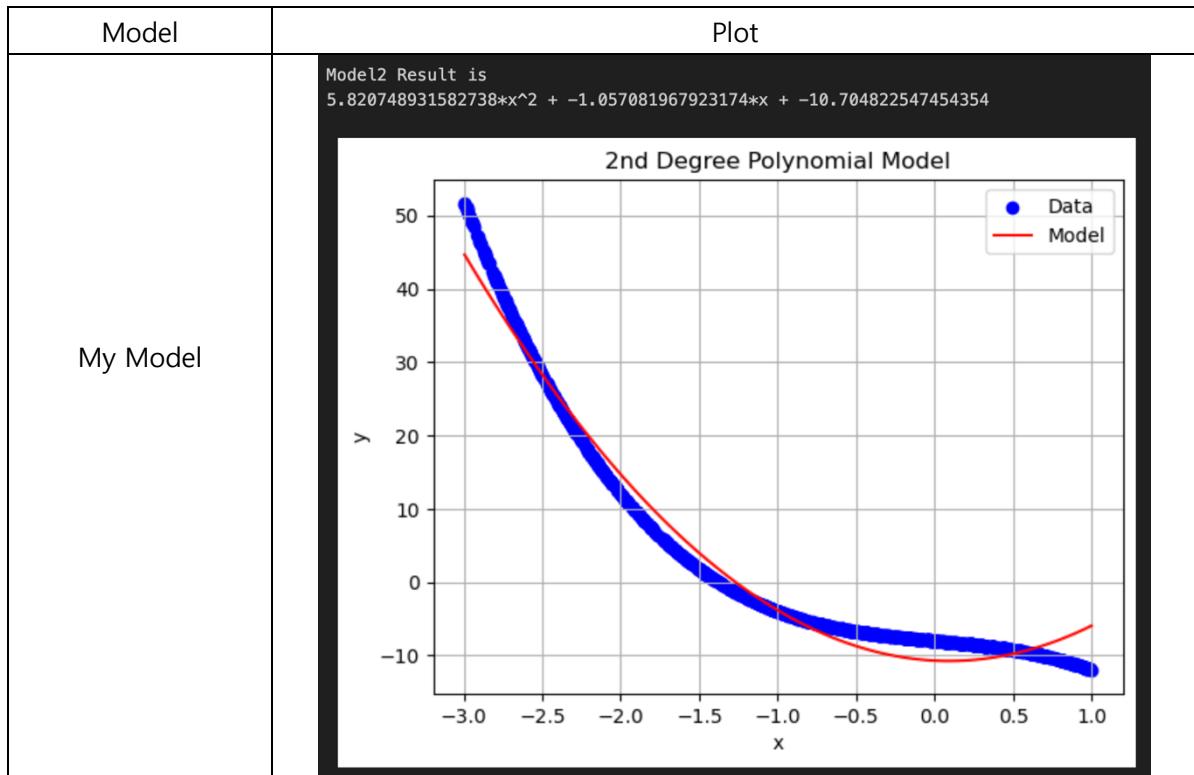
# 다항 특성 생성
degree = 2 # 다항식의 차수 설정
poly = PolynomialFeatures(degree=degree)
X_poly = poly.fit_transform(x.reshape(-1, 1))

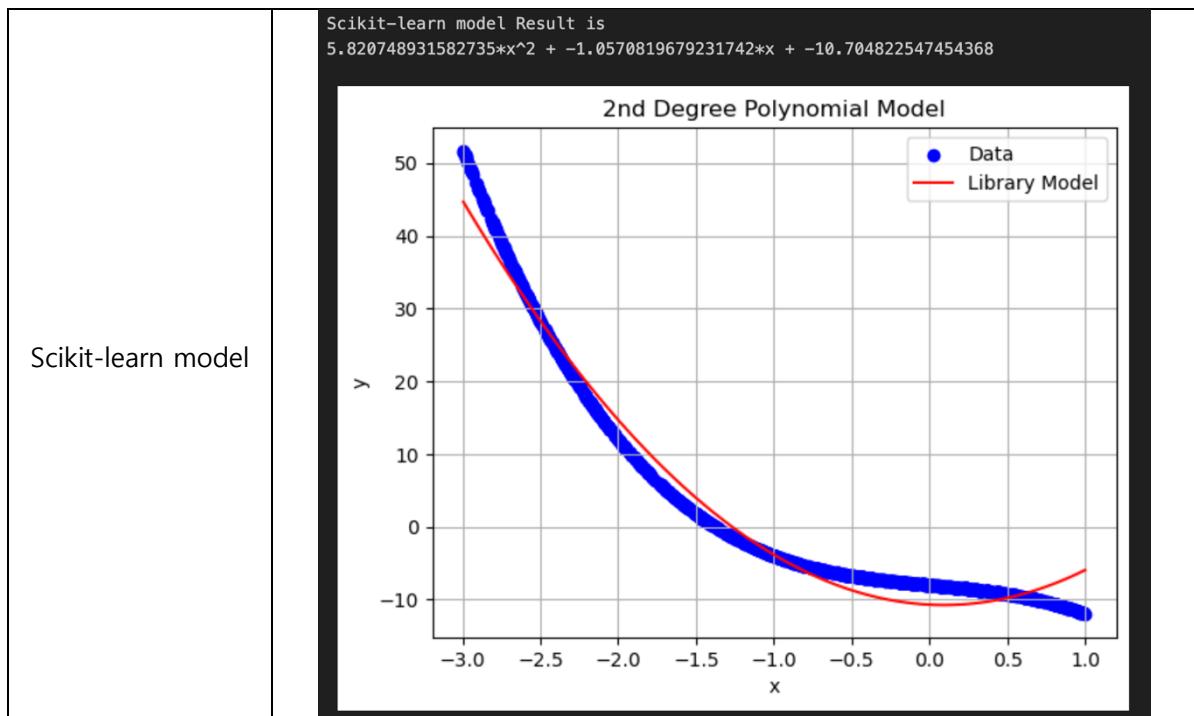
# 다항 회귀 모델 훈련
library_model = LinearRegression()
library_model.fit(X_poly, y)

# 회귀 결과 출력
print('Library Normal Equation model parameters')
print(f'parameter a : {library_model.coef_[2]}')
print(f'parameter b : {library_model.coef_[1]}')
print(f'parameter c : {library_model.intercept_}')

```

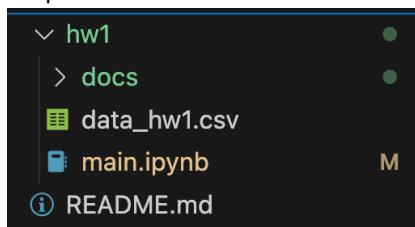
Figure 26 Scikit-Learn Library Model



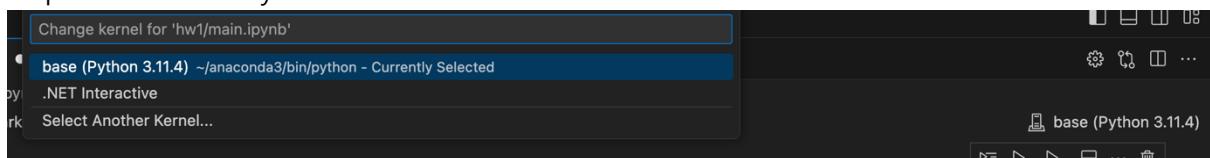


## 2-3 Run Code

Step 1 : Move to the Same Folder



Step 2 : Define the Python Base Kernel



Step 3 : Install the Libraries to Run the Code

- Pandas : 데이터를 읽을때 필요하다.
- Random : 첫 파라미터 지정하기 위해서 필요.
- Numpy : Matrix 연산에 필요
- Matplotlib : 결과 시각화를 하기 위해서
- Sklearn : 구현한 모델과 비교하기 위해서

Step 4: Execute the Cells one by one from top to bottom

```
In [20]: Execute Cell (^Enter) | t pandas as pd
          import random
          import numpy as np
[20]    ✓ 0.0s
Python
```

## Task 1

Model:  $y = ax + b$

Approach: Gradient Descent

```
class Model():
    def __init__(self, parameter1, parameter2, learning_rate):
        self.parameter1 = parameter1
        self.parameter2 = parameter2
```

### 3 느낀점

이번 과제를 통해서 Linear Regression 함수를 직접 구현을 해보니 함수가 어떻게 실행되는지 흐름을 정확하게 이해하는 게 가능해서 좋았다. 하지만 코드를 구현하면서 생기는 반복문으로 이론적으로 data set 이 늘어나면 Normal Equation 의 비효율성이 증가한다는 단점을 임의로 랜덤하게 데이터를 넣어서 파악하기 위해서 노력했지만 데이터가 직접적으로 확인하지 못한 건 조금 아쉬웠다.