Machine Learning

HW1

-Linear Regression-

Subject : 기계학습의 기초및응용
Professor : Kye-haeng Lee
Name : Dongho Bang
Student Number : 32181995
Data : 2022.09.28

Contents

1. HW Introduction

A two-column dataset file, data_hw1.csv, is provided. The first column is denoted as x, and the other one as y. The main purpose of this assignment is to write a program that performs linear regression on the given dataset with different models and approaches.

This assignment has two tasks, each of which uses different models and approaches as follows:

Task 1.

Model : y = ax + b

Approach : Gradient Descent

Task2.

Model : $y = ax^2 + bx + c$

Approach : Nomal Equation

2. Motivation

This assignment will give me a deeper understanding of linear regression.

3. Idea Sketch

1. Build Hypothesis (Assumption)
   - Ex) Seems to be a linear relationship between size and price.

Described as $h_\theta(x) = \theta_0 + \theta_1 x$ (x : size, $h_\theta(x)$: price)

2. Train the model (Optimization)

- Finds the best θ_0 and θ_1 values

3. Make prediction with the trained model

4. Concept

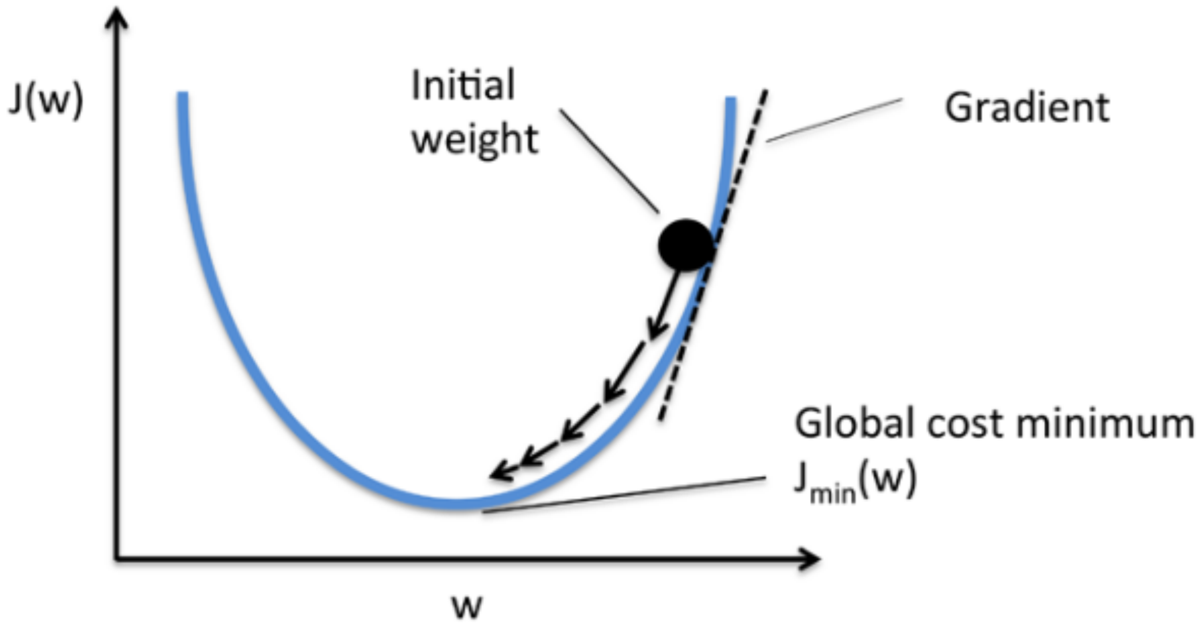
(1) Mean Squared Squared Error (MSE)

평균제곱오차(MSE)는 Regression과 같은 실수 기반의 결과에 대한 오차를 판별하는 방식이다.

Ex) MSE = (실제값 – 예측값)^2/크기

(2) 경사하강법 (Gradient Decent)

손실함수를 미분하면 어떤 특정 지점에서 어느 정도의 기울기가 나오는지 알 수 있다. 이러한 기울기의 절댓값이 작아지는 방향으로 그 지점을 옮기다 보면 손실 함수의 최솟값을 구할 수 있는데, 이 방법을 '경사 하강법'이라고 한다.



(3) 편미분

경사 하강법을 이해하기 위해선 우선 '편미분'에 대해 잘 알아야 한다.

고등학교때까지 배운 미분은 '상미분'으로 변수가 1개만 있는 함수이고, 대학교에와서 인공지능에서 사용하는 함수는 파라미터(매개변수)가 많기 때문에 함수가 매우 복잡하다. 그래서 보통 상미분을 하는 1변수 함수는 거의 없고, 여러 개의 변수를 미분하는 '편미분'이 중요하다. 편미분은 주로 미분하는 변수 하나를 지정하고, 다른 변수는 상수로 다룬다.

$$f(x,y) = 2x^2 + xy + 5y$$

$$\frac{\partial f(x,y)}{\partial x} = \frac{\partial(2x^2 + xy + 5y)}{\partial x} = 4x + y$$

$$\frac{\partial f(x,y)}{\partial y} = \frac{\partial(2x^2 + xy + 5y)}{\partial y} = 5$$

따라서 f(x, y)의 경우 x, y 각각을 기준으로 두면 도함수 2개를 계산해 x, y 각각에서의 기울기 2개 (미분계수 2개)를 구할 수 있다.

(4) Least Square Method (최소 제곱법)

최소 제곱법 : 오차의 제곱합이 최소가 되도록하는 추정값을 산출할 때 사용한다.
즉, 오차를 가장 적게하는 근사치를 구하는 것이다.
대부분의 데이터들은 완벽한 그래프를 그리는게 아니라, 오차가 존재하여 불완전한 그래프를 그린다. 그래서 이 점들을 가장 오차 없이 그리는 간단한 그래프를 구하기 위해 근사치를 구하는 것이 목적이다.

* 최소 제곱법으로부터 근사치 유도

오차를 e라고 하면,
오차 : $e_i = y_i - \hat{f}(x_i)$ 이기 때문에
오차의 제곱합$= \sum_{i=1}^{n} e_i^2$ 이다.
가중치(weight): $w = (w_1, w_2, \ldots, w_p)^T$
예측값: $\hat{f}(x_i) = \hat{y}_i = w^T x_i + b$
X는 주어진 입력이고, y는 정답레이블, $\hat{w}$은 우리가 구하려는 근사치이다.

$y = Xw$ <- 이 식을 만족할 만한 근사치를 구해보자.

$\hat{w} = argmin_w \sum_{i=1}^{n} e_i^2 = argmin_w (y - Xw)^T (y - Xw)$

위 식이 의미하는 바는, 오차 e를 가장 작게 만드는 w로 넣은 값을 근사치라고 하겠다라는 의미가 되겠다.

$$\begin{aligned} e &= (y - Xw)^T (y - Xw) \\ &= (y^T - w^T X^T)(y - Xw) \\ &= y^T y - y^T Xw - w^T X^T y + w^T X^T Xw \\ &= y^T y - 2w^T X^T y + w^T X^T Xw \end{aligned}$$

위처럼 변형될 수 있는 이유는 $y^T Xw$가 대칭행렬이라서 이다.

대칭행렬인 이유는 연산 결과가 스칼라(1 by 1)여서 전치행렬이 원래 행렬과 같기 때문

$\frac{\partial e}{\partial w} = \frac{\partial}{\partial w}(y^T y - 2w^T X^T y + w^T X^T Xw)$

$= -2X^T y + 2X^T Xw$

이 식을 만족하는 w를 구하면

$X^T y = X^T Xw$

$w = (X^T X)^{-1} X y^T$ <- 유도 결과

5. How to run

I made my code using 'jupyter notebook'. Therefore, you can check it using 'Jupyter notebook' or 'colaboratory'.

6. A set of snapshots

1 Machin Learning

1.1 HW1 Linear Regression

32181995 방동호

1.1.1 라이브러리 불러오기

```
import random
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

1.1.2 CSV 파일 불러오기

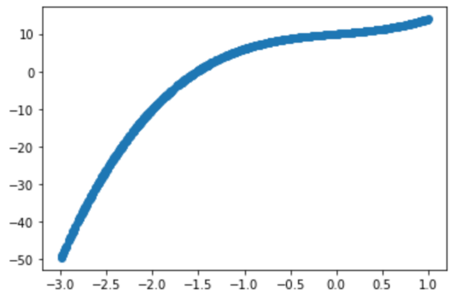
```
data = pd.read_csv('data_hw1.csv')
print(data)
```

```
            x          y
0   -1.572851  -0.927733
1   -2.424197 -23.341094
2    0.402431  10.935209
3   -1.648521  -2.257155
4   -1.899214  -7.499411
..        ...        ...
995 -0.388337   9.106198
996 -1.474044   0.646293
997 -1.799926  -5.262406
998 -1.639442  -2.091766
999  0.675706  11.968438

[1000 rows x 2 columns]
```

1.1.3 데이터 모양 확인하기

```
df = pd.read_csv('data_hw1.csv')

plt.plot(df["x"], df["y"], 'o', label='x')
plt.show
```

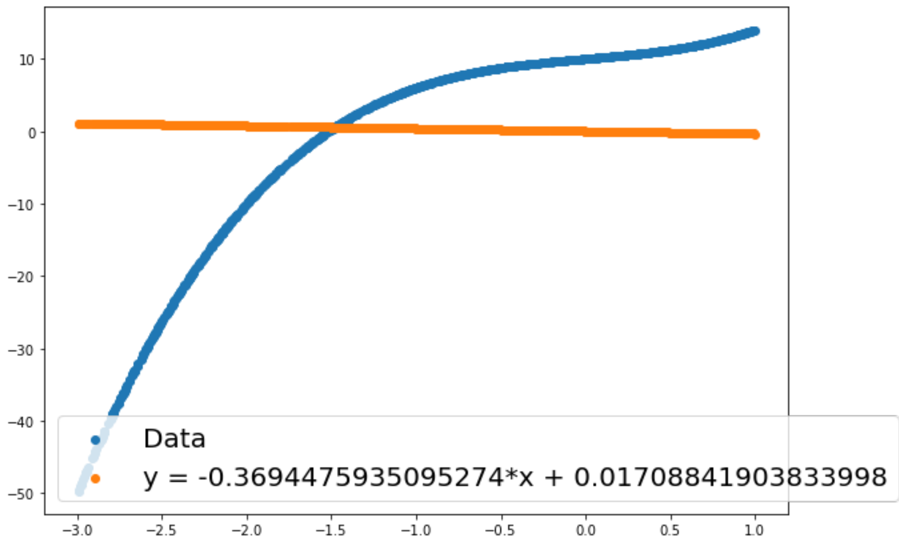
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



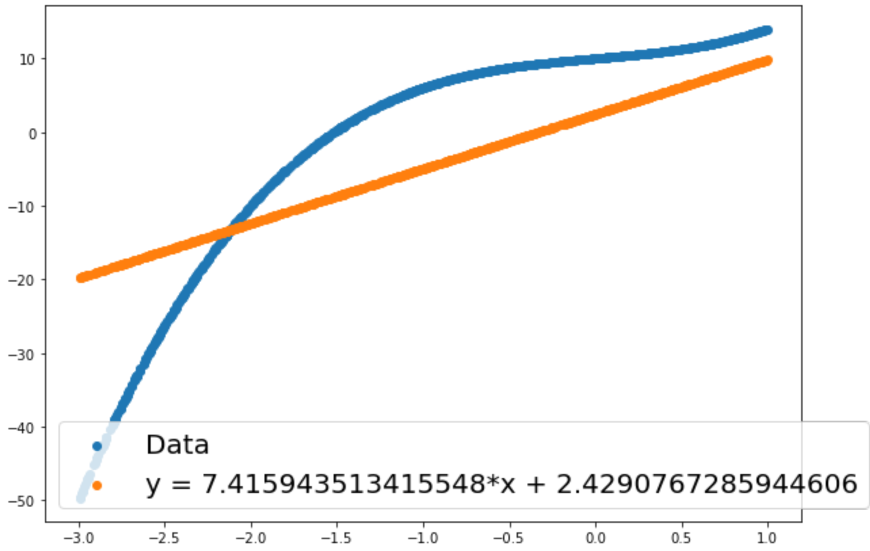
1.2 Task1

```python
# 그래프 출력 함수
def Predic_Model(y_pred, y):
    print('Result is')
    print("[{0}]x + [{1}]".format(a, b))
    plt.figure(figsize = (10, 7))
    plt.scatter(x, y, label='Data')
    plt.scatter(x, y_pred, label=f'y = {a}*x + {b}')
    plt.legend(fontsize=20)
    plt.show()


x = df['x']
y = df['y']

a = np.random.uniform(-1, 1)
b = np.random.uniform(-1, 1)
m = len(df)


learning_rate = 0.01

for epoch in range(m):
    y_pred = a*x + b

    # (2)pdf 10쪽, 25쪽
    MSE = sum(np.square(y_pred - y))/2*m
    if MSE < 10:
        break

    #(2)pdf 44쪽
    a_grad = learning_rate * sum(((y_pred - y)*x))/m
    b_grad = learning_rate * sum((y_pred - y))/m

    a = a - a_grad
    b = b - b_grad

    if epoch % 100 == 0:
        y_pred = a*x + b
        Predic_Model(y_pred, y)
```

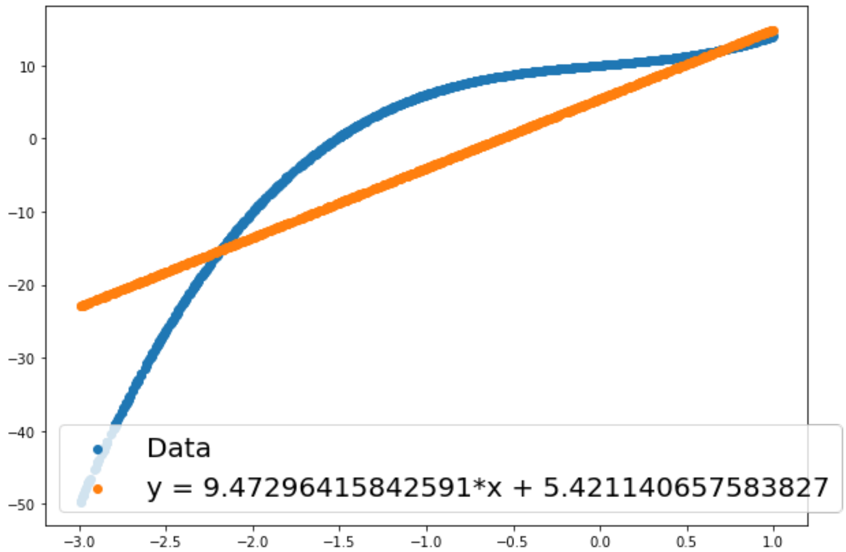
```
Result is
[-0.3694475935095274]x + [0.017088419038339 98]
```



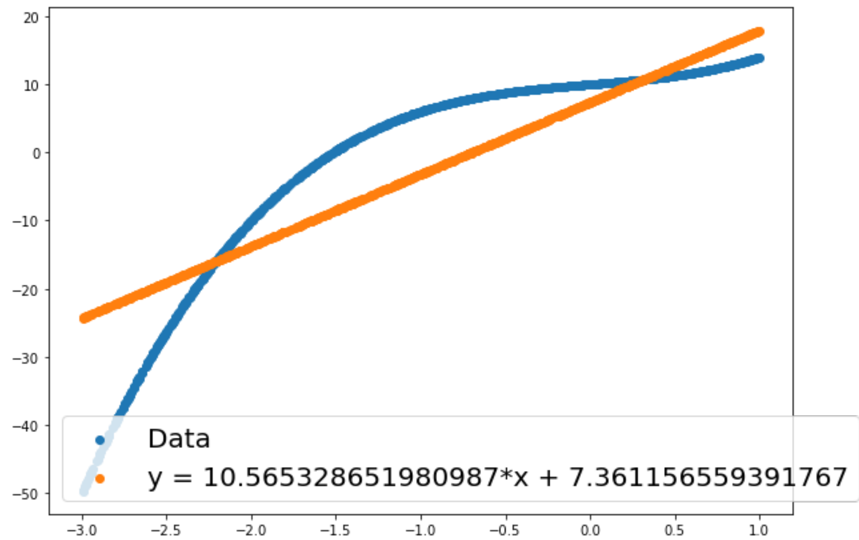

```
Result is
[7.415943513415548]x + [2.4290767285944606]
```



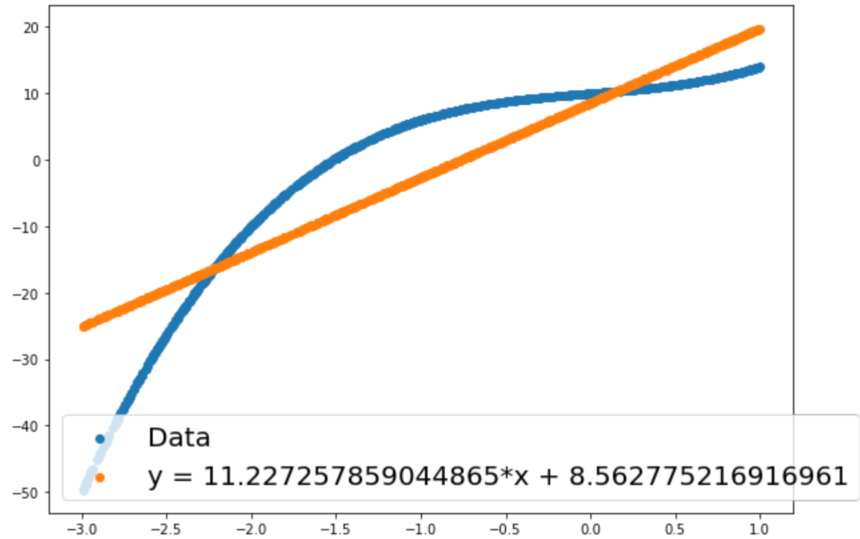

```
Result is
[9.47296415842591]x + [5.421140657583827]
```



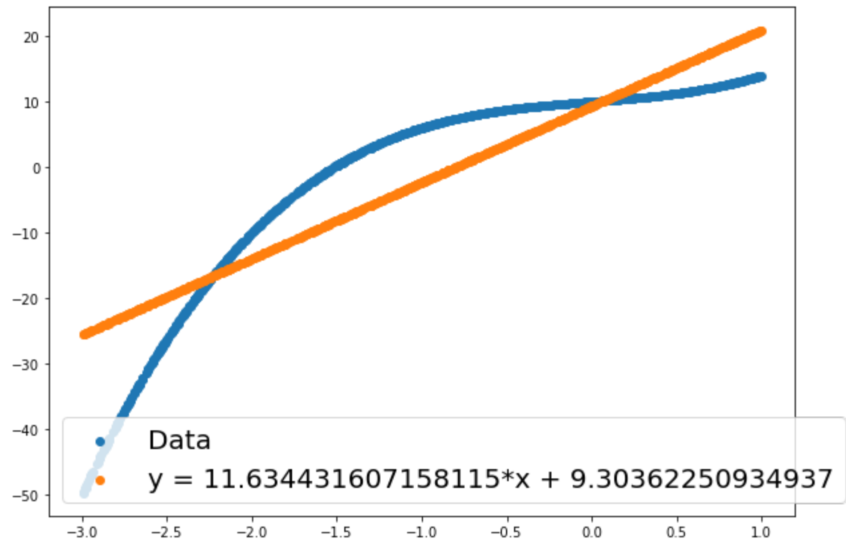

```
Result is
[10.565328651980987]x + [7.361156559391767]
```



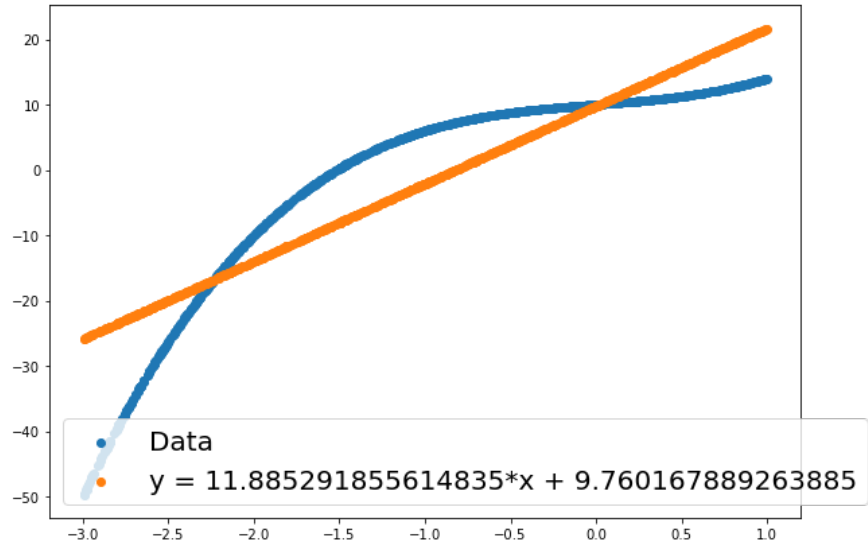

```
Result is
[11.227257859044865]x + [8.562775216916961]
```



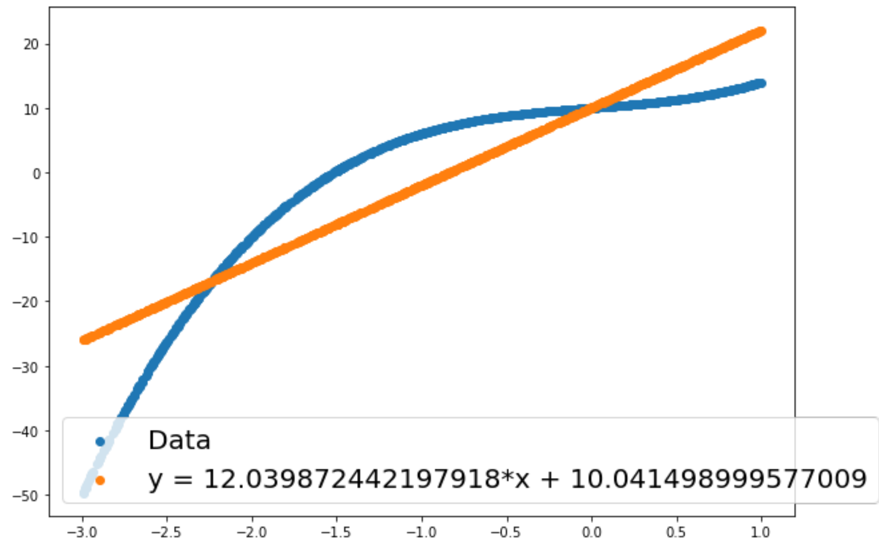

```
Result is
[11.634431607158115]x + [9.30362250934937]
```



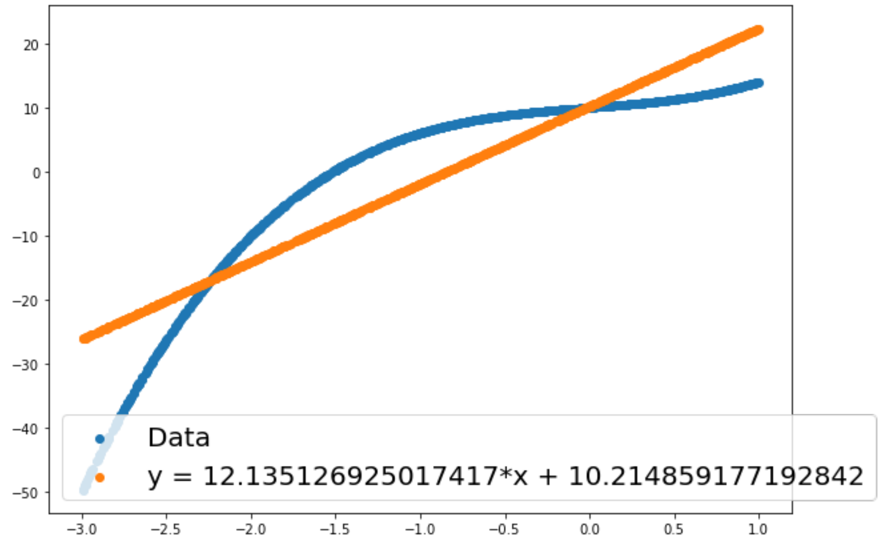
Result is
[11.885291855614835]x + [9.760167889263885]



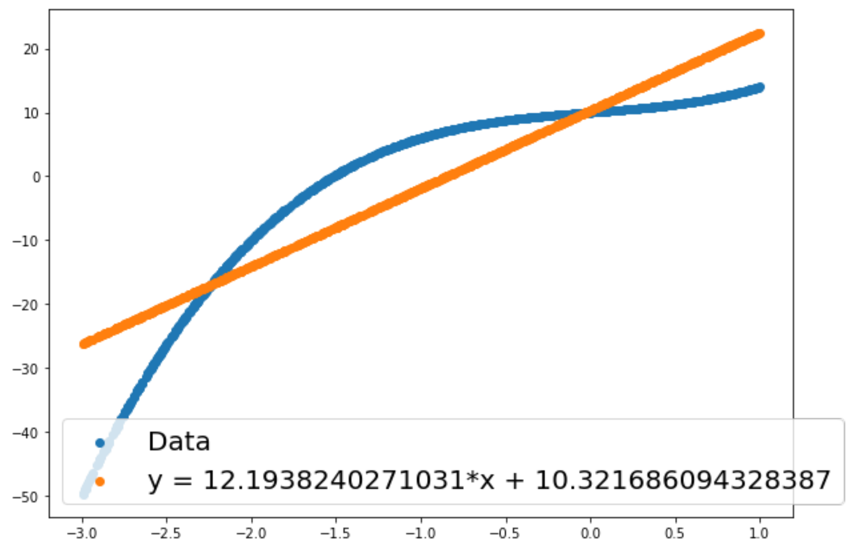
Result is
[12.039872442197918]x + [10.041498999577009]



```
Result is
[12.135126925017417]x + [10.214859177192842]
```



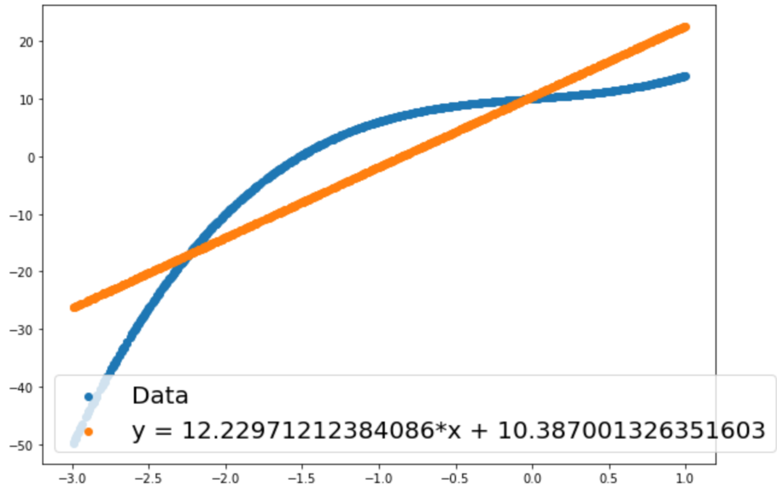

```
Result is
[12.1938240271031]x + [10.321686094328387]
```



1.2.1 Result of Task1

```
Predic_Model(y_pred, y)
print('a: ', a, 'b: ', b)
```

```
Result is
[12.22971212384086]x + [10.387001326351603]
```




```
a:  12.22971212384086 b:  10.387001326351603
```

1.3 Task2

- Model : y = ax^2 + b*x + c
- Approach : Normal Equation

1.3.1 y = *ax^2* + *bx* + c 에서

- a = theta2[2]
- b = theta2[1]
- c = theta2[0]

1.3.2 행렬에서 연산하는 순서

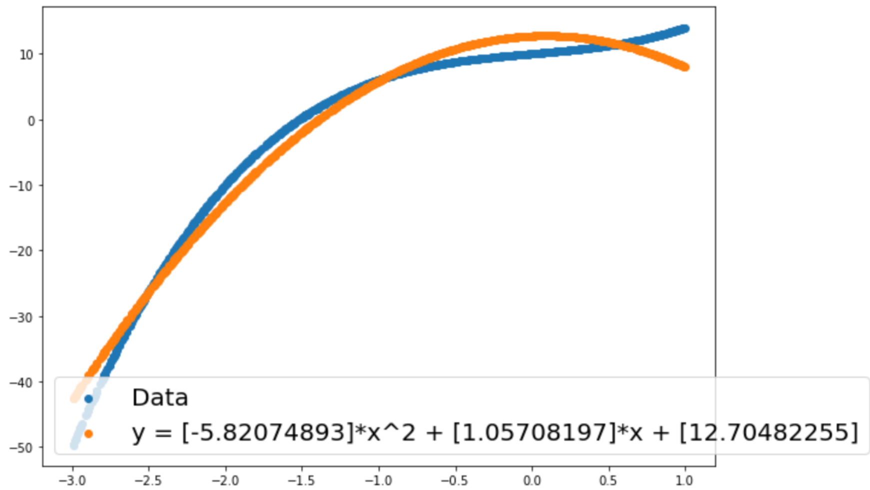
- X_transpose = X^T
- X_Transpose_dot = X^T * X
- Inverse_X_dot_X_1 = (X^T * X)-1 (역행렬)
- Inverse_X_dot_X_T = {(X^T X)-1} X^T
- theta = [{(X^T X)-1} X^T] * y

```python
# 그래프 출력 함수
def Predic_Model(y_pred, y):
    plt.figure(figsize = (10, 7))
    plt.scatter(x, y, label='Data')
    plt.scatter(x, y_pred, label=f'y = {a}*x^2 + {b}*x + {c}')
    plt.legend(fontsize=20)
    plt.show()


x = df['x']
y = df['y']

X = pd.DataFrame({'1': 1, '2': df['x'], '3': (df['x'])**2})

b = pd.DataFrame({'1': df['y']})

# 괄호 안 계산
pred_theta1 = (X.T).dot(X)
# 역행렬 취해줌
pred_theta1_reverse = np.linalg.inv(pred_theta1)
# A의 전치행렬
pred_theta2 = X.T
# b
pred_theta3 = b

theta1 = (pred_theta1_reverse).dot(pred_theta2)
theta2 = theta1.dot(b)

# print(temp2)

# a,b,c의 결과값
a = theta2[2]
b = theta2[1]
c = theta2[0]

y_pred = a*x*x + b*x + c
```

1.3.3 Result of Task2

```python
print('Result is')
print("[{0}]x^2 + [{1}]x + [{2}]".format(a, b, c))
Predic_Model(y_pred, y)
print('a: ', a, 'b: ', b, 'c: ', c)
```

```
Result is
[[-5.82074893]]x^2 + [[1.05708197]]x + [[12.70482255]]
```




```
a:  [-5.82074893] b:  [1.05708197] c:  [12.70482255]
```

2 Comparison

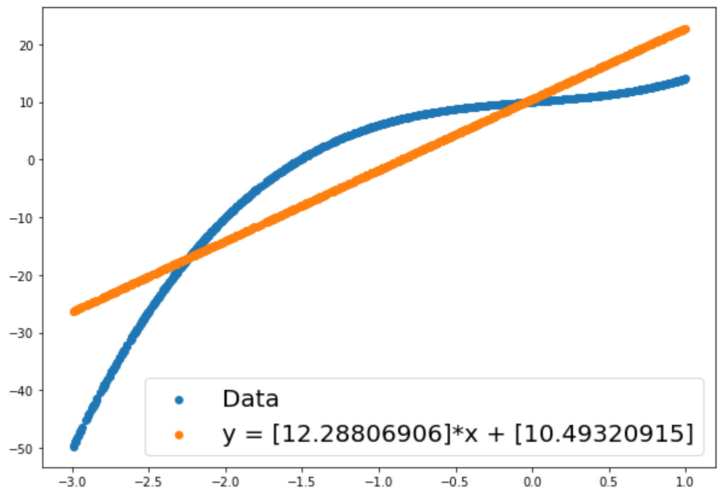
y = ax+b in normal equation

```
# 그래프 출력 함수
def Predic_Model(y_pred, y):
    plt.figure(figsize = (10, 7))
    plt.scatter(x, y, label='Data')
    plt.scatter(x, y_pred, label=f'y = {a}*x + {b}')
    plt.legend(fontsize=20)
    plt.show()


x = df['x']
y = df['y']

X = pd.DataFrame({'1': 1, '2': df['x']})

b = pd.DataFrame({'1': df['y']})

# 괄호 안 계산
pred_theta1 = (X.T).dot(X)
# 역행렬 취해줌
pred_theta1_reverse = np.linalg.inv(pred_theta1)
# A의 전치행렬
pred_theta2 = X.T
# b
pred_theta3 = b

theta1 = (pred_theta1_reverse).dot(pred_theta2)
theta2 = theta1.dot(b)

# print(temp2)

# a,b,c의 결과값
a = theta2[1]
b = theta2[0]

y_pred = a*x + b
```

2.0.1 Result of Comparison

```
print('Result is')
print("[{0}]x + [{1}]".format(a, b))
Predic_Model(y_pred, y)
print('a: ', a, 'b: ', b)
```

```
Result is
[[12.28806906]]x + [[10.49320915]]
```

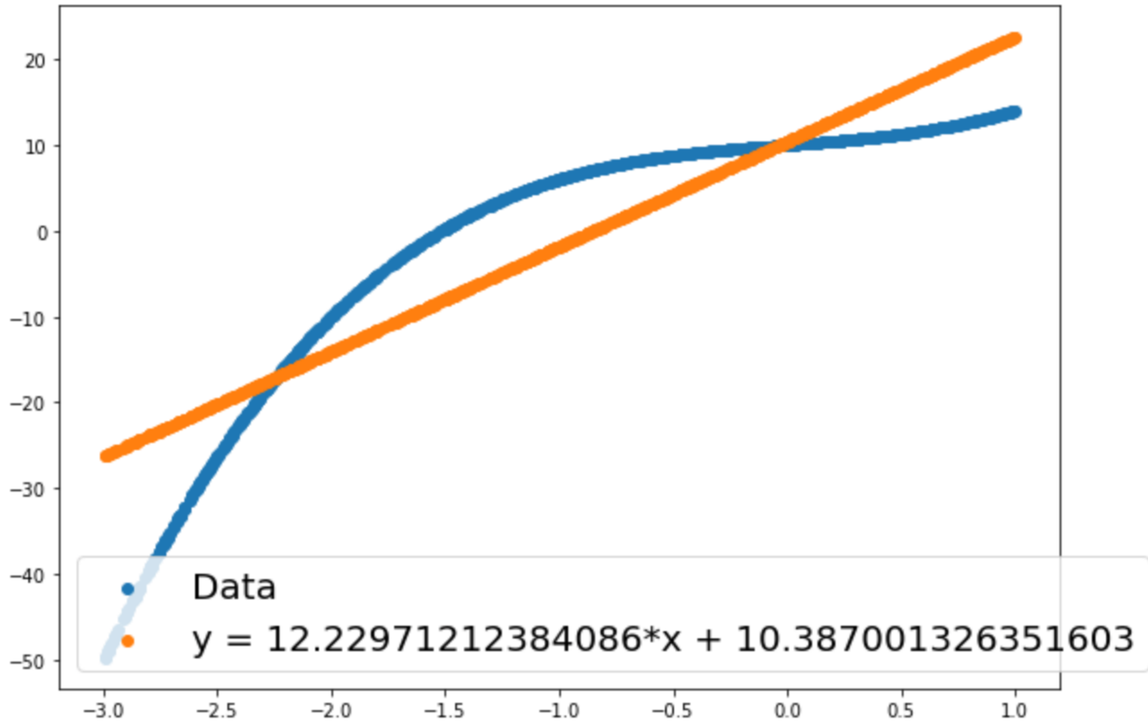



```
a:  [12.28806906] b:  [10.49320915]
```

7. Result

Task1

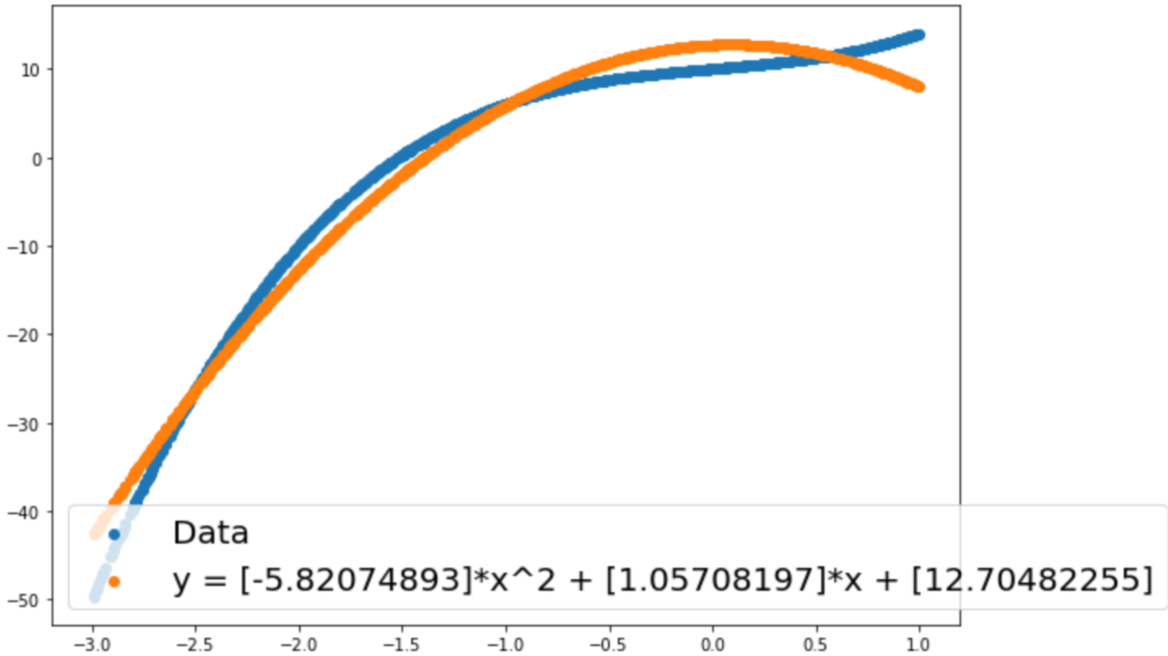
```
Result is
[12.22971212384086]x + [10.387001326351603]
```




```
a:  12.22971212384086 b:  10.387001326351603
```

Task2

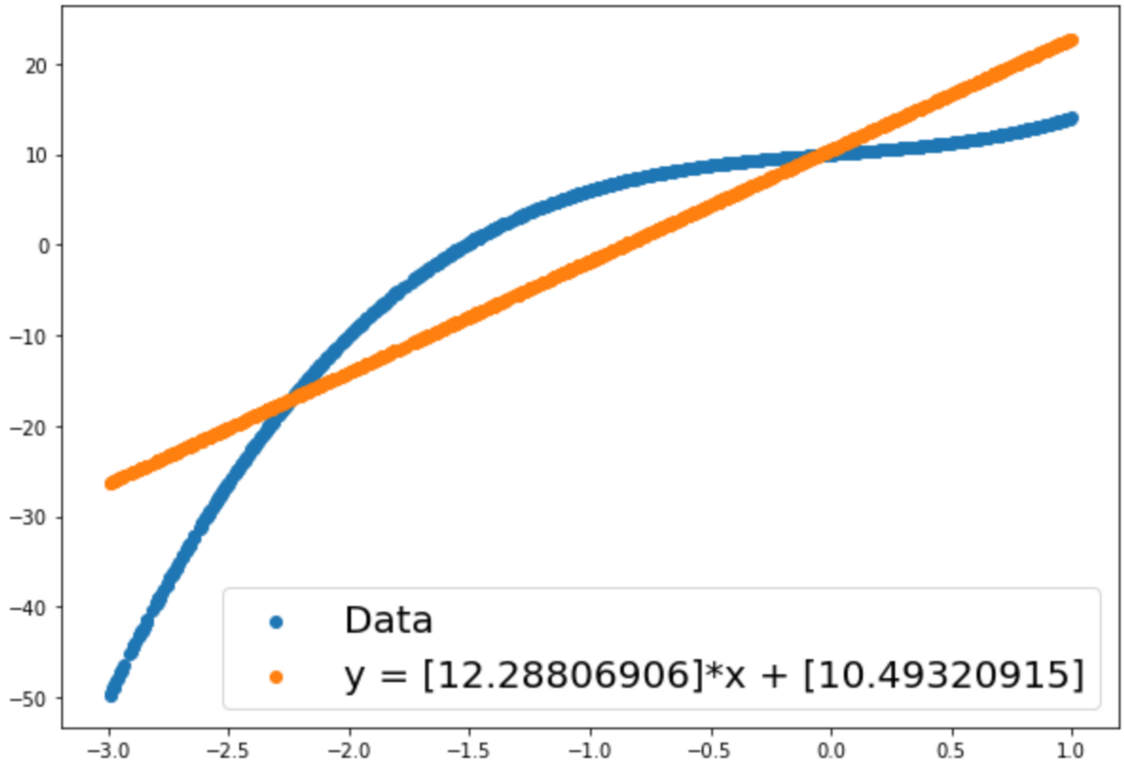
```
Result is
[[-5.82074893]]x^2 + [[1.05708197]]x + [[12.70482255]]
```




```
a:  [-5.82074893] b:  [1.05708197] c:  [12.70482255]
```

Comparison

```
Result is
[[12.28806906]]x + [[10.49320915]]
```




```
a:  [12.28806906] b:  [10.49320915]
```

Table of parameters

	A	B	C
Task1	12.22971212384086	10.387001326351603	x
Task2	-5.82074893	1.05708197	12.70482255
Comparison	12.28806906	10.49320915	x

8. Conclusion

(1) Gradient Descent

- Neet to choose $'\alpha'$.
- Needs many iterations.
- Works well even when 'n' is large.

(2) Normal Equation

- No need to choose $'\alpha'$.
- Don't need to iterate.
- Need to compute $(X^T X)^{-1}$.
- Slow if 'n' is very large.