HW#1
: Linear Regression

제출일 : 2023-10-15
과목명 : 기계학습의 기초 및 응용
담당교수 : 이규행
전공 : 모바일 시스템 공학과
학번 : 32191097
이름 : 김준형

Introduction

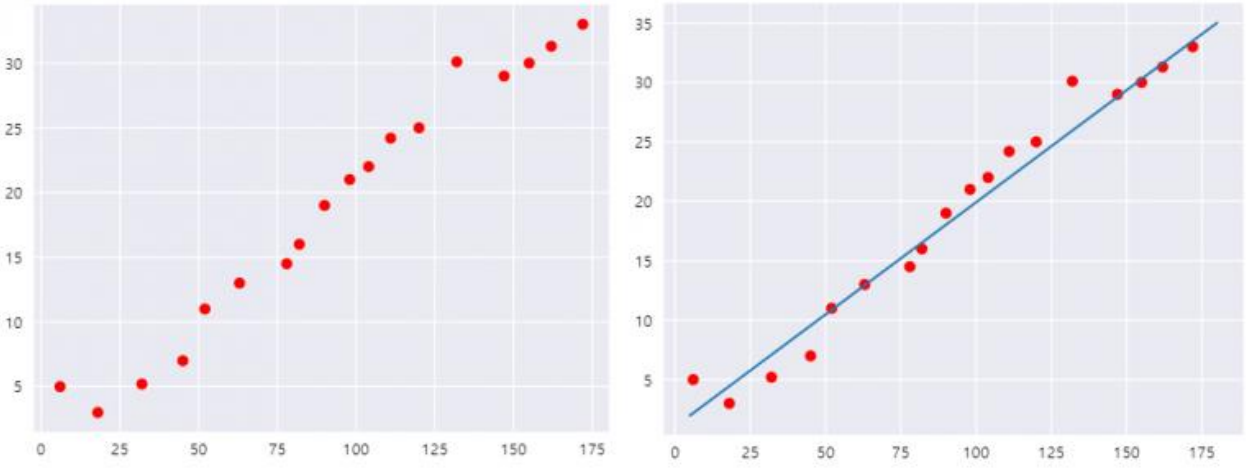
이번 과제는 x, y값에 대한 데이터가 저장되어 있는 csv파일을 읽어 경사 하강법과 정규 방정식을 통한 선형 회귀를 구현하는 것이다.

이 과제를 수행하며 학습률에 따라 지속적으로 변동하는 a와 b값을 확인하기 위해 일정한 학습 단계마다 a, b값을 출력하도록 했으며 모든 학습이 끝난 이후 matplotlib 라이브러리를 통해 그래프 위에 데이터셋과 선형회귀의 결과를 출력하도록 했다. matplotlib는 파이썬의 라이브러리 중 하나로 주어진 데이터를 차트, 그래프 등으로 시각화 해주는 라이브러리이다. 그 중 이번 과제에서는 matplotlib의 기능 중 하나인 라인플롯을 통해 그래프로 나타냈다.

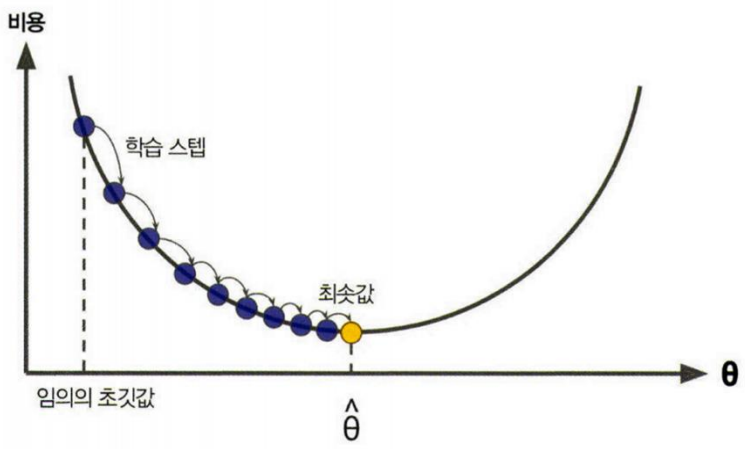
또한 정규 방정식의 행렬 연산을 위해 numpy 라이브러리를 사용했으며 이 결과 또한 matplot 라이브러리를 통해 그래프 위에 동시에 나타내었다. numpy는 다차원 배열의 연산에 특화된 라이브러리이다. numpy는 ndarray라는 객체를 지원하며 해당 객체는 inverse, transpose와 같은 다양한 배열 연산들을 지원한다. 이 과제에서는 정규 방정식의 공식에 사용되는 역행렬, 전치행렬, 행렬곱 연산을 위해 numpy 라이브러리를 사용했다.

Background

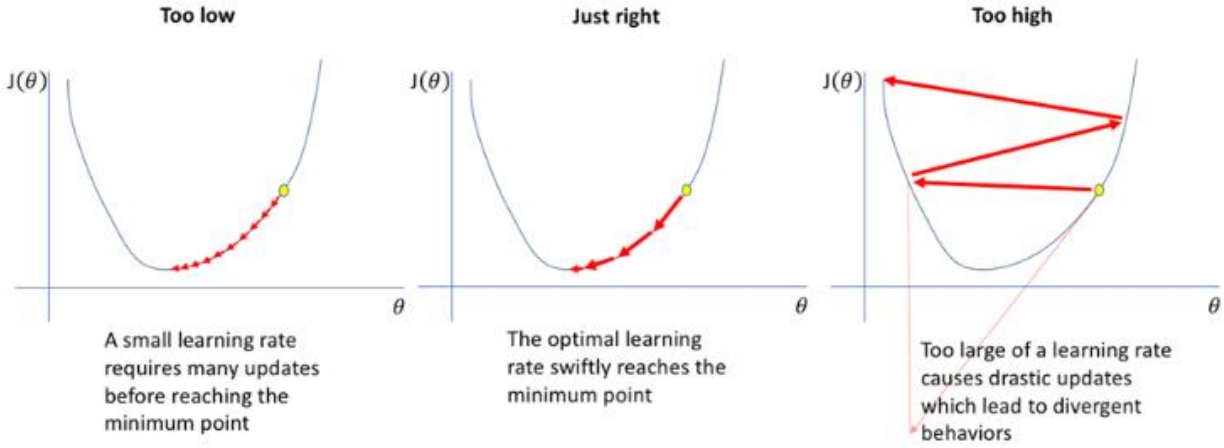
선형 회귀란 종속 변수와 해당 종속 변수에 영향을 주는 한 개 이상의 독립변수간 관계를 선형 함수로 나타내는 회귀분석 방법이다. 이 때 주어진 독립변수가 단 한개라면 단순 선형 회귀, 그 이상일 경우 다중 선형 회귀라고 한다. 즉 주어진 데이터를 가장 잘 나타내는 함수를 추정하고 이후 임의의 독립변수 값이 주어졌을 때 해당 값을 통해 나타날 종속 변수 값을 추정하는 과정이라 할 수 있다. 선형 회귀에는 다양한 방법이 있지만 이 과제에서는 경사 하강법과 정규 방정식을 사용한다.



경사 하강법이란 1차 미분계수를 통해 오차율의 극소값을 찾는 방법이다. 주어진 함수를 미분하면 그 함수에 대한 기울기 값을 알 수 있고 기울기가 0이 되는 지점의 값을 극값이라고 말한다. 이 때 cost function(MSE)를 각 가중치에 대하여 미분한 도함수의 값이 0에 수렴하도록 조절해 나가면 cost function의 극소값을 알 수 있다.



이 때 학습률의 정도에 따라 극값으로 수렴하는 속도와 극값의 수렴 여부가 달라지게 되기에 적절한 학습률을 설정하는 것이 중요하다.



가중치를 수정하는 과정은 다음과 같이 진행된다.

$$costfunction = \frac{1}{2m}\sum\left(y_{pred} - y\right)^2$$

$$w_{n+1} = w_n - \alpha\nabla f(w_n)$$

$$\nabla f(w_n) = \{(y_{pred} - y)x^0, (y_{pred} - y)x^1, \dots, (y_{pred} - y)x^i\}$$

이번 과제에서 경사 하강법의 목표값은 $ax + b$ 이므로 각각의 가중치들은

$$a = a - (y_{pred} - y)x$$

$$b = b - (y_{pred} - y)$$

의 식을 통해 수정된다.

정규 방정식이란 행렬 연산을 통해 cost function(MSE)의 최소값을 찾는 방법이다. 1차 미분 계수를 통해 지속적으로 가중치를 변화시켜나가는 경사하강법과 달리 정규방정식은 주어진 데이터에 대한 행렬 연산을 통해 바로 목표로 하는 가중치를 찾을 수 있다.

이 과제에서의 목표값은 $ax^2 + bx + c$이다. 이 식을 풀어서 정리하면 다음과 같이 정리된다.

$$c\sum 1 + b\sum x + a\sum x\text{^}2 = \sum y$$

$$c\sum x + b\sum x^2 + a\sum x^3 = \sum xy$$

$$c\sum x^2 + b\sum x^3 + a\sum x^4 = \sum x^2 y$$

이를 행렬로 표현하면

$$\begin{pmatrix} \sum 1 & \sum x & \sum x^2 \\ \sum x & \sum x^2 & \sum x^3 \\ \sum x^2 & \sum x^3 & \sum x^4 \end{pmatrix} \begin{pmatrix} c \\ b \\ a \end{pmatrix} = (X^T X) \begin{pmatrix} c \\ b \\ a \end{pmatrix} = X^T Y$$

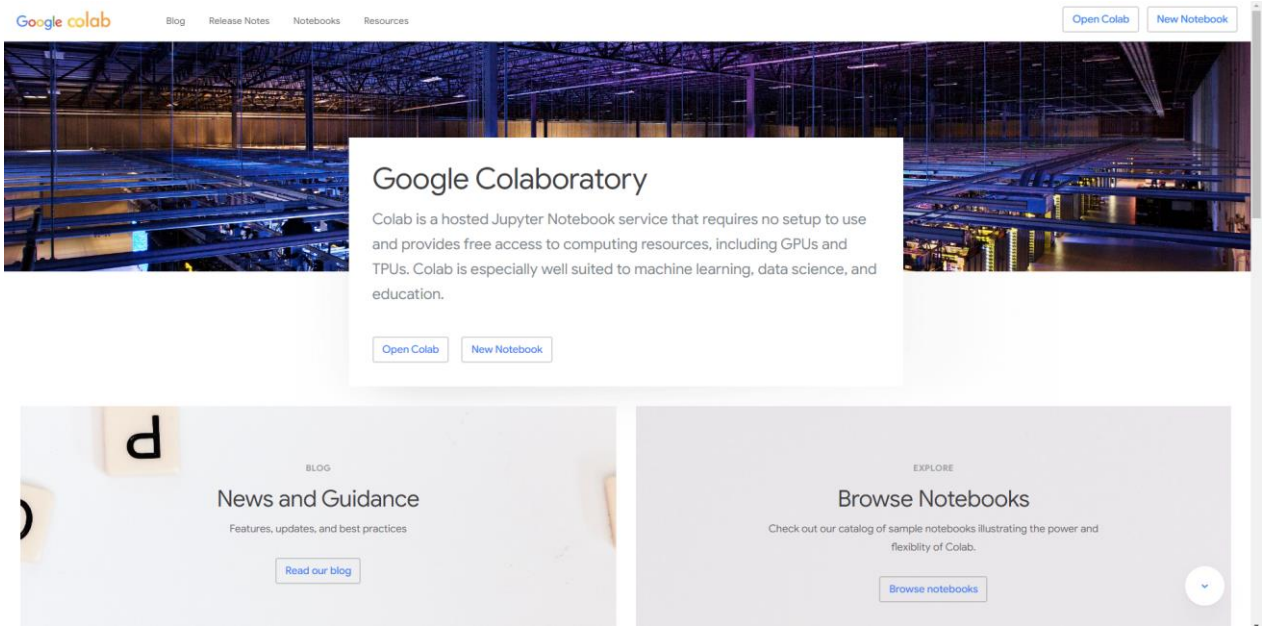
로 나타나며 이를 정리하면 다음과 같이 나타난다.

$$\begin{pmatrix} c \\ b \\ a \end{pmatrix} = (X^T X)^{-1} X^T Y$$

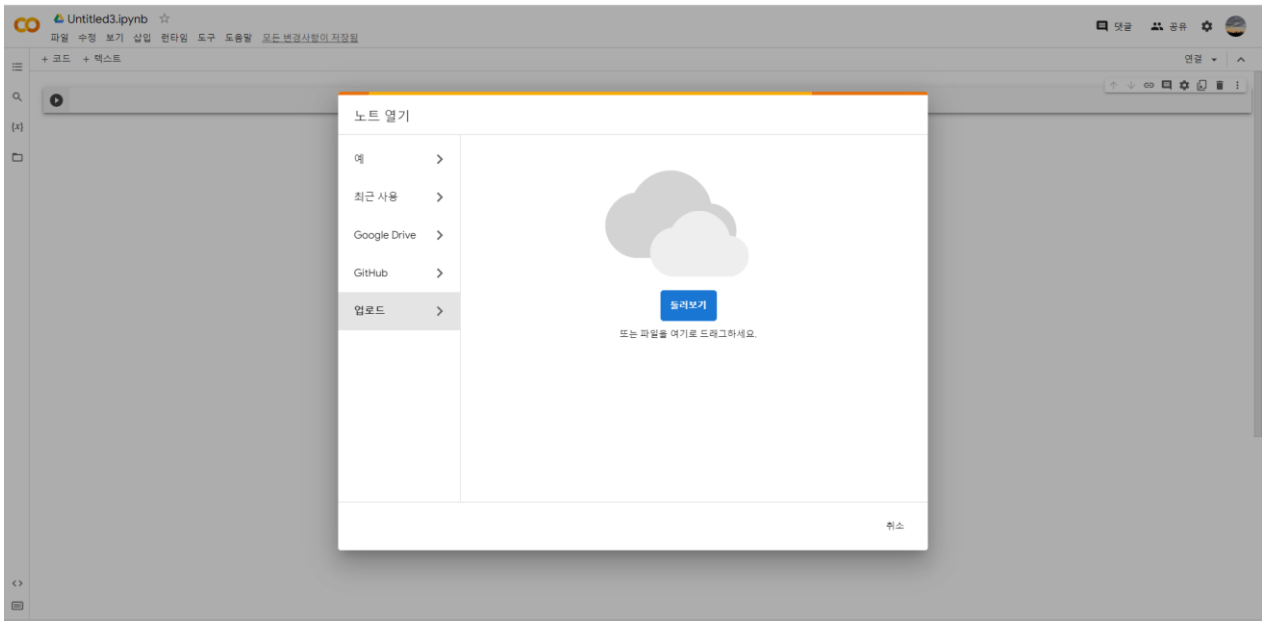
Implementation

이번 과제는 google colab에서 작성했으며 이를 실행하기 위한 몇 가지 전처리 과정이 존재한다.

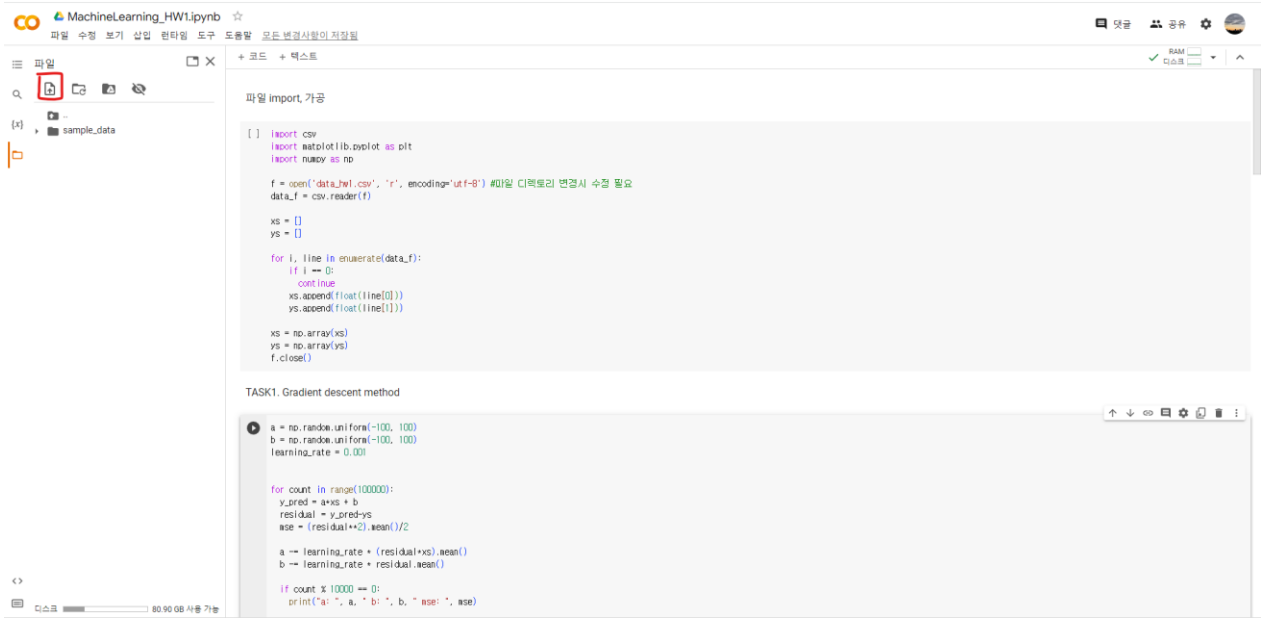
첫 번째로 google colab에 ipynb파일을 업로드한다. Google colab의 초기 화면은 다음과 같다.



이 화면에서 New Notebook을 선택후 파일-노트업로드를 통해 ipynb파일을 업로드한다.



두번째로 csv파일을 업로드한다. 좌측의 파일탭 - 세션 장소에 업로드를 통해 csv파일을 업로드 할 수 있다.



이 때 csv파일은 폴더의 최외곽 디렉토리에 존재해야 하며 만일 따로 디렉토리를 생성하여 csv파일을 저장할 경우 코드 탭의 파일 경로를 수정해 주어야 한다. 이후 colab 상단의 런타임 – 모두실행 버튼을 통해 코드를 실행하면 된다.

구현된 코드는 3가지 단계로 나누어진다.

첫번째 단계는 데이터 전처리 및 라이브러리 import 과정이다.

```
import csv
import matplotlib.pyplot as plt
import numpy as np
```

Impot한 라이브러리로는 csv파일을 쉽게 가공하기 위한 csv라이브러리, introduction에서 설명한 matplotlib, numpy 라이브러리가 있다. 이때 colab은 기본적으로 위의 라이브러리들을 지원해주기에 pip install과정을 통해 라이브러리를 설치할 필요 없이 사용할 수 있다.

```
f = open('data_hw1.csv', 'r', encoding='utf-8') #파일 디렉토리 변경시 수정 필요
data_f = csv.reader(f)

xs = []
ys = []

for i, line in enumerate(data_f):
    if i == 0:
      continue
    xs.append(float(line[0]))
    ys.append(float(line[1]))

xs = np.array(xs)
ys = np.array(ys)
f.close()
```

라이브러리를 import한 후 data_hw1.csv파일 open함수를 통해 파일 객체를 받아오며 csv.reader() 메소드를 통해 iterable한 객체를 반환받는다. 이후 반환받은 객체 내부 데이터를 반복문을 통해 xs, ys 리스트에 넣는데 이 때 csv파일의 첫 줄은 x y로 데이터 값이 아니기에 제외한다. 마지막으로 xs와 ys 리스트를 행렬 연산을 위해 ndarray객체로 변환한다.

두번째 단계는 task1. 경사 하강법 단계이다.

```
a = np.random.uniform(-100, 100)
b = np.random.uniform(-100, 100)
learning_rate = 0.001
```

경사 하강법은 초기에 설정한 가중치 값을 반복을 통해 수정해 나가는 방식이기에 초기 가중치 a, b를 -100부터 100사이의 임의 값으로 설정한다. 또한 학습률은 0.001로 설정하였는데 학습률이 0.7 이상의 높은 값으로 설정하게 되면 가중치가 발산하게 되어 정상적인 값을 반환받지 못하게 된다.

```
for count in range(100000):
  y_pred = a*xs + b
  residual = y_pred-ys
  mse = (residual**2).mean()/2

  a -= learning_rate * (residual*xs).mean()
  b -= learning_rate * residual.mean()

  if count % 10000 == 0:
    print("a: ", a, " b: ", b, " mse: ", mse)
```

이후 가중치 값을 수정하기 위해 필요한 값인 $y_{pred} - y$ (잔차, residual)을 통해 가중치를 수정한다. 이때

$$f(w_n) = \frac{1}{2m}\sum\left(y_{pred} - y\right)^2 = \frac{1}{2m}\sum\left(a^i x^i + b^i\right)^2$$

$$\nabla f(w_n) = \{(ax + b - y)x, (ax + b - y)\} = \left\{\left(y_{pred} - y\right)x, \left(y_{pred} - y\right)\right\}$$

이므로 a는 residual*xs의 평균값, b는 residual의 평균값만큼 수정하게 된다.

세번째 단계는 task2. 정규 방정식 단계이다.

```
def normal_equation(X, Y):
  return np.linalg.inv(np.dot(X.T, X)).dot(X.T).dot(Y)
```

정규 방정식의 수식은 다음과 같다.

$$(X^T X)^{-1} X^T Y$$

이를 구현하기 위해 numpy 라이브러리의 메소드 들을 사용했다. 각 메소드는 다음과 같다.

np.linalg.inv(ndarray): input으로 받은 ndarray객체를 역행렬로 변환하여 반환한다.

np.dot(ndarray, ndarray): input으로 받은 두 ndarray객체를 행렬곱하여 반환한다.

ndarray.dot(ndarray): input으로 받은 ndarray객체와 행렬곱을 하여 반환한다.

ndarray.T: 해당 ndarray 객체의 전치행렬을 반환한다.

```
new_xs = np.c_[(xs.T)**2, xs, np.ones((len(xs), 1))]
new_ys = ys.T

weight = normal_equation(new_xs, new_ys)
print(weight)

new_y_pred = np.dot(weight, new_xs.T)
new_residual = new_y_pred - ys
print((new_residual**2).mean()/2)
```

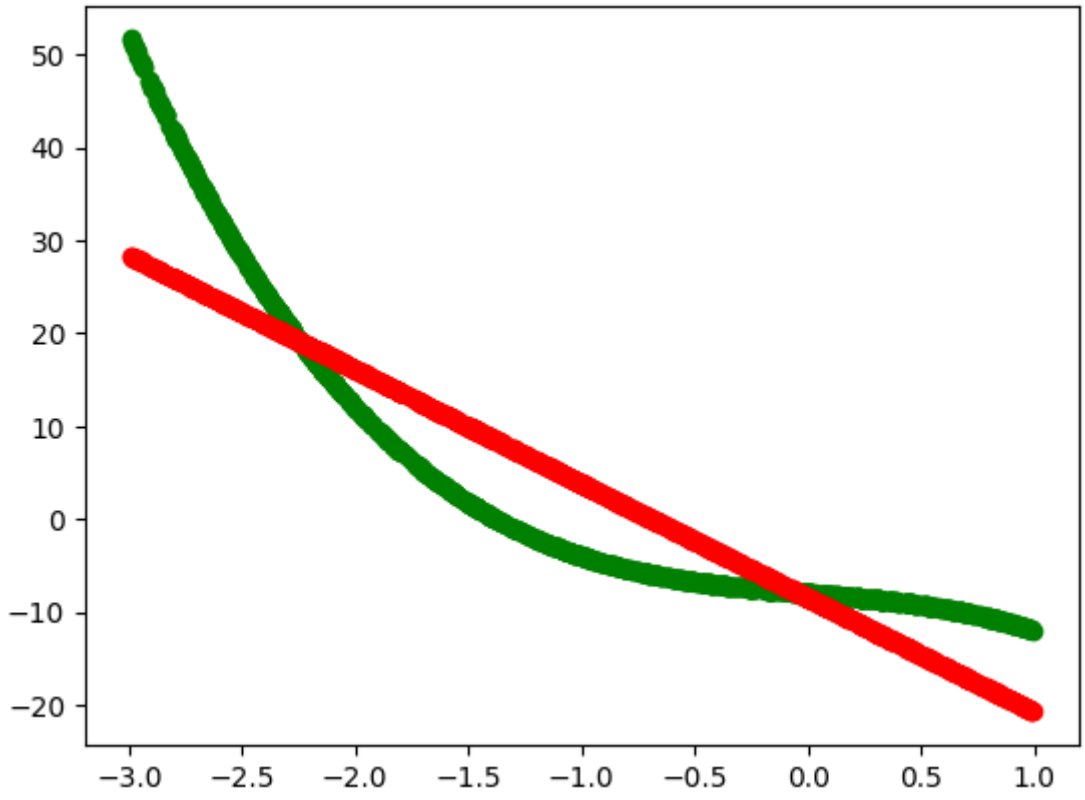
이번 과제에서의 목표값은 $ax^2 + bx + c$이므로 new_xs = $((x^2)^T \quad (x^1)^T \quad (x^0)^T)$, new_ys = y^T를 만들어 정규 방정식 함수에 입력해 가중치 배열을 반환받는다.

Result

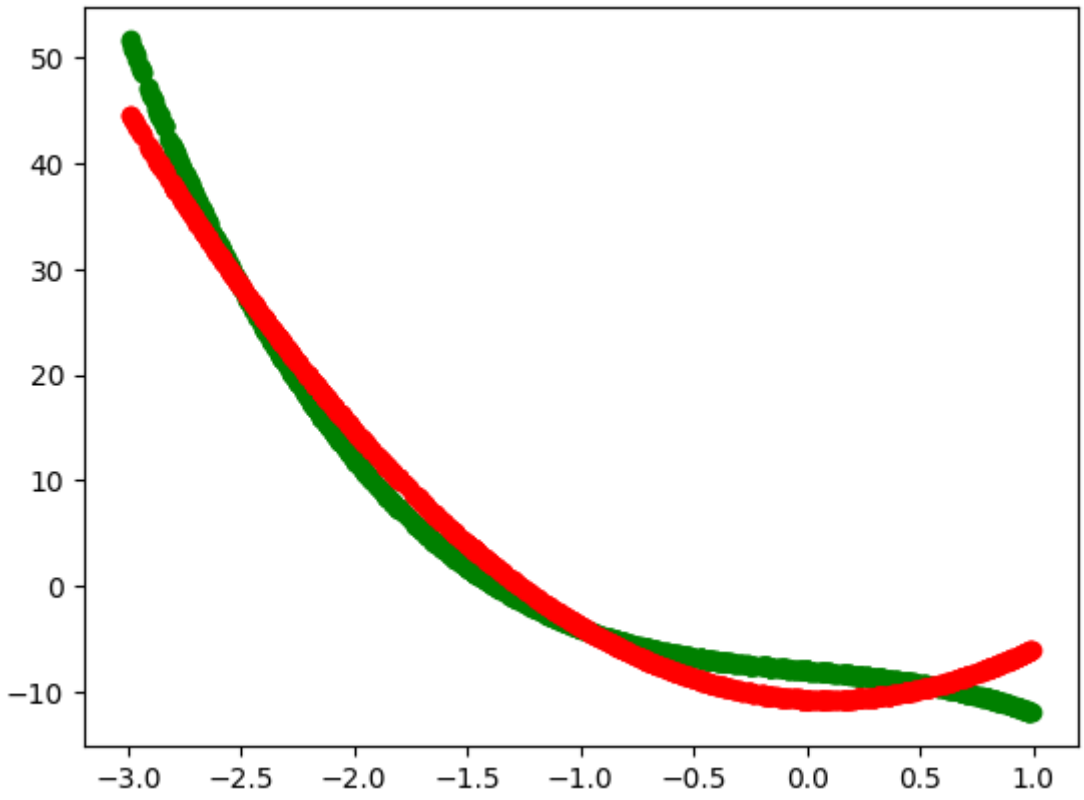
결과는 다음과 같다.

	a	b	c
Task1	-12.28806905988156	-8.49320915497584	0
Task2	5.82074893	-1.05708197	-10.70482255

Task1. Gradient descent method



Task2. Normal equation method



Conclusion

선형 회귀는 데이터를 분석, 예측하는 방법 중 하나로 주어진 데이터셋을 가장 잘 표현하는 함수를 찾는 회귀 분석 방법이다. 선형 회귀를 통한 예측 값과 실제 값의 오차를 최소화하는 것이 중요하다. 오차를 줄이기 위한 방법으로 이번 과제에서는 경사 하강법과 정규 방정식을 사용했으며 각각의 방법에서 평균제곱오차(MSE, $\frac{1}{2m}\sum(y_{pred}-y)^2$은 각각 25.8, 2.74의 값에 수렴했다.

이를 통해 알 수 있는 것으론 초기에 데이터 셋의 함수 형태를 적절히 예측하는 것이 중요하다는 것이다. 경사 하강법에서는 $ax+b$ 형태의 함수를 가정하고 진행했고 정규방정식에서는 ax^2+bx+c 형태의 함수를 가정하고 진행했으며 각 방식의 평균제곱오차 값을 통해 이 데이터 셋은 $ax+b$ 보다 ax^2+bx+c에 더 유사한 형태의 함수를 갖는 것을 알 수 있다. 이 외에 ax^3+bx^2+cx+d형태의 함수를 가정하고 정규 방정식을 진행했더니 2.3894833143950713^{-17}의 아주 작은 값의 오차를 가지며 두 함수가 거의 겹치는 것을 확인할 수 있었다.