Report

HW#1 Linear Regression

Machine Learning

Dpt. of Mobile System Engineering
32192530 양윤성

2023.10.14.

● Contents

1. Introduction

1-1. Linear Regression

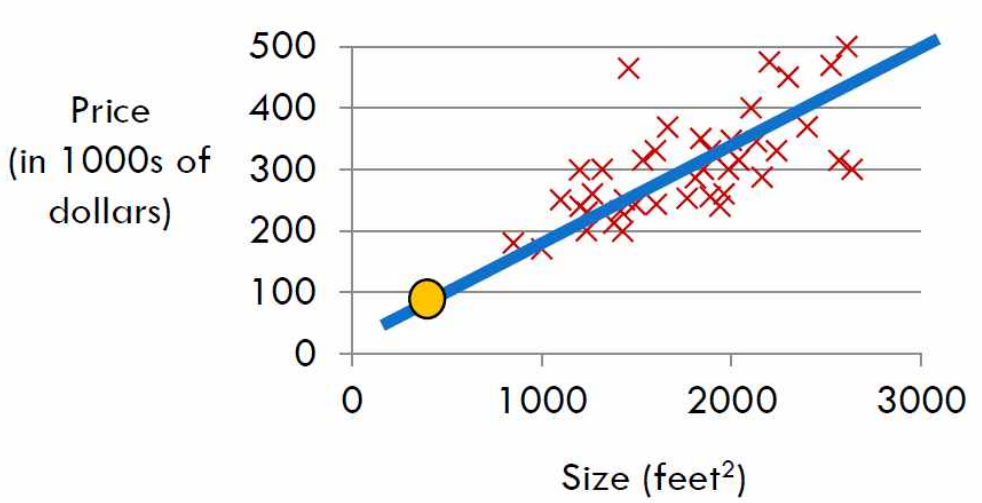

그림1. Housing Price Prediction을 위한 Linear Regression 예시

선형 회귀(Linear Regression)는 머신 러닝에서 사용하는 대표적인 회귀 분석 기법 중 하나로 데이터들로 특정 값을 예측하기 위함이다. 우리가 예측하고자 하는 변수를 종속변수라고 하며, 예측을 위해 필요로 하는 변수를 독립변수라고 한다. 이 둘 사이에 선형성이 만족하는 상황에서 Linear Regression을 효과적으로 사용할 수 있다. 선형성(Linearity)은 입력값 x, y가 있고 출력값 f(x), f(y)가 존재할 때, 가산성(Additivity)와 동차성(Homogeneity)을 동시에 만족하는 상황을 의미한다. 가산성은 x, y에 대해 f(x) + f(y) = f(x+y)를 만족하는 성질이고, 동차성은 임의의 스칼라값 a가 있다고 할 때 x와 a에 대해 af(x) = f(ax)를 만족하는 성질이다. 그러면 스칼라값 a, b가 있다고 할 때 af(x) + bf(y) = f(ax + by)를 만족하면 이 함수는 선형성을 만족한다고 한다. 이런 상황을 선형 회귀 모델에서는 종속변수를 y로, 독립변수를 x로 생각하면 되고 그럼 나머지 a와 b가 있을 것이다. 데이터들의 패턴을 파악해서 최적의 a와 b값을 찾아내야만 정확한 결과를 예측할 수 있고 이것이 바로 Linear Regression의 목적이다.

본 레포트에서는 선형 회귀 모델을 구현하기 위해 필요한 몇 가지 개념들을 알아보고, 2가지 방법(Gradient Descent, Least Squares)으로 실제 모델을 구현해 볼 것이다. 구현 후 예시 데이터들로 2가지의 Task를 통해 Optimal한 매개변수를 찾아내고, 마지막에는 결과를 분석하며 두 방법이 어떤 차이가 있는지를 확인할 것이다.

1-2. Cost Function

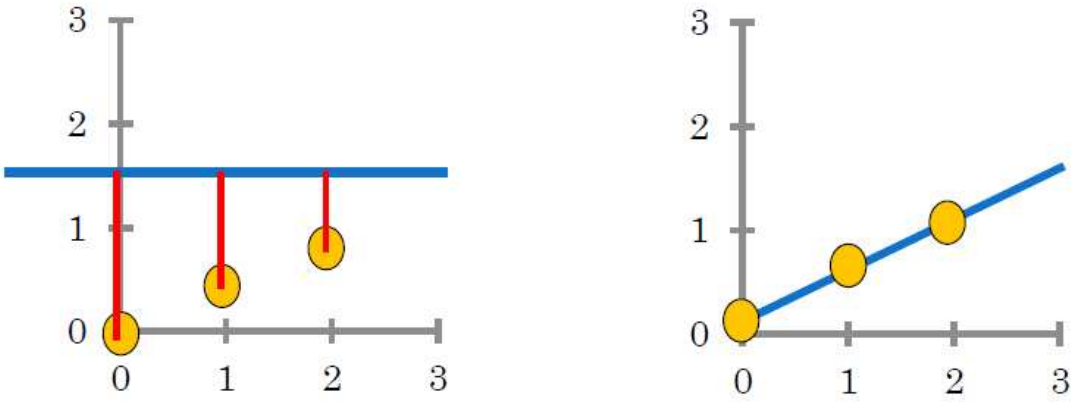

그림2. 예측값과 실제값의 차이 (왼쪽 : 차이가 나는 경우, 오른쪽 : 이상적인 경우)

선형 회귀 모델도 값을 예측하다 보면 왼쪽 그림과 같이 예측한 값과 이상적인 값이 차이가 나는 경우가 있다. 이 둘의 차이를 cost(error)라고 부르며, 이를 구하는 함수가 Cost Function이다. 즉 선형 회귀 분석의 목표는 이 Cost Function의 값이 최소화가 되도록 하는 것이다.

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

그림3. Mean Squared Error

선형 회귀 모델에서 Cost Function은 주로 평균 제곱 오차(Mean Squared Error, MSE)를 사용하여 계산한다. 모델의 예측값과 실제 값의 차이를 측정하며, 이 차이의 제곱을 평균화한 값이다. 정리하자면 선형 회귀 모델의 궁극적인 목적은 Cost Function인 MSE값을 최대한 작게 만드는 것이고 이 값이 최소가 되는 계수를 찾아내야 한다. 이러한 계수들을 찾는 접근 방식으로는 Gradient Descent와 Least Square Method가 있다.

1-3. Least Square Method

최소 제곱법(Least Square Method)은 MSE와 같은 cost function에 유용하게 사용 가능한 방법으로, 아래에서 설명할 Gradient Descent와 다르게 수식적인 접근으로 바로 Global Minimum을 찾을 수 있다는 장점이 있다. Least Square에서는 cost function의 매개변수들에 대해 각각 미분을 진행하여, 미분한 값이 0이 되도록 계산하여 optimal한 매개변수를 찾는다. y=ax+b와 같이 매개변수가 2개고 cost function으로 MSE를 사용한다면 아래와 같이 계산할 수 있다.

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

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

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

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

For θ_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 θ_1

$$\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x^{(i)} = \frac{1}{m}\sum_{i=1}^{m}(\theta_0 x^{(i)}+\theta_1 x^{(i)2}-y^{(i)}x^{(i)}) = 0$$

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

그림4. Least Square Method

$$\theta = (X^T X)^{-1} X^T y$$

그림5. Normal Equation

만약 위의 Least Square 수식을 행렬로 정리한다면, 그림 5와 같이 정형화된 식을 얻을 수 있다. 그림 5에서 세타 행렬이 바로 매개변수가 들어있는 행렬이라 볼 수 있다. 단 Normal Equation은 독립변수인 X가 역행렬이 존재해야만 사용할 수 있고, 역행렬을 구하면서 행렬을 곱하는 과정 자체에 cost가 높다는 단점도 존재한다.

1-4. Gradient Descent

$$\nabla_A f(A) \in \mathbb{R}^{m \times n} = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1n}} \\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}} \end{bmatrix}$$

$$(\nabla_A f(A))_{ij} = \frac{\partial f(A)}{\partial A_{ij}}.$$

그림6. Gradient

Gradient Descent를 살펴보기 전, Gradient를 먼저 알아보자면 간단하게 기울기라고 생각하면 된다. 그림 6의 의미는 함수 f가 있고, f의 출력값이 행렬 A라고 생각할 때 gradient를 아래와 같이 정의할 수 있다는 뜻이다. 즉 Gradient는 행렬의 각 요소별로 미분을 진행한 것으로, 각각 해당 요소에서의 증가율과 방향성을 가진다.

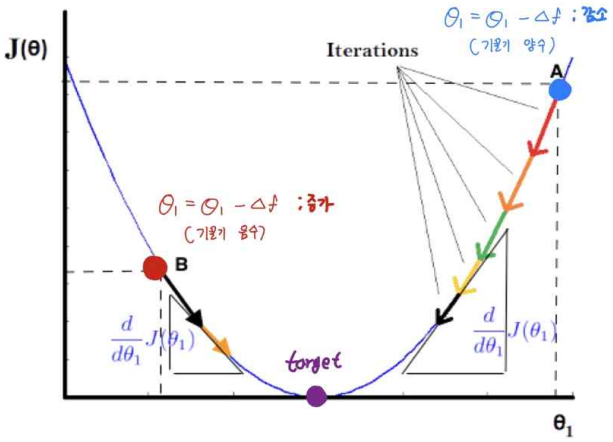

그림7. Gradient로 optimal한 J값을 찾는 과정

그림 7에서, 첫 세타 값들을 무작위로 선택한 다음, J 값이 최대한 작아지도록 세타 값을 계속 갱신해주는 작업을 나타낸다. 여기서 알파는 learning rate로, 얼마나 증가 혹은 감소할지를 수치상으로 결정하는 변수이다. 이렇게 기울기를 사용하여 optimal한 J 값을 찾아내는 방법이 Gradient Descent이며, 식은 아래와 같다.

$$\text{repeat until convergence } \{$$
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$
$$\}$$

그림8. Gradient Descent

세타j 오른쪽이 gradient를 나타내는 식이고, 이렇게 세타 값을 갱신시키며 target에 수렴하도록 한다. 만약 MSE를 cost function으로 사용한다면 아래와 같이 사용할 수 있다.

$$h_\theta(x) = \theta_0 + \theta_1 x$$
$$\text{repeat until convergence } \{$$
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$
$$\}$$

그림9. MSE에서의 Gradient Descent 예시

2. Implementation

2-1. Settings

본격적인 코드를 살펴보기 전, 준비해야 하는 기본 설정에 대한 코드를 먼저 소개할 것이다. 이를 먼저 해놓아야만 뒤의 Task들이 정상적으로 수행된다. Part 2에서는 코드의 핵심 부분만 설명하고 결과를 도출하거나 부수적인 코드 설명은 생략한다. 코드별 결과 사진은 *3-3. Working Snapshots* 에 있다.

```
1  import pandas as pd
2  import numpy as np
3  test = pd.read_csv("/content/drive/MyDrive/ML_HW1/data_hw1.csv")
4  test.isnull().any()
5  x_data = test.x
6  y_data = test.y
7  x = np.array(x_data).reshape(len(x_data), 1)
8  y = np.array(y_data).reshape(len(y_data), 1)
```

그림10. Setting Code

우선 Linear Regression에 필요한 라이브러리인 pandas와 numpy를 불러오고, task에 필요로 하는 데이터인 csv파일을 pandas로 불러와 test에 저장한다. 4번째 줄은 csv파일에 혹시 모를 null 값이 있는지 확인하기 위한 코드이고, 5~6번째 줄은 x값과 y값을 x_data와 y_data에 넣어주는 과정이다. 마지막으로 7~8번째 코드를 보면 x_data와 y_data를 Linear Regression할 수 있도록 Matrix 형태로 변형해 주었다.

2-2. Task Code

```
10  a1, b1 = 0, 0
11  learn_rate, epochs = 0.01, 10000
12  length = len(x)
13  for _ in range(epochs):
14      y_pred = a1*x + b1
15      gradient_a = (2/length) * np.sum((y_pred-y) * x)
16      gradient_b = (2/length) * np.sum(y_pred-y)
17      a1 -= learn_rate * gradient_a
18      b1 -= learn_rate * gradient_b
```

그림11. Task1 Code

Task1은 y = ax+b라는 모델이 있을 때 Gradient Descent로 optimal한 a와 b값을 구할 수 있도록 모델을 fitting하는 것이다. 초기 a, b값은 0으로 설정하고, learn_rate(learning rate)와 epochs 값은 0.01과 10000으로 설정했는데, 여러 테스트를 진행해 본 결과 0.01과 10000이 실제 값과 가장 유사하게 도출되어 두 수로 값을 정했다. epochs 만큼 gradient descent를 진행해 주고, 15~18번째 줄을 보면 앞서 설명한 개념이 모두 적용되어 있다. gradient descent 식을 사용하여 gradient값을 구하고, learn_rate와 곱하여 a값과 b값을 조절하고 있는 모습을 확인할 수 있다. y_pred이 모델의 예측값, y가 실제 값이다.

```
20  xx = np.column_stack((x**2, x, np.ones_like(x)))
21  xx_transpose = np.transpose(xx)
22  dot_x = np.dot(xx_transpose, xx)
23  dot_y = np.dot(xx_transpose, y)
24  theta = np.dot(np.linalg.inv(dot_x), dot_y)
25  a2, b2, c2 = theta
```

그림12. Task2 Code

Task2는 y = ax^2+bx +c라는 모델에 Normal Equation 방법으로 모델을 fitting하는 것이다. numpy의 column_stack()으로 입력 데이터인 x와 x^2으로 열을 쌓아 새로운 xx라는 2차원 배열을 만들었다. 또한 Normal Equation에는 역행렬 계산이 필요하기 때문에 numpy의 transpose()를 사용하였다. 이후에는 Normal Equation의 공식처럼 행렬 곱을 수행하여 세타 행렬을 만들어내고, 여기서 최종 a, b, c값을 정해준다.

3. Environment

그림13. Google Colab

3-1. Build Environment

Google Colaboratory virtual CPU

T4 GPU

Python 3.10.12

Pandas 1.5.3

NumPy 1.23.5

3-2. Compile

1. HW1_32192530_code.zip 압축풀고 hw1.ipynb, data_hw1.csv 저장
2. Google Colab 접속 → 업로드 → 둘러보기 클릭해서 hw1.ipynb 파일 불러오기
3. 우측 위의 연결 옆 화살표 클릭
4. 런타임 유형 변경 → 하드웨어 가속기를 T4 GPU로 변경
5. 우측 위의 연결 클릭
6. 연결이 정상적으로 되면 **drive.mount('/content/drive')**까지만 코드 실행
   (구글 드라이브 연동)
7. 왼쪽의 폴더 아이콘 클릭 후 drive/MyDrive에 data_hw1.csv 업로드
8. 이후 코드도 모두 실행

3-3. Working Snapshots

	x	y
0	-1.572851	2.927733
1	-2.424197	25.341094
2	0.402431	-8.935209
3	-1.648521	4.257155
4	-1.899214	9.499411
...	...	...
995	-0.388337	-7.106198
996	-1.474044	1.353707
997	-1.799926	7.262406
998	-1.639442	4.091766
999	0.675706	-9.968438

1000 rows × 2 columns

그림14. csv파일 업로드 확인

```
x :  [[-1.57285121]
 [-2.4241968 ]
 [ 0.40243062]
 [-1.64852104]
 [-1.89921403]
 [-2.97595611]
 [ 0.20764787]
 [-1.01036686]
 [-0.84862863]
 [ 0.4836501 ]]
x shape :  (1000, 1)
y :  [[ 2.92773279]
 [25.34109429]
 [-8.93520885]
 [ 4.25715489]
 [ 9.49941096]
 [50.66392018]
 [-8.43320232]
 [-3.91641805]
 [-5.08042801]
 [-9.19356856]]
y shape :  (1000, 1)
```

그림15. 데이터 Matrix 형태 (9번 idx까지만 출력)

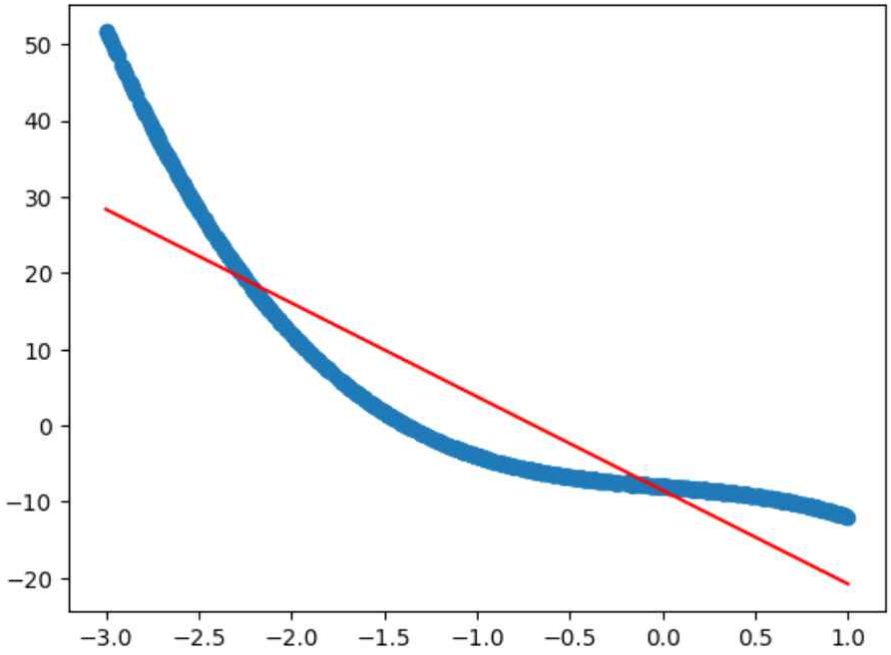

그림16. Task1 plot

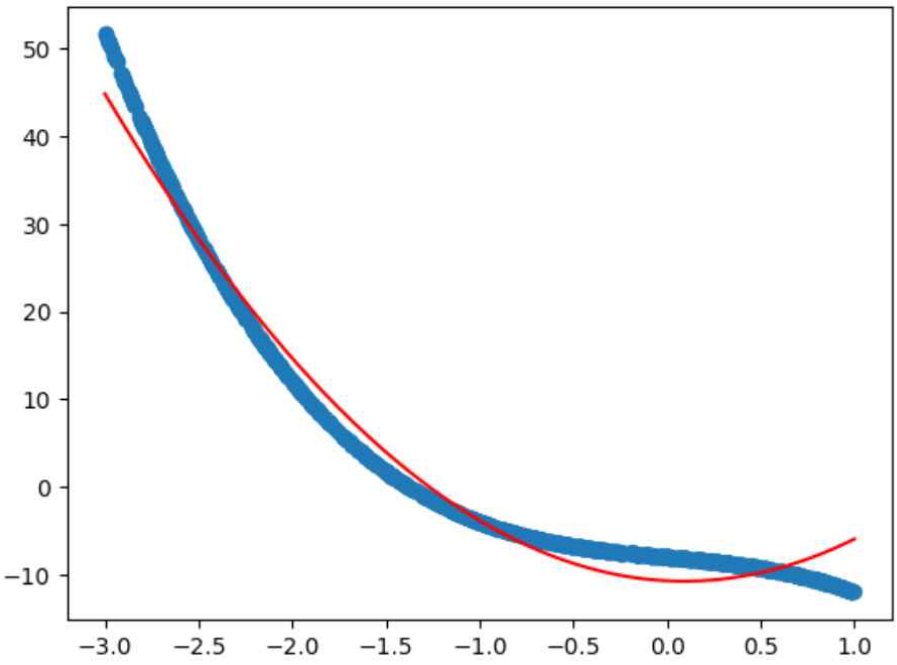

그림17. Task2 plot

4. Conclusion

아래 표는 Task1과 Task2의 최종 결과이다.

	a	b	c
Task1	-12.288	-8.493	-
Task2	5.821	-1.057	-10.705

(소수점 넷째 자리에서 반올림)

이번 레포트에서는 Linear Regression와 관련 개념들을 알고, Gradient Descent와 Normal Equation으로 task의 모델들을 fitting하며 optimal한 매개변수들을 찾아냈다. 테스트를 해보니 실제 값과 유사한 결과를 도출할 수 있었다. 처음 수업 때 들은 Linear Regression이 이론적이라 잘 와닿지 않았지만, 실제로 직접 구현하면서 실습해 보니 내용을 이해하는 데 큰 도움이 되었다. 특히 Gradient Descent와 Normal Equation의 차이점과 방법, 각각 사용할 때의 고려해야 할 사항들을 제대로 알 수 있었다. 동시에 NumPy와 Pandas같은 인공지능 관련 파이썬 라이브러리를 사용해 보며 이후에 있을 과제뿐만 아니라 혼자서 Machine Learning 코드를 구현하고 연습하기 위한 기반을 닦았다.