텍스트이(가) 표시된 사진

자동 생성된 설명

Gradient Descent

&

Normal Equation

Homework #1

32181928 박찬호

Dankook University

Mobile Systems Engineering

2022 Fall

**Contents 목차**

**Ⅰ 서론 01**

**Ⅱ 본론 02**

**Ⅲ 결론 09**

Linear Regression 01

Cost Function 01

Gradient Descent 02

Least Square 07

최종 결과 09

느낀 점 09

실행방법 09

서론

Linear Regression

Linear란 무엇일까? Linearity(Linear)는 additivity와 homogeneity를 동시에 만족하는 성질이다. 수식으로 보면, 임의의 수 x, y에 대해 를 항상 만족하면 함수 f는 linearity를 만족한다고 한다.

그렇다면Linearity가 왜 중요한가? Linearity는 단순한 입력들이 존재할 때 그 입력들의 스칼라배와 그 합으로 더 복잡한 입력을 표현할 수 있으면, 단순한 입력에서 얻어지는 단순한 출력으로 복잡한 출력을 구해낼 수 있다는 뜻이다. 따라서, 이러한 방법은 복잡한 관계를 단순한 속성들의 조합으로 표현할 수 있게 하기 때문에 어떠한 데이터를 설명하는 모델을 구성하거나, 또는 시스템을 설명할 때 막강한 도구가 된다. 따라서 linear하다는 것은 중요한 특성이다.

Linearity의 막강함을 자세히 들여다보자. 어떠한 데이터가 linearity를 만족하는 상황에서 이 데이터에 우리가 관심을 갖는 값 y가 존재한다고 생각하자. 이때, y가 데이터의 다른 속성과 어떤 관계를 갖는지 궁금하거나, 또는 데이터의 다른 속성을 통해 y의 값을 예측하고자 할 때 Linearity를 만족하는 것만으로 문제는 상당히 단순해진다.

Linear Regression은 이러한 상황에서 아주 효과적인 분석방법이다. 각 종속변수를 로, 독립변수를 로, bias를 포함한 독립변수의 계수를 행렬 로 표현하면 의 꼴로 문제를 나타낼 수 있게 되고, 결과적으로 우리가 해결할 문제는 에 한정되게 된다. 즉, 데이터를 잘 설명하는 어떠한 model을 찾는 과정이 linear regression에서는 적절한 를 구하는 문제로 바뀌는 것이다. Linear regression에서 를 찾는 방법은 Gradient Descent와 Least Square가 있다.

Cost Function

Gradient Descent와 Least Square로 향하기 이전에 optimal 와 잘 맞는 model이라는 것은 과연 무엇일까? 어떠한 데이터가 존재할 때 그 데이터가 갖는 y의 값과 model에 x를 입력해 얻은 출력이 비슷하다면, 우리는 직관적으로 그 model이 그 데이터를 잘 설명한다고 생각한다. 이를 조금 더 엄밀히 정의하기 위해, 우리는 cost function에 대해 알아보자.

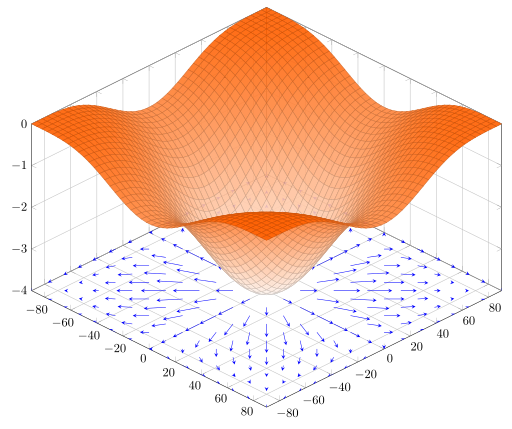
Cost function은 model의 예측과 데이터의 y가 얼만큼 비슷한지, 다른 말로는 얼만큼 차이나는지 설명하는 함수이다. 이때 이 차이를 Error 또는 Cost라고 한다. Cost는 다양한 방식으로 정의될 수 있는데, 주로 MSE, Cross-Entropy를 사용한다.

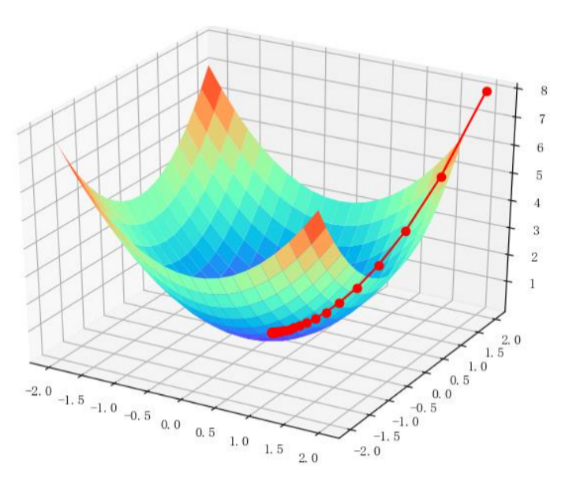
다시 Optimal 로 돌아가면, 결국 optimal 를 찾는다는 것은 Cost function의 결과가 가장 작은, model의 error가 가장 적은, 즉 model이 데이터를 가장 비슷하게 설명하는 를 찾는다는 것이다. 따라서, Linear Regression을 수행할 때의 우리의 목적은 cost function J에 대해 로 바뀌게 된다.

본론

Gradient Descent

Gradient Descent는 Gradient라는 벡터장을 이용하여 cost function의 optimal 를 찾는 방법이다. 함수 F가 존재하고, 이 함수의 출력이 인 행렬 A일 때, gradient는

위와 같은 꼴을 가진다. 즉, Gradient는 해당 성분에서의 편미분으로 해당 성분에서의 증가율과 그 방향을 갖는다. 이를 함수의 그래프와 비교해서 확인해보면 아래와 같다.

그래프에서 확인할 수 있듯 어떤 지점에서 함수의 기울기가 가파르면, 그 아래 투영된 gradient도 크게 나타난다. 따라서 Convex한 cost function에서 gradient에 음수를 곱하면, 크기는 gradient의 크기에 비례하고 방향은 optimal point를 향하는 벡터들을 얻을 수 있다(Convex하지 않은 경우 local minimum에 빠지는 것과 같은 문제가 발생할 수 있다. 그러나 여전히 gradient descent를 사용할 수 있다). 이는 optimal point에 가까워질수록 벡터의 크기가 더 작아지므로, 결과적으로 gradient를 구하고 음수를 곱한 후, 이를 적용해서 이동한 다음 성분에서 다시 gradient를 구하는 것을 반복하면 optimal point로 향하게 될 수 있음을 보인다.

이를 다시금 정리하면 다음과 같다. 이때 alpha는 step size로 gradient를 적용하는 정도이다.

이러한 과정을 조금 특수한 경우에서 풀어 써보겠다. 먼저 parameter로 와 이 존재하고,

인 h를 hypothesis라고 가정한다. 이때 cost function J로 MSE를 사용한다고 생각하자. 그렇다면, m이 전체 샘플의 수일 때 J는

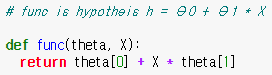
로 정의할 수 있다. 그렇다면, gradient descent는

가 된다.

과제의 Task1이 위의 공식을 그대로 이용하는 문제이다.

Task1에서는 y = ax + b라는 모델을 gradient descent를 이용하여 fitting한다.

코드를 통해 살펴보겠다.

먼저 모델을 func로 정의한다

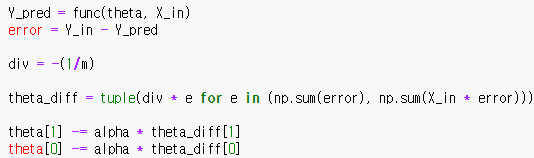
Gradient Descent는 X, Y, 그리고 func를 필수 인자로 받는다.

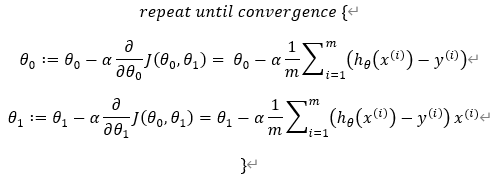
Epsilon은 이전 iteration과의 error 차이가 epsilon보다 작으면 convergence라고 판단하여 iteration을 멈춘다

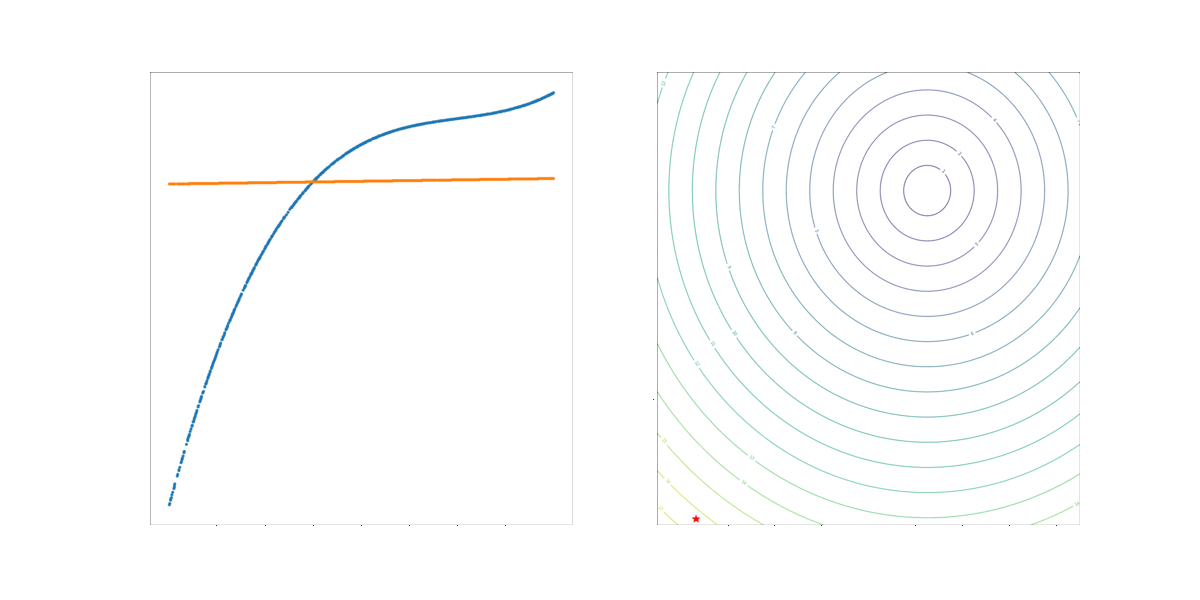
Alpha는 step size, Epoch은 iteration 횟수이다.



핵심 로직 부분이다. 각 코드에 맞는 부분을 수식에서 같은 색으로 표시하였다





이를 통해 얻어지는 full batch gradient의 결과는 아래와 같다

θ0 = 10.475902, θ1 = 12.278560, epoch = 267, error = 51.713806419871865

(error는 mse이다)

Gradient Descent는 m의 크기에 따라 다른 batch size를 가질 수 있다. 앞서 설명한 gradient descent는 (full) batch gradient descent로 전체 데이터를 batch로 하여 gradient를 계산하는 방법이다. 이러한 방법은 convergence로 부드럽게 이어지지만, 속도가 느리고 계산 양이 많으며, local minimum에 빠질 수 있다는 단점이 있다.

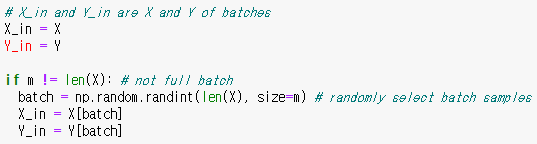
반면 Stochastic Gradient Descent(SGD)는 batch size (m)가 1일 때의 경우로, 무작위로 하나의 샘플을 선택하여 gradient를 계산하고 gradient descent를 수행한다. 이는 빠르고, shooting 특성을 가지기 때문에 local minimum을 피할 수 있지만, global minimum에 도달한 후에도 계속 shooting을 거듭하기 때문에 convergence에 이르러서도 계속 에러가 들쑥날쑥한 경향이 있다.

이러한 문제를 해결하기 위해, mini-batch (stochastic) gradient descent가 등장했다. Mini-batch 방법은 batch size를 1과 full batch 사이로 정함으로써, full batch에서 생기는 속도, 계산 양 등의 문제를 해결하면서 동시에 SGD에서 값이 튀는 문제도 해결하였다.

위의 세 방법은 아래의 코드에서 m을 설정하면서 구현하였다.



또한, 매 iteration에서 stochastic 하게 batch를 구성하는 코드도 구현하였다.

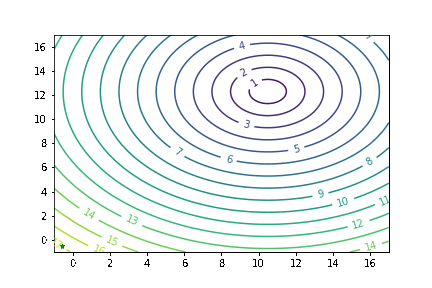


이를 통해 최종적으로 아래와 같은 결과를 얻을 수 있었다.

녹색 점 : full batch gradient descent

빨간 점 : mini batch gradient descent (batch size = 30)

파란 점 : stochastic gradient descent



Least Square

Least Square는 Mean Squared Error과 같은 squared error를 사용한 cost function에 한정해서 쓸 수 있는 cost optimization 방식이다. Squared Error는 convex function이고, squared term으로 구성되어 있기 때문에 global minimum을 바로 찾을 수 있다.

Convex function에서 global minimum은 derivative 가 0이 되는 지점에 존재하는데, mean squared과 같은 cost function J는 multi variable function이므로, 각 변수에 대해 partial derivative를 수행하여 derivative가 0이 되는 지점을 찾아야 한다.

와 같은 가장 단순한 경우를 생각해보자. 이때 h의 각 변수에 대한 partial derivative는 앞서 gradient 파트에서 다룬 것과 같이, 아래의 수식으로 정리된다.

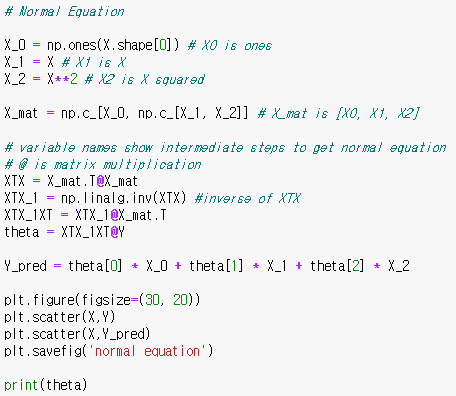
이를 통해 각 partial derivative가 0이 되는 와 을 찾을 수 있다

위의 식을 다변수에 대해 일반적인 수식으로 행렬을 통해 정리하면 아래와 같다. 증명은 생략한다.

이처럼, least square를 이용하면 수식적으로 global minimum을 바로 도출할 수 있다는 장점이 있다. 그러나, 행렬 곱을 수행하는 것이 많은 computation을 요구하고, 또한 행렬이 symmetric이고 inverse 가 존재해야 한다는 제약이 있다.

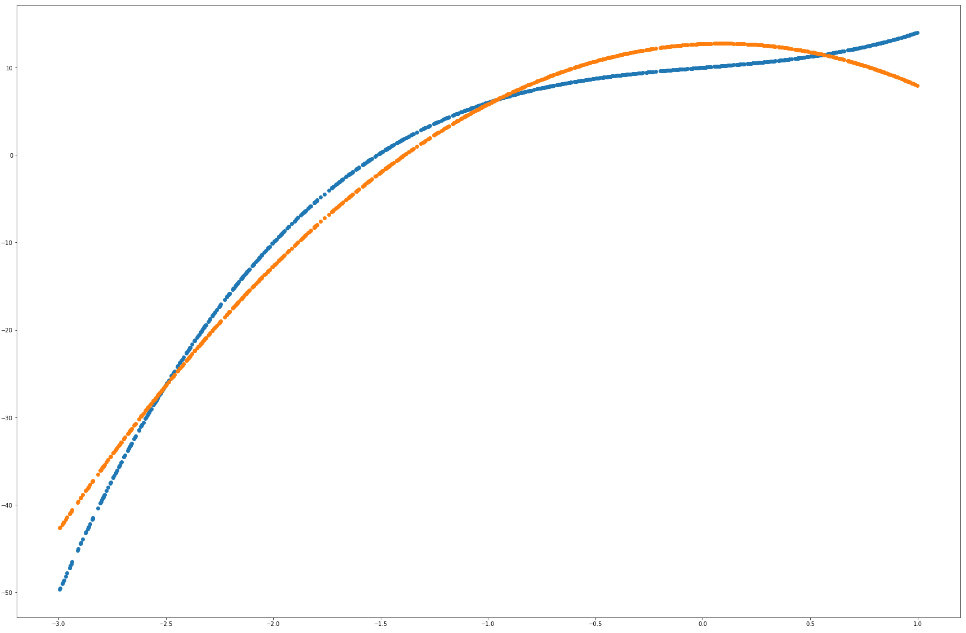
이를 코드로 구현하기 위해서는 위의 행렬 곱을 구현하면 된다.

Task2에서는 모델에 대해 Normal Equation을 통해 fitting을 수행한다.



결과는 아래와 같이 나온다.

[12.70482255 1.05708197 -5.82074893]



최종 결과

|  |  |  |  |
| --- | --- | --- | --- |
|  | a | b | c |
| Task1 | 12.278560 | 10.475902 | 0 |
| Task2 | -5.82074893 | 1.05708197 | 12.70482255 |

결론

느낀 점

Gradient Descent를 잘 안다고 생각했는데, 막상 구현을 하는 과정에서 깔끔하게 로직을 작성하려다 보니 이해가 조금 얕은 부분을 발견할 수 있었습니다. 그러한 부분을 메꿀 수 있는 좋은 경험이었습니다.

실행 방법

HW1.ipynb를 jupyter notebook이나 colab에서 열고 하나씩 셀을 실행하면 됩니다. Numpy, matplotlib이 설치되어 있어야합니다.

제공해드리는 HW1.py는 HW1.ipynb와 동일한 코드입니다. conda run HW1.py를 통해서 실행할 수 있습니다. 그러나 셀 환경에서 실행되지 않기 때문에 원활한 동작이 이뤄지지 않습니다. Plot을 마지막에만 출력하도록 수정 후 실행해야 합니다