머신러닝 기초 & 선형회귀

오늘의 학습내용

- 머신러닝의 기본 프로세스
- 선형회귀(Linear Regression)
- Overfitting
- Mini-Batch Gradient Descent
- PyTorch를 이용한 선형회귀 구현

머신러닝의 기본 프로세스 - 가설 정의, 손실함수 정의, 최적화 정의

❖ 모든 머신러닝 모델은 다음의 3가지 과정을 거칩니다. 이 과정은 앞으로 배우는 모든 머신러닝 알고리즘의 기본 토대이기 때문에 꼭 제대로 숙지하고 넘어가셔야 합니다.

① 학습하고자 하는 가설Hypothesis h(θ)을 수학적 표현식으로 나타낸다.

② 가설의 성능을 측정할 수 있는 손실함수Loss Function J(θ)을 정의한다.

③ 손실 함수 J(θ)을 최소화Minimize 할 수 있는 학습 알고리즘을 설계한다.

1. 가설 정의

- 이 3가지 과정을 선형 회귀 모델에 대입해서 생각해보면 다음과 같습니다.
- 선형 회귀 모델은 선형 함수를 이용해서 회귀를 수행하는 기법입니다.
- 선형 회귀 함수는 학습하고자 하는 가설Hypothesis h(θ)을 아래와 같은 선형 함수 형태로 표현합니다.

$$y=Wx+b$$

- 이때 x와 y는 데이터로부터 주어지는 인풋 데이터, 타겟 데이터이고, W와 b는 파라미터Parameter θ라고 부르며 트레이닝 데이터로부터 학습을 통해 적절한 값을 찾아내야만 하는 값입니다.

2. 손실 함수 정의

- 적절한 파라미터값을 알아내기 위해서는 현재 파라미터값이 우리가 풀고자 하는 목적Task에 적합한 값인지를 측정할 수 있어야 합니다. 이를 위해 손실 함수Loss Function J(θ)를 정의합니다.
- 손실 함수는 여러가지 형태로 정의될 수 있습니다. 그 중 가장 대표적인 손실 함수 중 하나는 평균제곱오차Mean of Squared Error(MSE)입니다. MSE는 다음 수식으로 정의됩니다.

$$MSE = \frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

2. 손실 함수 정의

❖ 예를 들어, 정답이 y=[1, 10, 13, 7]이고 우리의 모델의 예측값이 $\hat{y}$=[10, 3, 1, 4]와 같이 잘못된 값을 예측한다면 MSE 손실 함수는 아래와 같이 35.375라는 큰 값을 갖게 될 것입니다.

$$MSE = \frac{1}{2*4}\{(10-1)^2+(3-10)^2+(1-13)^2+(4-7)^2\} = \mathbf{35.375}$$

❖ 하지만, 정답이 y=[1, 10, 13, 7]이고 우리의 모델의 예측값이 $\hat{y}$=[2, 10, 11, 6]와 같이 비슷한 값을 예측한다면 MSE 손실 함수는 아래와 같이 1.5라는 작은 값을 갖게 될 것입니다.

$$MSE = \frac{1}{2*4}\{(2-1)^2+(10-10)^2+(11-13)^2+(6-7)^2\} = \mathbf{1.5}$$

❖ 이처럼 손실 함수는 우리가 풀고자 하는 목적에 가까운 형태로 파라미터가 최적화 되었을 때(즉, 모델이 잘 학습되었을 때) 더 작은 값을 갖는 특성을 가져야만 합니다. 이런 특징 때문에 손실 함수를 다른 말로 비용 함수Cost Function라고도 부릅니다.

3. 최적화 정의 – Gradient Descent

③ 마지막으로 손실 함수를 최소화하는 방향으로 파라미터들을 업데이트할 수 있는 학습 알고리즘을 설계해야 합니다.

❖ 머신러닝 모델은 보통 맨 처음에 랜덤한 값으로 파라미터를 초기화한 후에 파라미터를 적절한 값으로 계속해서 업데이트합니다.

❖ 이때, 파라미터를 적절한 값으로 업데이트하는 알고리즘을 최적화Optimization 기법이라고 합니다. 여러 최적화 기법 중에서 대표적인 기법은 경사하강법Gradient Descent입니다.

❖ 경사하강법은 현재 스텝의 파라미터에서 손실 함수의 미분값에 러닝레이트 α를 곱한만큼을 빼서 다음 스텝의 파라미터값으로 지정합니다.

❖ 따라서 손실 함수의 미분값이 크면 하나의 스텝에서 파라미터가 많이 업데이트되고 손실 함수의 미분값이 작으면 적게 업데이트 될 것입니다. 또한 러닝레이트 α가 크면 많이 업데이트, α가 작으면 적게 업데이트 될 것입니다.

3. 최적화 정의 – Gradient Descent

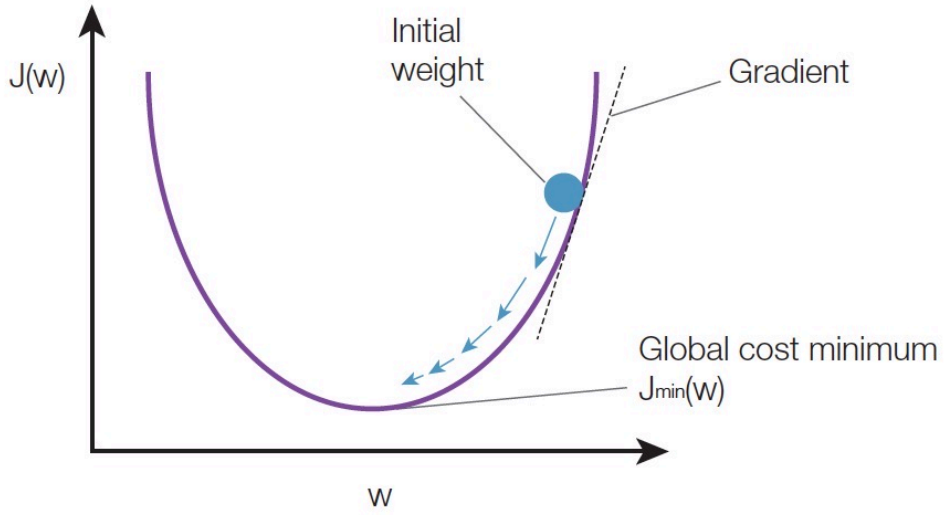
❖ 경사하강법의 파라미터 한 스텝 업데이트 과정을 수식으로 나타내면 다음과 같습니다.

$$\theta_i = \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1)$$

❖ 위 표현에서 θ는 우리가 학습하고자하는 파라미터를 나타냅니다. 선형회귀 모델에서 파라미터는 W와 b 입니다. 즉, θ=(b,W), θ_0 =b, θ_1 =W 입니다.

3. 최적화 정의 – Gradient Descent

❖ 아래 그림은 경사하강법의 동작 과정을 나타냅니다.

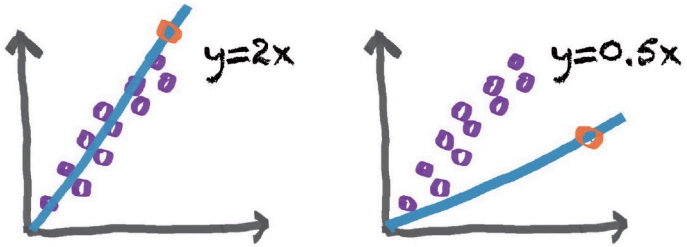


머신러닝의 기본 프로세스 - 가설 정의, 손실함수 정의, 최적화 정의

- 경사하강법은 손실 함수가 최소가 되는 지점에서 종료하는 것이 가장 이상적이지만 현실적으로 언제 손실 함수가 최소가 될지 알기 어렵기 때문에 충분한 횟수라고 생각되는 횟수만큼 업데이트를 진행한 후 학습을 종료합니다.
- 이제 머신러닝에 필요한 3가지 과정(모델 정의, 손실 함수 정의, 옵티마이저 정의)을 모두 살펴 보았습니다.

선형 회귀 (Linear Regression)

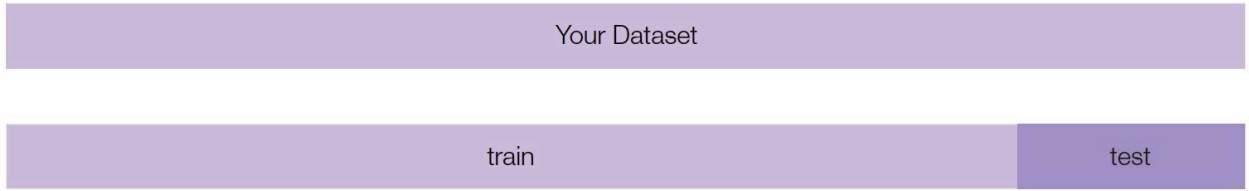
- 이제 머신러닝의 3가지 프로세스를 이용해서 선형 회귀 모델을 구현해봅시다. 아래 그림은 선형 회귀의 예시를 보여줍니다.



- 보라색 동그라미는 트레이닝 데이터, 파란색 라인은 선형 회귀 기법이 학습한 가설, 주황색 동그라미는 학습한 가설을 바탕으로 테스트 데이터에 대해 예측을 수행한 결과입니다.
- 왼쪽 그림은 선형 회귀 모델이 y=2x(W=2, b=0)로 가설을 학습한 경우, 오른쪽 그림은 선형 회귀 모델이 y=0.5x(W=0.5, b=0)로 가설을 학습한 경우입니다.
- 그림에서 볼 수 있듯이 보라색의 트레이닝 데이터는 y=2x 형태의 경향성을 띠고 있기 때문에 선형 회귀 모델이 잘 학습된 경우 y=2x 형태의 가설을 가지고 있어야 합니다. 만약 잘못된 선형 함수가 학습된 경우 오른쪽 그림과 같이 테스트 데이터에 대해 부정확한 예측값을 출력하게 될 것입니다.

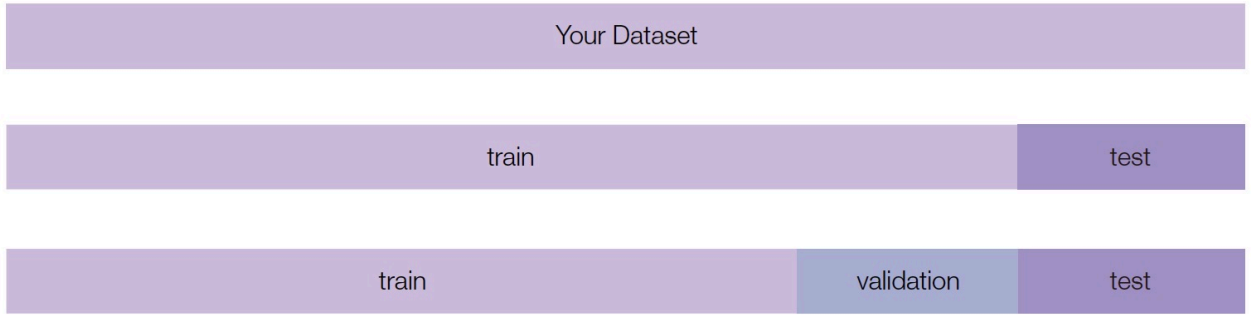
트레이닝 데이터, 테스트 데이터 나누기(split)

- 따라서 머신러닝 모델을 학습시키기 위해서 전체 데이터의 일부를 Training Data, 일부는 Test Data로 나눠서 사용합니다.
- 일반적으로 데이터의 80% 정도는 트레이닝 데이터, 20% 정도는 테스트 데이터로 나눠서 사용합니다.
- 예를 들어 1000명의 (키, 몸무게) 데이터가 있다면 800명분의 데이터는 트레이닝 데이터, 200명분의 데이터는 테스트 데이터로 나눠서 사용합니다.



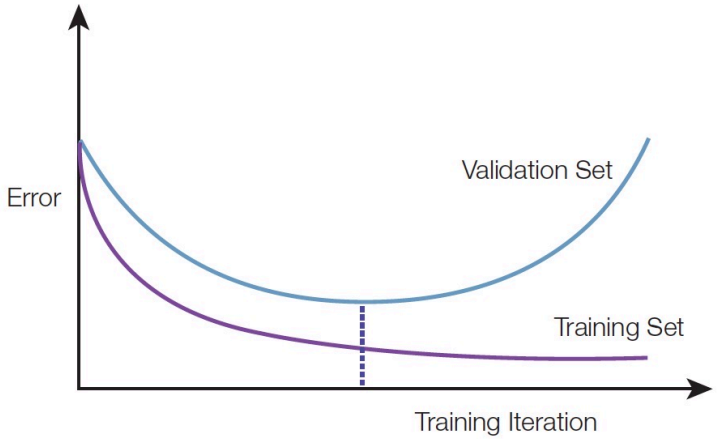
Validation Data(검증용 데이터)

- 여기에 더 나아가서 전체 데이터를 트레이닝$_{training}$ 데이터, 검증용$_{validation}$ 데이터, 테스트$_{test}$ 데이터로 나누기도 합니다.
- 검증용 데이터는 트레이닝 과정에서 학습에 사용하지는 않지만 중간중간 테스트하는데 사용해서 학습하고 있는 모델이 오버피팅에 빠지지 않았는지 체크하는데 사용됩니다.
- 즉, 직관적으로 설명하면 검증용 데이터는 트레이닝 과정 중간에 사용하는 테스트 데이터로 볼 수 있습니다.



Overfitting, Underfitting

- 아래 그림은 학습 과정에서 트레이닝 에러와 검증 에러를 출력한 그래프입니다.
- 처음에는 트레이닝 에러와 검증 에러가 모두 작아지지만 일정 횟수 이상 반복할 경우 트레이닝 에러는 작아지지만 검증 에러는 커지는 오버피팅(Overfitting)에 빠지게 됩니다.
- 따라서 트레이닝 에러는 작아지지만 검증 에러는 커지는 지점에서 업데이트를 중지하면 최적의 파라미터를 얻을 수 있습니다.



Overfitting, Underfitting

- 오버피팅(Overfitting)은 학습 과정에서 머신러닝 알고리즘의 파라미터가 트레이닝 데이터에 과도하게 최적화되어 트레이닝 데이터에 대해서는 잘 동작하지만 새로운 데이터인 테스트 데이터에 대해서는 잘 동작하지 못하는 현상을 말합니다. 오버피팅은 모델의 표현력이 지나치게 강력할 경우 발생하기 쉽습니다.

- 그림 4-3은 오버피팅, 언더피팅의 경우를 보여줍니다. 그림 4-3의 가장 오른쪽 그림을 보면 모델이 트레이닝 데이터의 정확도를 높이기 위해 결정 직선(Decision Boundary)을 과도하게 꼬아서 그린 모습을 볼 수 있습니다. 이럴 경우 트레이닝 데이터와 조금 형태가 다른 새로운 데이터를 예측할 때도 성능이 떨어지게 됩니다.

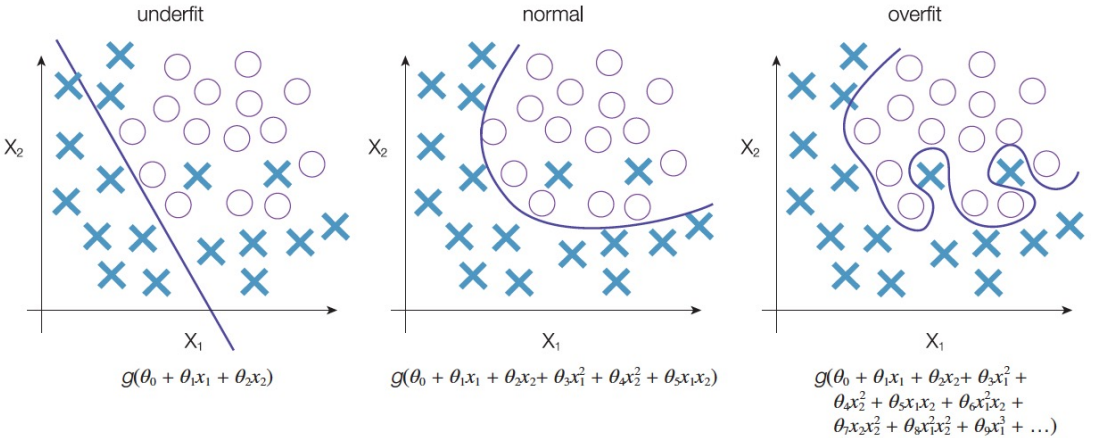


그림 4-3 | 언더피팅, 적절히 학습된 경우, 오버피팅

Overfitting, Underfitting

- 이에 반해 그림 4-3의 가장 왼쪽 그림은 언더피팅(Underfitting)에 빠진 상황을 보여줍니다. 언더피팅은 오버피팅의 반대 상황으로 모델의 표현력이 부족해서 트레이닝 데이터도 제대로 예측하지 못하는 상황을 말합니다. 마지막으로 그림 4-3의 중앙에 있는 그림은 오버피팅과 언더피팅에 빠지지 않고 파라미터가 적절히 학습된 경우를 보여줍니다.
- 이런 오버피팅을 방지하기 위한 기법들을 Regularization 기법이라고 부릅니다.

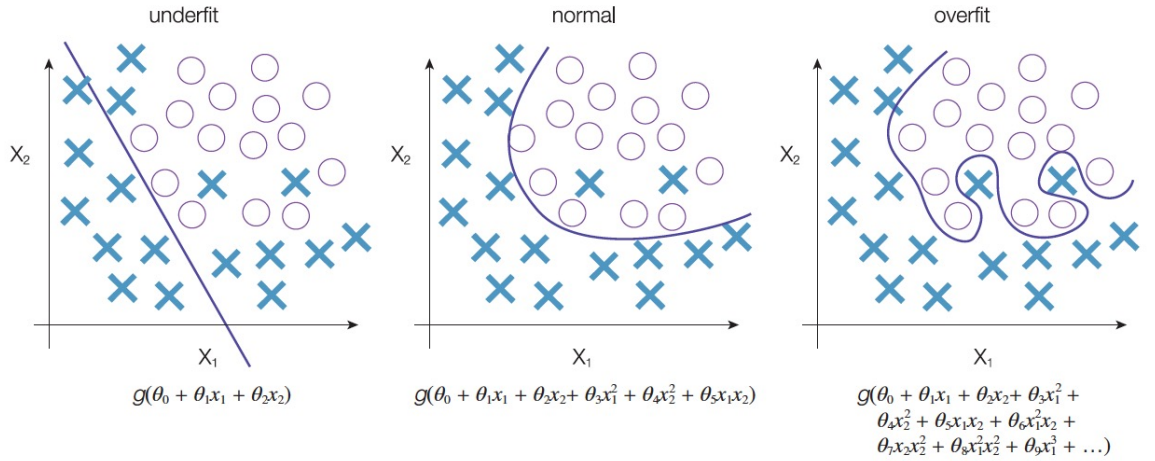


그림 4-3 | 언더피팅, 적절히 학습된 경우, 오버피팅

Batch Gradient Descent, Mini-Batch Gradient Descent, Stochastic Gradient Descent

- 이 3가지 과정 중에서 손실 함수 J(θ)를 최소화할 수 있는 최적화 알고리즘으로 가장 일반적으로 사용되는 기법이 경사하강법입니다.
- 경사하강법은 다음 수식처럼 손실 함수의 미분값과 러닝 레이트의 곱만큼을 원래 파라미터에 뺀 값으로 파라미터를 한 스텝 업데이트합니다.

$$\theta_i = \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta_0, \theta_1)$$

- 가설로 선형회귀 모델(y=Wx)을 사용하고 손실함수로 $MSE(\frac{1}{2n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2)$를 사용할 경우 경사하강법의 한 스텝 업데이트를 위해 계산하는 손실함수의 미분값은 아래와 같이 나타낼 수 있습니다.

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0} \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_1} \frac{1}{2n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

Batch Gradient Descent

- 위 수식에서 우리는 트레이닝 데이터 n개의 손실함수 미분값을 모두 더한 뒤 평균을 취해서 파라미터를 한 스텝 업데이트하였습니다.
- 이런 방법은 경사하강법의 한 스텝 업데이트할 때 전체 트레이닝 데이터를 하나의 Batch로 만들어 사용하기 때문에 Batch Gradient Descent라고 부릅니다.
- 하지만 실제 문제를 다룰 때는 트레이닝 데이터의 개수가 매우 많아 질 수 있습니다. 이런 상황에서 Batch Gradient Descent를 사용할 경우 파라미터를 한 스텝 업데이트하는데 많은 시간이 걸리게되고(한 스텝 업데이트하는데 전체 배치에 대한 미분값을 계산해야하므로), 결과적으로 최적의 파라미터를 찾는데 오랜 시간이 걸리게 됩니다.

Stochastic Gradient Descent

- 이와 반대로 경사하강법의 한스텝 업데이트를 진행할때 1개의 트레이닝 데이터만 사용하는 기법을 Stochastic Gradient Descent 기법이라고 부릅니다. Stochastic Gradient Descent 기법에서 한 스텝 업데이트를 위해 계산하는 손실함수의 미분값은 아래 수식으로 나타낼 수 있습니다.

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0} (\hat{y} - y)^2$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_1} (\hat{y} - y)^2$$

- Stochastic Gradient Descent 기법을 사용할 경우 파라미터를 자주 업데이트 할 수 있지만 파라미터를 한번 업데이트 할 때 전체 트레이닝 데이터의 특성을 고려하지 않고 각각의 트레이닝 데이터의 특성만을 고려하므로 부정확한 방향으로 업데이트가 진행될 수도 있습니다.

Mini-Batch Gradient Descent

- 따라서, 실제 문제를 해결할 때는 Batch Gradient Descent와 Stochastic Gradient Descent의 절충적인 기법인 Mini-Batch Gradient Descent를 많이 사용합니다.
- Mini-batch Gradient Descent는 전체 트레이닝 데이터 Batch가 1000(n)개라면 이를 100(m)개씩 묶은 Mini-Batch 개수만큼의 손실 함수 미분값 평균을 이용해서 파라미터를 한 스텝을 업데이트하는 기법입니다.
- Mini-Batch Gradient Descent 기법에서 한 스텝 업데이트를 위해 계산하는 손실 함수의 미분값은 아래 수식으로 나타낼 수 있습니다.

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0} \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_1} \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

파이토치(PyTorch)를 이용한 선형회귀(Linear Regression) 구현하기

- 파이토치(PyTorch)를 이용해서 선형회귀(Linear Regression) 알고리즘을 실제로 구현해봅시다.
- 2강_PyTorch를_이용한_선형회귀.ipynb
- https://colab.research.google.com/drive/1Vsyb2l9J7adgQ8jA-6izSiUdRdzcl6ye?usp=sharing