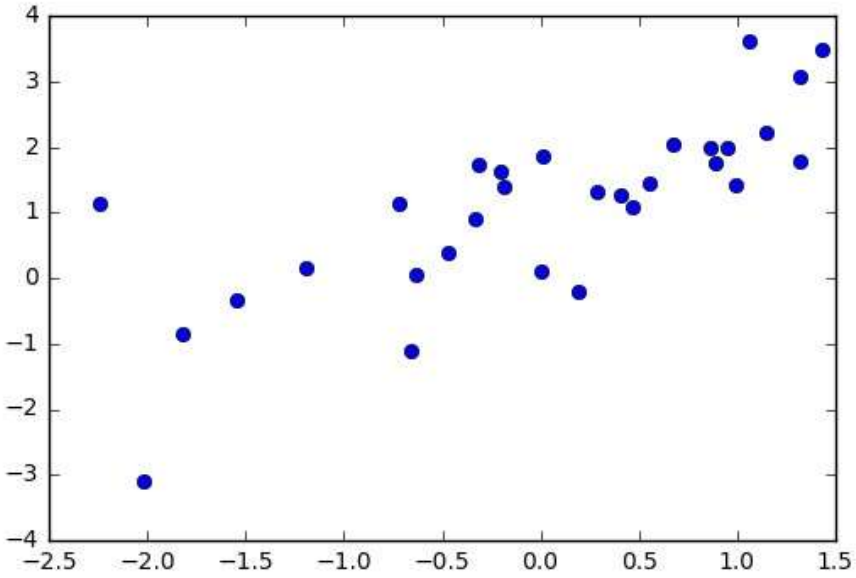
2. 선형 회귀

2.1. 선형 모형



우리는 데이터가 어떤 형태의 패턴을 가지고 있을 것이라 가정을 한다. 예를 들면 위의 그림에서 데이터들은 x가 증가할 수록 y도 증가하는 선형적인 모습을 보인다. 이런 모습을 수식으로 나타내면 $y = wx + b$로 정의할 수 있다. 이 수식에서 w는 직선의 **기울기**(slope), b는 y **절편**(intercept)이다.

b를 절편이라고 하는 이유는 x가 0일 때 이 직선이 y축을 자르고 지나가기 때문이다. 위의 식에 $x = 0$을 대입해보면 그때의 $y = b$가 되는 것을 알 수 있다.

w는 기울기라고도 하지만 가중치(weight)라고도 한다. 그림으로 그릴 때는 직선의 기울기에 해당하기도 하지만, 수식만 볼 때는 x에 일정한 가중치를 곱하는 것이므로 그렇게도 부른다.

이렇게 데이터의 패턴을 정의한 것을 **모형**(model)이라고 한다. 그 중에서도 $y = wx + b$는 직선적인 패턴을 나타내므로 **선형 모형**(linear model)이라고 부른다.

똑같은 선형 모형이라도 기울기 w와 절편 b에 따라 직선의 모양은 달라진다. 이렇게 모형의 형태를 결정짓는 변수를 **파라미터**(parameter) 또는 모수(母數)라고 한다.

선형 모형은 파라미터의 종류나 개수가 정해져 있다. 이런 모형을 **모수적**(parametric) 모형이라고 한다. 반대로 파라미터의 개수나 종류가 고정되어 있지 않는 모형을 **비모수적**(nonparametric) 모형이라고 한다. 일단 우리는 모수적 모형을 중심으로 살펴볼 것이다.

기계 학습은 모형을 데이터에 맞춰서 데이터의 패턴을 잡아내는 방식으로 학습이 이뤄진다. 이 과정을 **훈련**(training)이라고도 하고 **적합**(fitting)이라고도 한다. 영어로 fit은 맞춘다는 뜻이다. 모

수적 모형의 경우 적합은 모수, 즉 파라미터를 찾는 것으로 끝이다.

2.2. 회귀의 손실 함수

적합을 위해서는 두 가지가 먼저 필요하다. 한 가지는 모형이 데이터에 얼마나 잘 맞는지를 측정할 방법이 필요하다. 모형이 데이터에 **안** 맞는 정도를 나타내는 함수를 **손실 함수**(loss function)이라고 부른다.

2.2.1. MSE

가장 널리 쓰이는 손실 함수는 **평균 제곱 오차**(mean squared error, 이하 MSE)이다. 예측과 실제의 차이를 오차라고 한다. 이 오차를 모두 제곱한 다음에 평균 낸 것이 MSE이다.

$$MSE = \frac{1}{N}\sum_i (y_i - \hat{y}_i)^2$$

MSE의 정의를 가만히 보면 분산과 똑같다는 것을 알 수 있다. 분산은 각 데이터와 평균의 차이를 제곱해서 평균낸 것이다.

$$Var(y) = \frac{1}{N}\sum_i (y_i - \bar{y})^2$$

분산은 데이터가 평균으로부터 얼마나 퍼져 있는지를 나타내는 통계치이다. MSE도 데이터가 예측으로부터 얼마나 퍼져있는지를 나타낸다고 할 수 있다.

여기서 한 가지 의문이 들 수 있다. 왜 하필 제곱을 하는 것일까? 예측을 두 번 했는데 오차가 +1, -1인 경우와, 오차가 +3, -3인 경우를 생각해보자. 제곱을 하지 않고 오차의 평균을 구하면 두 경우 모두 오차의 평균은 0이 된다. 1-1도 0이고, 3-3도 0이기 때문이다. 따라서 오차를 +로 바꿔줄 필요가 있는다. 그래서 제곱을 하는 것이다.

2.2.2. MAE

물론 제곱 대신 절대값을 사용할 수도 있다. 절대값은 단순히 +, - 부호만 뗀 것이다. 이것을 평균 절대 오차(mean absolute error, 이하 MAE)라고 부른다.

$$MAE = \frac{1}{N}\sum_i |y_i - \hat{y}_i|$$

일반적으로는 MAE보다는 MSE를 더 많이 쓴다. 무슨 거창한 이유 때문이 아니라 계산하기가 더 쉽기 때문이다. 물론 지금 보기에는 부호만 떼면 되는 절대값이 제곱보다 간단해 보이겠지만 계산이 더 복잡해지면 그렇지 않다.

다만 MAE에도 장점이 있다. MSE는 오차를 제곱하기 때문에 오차가 커지면 커질 수록 손실이 제

곱으로 커진다. 따라서 100개의 예측 중에 99개가 잘 맞아도 하나에서 크게 틀리면 그 하나에 영향을 많이 받게 된다. 그런데 그 하나는 데이터가 잘못된 것일 수도 있고, 우연히 패턴에서 벗어난 것일 수도 있다. 이런 예외적인 데이터를 **이상점**(outlier)라고 한다. MAE는 MSE에 비해 이상점에 영향을 적게 받는다. 이렇게 이상점에 적게 영향 받는 것을 통계용어로는 **강건**(robust)하다고 한다. 즉, MAE는 강건한 특성이 있다.

2.2.3. 후버 손실

MSE와 MAE를 절충한 후버 손실(Huber loss)이라는 것도 있다. 통계학자 피터 J. 후버(Peter J. Huber)가 제안한 것으로 일정한 범위(δ, '델타'라고 읽는다)를 정해서 그 안에 있으면 오차를 제곱하고, 그 밖에 있으면 오차의 절대값을 구하는 것이다.

$$L_\delta(e) = \begin{cases} \frac{1}{2}e^2 & \text{for } |e| \le \delta, \\ \delta(|e| - \frac{1}{2}\delta), & \text{otherwise.} \end{cases}$$

위의 식에서 $\frac{1}{2}$가 붙는 이유는 δ에서 오차 제곱과 오차 절대값이 만날 때 매끄럽게 이어지도록 하게 만들기 위해서다.

2.2.4. 결정 계수 R^2

MSE의 단점은 크고, 작음을 직관적으로 알기 어렵다는데 있다. 예를 들어 MSE가 10이라면, 또는 100이라면 이 모형은 얼마나 잘 적합된 것인지 잘 와닿지 않는다. 이를 보완하는 수치가 **결정 계수**(coefficient of determination)이다. 보통 R^2이라고 쓴다.

$$R^2 = 1 - \frac{MSE}{VAR}$$

앞에서 우리는 MSE와 분산의 식이 같은 형태라는 것을 살펴보았다. 만약에 모든 경우에 평균으로 예측을 한다면 MSE와 분산은 같은 값이 된다. 따라서 $MSE/VAR = 1$이 되고 $R^2 = 0$이 된다. 오차가 하나도 없다면 $MSE = 0$이 되고, $R^2 = 1$이 된다.

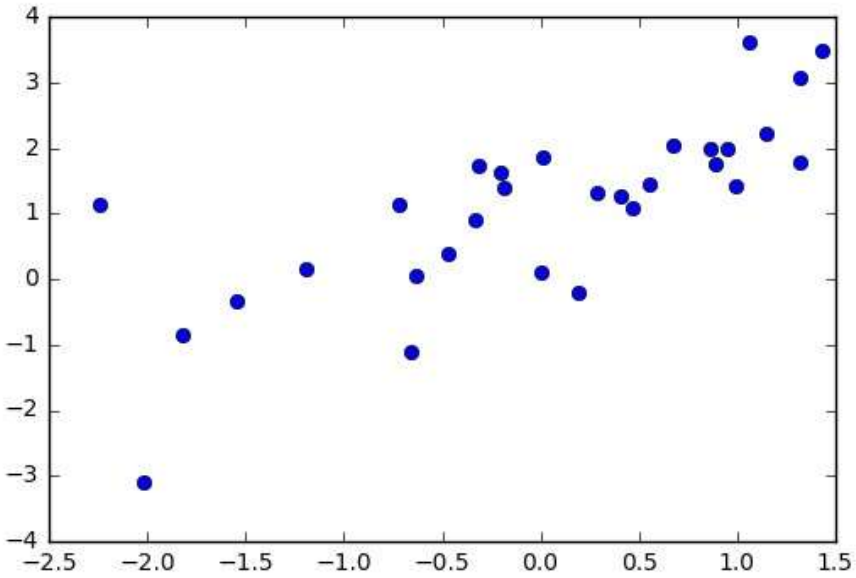
R^2은 대개 0에서 1까지 범위로 나오기 때문에 0%~100%로 읽기도 한다. 퍼센트(percent)는 "백분의"라는 뜻으로 100%는 100분의 100, 즉 1이라는 뜻이다. 익숙한 크기의 값이기 때문에 더 쉽게 이해할 수 있다.

보통 "이 모형은 분산의 ~%를 설명한다"라는 식으로 해석한다. 예를 들어 $R^2 = 0.7$이라면 "이 모형은 분산의 70%를 설명한다"라고 해석한다.

분산은 데이터에 고정된 값이기 때문에 R^2은 실제로는 MSE를 정확히 반영하는 값이다. 즉, MSE가 커지면 작아지고, MSE가 작아지면 커진다. 컴퓨터의 입장에서는 MSE와 R^2 아무 차이가 없지만 사람의 입장에서는 MSE보다 이해하기 쉽기 때문에 실제로 계산은 MSE로 하고 사람이 결과를 살펴볼 때는 R^2을 사용하는 경우가 많다.

2.3. 경사하강법

2.3.1. 모형 적합



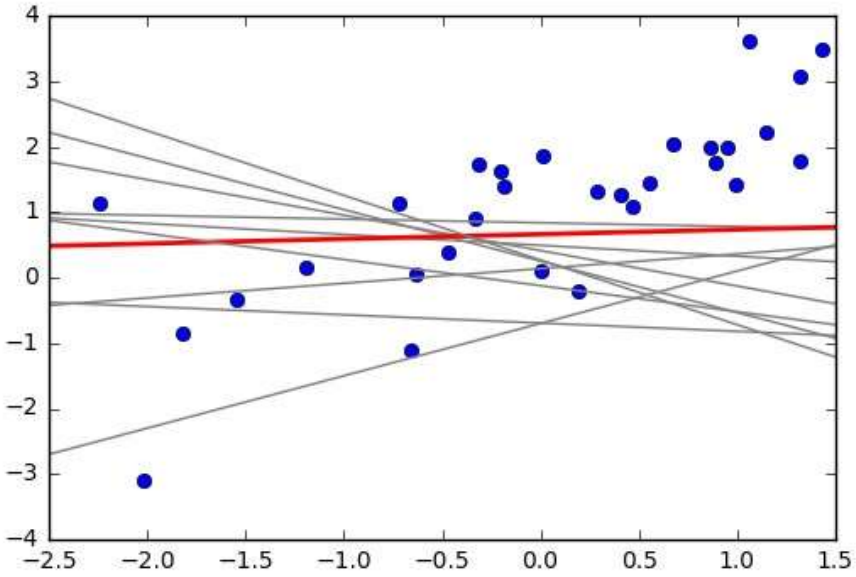
위의 그래프는 $w = 1, b = 1$인 직선 패턴에서 노이즈를 섞어 만든 데이터를 나타낸 것이다. 선형 모형에서 데이터에 대한 오차가 가장 적은 파라미터를 찾는다면, 데이터가 생성된 실제 패턴에 가까워질 것이다.

문제는 오차가를 가장 적은 파라미터를 찾는 방법이다. 이러한 종류의 문제를 최적화(optimization)이라고 한다. 최적화 문제는 주어진 제약 조건 아래 목적 함수의 값을 최소화 또는 최대화하는 문제를 말한다. 최소화하는 경우에 목적 함수를 손실 함수(loss function) 또는 비용 함수(cost function)라고 한다.

무작위 대입

최적화에는 여러 가지 알고리즘이 있을 수 있다. 가장 쉽게 생각해볼 수 있는 방법은 모든 파라미터를 하나씩 다 대입해보는 방법이다. 그러나 파라미터 값은 무한히 많기 때문에 전부 대입해볼 수는 없다. 그렇다면 일정 범위를 정해서 그 범위 내에서 고르게 대입해보거나 무작위로 대입해보는 방법을 생각해볼 수 있다.

아래는 기울기를 -1에서 1까지, 절편도 -1에서 1까지 범위에서 무작위로 10가지씩 골라 직선으로 그린 것이다. 10개의 직선 중에 빨간색 직선($w = 0.07, b = 0.66$)이 MSE가 1.91로 가장 오차가 적다.



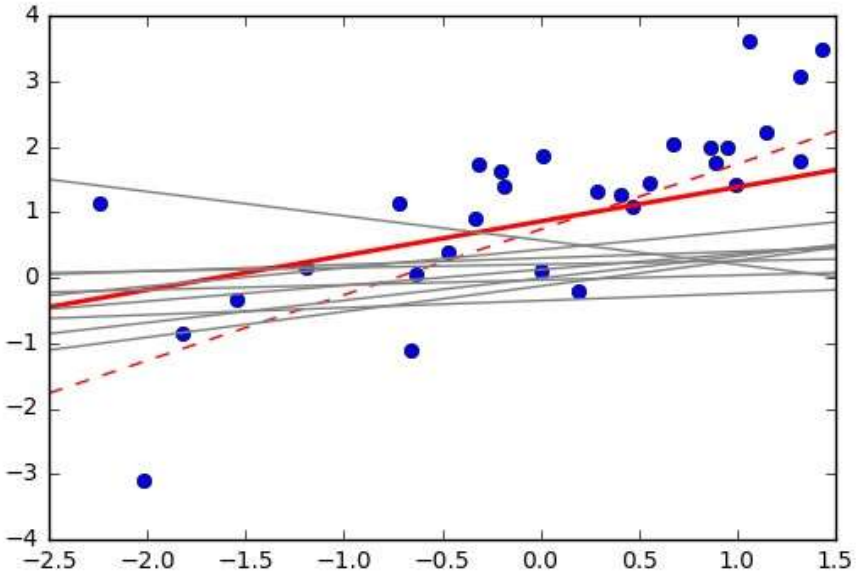
그렇지만 실제 패턴($w = 1, b = 1$)과는 꽤 거리가 멀어 보인다. 더 오차를 줄이려면 더 많은 직선을 무작위로 시도해봐야 한다. 그런데 그릴 수 있는 직선의 종류는 무한히 많기 때문에 이렇게 무작위로 시도하는 것은 매우 비현실적이다.

진화 알고리즘

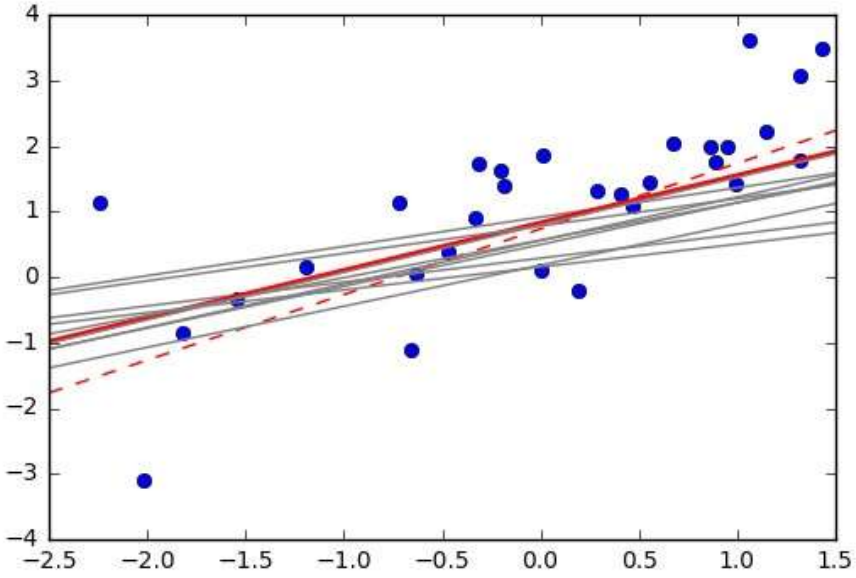
자연계에도 모형 적합과 비슷한 현상이 있다. 생물들은 모양이나 행동을 결정하는 유전자를 가지고 있다. 유전자는 생물의 파라미터라고 할 수 있다. 생물들은 세대를 거듭하면서 돌연변이가 생겨난다. 파라미터가 바뀌는 것이다. 그 중에 환경에 더 적합한 유전자를 가진 생물은 더 많은 후손을 남기게 된다. 오랜 시간이 걸리면 환경에 맞는 유전자만 남게 된다. 이것이 진화다.

생물의 진화를 흉내내어 진화 알고리즘(evolutionary algorithm)을 시도해보자. 일단 무작위로 파라미터를 여러 가지 대입해본다. 그 중에 가장 손실이 적은 파라미터를 고른다. 여기까지는 무작위 대입과 같다.

이제 무작위 탐색으로 찾은 가장 좋은 파라미터를 부모로 삼아 부모와 비슷하지만 조금 다른 자식들을 만들어낸다. 그리고 자식들 중에 가장 손실이 적은 파라미터를 찾는다. 아래의 경우에 새로 찾아낸 직선($w = 0.52, b = 0.86$)은 MSE=1.10으로 줄었고 파라미터도 실제와 더 비슷해졌다.



이런 과정을 다시 반복한다. 그러면 더 실제와 비슷한 직선($w = 0.72, b = 0.83$)을 찾았다. MSE=0.95 역시 줄었다.



이와 같은 과정을 계속해서 반복하다보면 점점 정답에 가까워져간다.

진화 알고리즘은 무작위로 시도하는 것보다는 낫지만 여전히 문제가 있다. 지구에 첫 생명이 나타난 후로 인류가 등장할 때까지는 약 30억년의 시간이 걸렸다. 진화는 뚜렷한 방향성이 없고 무작위적인 과정이기 때문에 속도가 느리다. 진화 알고리즘도 같은 이유로 속도가 느리다.

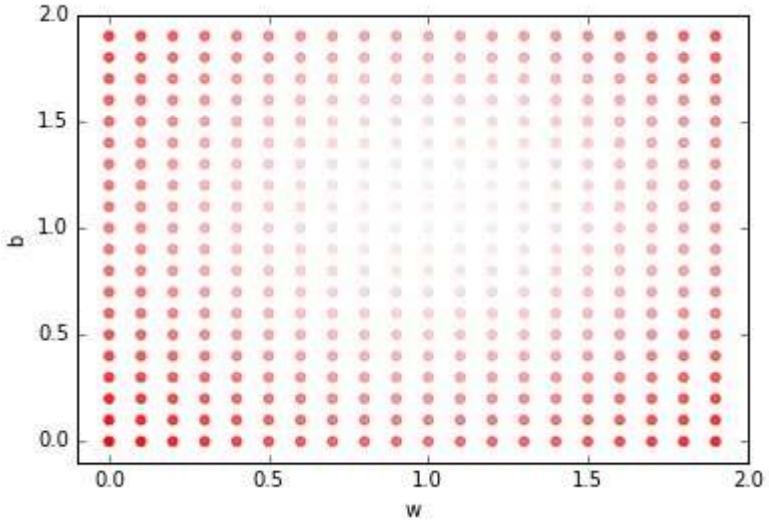
위의 예에서 절편의 변화를 보면 1세대 0.66에서 2세대 0.86으로 정답에 가까워졌지만 3세대에서는 0.83으로 정답에서 약간 멀어졌다. 무작위적으로 답을 찾기 때문에 항상 정답을 향해갈 수는 없는 것이다.

모형을 더 빨리 적합시키려면 파라미터를 개선하는 과정에 방향성을 더 할 필요가 있다.

2.3.2. 데이터 공간과 파라미터 공간

지금까지는 그래프를 그릴 때 가로축은 데이터의 x, 세로축은 데이터의 y로 그렸다. 데이터를 바탕으로 그리기 때문에 이러한 공간은 데이터 공간(data space)라고 한다.

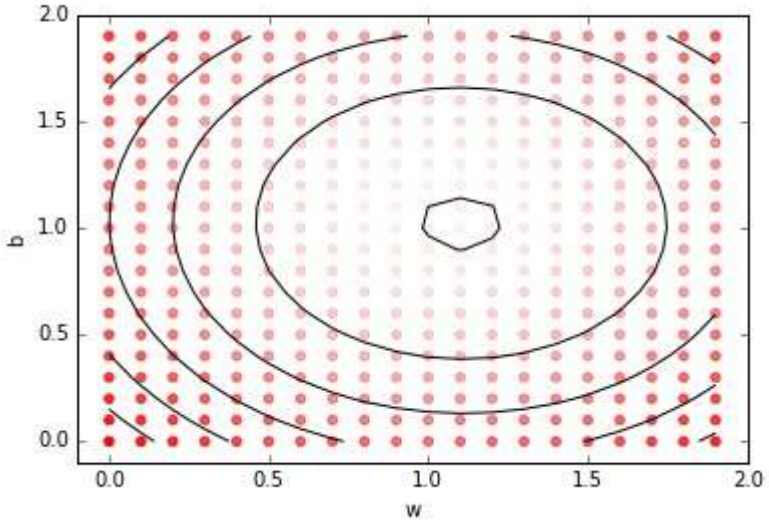
데이터 공간에서 하나의 직선은 w와 b 두 개의 파라미터에 따라 그려진다. 파라미터가 2개이기 때문에 데이터 공간에서 하나의 직선은 2차원 공간에 하나의 점으로 나타낼 수 있다. 이 공간을 파라미터 공간(parameter space)라고 한다. 아래 그림은 파라미터 공간의 예시이다.



위의 파라미터 공간에서 예를 들어 왼쪽 아래 구석의 빨간 점은 $w = 0, b = 0$이다. 따라서 데이터 공간에서는 기울기도 절편도 0인 직선이 된다.

각각의 직선마다 MSE를 구해서 손실이 큰 직선은 점의 색깔을 진하게, 손실이 작은 직선은 점을 옅게 칠했다. 정답에 해당하는 $w = 1, b = 1$ 근처에서는 오차가 적기 때문에 점이 매우 옅은 것을 볼 수 있다.

아래 그림은 MSE가 같은 점들을 이어 MSE의 등고선을 나타낸 것이다.



위의 등고선을 보면 파라미터 공간에서 MSE는 매끄럽게 변한다는 사실을 알 수 있다. 즉, 한 점과 그 주변은 MSE가 비슷하고, 어떤 방향으로 가면 MSE가 증가하고 반대 방향으로 가면 감소하는 식이다.

앞서 진화 알고리즘으로 3세대만에 찾아냈던 $w = 0.72, b = 0.83$를 기준으로 생각해보자. 위의 그림을 보면 w와 b를 증가시키는 방향으로 파라미터를 수정한다면 MSE가 더 감소하게 된다.

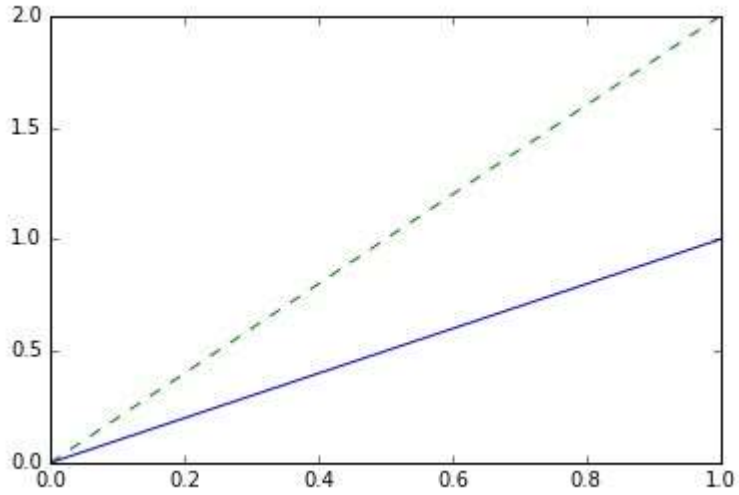
MSE는 어떤 방향으로 줄어들거나 늘어나기 때문에, 파라미터 공간에서 MSE가 줄어드는 방향성을 알 수 있다면 그 방향으로 파라미터를 개선하여 MSE를 최소화할 수 있다.

2.3.3. 경사하강법

경사하강법(gradient descent)은 파라미터를 더 손실이 적은 방향으로 개선해나가는 방법이다. 여기서 '경사'라는 말의 의미를 살펴보자.

직선의 기울기

기울기는 어떻게 정의할 수 있을까? 다음 두 직선을 비교해보자. 어느 쪽이 더 가파른가?



초록색 점선이 파란색 실선보다 더 **가파르다**고 할 것이다. 그렇다면 이를 수치적으로 표현할 수 있을까?

다들 알겠지만 수학에서는 관습적으로 가로축을 x라고 하고 세로축을 y라고 한다. 앞으로는 이 관습대로 읽을 것이다.

파란색 실선은 x가 0에서 1까지 변하는 동안 y가 0에서 1까지 변했다. 즉 x가 1만큼$(= 1 - 0)$ 변하는 동안, y도 1만큼 변한 것이다$(= 1 - 0)$. 반면, 초록색 실선은 같은 범위에서 y가 2만큼$(= 2 - 0)$ 변했다.

수학에서 변화량을 나타낼 때는 Δ라는 표기를 쓴다. Δ는 그리스어 알파벳으로 '델타(delta)'라고 읽는데, 변화(difference)와 첫 글자가 똑같이 d이기 때문에 그렇게 쓴다. 앞에서 한 말을 다시 반

복하면 $\Delta x = 1$일 때 파란 실선의 $\Delta y = 1$이고 초록 점선의 $\Delta y = 2$이다.

Δx에 비해서 Δy가 클 수록 기울기가 가파르다는 것은 직관적으로 알 수 있다. 이것을 수식으로 나타내면 이렇게 쓸 수 있다.

$$\frac{\Delta y}{\Delta x}$$

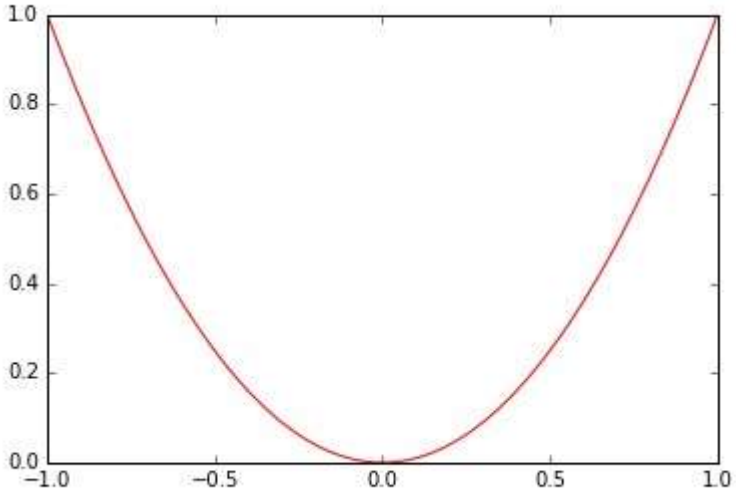
기울기가 +라는 것은 x가 커질 수록 y도 커진다는 뜻이다. 기울기가 -면 반대로 x가 커지면 y가 작아진다.

그림으로 그린다면 기울기가 +일 때는 오른쪽으로 올라가는(우상향)하는 형태가 되고,

기울기가 -일때는 오른쪽으로 내려가는(우하향)하는 형태가 된다.

곡선의 기울기

이번에는 곡선의 기울기에 대해 생각해보자. 아래 곡선은 $y = x^2$인 곡선이다.



곡선은 휘어있지만 한 점을 크게 확대해보면 직선과 별 차이가 없다. 마치 지구는 둥글지만 우리가 땅 위에서 보면 평평해 보이는 것과 같다.

곡선을 확대해보려면 Δx를 아주 작게 만드는 것과 같다. 곡선을 확대하면 x의 범위가 좁아지기 때문이다. Δx를 아주 작게 만든 경우는 특별히 Δ 대신 d를 쓴다. 그러니까 곡선의 기울기는 $\Delta y / \Delta x$ 대신에

$$\frac{dy}{dx}$$

를 사용한다. 위 식은 **x에 대한 y의 미분**(deravative)이라고 읽는다.

직선과 달리 곡선은 위치에 따라 기울기가 다르다. 위의 빨간 곡선을 보면 0 근처에서는 기울기가 완만하지만, 왼쪽이나 오른쪽은 기울기가 급하다. 또한, 0의 왼쪽은 오른쪽으로 내려가는 형태(우

하향)이므로 기울기가 -이고, 오른쪽은 오른쪽으로 올라가는 형태(우상향)이므로 기울기가 +이다.

따라서 x에 대한 y의 미분도 식으로 나타낼 수 있다. $y = x^2$일 때 x에 대한 y의 미분은 다음과 같다.

$$\frac{dy}{dx} = 2x$$

따라서 $x = 0$이면 기울기도 0이고 $x = 1$이면 기울기는 2, $x = -1$이면 기울기는 -2가 된다.

경사하강법

곡선의 기울기가 우리에게 왜 중요할까? 기계학습에서 대부분의 모형은 오차를 가장 작아지도록 하는 파라미터를 찾는 방식으로 학습이 이뤄진다.

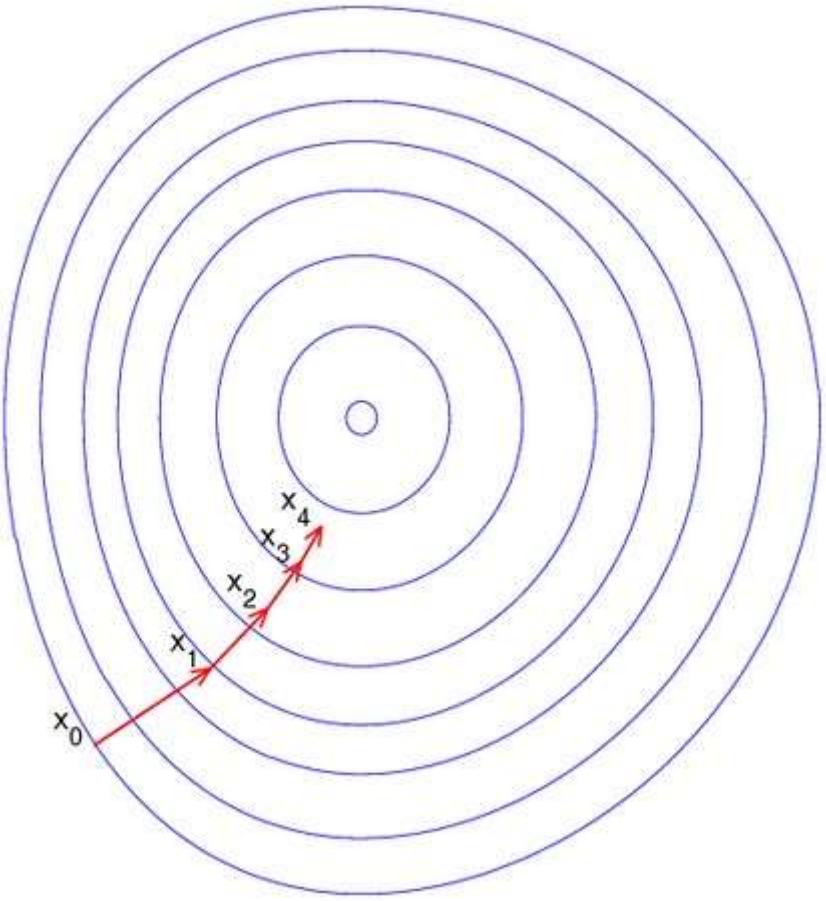
중고등학교 때 배우는 간단한 방정식들과 달리 기계학습에서 사용하는 모형과 오차는 매우 복잡한 형태여서 최적의 파라미터를 구할 수 있는 간단한 공식이 있는 경우가 거의 없다.

따라서 파라미터를 임의로 정한 다음에 조금씩 변화시켜가며 오차를 점점 줄여가는 방법으로 최적의 파라미터를 찾아간다. 이때 미분을 사용한다.

앞에서 기울기가 +이면 그래프가 우상향, 기울기가 -이면 우하향한다고 이야기했다. 따라서 기울기가 +일 때 x를 감소시키면 y는 감소하게 된다. 반대로, 기울기가 -일 때는 x를 증가시켜야 y를 감소시킬 수 있다.

x가 파라미터이고, y가 오차라고 해보자. 파라미터에 대한 오차의 미분을 구해서 이것이 +라면 파라미터를 줄여야 한다. 반대로 -면 파라미터를 키워야 한다. 이렇게 파라미터를 줄이고 키우는 과정을 더이상 오차가 줄어들지 않을 때까지 반복하는 것이다.

이런 방법을 기울기를 따라 조금씩 오차를 낮춰간다고 해서 **경사 하강법**(gradient descent)이라고 한다. 여기서 gradient에는 수학적인 의미가 있는데 일단은 기울기나 미분과 같은 의미라고 이해해두자.



선형 회귀의 경사하강법*

아래의 설명은 잘 이해하지 못할 경우 건너 뛰어도 좋다.

MSE의 식은 아래와 같다.

$$E(w) = \frac{1}{N}\sum(y_i - \hat{y}_i)^2$$
$$\hat{y}_i = wx_i + b$$

우선 미적분학에서 연쇄 규칙(chain rule)이라는 규칙이 있다.

$$\frac{dE}{dw} = \frac{dE}{df}\frac{df}{dw}$$

다음으로 x^n 꼴의 경우 nx^{n-1}의 꼴로 미분이 된다. 따라서 $f = y_i - \hat{y}_i$라 하면

$$\frac{dE}{df} = 2(y - \hat{y}_i)$$

가 되고

$$\frac{df}{dw} = -x_i$$

가 된다($w^0 = 1$). 이제 연쇄 규칙의 식에 대입하면 다음과 같은 기울기를 얻는다.

$$\frac{dE}{dw} = \frac{dE}{df}\frac{df}{dw} = \frac{1}{N}\sum -2x_i(y_i - \hat{y}_i)$$

위의 식에서 $\frac{1}{N}\sum$은 평균을 구하는 부분이니 무시하고 2도 무시하면 핵심은 $-x_i(y_i - \hat{y}_i)$이다. 이것을 하나씩 뜯어보자.

$x_i > 0, (y_i - \hat{y}_i) > 0$인 경우 둘다 +이므로 오차의 기울기는 -가 된다. 오차의 기울기가 -이면 우하향하는 형태가 되므로 w를 키워야 오차가 줄어든다.

다른 방향에서 살펴보자. 직선의 오른쪽($x > 0$) 에서 실제값(y_i)이 예측값($\hat{y}_i$)보다 더 위에($y_i - \hat{y}_i > 0$) 있다. 이 차이를 좁히려면 직선이 더 우상향하도록 w를 키워야 한다.

신기하게도 경사하강법은 직관과 잘 맞는 것을 알 수 있다. 물론 수학적으로는 당연한 결과다.

학습률

경사하강법은 최종적으로 다음과 같은 형태의 식을 따라 파라미터를 업데이트 한다.

$$w_{t+1} = w_t - \eta\frac{dE}{dw}$$

여기서 η는 학습률이라고 한다. 학습률이 크면 경사를 많이 따라가게 된다. 따라서 학습도 많이 일어난다. 단점은 최적점 근처에서 지나치게 많이 파라미터를 업데이트 하면 최적점을 지나칠 수 있다는 것이다.

반대로 학습률이 작으면 경사를 적게 따라가게 된다. 이 경우 앞의 단점은 해결되지만 전반적으로 학습이 느려지게 된다.

이를 해결하여 학습률을 조정하는 방법은 다음에 알아보기로 한다.

2.3.4. 국소 최적과 전역 최적

어떤 파라미터가 모든 경우를 통틀어 최적인 경우를 **전역 최적**(global optimum)이라 하고, 주변의 다른 파라미터들보다만 더 나은 경우를 **국소 최적**(local optimum)라고 한다.

(토막 상식: 참고로로 최적을 뜻하는 optimum은 라틴어이다. 복수로는 optima. 비슷하게 최소는 minimum, minima 최대는 maximum, maxima라고 한다. 단수 대신 복수로 쓰는 경우도 많다)

국소 최적이든 전역 최적이든 최적점에서는 기울기가 항상 0이 된다. 왜냐하면 최적점에서는 어떤 방향으로 파라미터를 바꾸어봐도 오차가 더 이상 줄어들지 않기 때문이다. 따라서 파라미터가 국소 최적에 빠지면 더이상 파라미터를 개선할 수 없고, 학습이 거기서 멈추게 된다.

다행히도 선형 회귀의 경우에는 국소 최적이 곧 전역 최적이기 때문에 항상 오차가 가장 적은 파라

미터를 찾을 수 있다. 그러나 딥러닝과 같이 복잡한 모형에서는 많은 국소 최적해가 존재하기 때문에 전역 최적에 도달한다는 보장이 없다.

이를 보완하기 위해 다양한 전략들이 동원된다. 가장 간단한 방법은 출발점을 여러 가지로 다양하게 시도하는 것이다. 경사하강법은 가장 가까운 국소최적으로 수렴하므로 출발점이 다양하다면 다양한 국소최적에 수렴할 수 있다. 이러한 전략들이 전역 최적에 수렴하는 것을 보장해주지는 않지만, 국소최적 중에서 더 나은 국소 최적에 수렴하도록 도와줄 수는 있다.

2.3.5. 닫힌 해

실제로는 선형 회귀는 **닫힌 해**(closed form solution), 즉 공식으로 한 번에 구할 수 있는 최적해가 존재한다. 경사하강법으로 한 단계씩 파라미터를 업데이트해가면서 최적 파라미터를 찾을 필요는 없다는 이야기다. 실제로도 선형 회귀에 굳이 경사하강법을 적용하지는 않는다.

그러나 선형 회귀 모형을 더 복잡하게 확장하거나 데이터가 많아지면 더이상 닫힌 해가 없기 때문에 경사하강법이 필요하게 된다.

2.4. 선형 회귀

2.4.1. 데이터 불러오기

```
data.url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/w
```

```
red.wine = read.csv(url(data.url), sep = ';')
```

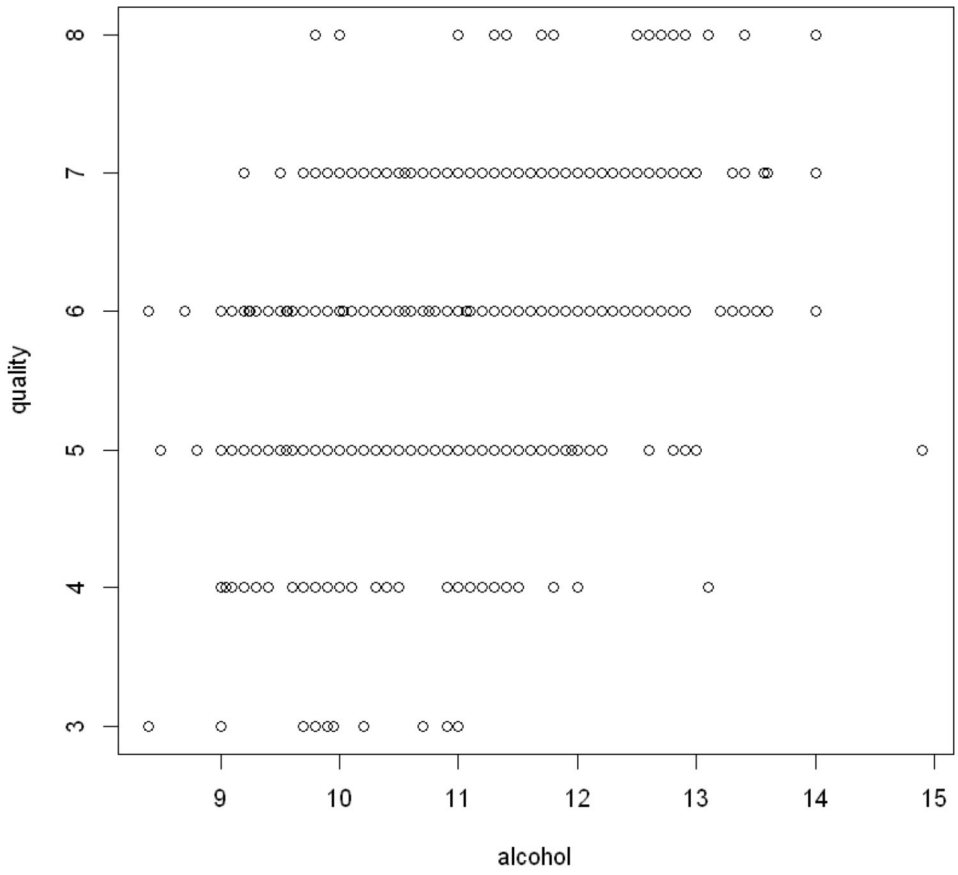
```
head(red.wine)
```

```
fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
1           7.4             0.70        0.00            1.9     0.076
2           7.8             0.88        0.00            2.6     0.098
3           7.8             0.76        0.04            2.3     0.092
4          11.2             0.28        0.56            1.9     0.075
5           7.4             0.70        0.00            1.9     0.076
6           7.4             0.66        0.00            1.8     0.075
  free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
1                  11                   34  0.9978 3.51      0.56     9.4
2                  25                   67  0.9968 3.20      0.68     9.8
3                  15                   54  0.9970 3.26      0.65     9.8
4                  17                   60  0.9980 3.16      0.58     9.8
5                  11                   34  0.9978 3.51      0.56     9.4
6                  13                   40  0.9978 3.51      0.56     9.4
  quality
1       5
2       5
3       5
4       6
5       5
6       5
```

2.4.2. 시각화

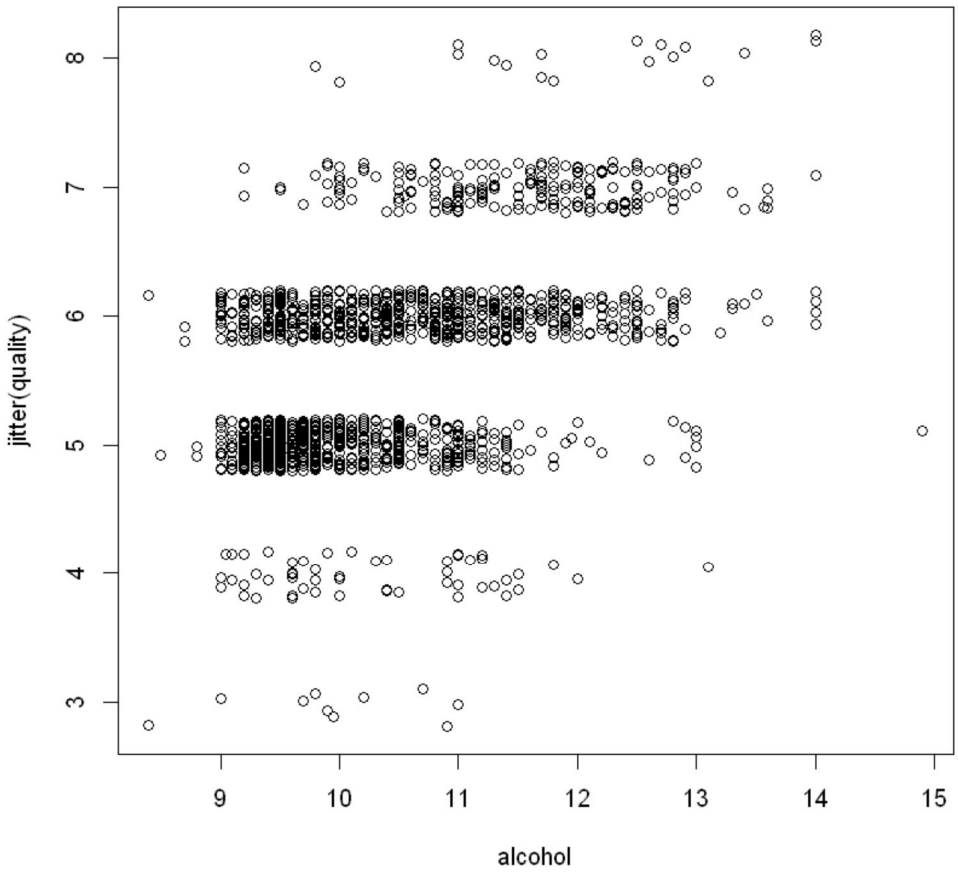
```
plot(quality ~ alcohol, red.wine)
```

plot without title




```
plot(jitter(quality) ~ alcohol, red.wine)
```

plot without title



2.4.3. 단순 선형 회귀

```
model = lm(quality ~ alcohol, red.wine)
```

```
summary(model)
```

```
Call:
lm(formula = quality ~ alcohol, data = red.wine)

Residuals:
    Min      1Q  Median      3Q     Max
-2.8442 -0.4112 -0.1690  0.5166  2.5888

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.87497    0.17471   10.73   <2e-16 ***
alcohol      0.36084    0.01668   21.64   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7104 on 1597 degrees of freedom
Multiple R-squared:  0.2267,    Adjusted R-squared:  0.2263
F-statistic: 468.3 on 1 and 1597 DF,  p-value: < 2.2e-16
```

2.4.4. 중다 선형 회귀

```
model = lm(quality ~ alcohol, red.wine)
```

```
summary(model)
```

```
Call:
lm(formula = quality ~ alcohol, data = red.wine)

Residuals:
    Min      1Q  Median      3Q     Max
-2.8442 -0.4112 -0.1690  0.5166  2.5888

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.87497    0.17471   10.73   <2e-16 ***
alcohol      0.36084    0.01668   21.64   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.7104 on 1597 degrees of freedom
Multiple R-squared:  0.2267,    Adjusted R-squared:  0.2263
F-statistic: 468.3 on 1 and 1597 DF,  p-value: < 2.2e-16
```

2.4.5. caret

```
library(caret)
```

2.4.6. 데이터 분할

```
set.seed(1234)
```

```
idx = createDataPartition(red.wine$quality, p=.8, list=F)
```

```
data.train = red.wine[idx, ]
data.test = red.wine[-idx, ]
```

2.4.7. 훈련

```
lin.model = train(
    quality ~ .,
    data=data.train,
    method='lm')
```

2.4.8. 성능 검증

훈련용 데이터

```
y.train.pred = predict(lin.model, data.train)
```

```
RMSE(data.train$quality, y.train.pred)
```

```
[1] 0.6522562
```

```
R2(data.train$quality, y.train.pred)
```

```
[1] 0.3592011
```

테스트용 데이터

```
y.pred = predict(lin.model, data.test)
```

```
RMSE(data.test$quality, y.pred)
```

```
[1] 0.6211207
```

```
R2(data.test$quality, y.pred)
```

```
[1] 0.360987
```