

03-2

경사 하강법(Gradient descent) 으로 학습하기

특성이 많은 경우 그래프로 나타내기에 너무 복잡하다.
따라서, 보통 여러 개의 특성을 가진 데이터에서 특성의 개수를 1,2개만 사용하여 2차원이나 3차원 그래프로 그리는 경우가 많다.

경사 하강법은 모델이 데이터를 잘 표현할 수 있도록 기울기(변화율)를 사용하여 모델을 조금씩 조정하는 최적화 알고리즘이다.

경사 하강법은 많은 양의 데이터에 사용하기 좋은 알고리즘이다.

모델 $y = ax + b$ 에서

기울기 $a \Rightarrow$ 가중치 w

$y \Rightarrow$ 예측값 y_{hat}

: 우리가 구한 모델에 새로운 입력값을 넣었을 때 나온 출력

$\Rightarrow y_{\text{hat}} = wx + b$

cf) y : 타깃 데이터

<훈련 데이터에 잘 맞는 w와 b를 찾는 방법>

(1) 무작위로 w와 b를 정합니다.

(무작위로 모델 만들기)

(2) x에서 샘플 하나를 선택하여 y_hat을 계산합니다.

(무작위로 모델 예측하기)

(3) y_hat과 선택한 샘플의 진짜 y를 비교합니다.

(예측값과 진짜 정답 비교하기)

(4) y_hat이 y와 더 가까워지도록 w,b를 조정합니다.

(모델 조정하기)

(5) 모든 샘플을 처리할 때까지 다시 (2)~(4) 과정을 반복합니다.

(1) w와 b 초기화하기

w와 b를 무작위로 초기화합니다.

여기에서는 간단하게 두 값을 모두 실수 1.0으로 정하겠습니다.

$$w=1.0$$

$$b=1.0$$

(2) 훈련 데이터의 첫 번째 샘플 데이터로 y_hat 얻기

임시로 만든 모델 $y_{\text{hat}} = w(=1.0)x + b(=1.0)$

로 훈련 데이터의 첫 번째 샘플 $x[0]$ 에 대한 y_{hat} 을 계산해 보겠습니다.

```
y_hat=w*x[0]+b  
print("x[0]=",x[0])  
print("y_hat=",y_hat)
```

```
x[0]= 0.0616962065186885  
y_hat= 1.0616962065186886
```

(3) 타깃과 예측 데이터 비교하기

첫 번째 샘플 $x[0]$ 에 대응하는 타깃값 $y[0]$ 을 출력하여 y_{hat} 의 값과 비교합니다.

```
print("y=",y[0],"\n\n")
```

```
y= 151.0
```

(4) w값을 조절해 예측값 y_hat 바꾸기

x[0]에 대응하는 예측값 $y_{\text{hat}}=1.06$

타깃값 $y=151$

-> y_{hat} 과 y 의 차이를 줄이는 방향으로 w 를 조절해야함.

w 와 b 를 조금씩 변경해서 y_{hat} 이 증가하는지 또는 감소하는지 살펴봅시다.

먼저 w 를 0.1만큼 증가시키고 y_{hat} 의 변화량을 관찰해 봅시다.

```
w_inc=w+0.1  
y_hat_inc=w_inc*x[0]+b  
print("y_hat_inc=",y_hat_inc,"\\n\\n")
```

$y_{\text{hat}}_{\text{inc}}= 1.0678658271705574$

-> $y_{\text{hat}}=1.0616$

$y_{\text{hat}}_{\text{inc}}=1.0678$

로 조금 증가하였습니다.

(5) w값을 조정한 후 예측값 증가 정도 확인하기

w 가 0.1만큼 증가했을 때, y_{hat} 이 얼마나 증가했는지 계산해 보겠습니다.

y_{hat} 이 증가한 양을 w 가 증가한 양으로 나누면 됩니다.

```
#변화율
```

```
w_rate=(y_hat_inc-y_hat)/(w_inc-w)  
print("w_rate=",w_rate,"\\n\\n")
```

w_rate= 0.061696206518688734

-> w_rate : 첫 번째 훈련 데이터 x[0]에 대한 w의 변화율.

w_rate는 훈련 데이터의 첫 번째 샘플인 x[0]과 동일하다는 것을 알 수 있습니다.

위의 경우 $y_{\text{hat}} (=1.06)$ 의 값은 $y (=151)$ 보다 작으므로 y_{hat} 의 값을 증가시켜야 합니다.

이 때 변화율 $w_{\text{rate}} (=0.06)$ 은 양수이므로 w 값을 증가시키면 y_{hat} 의 값을 증가시킬 수 있습니다.

만약 변화율이 음수일 때 y_{hat} 의 값을 증가시켜야 한다면 w 값을 감소시켜야 합니다.

하지만 이 방법은 변화율이 양수일 때와 음수일 때를 구분해야 하므로 번거롭습니다.

y에 가까운 y_{hat} 을 출력하는 (=잘 예측하는) 모델 (=w와 b)을 찾아내는 것이 선형 회귀의 목표입니다.

지금부터는 w와 b를 변화율로 업데이트 하는 방법을 알아봅시다.

<변화율로 가중치 업데이트하기>

- 변화율이 양수일 때 가중치를 업데이트하는 방법
이 상황에서는 w가 증가하면 y_{hat} 도 증가합니다.
이 때 변화율을 w에 더하는 방법으로 w를 증가시킬 수 있습니다.
- 변화율이 음수일 때 가중치를 업데이트하는 방법
이 상황에서는 w가 증가하면 y_{hat} 은 감소합니다.
 \Leftrightarrow w가 감소하면 y_{hat} 은 증가합니다.

이 때 변화율을 w에 더하는 방법으로 w를 증가시킬 수 있습니다.

즉, 가중치 w를 업데이트하는 방법은 두 경우 모두
 $w=w+w_rate$
입니다.

```
#w 업데이트  
w_new=w+w_rate  
print("w_new=",w_new,"\\n\\n")
```

w_new= 1.0616962065186888

```
# w=1.0 , w_rate=0.06  
# w_new=w+w_rate=1.06
```

<변화율로 절편 업데이트하기>

이번에는 절편 b 에 대한 변화율($=b_rate$)을 구한 다음 변화율로 b 를 업데이트하겠습니다.

b 를 0.1만큼 증가시킨 후 $y_{\hat{}}^{}$ 이 얼마나 증가했는지 계산하고 변화율도 계산합니다.

```
b_inc=b+0.1  
y_hat_inc=w*x[0]+b_inc  
print("y_hat_inc=",y_hat_inc,"\\n\\n")  
  
#변화율  
b_rate=(y_hat_inc-y_hat)/(b_inc-b)  
print("b_rate=",b_rate,"\\n\\n")
```

```
y_hat_inc= 1.1616962065186887
```

```
b_rate= 1.0
```

-> 변화율 b_rate 은 1입니다.

-> 즉, b가 1만큼 증가하면 y_hat도 1만큼 증가합니다.

즉, 절편 b를 업데이트하는 방법은

[b=b+1](#)

입니다.

```
#b 업데이트  
b_new=b+1  
print("b_new=",b_new,"\\n\\n")
```

```
b_new= 2.0
```

```
# b=1.0, b_rate=1.0
```

```
# b_new=b+b_rate=2.0
```

그런데 이 방법은 다음 두 상황에 대해 적합하게 대처하지 못하기 때문에, 조금 수동적인 방법입니다.

- y_{hat} 이 y 에 한참 미치지 못하는 값인 경우,
 w 와 b 를 더 큰 폭으로 수정할 수 없습니다.
- y_{hat} 이 y 보다 커지면 y_{hat} 을 감소시키지 못합니다.

-> 다음 실습에서는 w 와 b 를 더 능동적으로 업데이트하는 방법인 "오차 역전파"에 대해 알아보겠습니다.

<오차 역전파로 가중치와 절편을 업데이트하기>

오차 역전파(backpropagation) :

y 와 y_{hat} 의 차이를 이용하여 w 와 b 를 업데이트합니다.

오차 역전파라는 이름에서 알 수 있듯이,
이 방법은 오차가 연이어 전파되는 모습으로 수행됩니다.

앞의 예제에서는 변화율만으로 w 와 b 를 업데이트했습니다.
이번에는 y 에서 y_{hat} 을 뺀 오차의 양을 변화율에 곱하는
방법으로 w 를 업데이트해 보겠습니다.

이렇게 하면 y_{hat} 이 y 보다 많이 작은 경우 w 와 b 를 많이
바꿀 수 있습니다.

또, y_{hat} 이 y 보다 커지면 w 와 b 의 방향도 바꿔줍니다.

(1) 오차와 변화율을 곱하여 가중치 업데이트하기

먼저 $x[0]$ 일 때 w 의 변화율($=w_rate$)과

b의 변화율(=b_rate)에 오차를 곱한 다음,
업데이트된 w_new와 b_new를 출력해 보겠습니다.

결과값을 보면 w와 b가 각각 큰 폭으로 바뀌었음을 알 수 있습니다.

```
print("<x[0],y[0] 에 따른 오차(y_hat)를 이용하여 w_new 와 b_new 업데이트>\n")  
err=y[0]-y_hat  
print("err=",err,"\\n")  
w_new=w+(w_rate*err) # => w+(x[0]*err)  
b_new=b+(b_rate*err) # => b+(1*err)  
print("w_new=",w_new,"\\n")  
print("b_new=",b_new,"\\n\\n")
```

```
<x[0],y[0] 에 따른 오차(y_hat)를 이용하여 w_new 와 b_new 업데이트>  
err= 149.9383037934813  
w_new= 10.250624555904514  
b_new= 150.9383037934813
```

```
# w=1.0 -> w_new=10.25  
# b=1.0 -> b_new=150.94
```

(2) 두 번째 샘플 x[1]을 사용하여 오차를 구하고
새로운 w와 b를 구해 보겠습니다.

앞에서 w_rate 식을 정리했을 때 샘플값과 같아진다는 것을 알았으므로, 앞으로는 w_rate를 별도로 계산하지 않고 샘플값을 그대로 사용합니다.

즉, 여기서는 w_rate=x[1] 입니다.

```

print("<x[1],y[1] 에 따른 오차(y_hat)를 이용하여 w_new 와 b_new 업데이트>\n")
y_hat=w_new*x[1]+b_new
err=y[1]-y_hat
print("err=",err,"\\n")
w_new=w_new+(x[1]*err)
b_new=b_new+(1*err)
print("w_new=",w_new,"\\n")
print("b_new=",b_new,"\\n\\n")

```

<x[1],y[1] 에 따른 오차(y_hat)를 이용하여 w_new 와 b_new 업데이트>

err= -75.41066251735467

w_new= 14.132317616381767

b_new= 75.52764127612664

w_new=10.25 -> w_new=14.13

b_new=150.94 -> b_new=75.53

이런 방식으로 모든 샘플을 사용해 가중치와 절편을 업데이트합니다.

(3) 전체 샘플을 반복하기

```

print("<반복>\n")
for x_i,y_i in zip(x,y): #zip 함수는 여러 개의 배열에서 동시에 요소를 하나씩 꺼내준다.
    y_hat=w*x_i+b
    err=y_i-y_hat
    w=w+(x_i*err)
    b=b+(1*err)
print("w=",w,"\\n")
print("b=",b,"\\n\\n")

```

<반복>

w= 587.8654539985689

b= 99.40935564531424

(4) 과정 3을 통해 얻어낸 모델이 전체 데이터 세트를 잘 표현하는지 그래프를 그려 알아보겠습니다.

산점도 위에 w와 b를 사용한 직선을 그려보면 금방 알 수 있습니다.

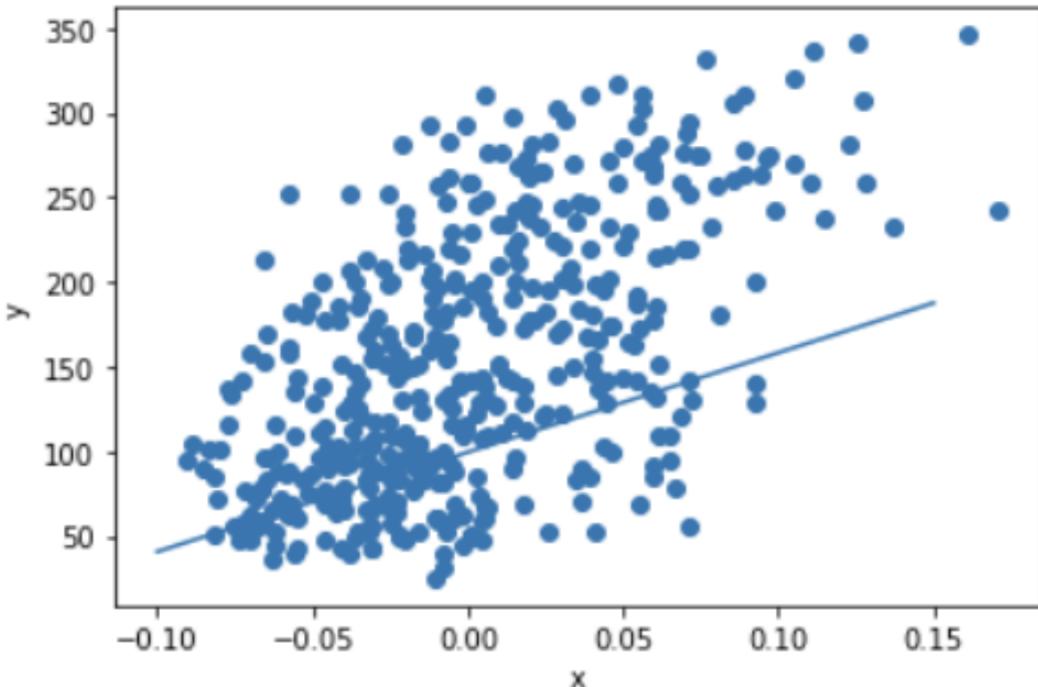
직선 그래프를 그리려면 시작점과 종료점의 x 좌표와 y 좌표를 plot() 함수에 전달하면 됩니다.

x 좌표 2개 : -0.1 , 0.15

y 좌표 2개 : -0.1*w+b , 0.15*w+b

```
plt.scatter(x,y)
pt1=(-0.1,w*(-0.1)+b)
pt2=(0.15,w*(0.15)+b)
plt.plot([pt1[0],pt2[0]],[pt1[1],pt2[1]])
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

pt1 , pt2 는 tuple



-> 직선이 산점도를 어느 정도 잘 표현한 것 같습니다.
어떻게 하면 좀 더 그럴싸한 직선을 얻을 수 있을까요?

(5) 여러 에포크를 반복하기

보통 경사 하강법에서는 주어진 훈련 데이터로 학습을
여러 번 반복합니다.

이렇게 전체 훈련 데이터를 모두 이용하여 한 단위의 작업
을 진행하는 것을 특별히 에포크(epoch) 라고 부릅니다.

일반적으로 수십~수천 번의 에포크를 반복합니다.

100번의 에포크를 반복하면서 직선이 어떻게 이동하는지
확인해 보겠습니다.

```

print("<100회의 에포크 반복>\n")
for i in range(1,100):
    for x_i,y_i in zip(x,y): #zip 함수는 여러 개의 배열에서 동시에 요소를 하나씩 꺼내준다.
        y_hat=w*x_i+b
        err=y_i-y_hat
        w=w+(x_i*err)
        b=b+(1*err)
print("w=",w,"\\n") #w=913.5973364345905
print("b=",b,"\\n\\n") #b=123.39414383177204

```

<100회의 에포크 반복>

w= 913.5973364345905

b= 123.39414383177204

100번의 에포크를 반복하여 찾은

w=913.60

b=123.40

(6) 모델로 예측하기

위의 과정을 수행하여 도출한 모델은 다음과 같습니다.

$$y_{\text{hat}} = 913.6x + 123.4$$

이번에는 입력 x에 없었던 새로운 데이터가 발생했다고 가정해 보겠습니다.

이 데이터에 대해 예측값을 얻으려면 모델에 x를 넣고 계산하기만 하면 됩니다.

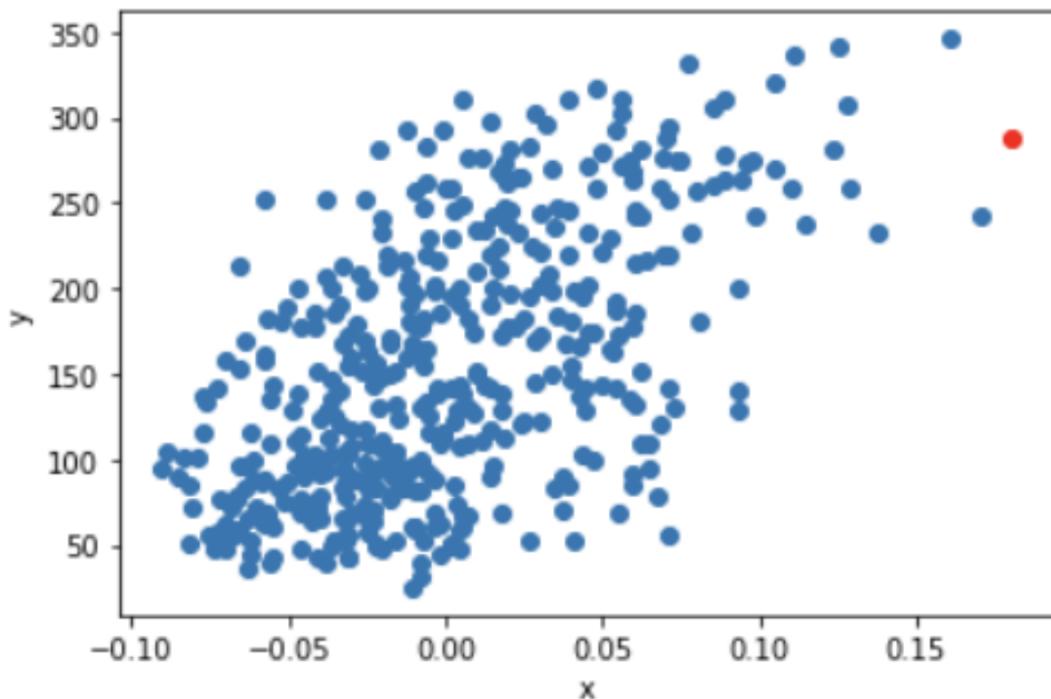
x=0.18 일 때 y_hat 의 값을 예측해 보겠습니다.

```
#모델 완성  
# y_hat=(913.6)*x+(123.4)  
  
#예측 수행  
x_new=0.18  
y_pred=w*x_new+b  
print("x_new:0.18에 대한 예측값 y_pred=",y_pred,"\\n\\n")
```

x_new:0.18에 대한 예측값 y_pred= 287.8416643899983

다음은 이 데이터를 산점도 위에 나타낸 것입니다.

```
plt.scatter(x,y)  
plt.scatter(x_new,y_pred,c='red')  
plt.xlabel('x')  
plt.ylabel('y')  
plt.show()
```



그래프 우측 상단에 붉은 색으로 표시된 점이 새 데이터.

