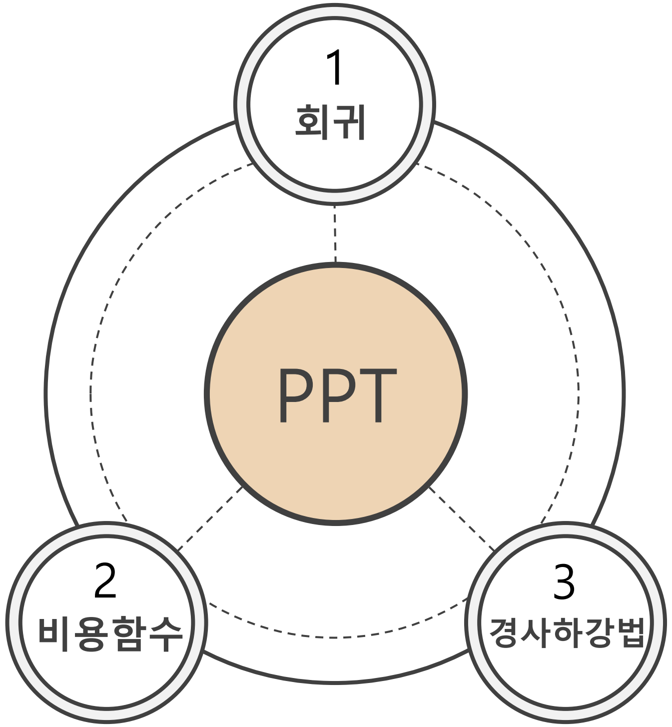
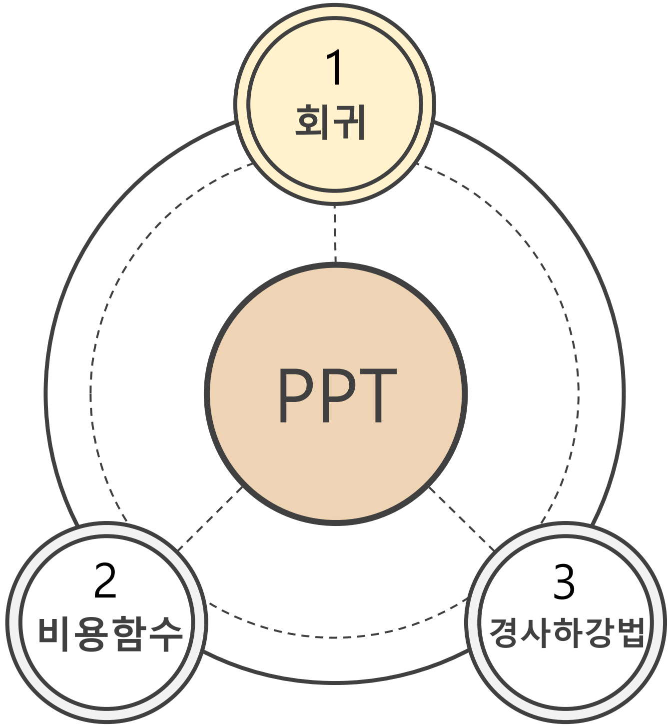
경사하강법

91714167 유재겸

목차



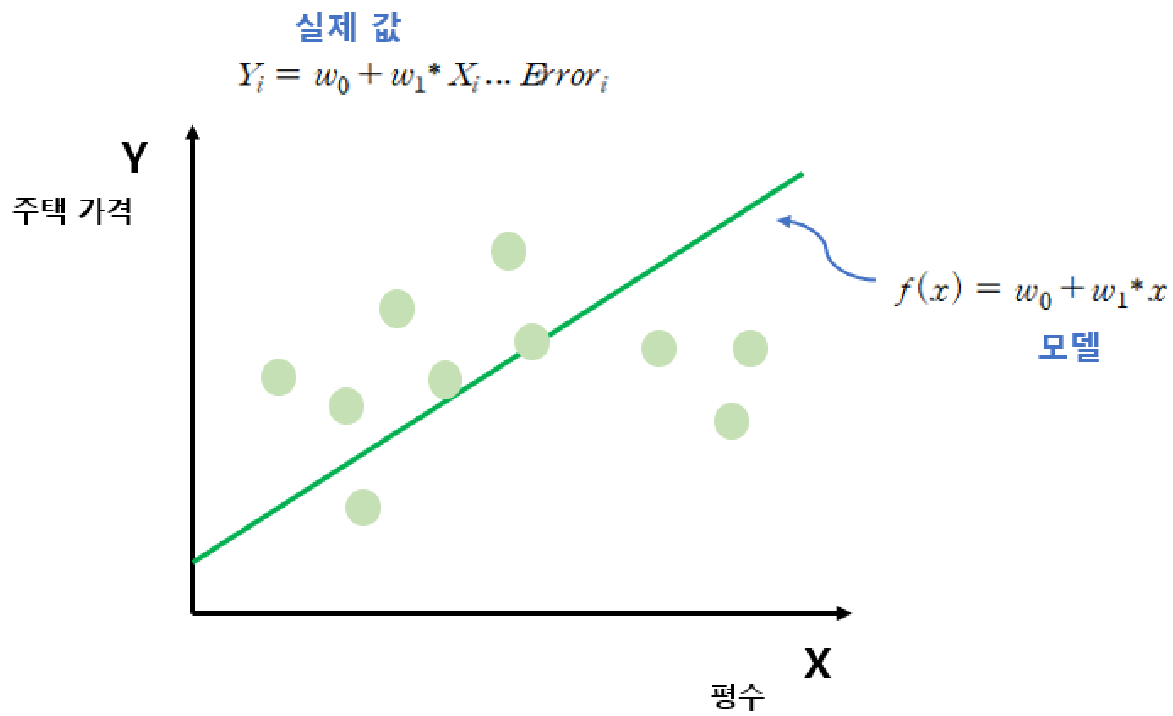

1
회귀
PPT
2
비용함수
3
경사하강법

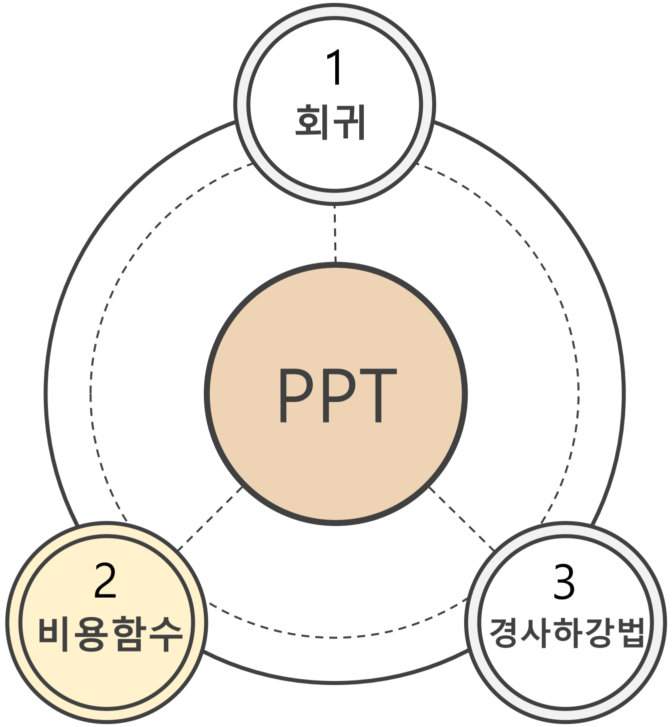
1

회귀

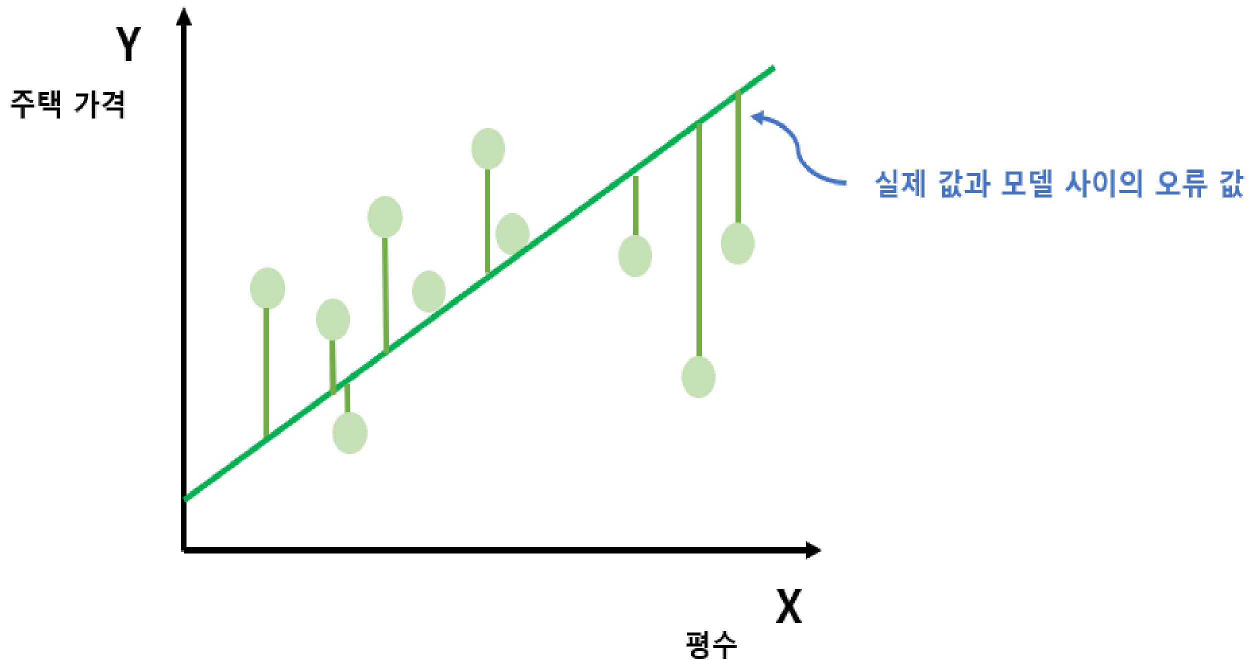
데이터 값이 평균과 같은 일정한 값으로 돌아가려는 경향을 이용한 통계학 기법

1
회귀



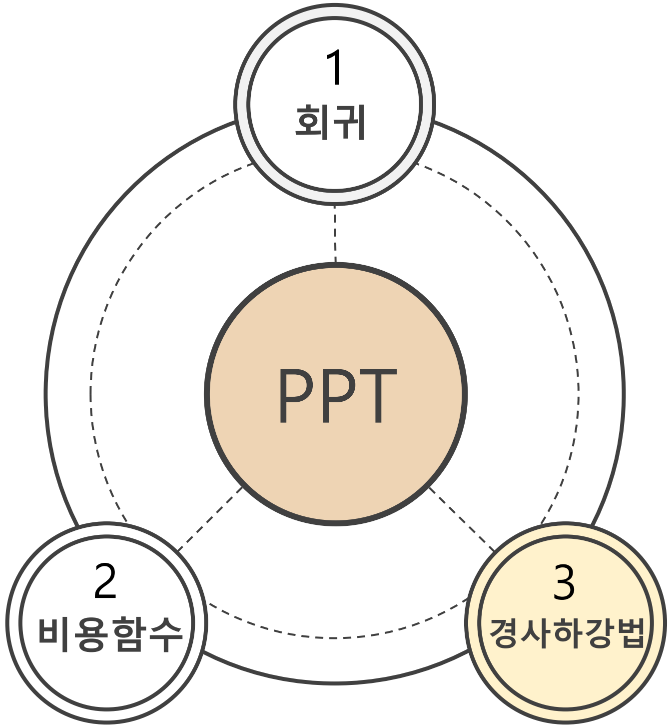

1
회귀
PPT
2
비용함수
3
경사하강법

2
비용함수

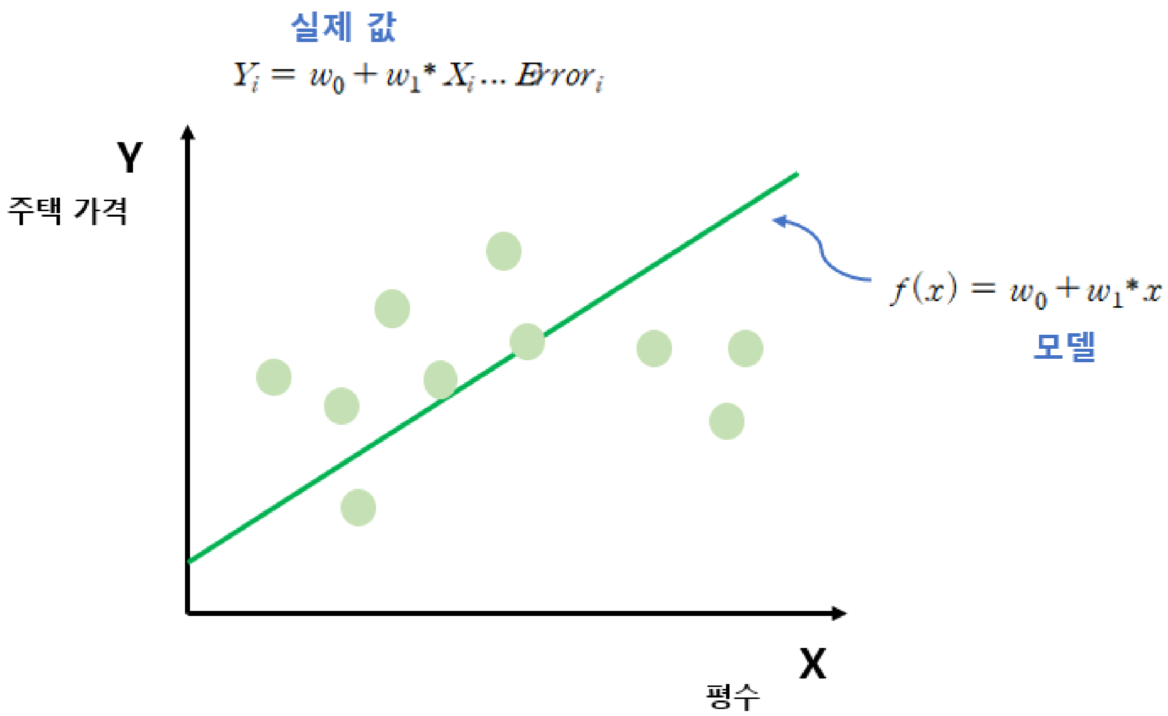


2
비용함수

$$RSS(w_0,\ w_1) = \frac{1}{N}\Sigma_{i=1}^{N}(y_i - (w_{0+}w_1 * x_i))^2$$

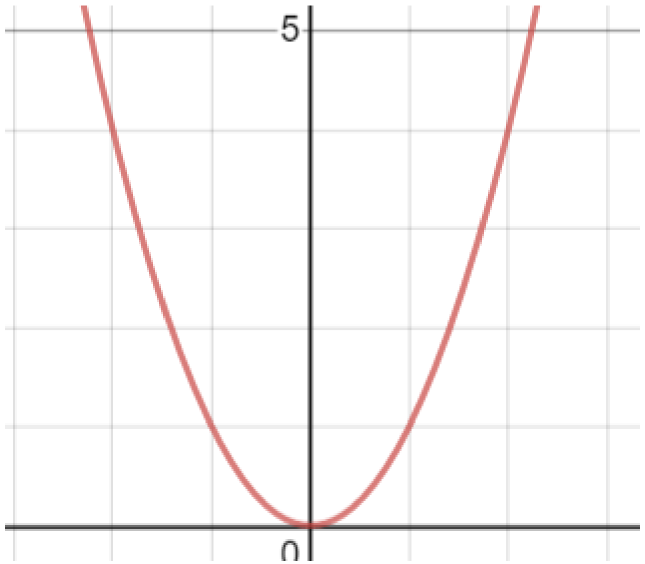

1
회귀
PPT
2
비용함수
3
경사하강법

3
경사하강법



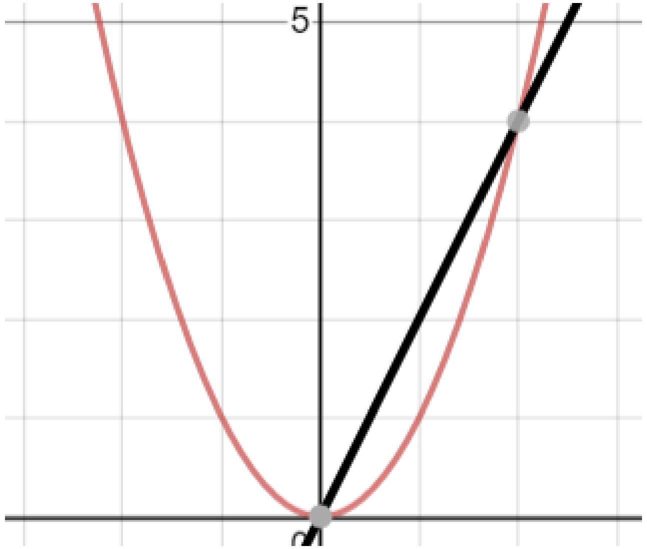
3

경사하강법



f(x) = x^2

3
경사하강법



$f(x) = x^2$

$f'(x) = 2x$

3
경사하강법

$$\mathrm{RSS}(w_0,\ w_1) = \frac{1}{N}\Sigma_{\mathrm{i=1}}^{N}(y_i - (w_0\ +\ w_1 * x_i))^2$$

$$R(W) = \frac{1}{N}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$$

W_0에 대하여 미분

$$\frac{1}{N}\sum_{i=1}^{n}(y_i^2 - 2y_i(w_0 + w_1 x_i) + (w_0^2 + 2w_0 w_1 x_i + w_1^2 x_i^2))$$

$$\Rightarrow \frac{1}{N}\sum_{i=1}^{n}(-2y_i + 2w_0 + 2w_1 x_i)$$

$$= -\frac{2}{N}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))$$

$$\frac{1}{N}\sum_{i=1}^{n}(y_i^2 - (2y_i(w_0 + w_1 x_i)) + (w_0^2 + 2w_0 w_1 x_i + w_1^2 x_i^2))$$

W_1에 대하여 미분

$$\frac{1}{N}\sum_{i=1}^{n}(-(2y_i x_i + 2w_0 x_i + 2x_i^2 w_1)$$

$$= -\frac{2}{N}\sum_{i=1}^{n}(y_i x_i - w_0 x_i - x_i^2 w_1$$

$$= -\frac{2}{N}\sum_{i=1}^{n} \cdot x_i(y_i - (w_0 + x_i w_1))$$

$$= -\frac{2}{N}\sum_{i=1}^{n} \cdot x_i(y_i - (w_0 + w_1 x_i))$$

실제값 − 예측값

3
경사하강법

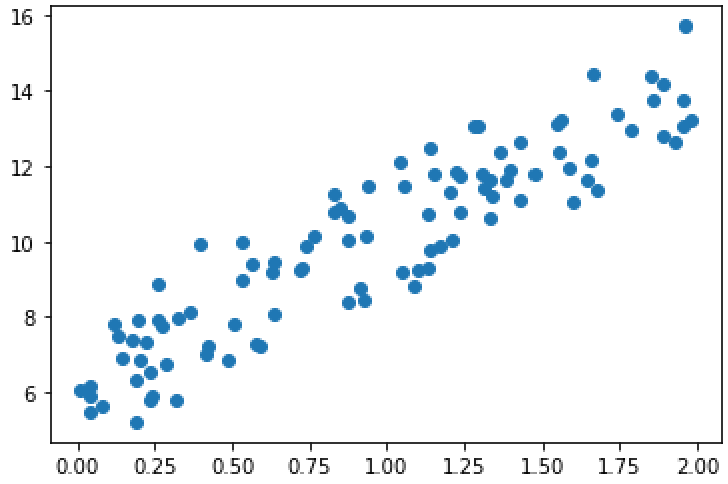
$$\mathrm{RSS}' = -\frac{2}{N}\Sigma_i^n * x_i * (y_i - (w_0 + w_1 * x_i)) \quad - w_1\text{미분}$$

$$\mathrm{RSS}' = -\frac{2}{N}\Sigma_i^n * (y_i - (w_0 + w_1 * x_i)) \quad - w_0\text{미분}$$

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [4]:
```
np.random.seed(0)
X = 2 * np.random.rand(100,1)
y = 6 + 4 * X+np.random.randn(100,1)

plt.scatter(X,y)
```

Out[4]: <matplotlib.collections.PathCollection at 0x13e847191c0>



In [40]:

```python
def get_cost(y,y_pred):
    N = len(y)
    cost = np.sum(np.square(y-y_pred))/N
    return cost
```

In [44]:

```python
def get_weight_updates(w1, w0, X, Y, learning_rate=0.01):
    N = len(y)
    w1_update = np.zeros_like(w1)
    w0_update = np.zeros_like(w0)
    y_pred = np.dot(X,w1.T) + w0
    diff = y-y_pred
    w0_factors = np.ones((N,1))
    w1_update = -(2/N)*learning_rate*(np.dot(X.T,diff))
    w0_update = -(2/N)*learning_rate*(np.dot(w0_factors.T,diff))
    return w1_update, w0_update
```

In [47]:

```python
 def gradient_descent_steps(X,y,iters=10000):
        w0=np.zeros((1,1))
        w1=np.zeros((1,1))
        for ind in range(iters):
            w1_update, w0_update = get_weight_updates(w1,w0,X,y,learning_rate=0.01)
            w1=w1-w1_update
            w0=w0-w0_update
        return w1,w0
```

In [49]:

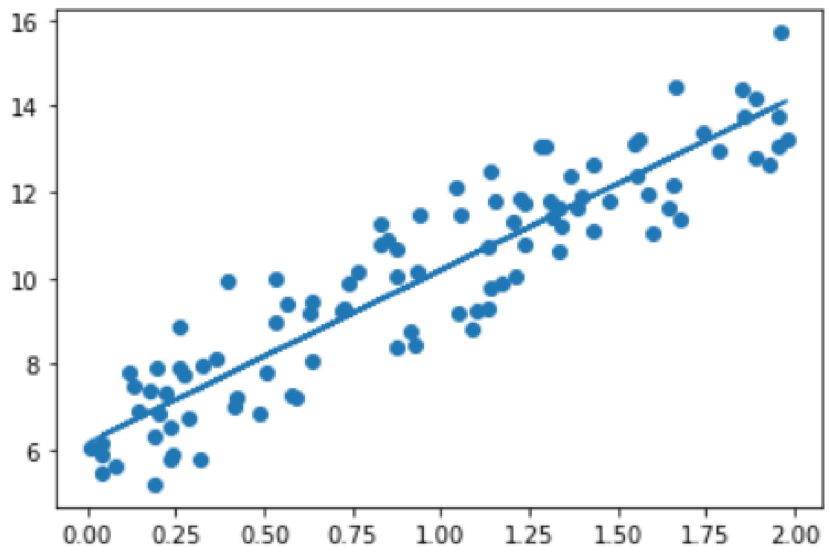
```python
def get_cost(y,y_pred):
    N=len(y)
    cost=np.sum(np.square(y-y_pred))/N
    return cost

w1,w0 = gradient_descent_steps(X,y,iters=1000)
print("w1:{0:.3f} w0:{1:.3f}".format(w1[0,0], w0[0,0]))
y_pred = w1[0,0] * X + w0
print('Gradient Descent Total Cost:{0:.4f}'.format(get_cost(y,y_pred)))
```

```
w1:4.022 w0:6.162
Gradient Descent Total Cost:0.9935
```

In [50]:

```python
plt.scatter(X,y)
plt.plot(X,y_pred)
```

Out[50]: [<matplotlib.lines.Line2D at 0x13e84d06130>]



ANY QUESTION??

감사합니다ㅎㅎ