Hands-On Machin Learning With Scikit-Learn, Keras & TensorFlow

PAPATA LABS

Hands-On Machine Learning 4장

Index

00 서론

- 지금까지는 머신러닝 모델과 훈련 알고리즘의 라이브러리를 그저 사용하고 내부가 어떻게 작동하는지를 확인하지 않았다.

- 하지만 어떻게 작동하는지를 잘 이해하고 있으면 적절한 모델, 올바른 훈련 알고리즘, 작업에 맞는 좋은 하이퍼 파라미터를 빠르게 찾을 수 있다.

선형 회귀

하루에 걷는 횟수를 늘릴 수록 몸무게가 줄어드는 것처럼

집의 평수가 클수록 집의 매매 가격이 비싼 것 처럼

어떤 요인의 수치에 따라서 특정 요인의 수치가 영향을 받는 회귀

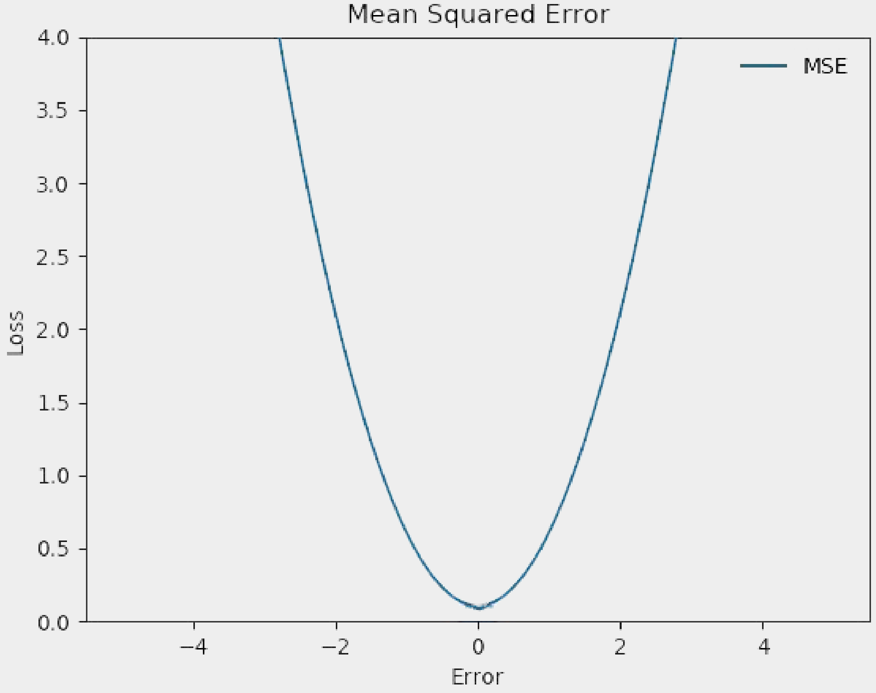
$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

$$\hat{y} = h_0(x) = \theta \cdot x$$

01 선형 회귀

비용 함수

$$\mathrm{MSE}(\mathbf{X}, h_\theta) = \frac{1}{m}\sum_{i=0}^{m-1}(\theta^T\,(\mathbf{x}_b^{(i)})^T - y^{(i)})^2$$



01 선형 회귀

비용 함수

$$MSE(W) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2$$

$$= \frac{1}{m}(WX - y)^2$$

$$= \frac{1}{m}(WX - y)^T(WX - y)$$

$$= \frac{1}{m}((WX)^T - y^T)(WX - y)$$

$$= \frac{1}{m}((WX)^T WX - y^T WX - (WX)^T y + y^T y$$

$$= \frac{1}{m}(X^T W^T WX - (WX)^T y - (WX)^T y - y^T y)$$

$$= \frac{1}{m}(X^T X W^2 - 2X^T W^T y - y^T y)$$

$$\frac{dMSE}{dW} = \frac{1}{m}(2X^T X W - 2X^T y) = 0 \quad \text{(최소)}$$

$$2X^T X W - 2X^T y = 0$$

$$X^T X W = X^T y \quad \therefore W = (X^T X)^{-1} X^T y$$

01 선형 회귀

비용 함수 – 정규 방정식

$$\mathrm{MSE}(\mathbf{X}, h_\theta) = \frac{1}{m}\sum_{i=0}^{m-1}(\theta^T (\mathbf{x}_b^{(i)})^T - y^{(i)})^2$$

$$\theta = (X^T X)^{-1} X^T y$$

X : x_0 에서 x_n까지 담은 샘플의 특성 백터
y : 실제 값을 담은 특성 백터

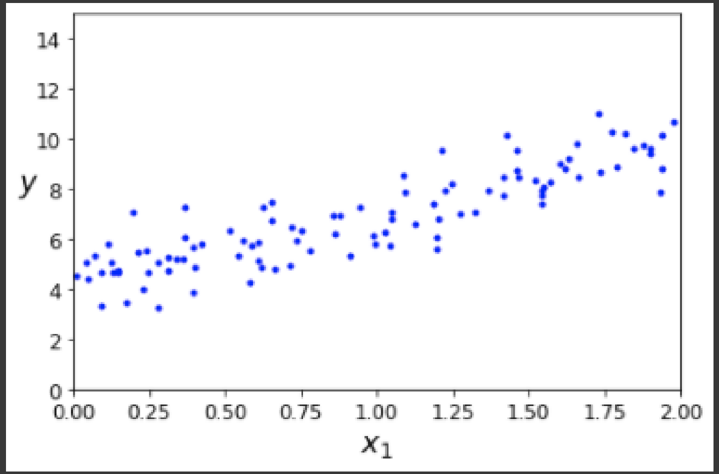
01 선형 회귀

비용 함수 – 정규 방정식

```
[ ] import numpy as np

    X = 2 * np.random.rand(100, 1)
    y = 4 + 3 * X + np.random.randn(100, 1)
```

```
[ ] plt.plot(X, y, "b.")

    plt.xlabel("$x_1$", fontsize=18)
    plt.ylabel("$y$", rotation=0, fontsize=18)

    plt.axis([0, 2, 0, 15])

    plt.show()
```

그림 저장: generated_data_plot



01 선형 회귀

비용 함수 – 정규 방정식

식 4-4: 정규 방정식

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

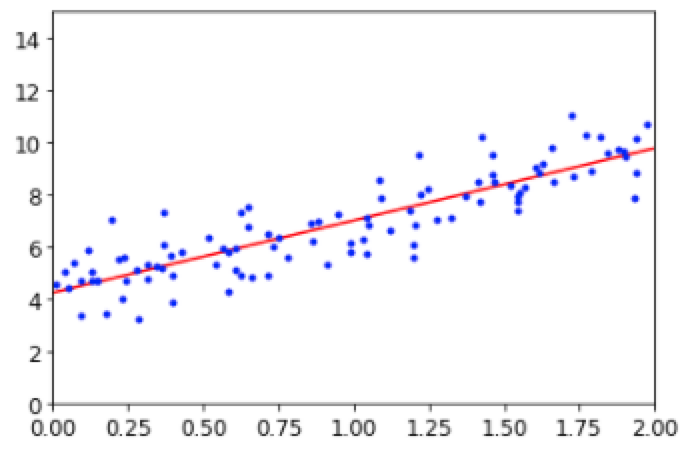
```
[ ]  X_b = np.c_[np.ones((100, 1)), X]

     theta_best = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```

```
[ ]  theta_best

     array([[4.21509616],
            [2.77011339]])
```

01 선형 회귀

비용 함수 – 정규 방정식

```
[ ] plt.plot(X_new, y_predict, "r-")
    plt.plot(X, y, "b.")
    plt.axis([0, 2, 0, 15])
    plt.show()
```



01 선형 회귀

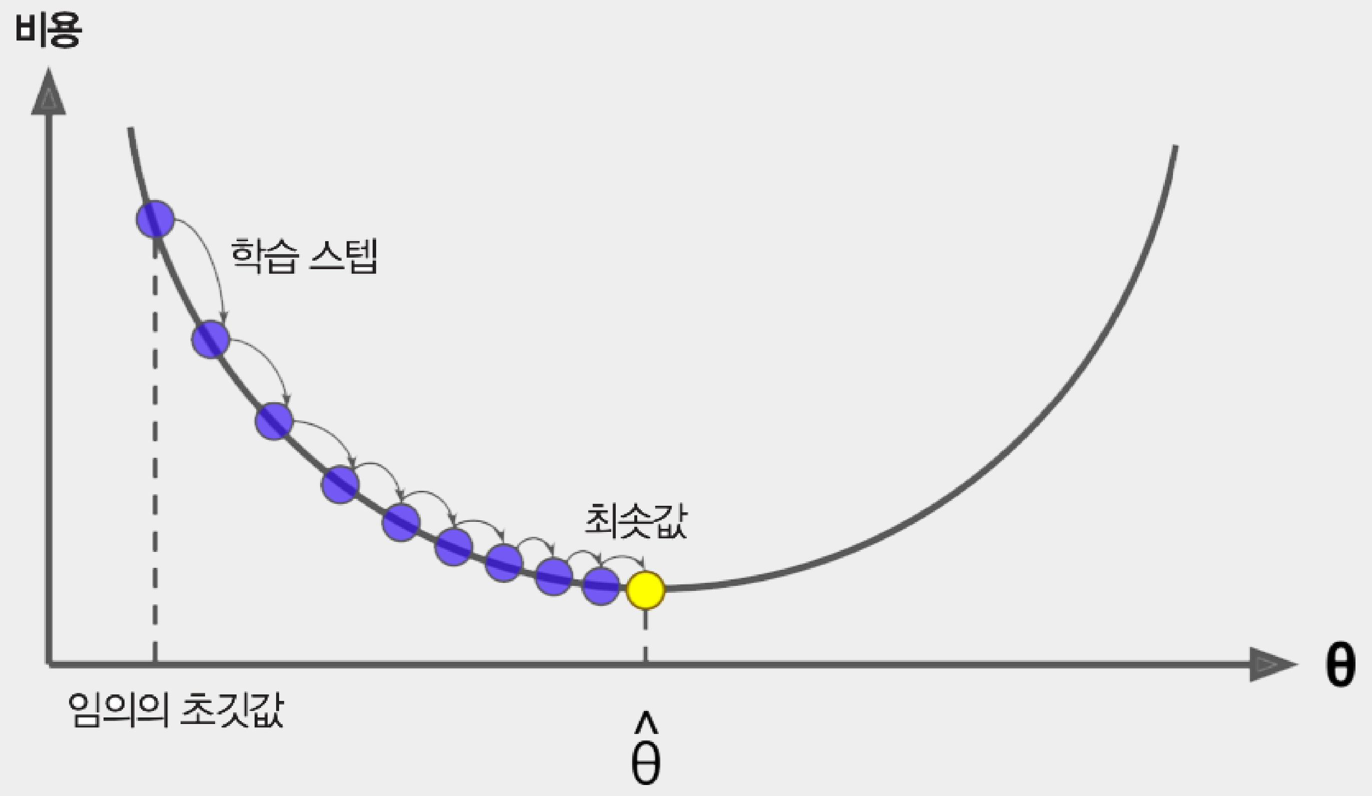
비용 함수 – 정규 방정식

정규 방정식은 $(n+1) \times (n+1)$크기가 되는 $X^T X$의 역행렬을 계산한다

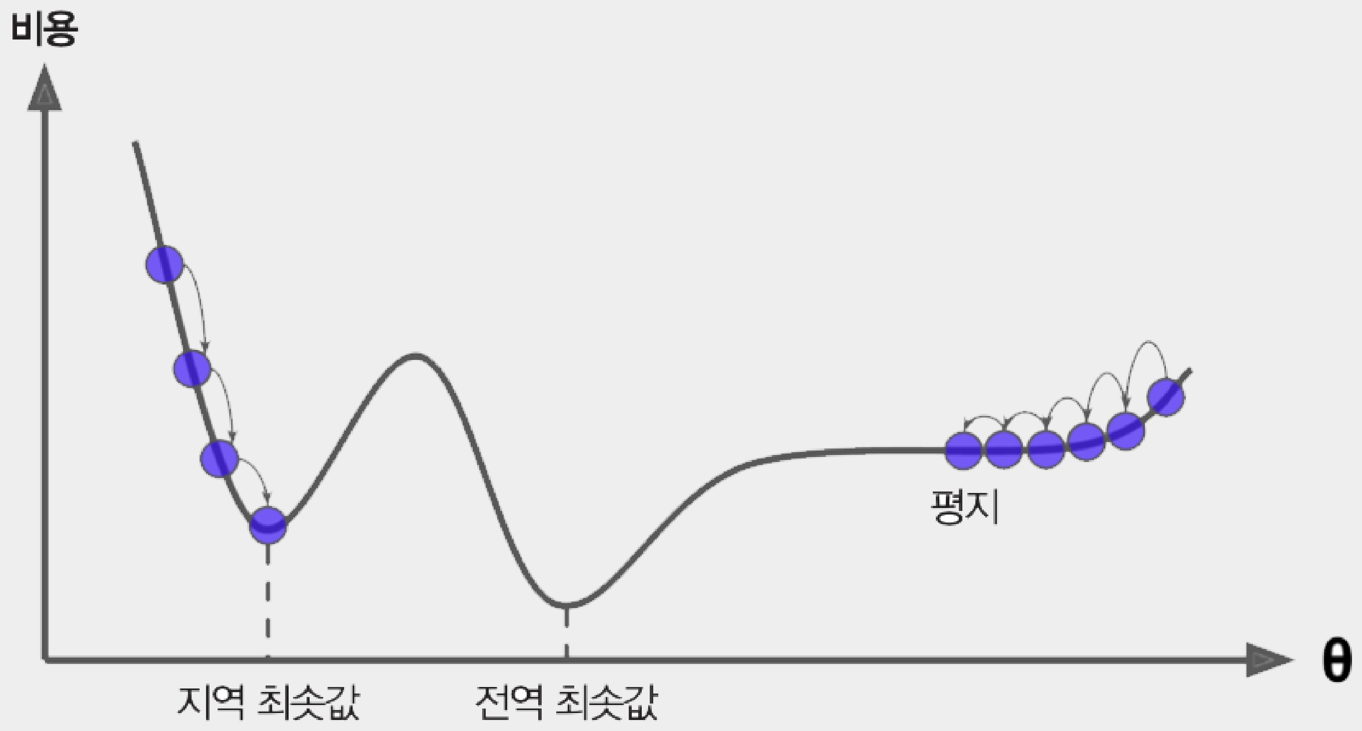
역행렬을 계산하는 계산 복잡도는 일반적으로 $O(n^{2.4})$ 에서 $O(n^3)$ 사이이다

즉 n, 특성 수가 많아질수록 계산 복잡도는 기하 급수적으로 증가한다

02 경사 하강법



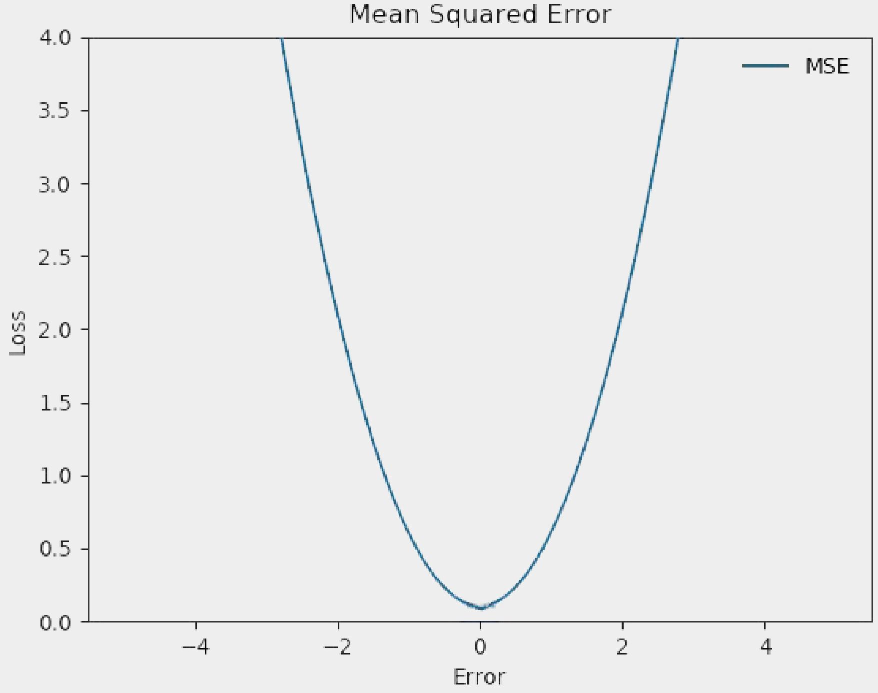
02 경사 하강법



02 경사 하강법

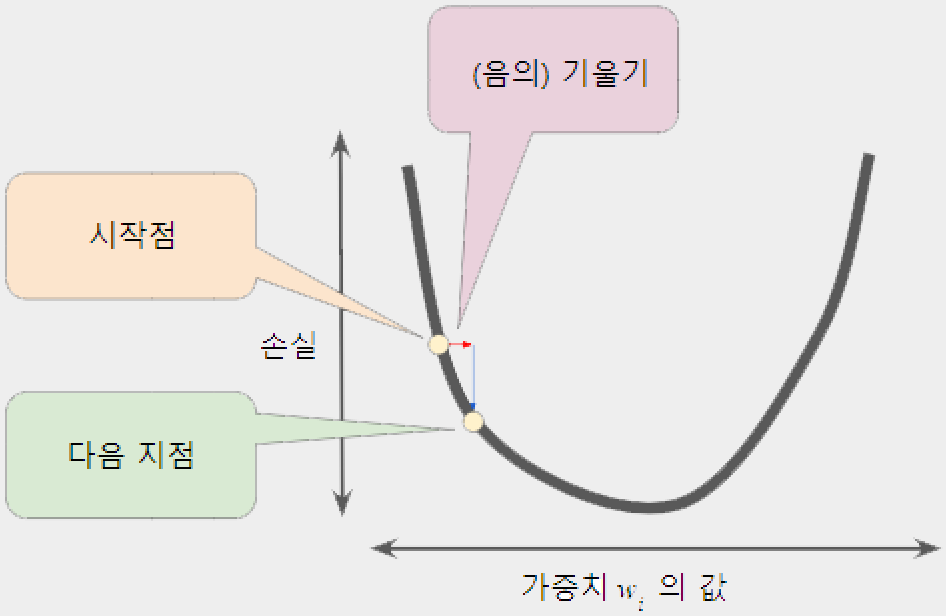
MSE의 경우

$$\mathrm{MSE}(\mathbf{X}, h_\theta) = \frac{1}{m}\sum_{i=0}^{m-1}(\theta^T\,(\mathbf{x}_b^{(i)})^T - y^{(i)})^2$$



02 경사 하강법

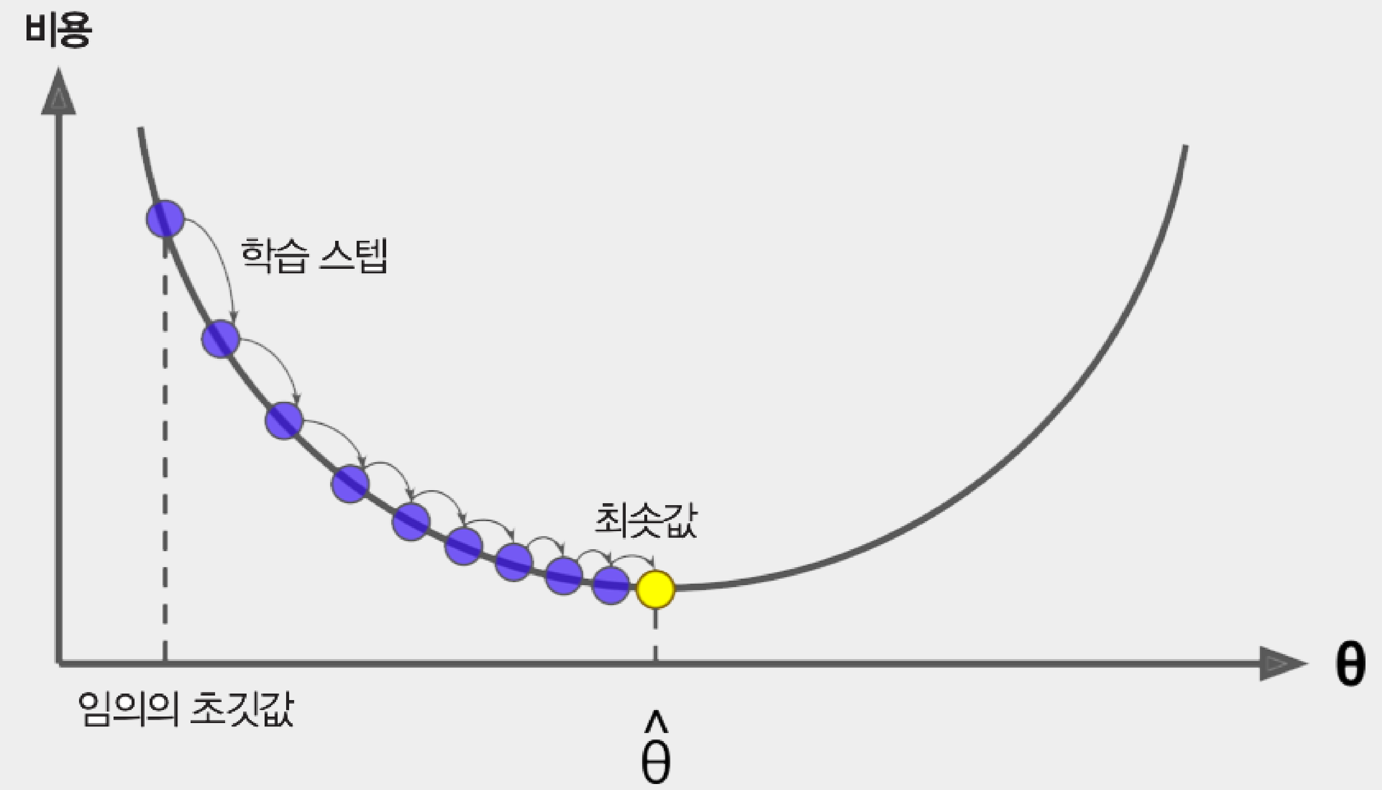
최적의 학습률



$$\boldsymbol{\theta}^{(\text{ next step })} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \ \ \text{MSE}(\boldsymbol{\theta})$$

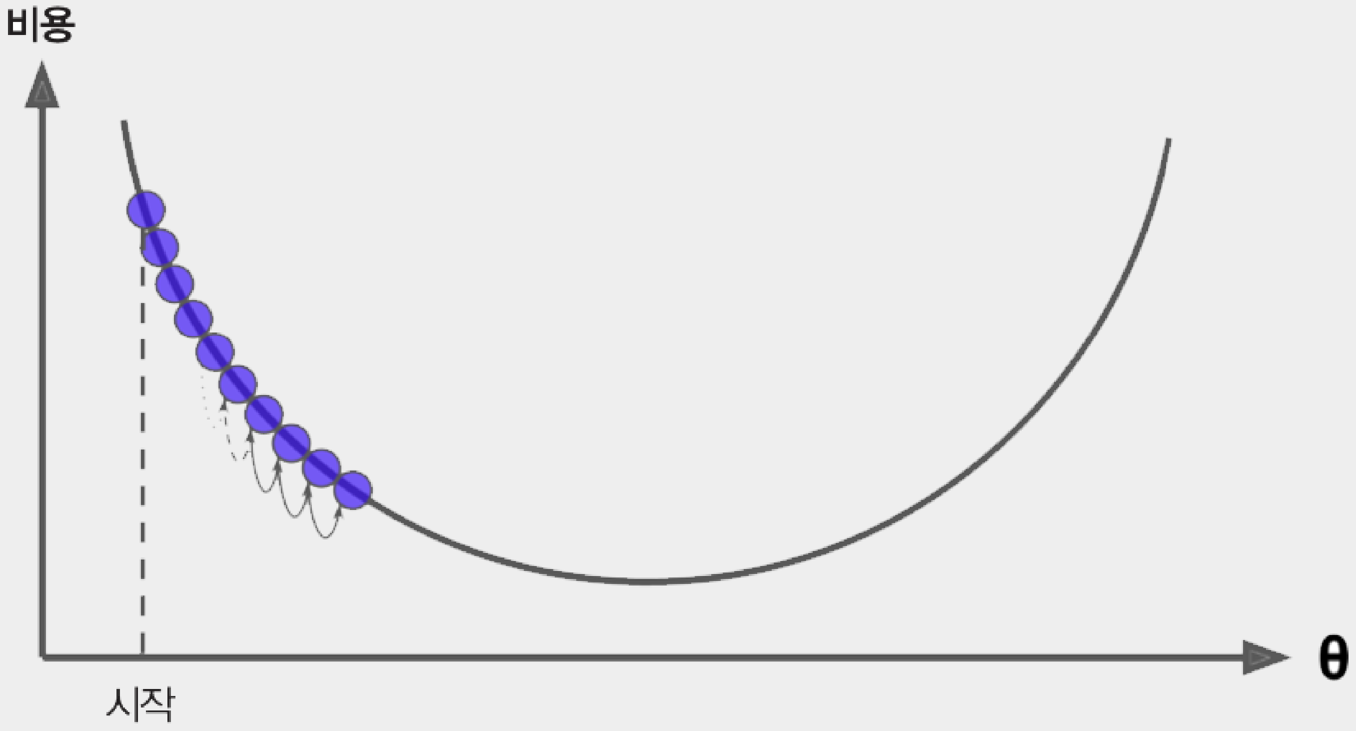
02 경사 하강법

최적의 학습률



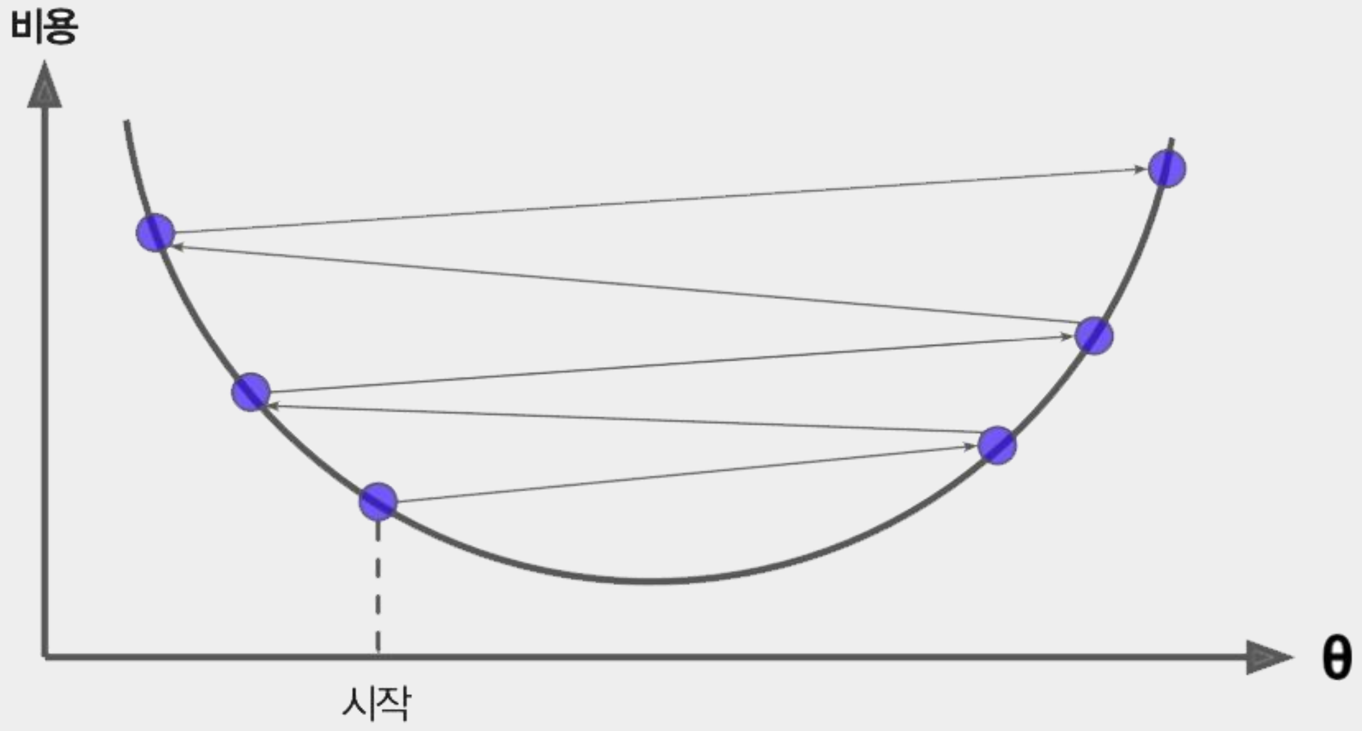
02 경사 하강법

학습률이 작을 경우



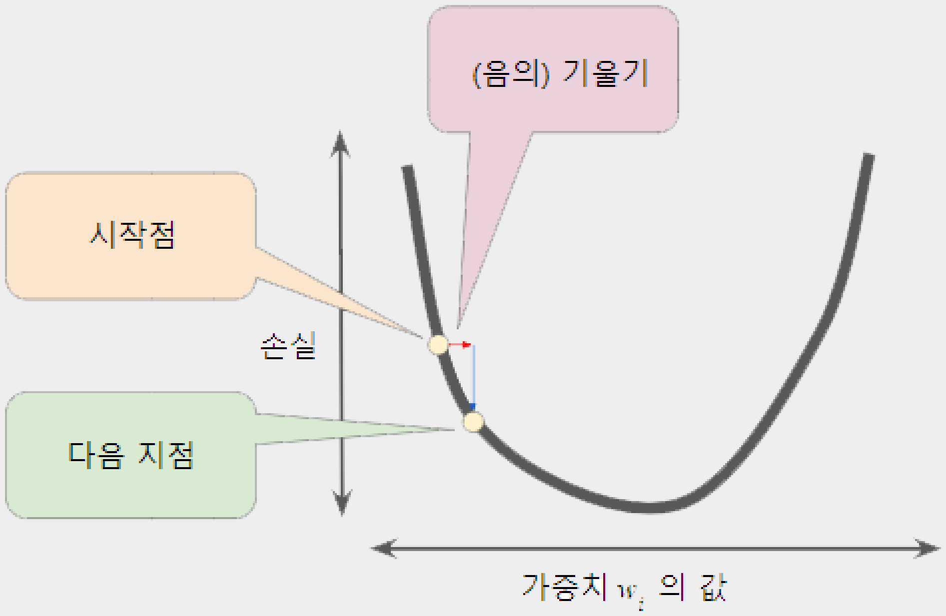
02 경사 하강법

학습률이 큰 경우



02 경사 하강법

최적의 학습률



$$\boldsymbol{\theta}^{(\text{ next step })} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \ \ \mathrm{MSE}\,(\boldsymbol{\theta})$$

경사 하강법 종류

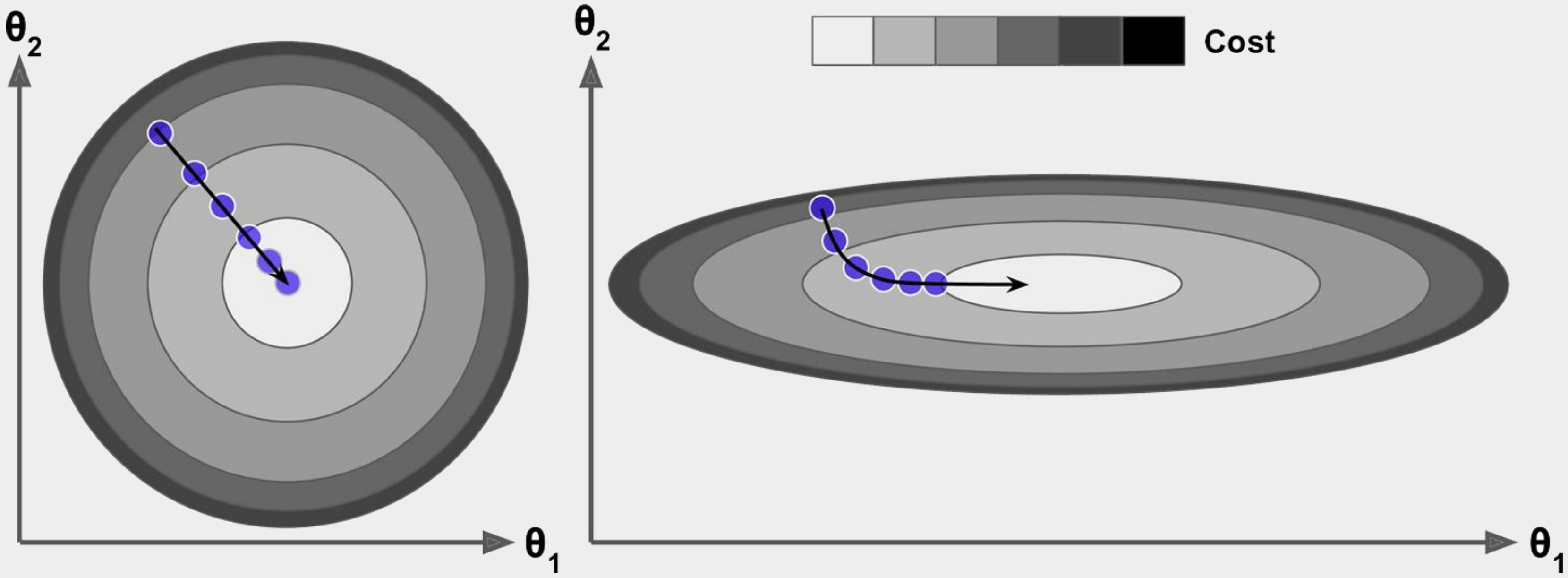
- 배치 경사 하강법
- 확률적 경사 하강법
- 미니 배치 경사 하강법

배치 경사 하강법

$$\nabla_{\boldsymbol{\theta}}\, \mathrm{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \mathrm{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \mathrm{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \mathrm{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

02 경사 하강법

배치 경사 하강법



02 경사 하강법

배치 경사 하강법

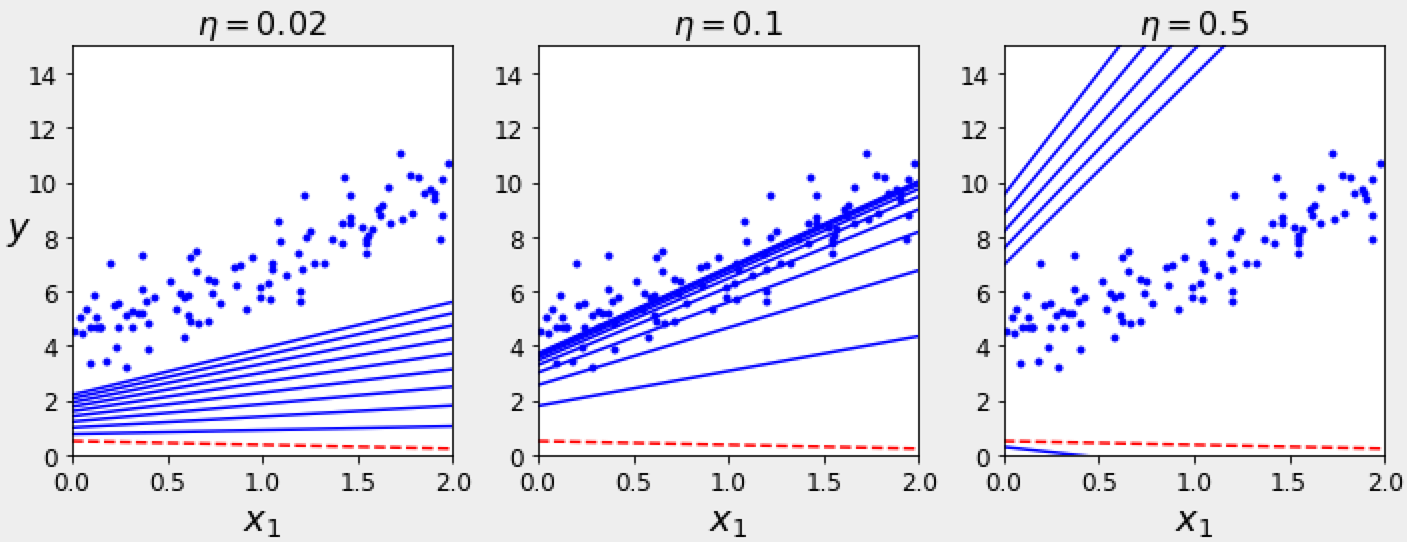
```
[ ]  eta = 0.1
     n_iterations = 1000
     m = 100

     theta = np.random.randn(2,1)

     for iteration in range(n_iterations):
         gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
         theta = theta - eta * gradients
```

```
[ ]  theta
```

```
     array([[4.21509616],
            [2.77011339]])
```

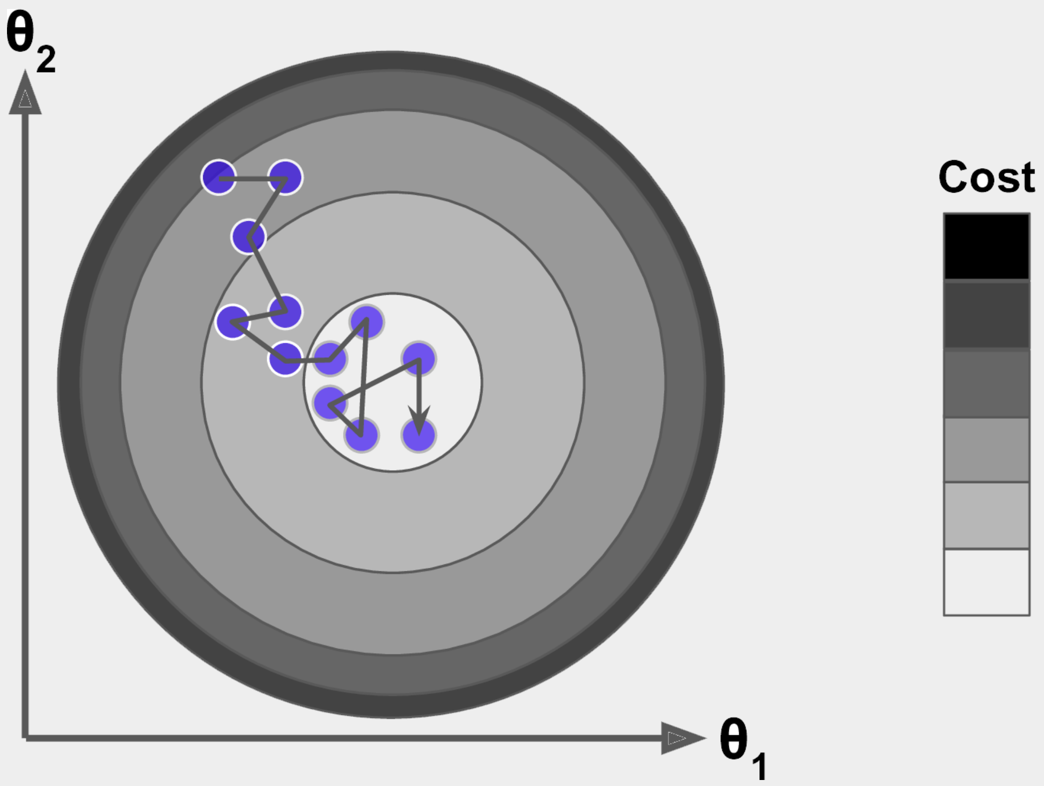
02 경사 하강법

배치 경사 하강법



02 경사 하강법

확률적 경사 하강법



02 경사 하강법

확률적 경사 하강법

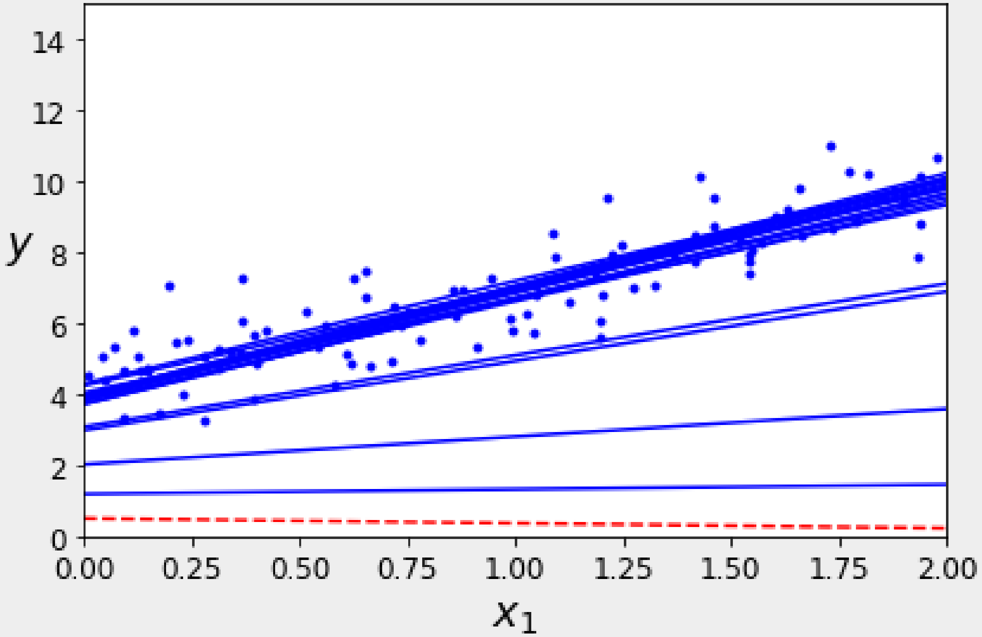
```
n_epochs = 50
t0, t1 = 5, 50

def learning_schedule(t):
    return t0 / (t + t1)

theta = np.random.randn(2,1)

for epoch in range(n_epochs):
    for i in range(m):
        if epoch == 0 and i < 20:
            y_predict = X_new_b.dot(theta)
            style = "b-" if i > 0 else "r--"
            plt.plot(X_new, y_predict, style)
        random_index = np.random.randint(m)
        xi = X_b[random_index:random_index+1]
        yi = y[random_index:random_index+1]
        gradients = 2 * xi.T.dot(xi.dot(theta) - yi)
        eta = learning_schedule(epoch * m + i)
        theta = theta - eta * gradients
        theta_path_sgd.append(theta)
```

```
theta
```

```
array([[4.21076011],
       [2.74856079]])
```

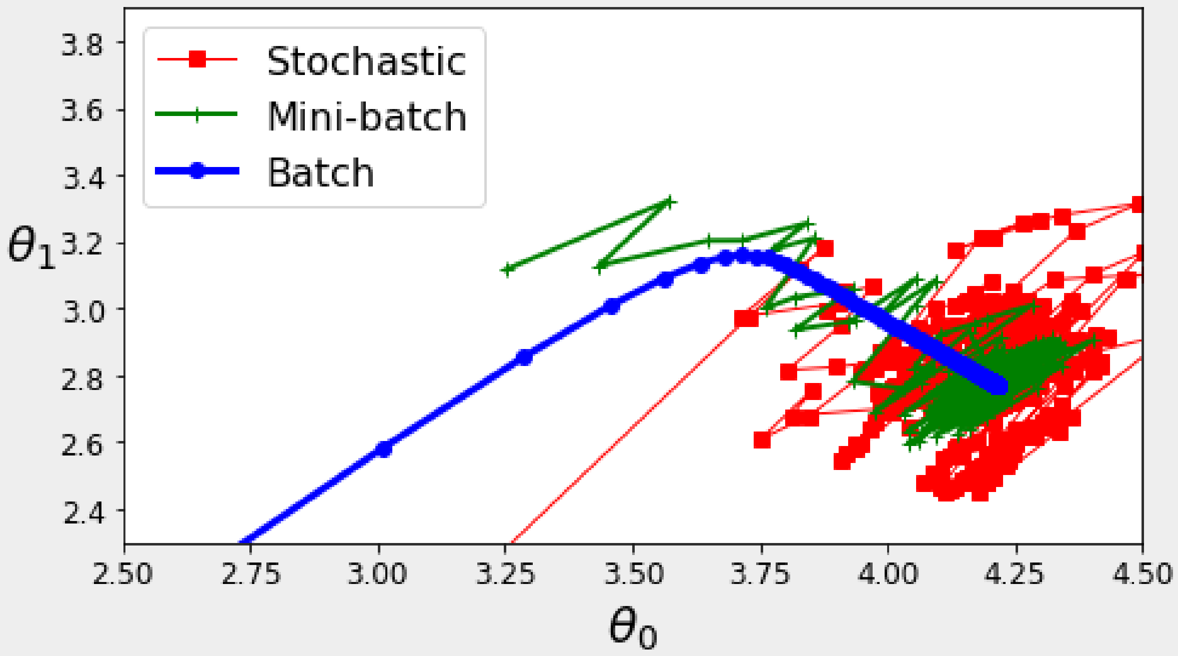
02 경사 하강법

확률적 경사 하강법



02 경사 하강법

미니배치 경사 하강법



02 경사 하강법

미니배치 경사 하강법

알고리즘	많은 샘플 수	외부 메모리 학습	많은 특성 수	하이퍼 파라미터 수	스케일 조정	사이킷런 지원
정규방정식	빠름	지원 안됨	느림	0	불필요	지원 없음
SVD	빠름	지원 안됨	느림	0	불필요	LinearRegression
배치 GD	느림	지원 안됨	빠름	2	필요	LogisticRegression
SGD	빠름	지원	빠름	>= 2	필요	SGDRegressor
미니배치 GD	빠름	지원	빠름	>=2	필요	SGDRegressor

선형 회귀

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

다항 회귀

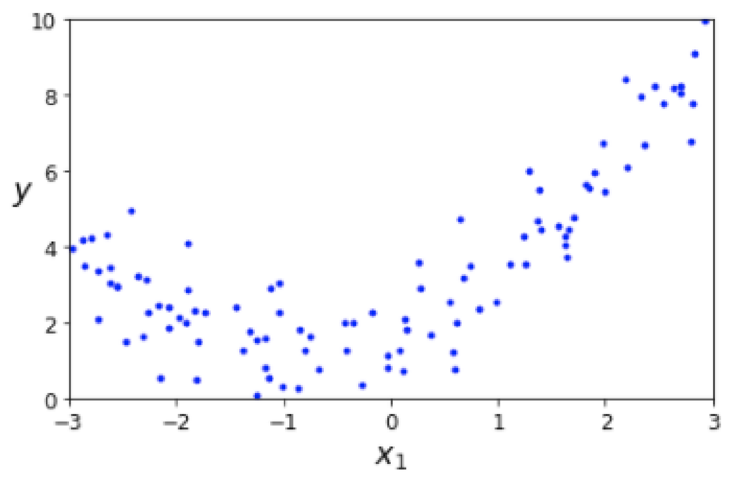
$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_1 x_2 + \theta_3 {x_1}^2 + \cdots$$

03 다항 회귀

```
[ ]  m = 100
     X = 6 * np.random.rand(m, 1) - 3
     y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)
```

```
[ ]  plt.plot(X, y, "b.")

     plt.xlabel("$x_1$", fontsize=18)
     plt.ylabel("$y$", rotation=0, fontsize=18)
     plt.axis([-3, 3, 0, 10])

     plt.show()
```

그림 저장: quadratic_data_plot



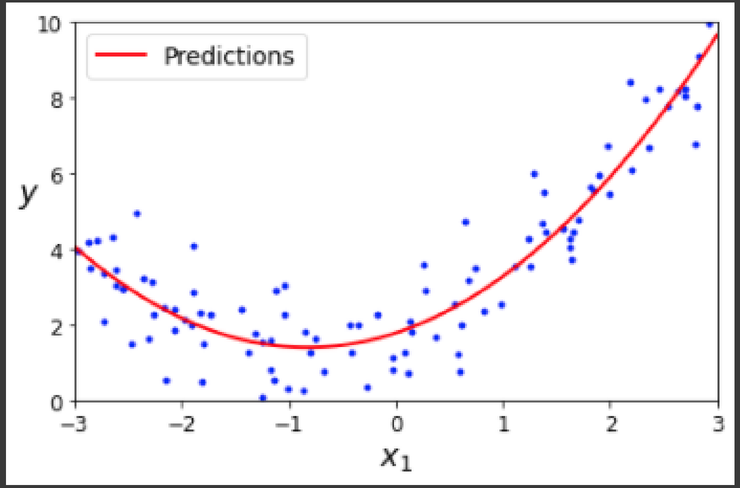
03 다항 회귀

```
[ ] from sklearn.preprocessing import PolynomialFeatures
    poly_features = PolynomialFeatures(degree=2, include_bias=False)
    X_poly = poly_features.fit_transform(X)
    X[0]

    array([-0.75275929])

[ ] X_poly[0]

    array([-0.75275929,  0.56664654])

[ ] lin_reg = LinearRegression()
    lin_reg.fit(X_poly, y)
    lin_reg.intercept_, lin_reg.coef_

    (array([1.78134581]), array([[0.93366893, 0.56456263]]))
```

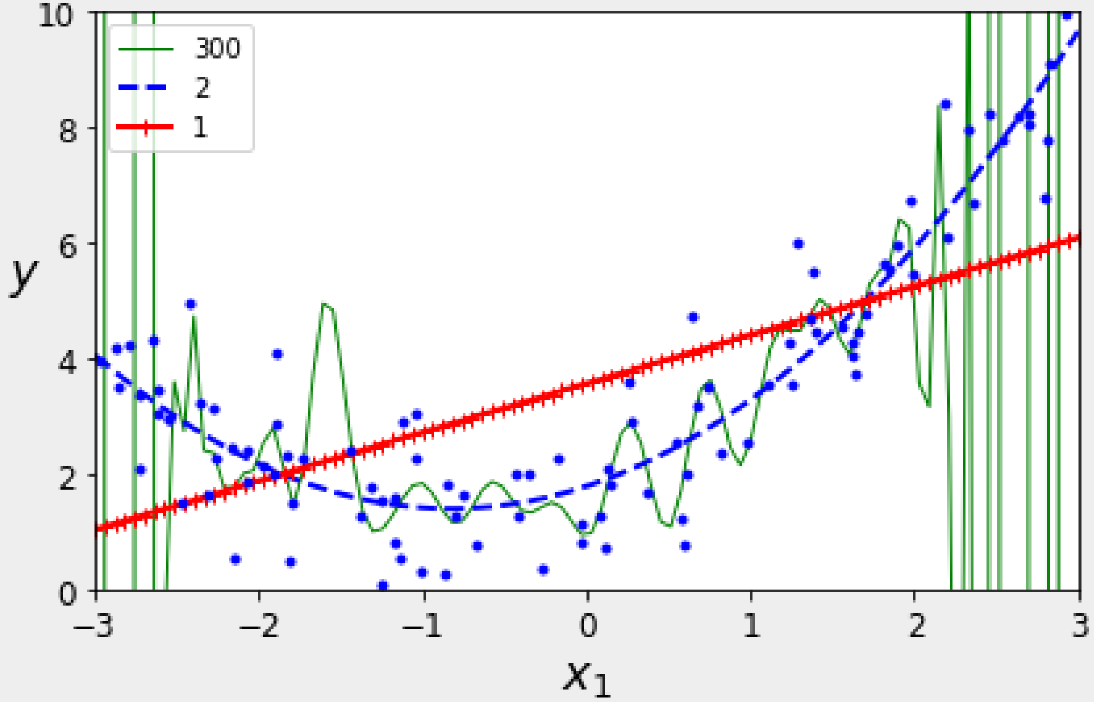
03 다항 회귀

```
[ ] X_new=np.linspace(-3, 3, 100).reshape(100, 1)
    X_new_poly = poly_features.transform(X_new)
    y_new = lin_reg.predict(X_new_poly)

    plt.plot(X, y, "b.")
    plt.plot(X_new, y_new, "r-", linewidth=2, label="Predictions")

    plt.xlabel("$x_1$", fontsize=18)
    plt.ylabel("$y$", rotation=0, fontsize=18)
    plt.legend(loc="upper left", fontsize=14)
    plt.axis([-3, 3, 0, 10])

    plt.show()
```

그림 저장: quadratic_predictions_plot



04 학습 곡선



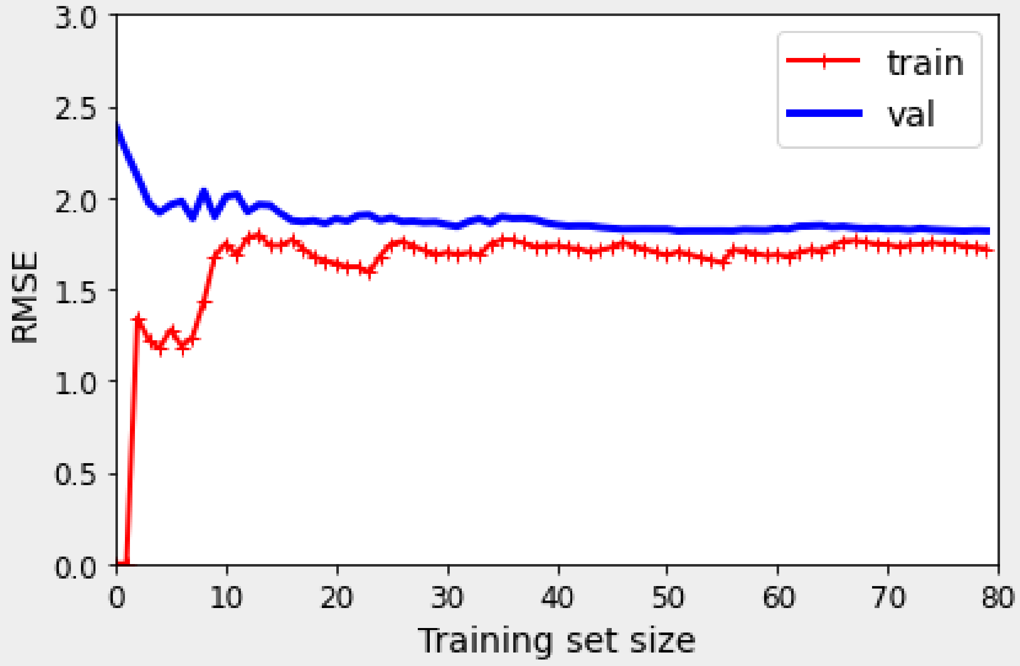
학습 곡선

- 훈련 데이터의 양에 대해, 모델의 범화 성능을 그린 것

- 훈련 데이터에 대한 모델의 성능과 검증 데이터에 대한 모델의 성능이 훈련 데이터의 양에 따라 어떻게 변화해가는지를 표시한 그래프

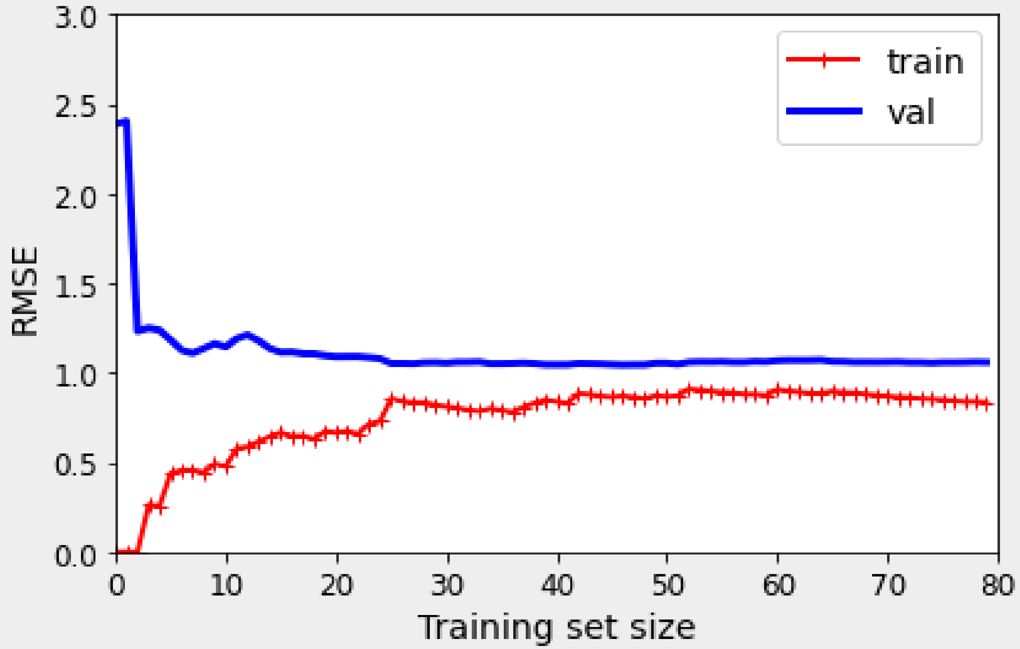
04 학습 곡선

1차항의 경우



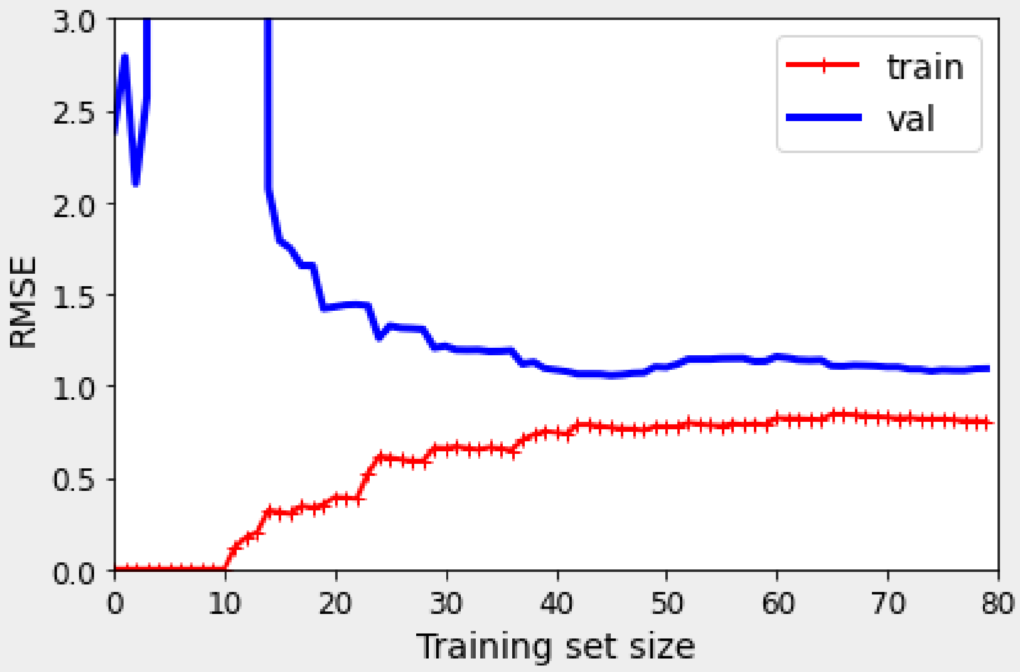
04 학습 곡선

2차항의 경우



04 학습 곡선

10차항의 경우



편향/분산 트레이드 오프

- 편향
  - 일반화 오차 중에서 편향은 잘못된 가정으로 인한 것
  - 편향이 큰 모델은 훈련 데이터에 과소적합되는 경향이 있다

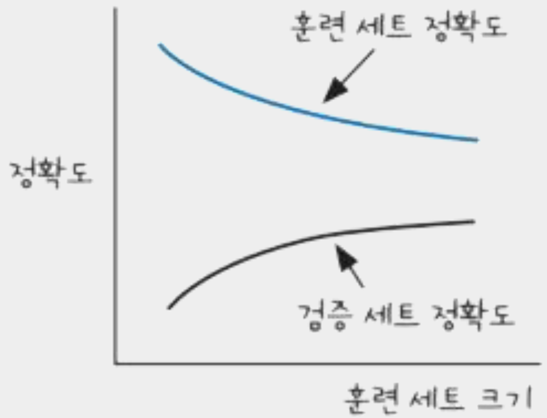
- 분산
  - 훈련 데이터에 있는 작은 변동에 모델이 과도하게 민감하기 때문에 나타남
  - 분산이 큰 모델은 훈련 데이터에 과대적합되는 경향이 있다

- 줄일 수 없는 오차
  - 데이터 자체에 있는 잡음 때문에 오차는 항상 발생하며,

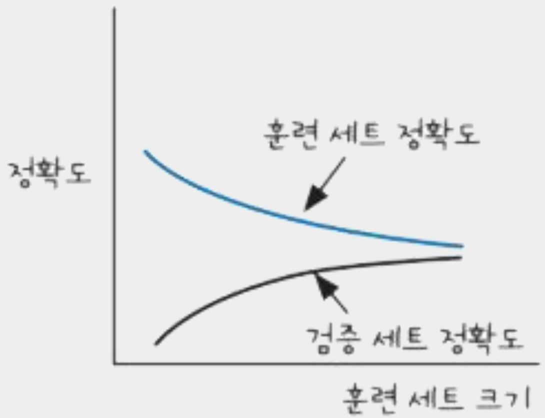
- 모델의 복잡도가 커지면, 통상적으로 분산이 늘어나고 편향을 줄어들지만
- 모델의 복잡도가 줄어들면 편향이 커지고 분산이 작아진다

04 학습 곡선

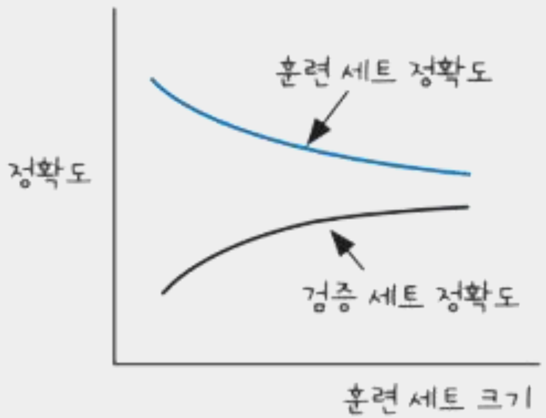
편향-분산 트레이드 오프



분산이 높은 경우



편향이 높은 경우



최적의 트레이드 오프

END OF DOCUMENT

THANK YOU

PAPATA LABS