머신러닝
(MACHINE LEARNING)

LECTURE IX: 지도 학습 3
(Supervised Learning)

Dai-Gyoung Kim

Department of Applied Mathematics

Hanyang University ERICA

 ⓒ dgkim@hanyang.ac.kr

지도 학습 (Supervised Learning)

Contents

- 분류와 회귀
- 일반화, 과대적합, 과소적합
- 지도 학습 알고리즘
    - k-최근접 이웃
    - 선형모델
    - 나이브베이즈 모델
    - 결정트리
    - 결정트리의 앙상블
    - 커널 서포트 벡터 머신
    - 신경망, 딥러닝
- 분류 예측의 불확실성 추정

▶ 선형 모델

- 선형 모델은 입력 특성에 대한 선형 함수를 만들어 예측을 수행함.

❖ 회귀의 선형 모델

- 선형 회귀 모델은 학습 모델 중에서 가장 간단하면서도 기본적인 모델 중에 하나임.
- 선형 회귀 모델은 입력 특성이 **가중치 합**과 **편향**(절편)을 이용해 예측을 만들어냄.
- 선형 회귀 모델 식:

$$\hat{y} = w_0 x_0 + w_1 x_1 + \cdots + w_p x_p + b$$

▷ x_j : j 번째 특성 (특성의 개수: $p+1$)

▷ w_j, b : 모델이 학습할 파라미터

▷ $\hat{y}$: 예측값

✓ 특성의 수: $p+1$개

✓ 모델 파라미터 수: $p+2$개

- 선형 회귀 모델 식의 벡터형

$$\hat{y} = \boldsymbol{w}^T\boldsymbol{x} + b$$

▷ $\boldsymbol{x} = (x_0, x_1, \ldots, x_p)^T \in \mathbb{R}^{p+1}$: 특성 데이터 벡터

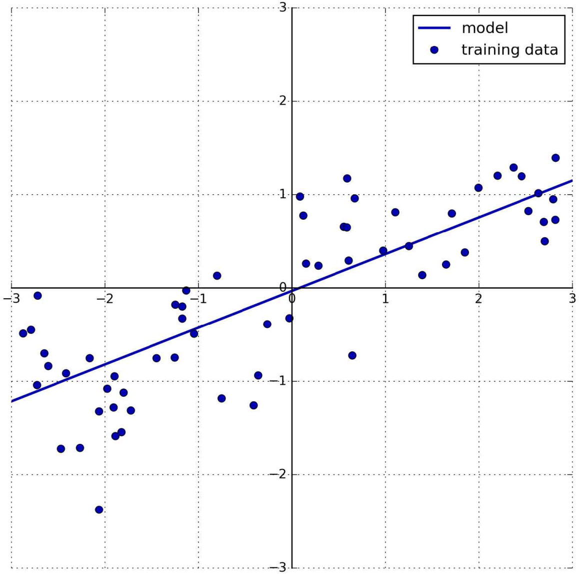
▷ $\boldsymbol{w} = (w_0, w_1, \ldots, w_p)^T \in \mathbb{R}^{p+1}$: 가중치 벡터

▷ b : 편향 (절편)

- 다음은 1차원 wave 데이터셋으로 직선의 예측을 만드는 모델 파라미터 w_0, b를 학습시킨 것임.

```
mglearn.plots.plot_linear_regression_wave()
```

```
w[0]: 0.393906   b: -0.031804
```



- 선형 회귀 모델은 특성 수가 많아지면 성능이 좋아지는 경향이 있음.
  - ✓ 특성이 하나일 때 직선으로 예측
  - ✓ 특성이 두 개일 때 평면으로 예측
  - ✓ 특성이 세 개 이상일 때 초평면(hyperplane)으로 예측

- 특히 훈련 데이터셋 보다 특성이 더 많으면 어떤 타깃(훈련데이터)도 정확하게 선형 함수로 모델링할 수 있음.
  - ✓ 훈련 데이터(방정식)보다 특성 또는 모델 파라미터(미지수)가 많은 경우 불충분한 시스템 (underdetermined system)이 되어 무수히 많은 해가 존재함.

- 모델을 훈련시킨다는 것은 모델이 훈련 세트에 가장 잘 맞도록 모델 파라미터를 설정하는 것임.
  - ✓ 이를 위해 먼저 모델이 훈련 데이터에 얼마나 잘 들어맞는지 측정해야함. 이때, 성능 측정 시스템이 필요함.
  - ✓ 회귀에서 가장 널리 사용되는 성능 측정 지표는 **평균제곱근오차**(**RMSE**Root Mean Square Error) 또는 **평균제곱오차**(**MSE**Mean Square Error)임.

- 선형 회귀를 위한 선형 모델은 다양하며, 이 모델들은 훈련 데이터로부터 모델 파라미터를 학습하는 방법과 모델의 복잡도를 제어하는 방법에 따라 다름.

1) 선형 회귀 또는 최소제곱법 (OLS Ordinary Least Squares)

2) 릿지 회귀 (Ridge Regression)

3) 라쏘 회귀 (Lasso Least Absolute Shrinkage and Selection Operator Regression)

❖ 선형 모델(최소 제곱법)

▷ 선형 회귀 모델의 MSE 비용함수

- 훈련 데이터 세트 $\{x_i, y_i\}_{i=1}^{m}$: m개의 데이터셋

$$\boldsymbol{x}_i = \left(x_{i0}, \ \ldots \ , x_{ip}\right)^T \in \mathbb{R}^{p+1}, \qquad y_i \in \mathbb{R}$$

- 훈련 데이터 세트에 대한 선형 회귀 모델의 MSE 비용함수

$$\boxed{\mathrm{MSE}(\boldsymbol{w}, b) = \frac{1}{m}\sum_{i=1}^{m}\left(\boldsymbol{w}^T\boldsymbol{x}_i + b - y_i\right)^2}$$

- MSE 비용함수를 최소화하는 해를 구하는 방법.

1) 정규 방정식 (Normal Equation)
2) 경사 하강법 (Gradient Descent Method)

정규 방정식

- 비용 함수를 최소화 하는 파라미터 벡터 θ의 값을 찾기 위한 공식:

$$\hat{\theta} \equiv \begin{bmatrix} \hat{\boldsymbol{w}} \\ \hat{b} \end{bmatrix} = (A^T A)^{-1} A^T \boldsymbol{y}$$

▷ 파라미터 벡터 $\theta \in \mathbb{R}^{n+1}$: $\theta = (w_0, \dots, w_p, b)^T$, $n = p+1$.

▷ 디자인 행렬 $A \in \mathbb{R}^{m \times (n+1)}$:

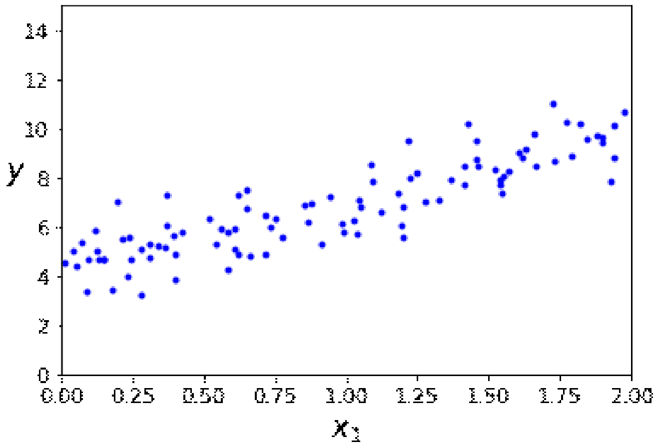
$$A = \begin{pmatrix} \boldsymbol{x}_1^T & 1 \\ \vdots & \vdots \\ \boldsymbol{x}_m^T & 1 \end{pmatrix} = \begin{pmatrix} x_{10} & \cdots & x_{1p} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_{m0} & \cdots & x_{mp} & 1 \end{pmatrix}$$

▷ $\boldsymbol{y} = (y_1, \dots, y_m)^T \in \mathbb{R}^m$: 훈련 데이터의 타깃 벡터

- 넘파이를 사용하여 정규 방정식 해법을 확인하기 위해 데이터를 생성함.

```
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()
```



- 넘파이 선형대수 모듈(np.linalg)에 있는 'inv()' 함수를 사용해 역행렬을 계산함.

```
A = np.c_[X, np.ones((100, 1))] # 모든 샘플에 1을 추가

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

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

✓ θ의 추정: $\hat{\theta} = (\hat{w}, \hat{b})$, $\hat{w} = 2.770$, $\hat{b} = 4.215$

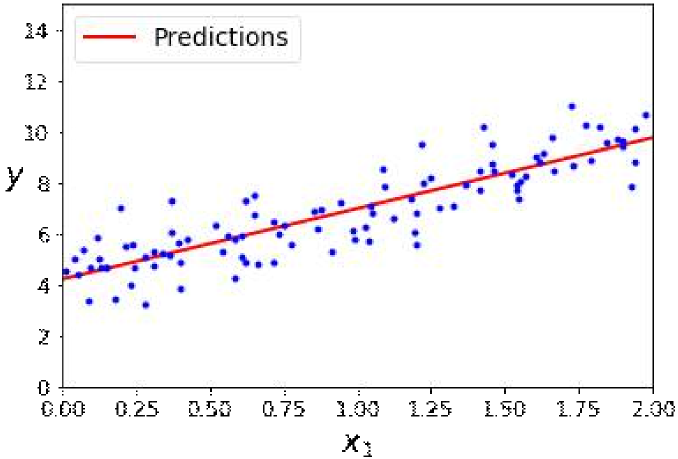
- 위 훈련 데이터셋으로 훈련된 선형 회귀 예측

```
X_new = np.array([[0], [2]])
X_new_b = np.c_[X_new, np.ones((2, 1))] # 모든 샘플에 1을 추가
y_predict = X_new_b.dot(theta_best)
y_predict
```

```
array([[ 4.21509616],
       [ 9.75532293]])
```

- 선형 회귀 모델 예측의 그래프

```
plt.plot(X_new, y_predict, "r-", linewidth=2, label="Predictions")
plt.plot(X, y, "b.")
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.legend(loc="upper left", fontsize=14)
plt.axis([0, 2, 0, 15])
plt.show()
```



- 같은 작업을 하는 사이킷런 코드는 “LinearRegression”을 사용함.

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, y)
lin_reg.intercept_, lin_reg.coef_
```

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

[계산 복잡도]

- 정규 방정식에서 $(n+1)\times(n+1)$ 크기인 행렬 A^TA의 역행렬 계산의 복잡도.
  - ✓ 이때, $n=p+1$은 특성의 수
  - ✓ 역행렬을 계산하는 계산 복잡도(computational complexity)는 일반적으로 $O(n^{2.4})$에서 $O(n^3)$임. (특성 수가 2배로 증가하면, 계산 시간이 대략 $2^{2.4}$에서 2^3배로 증가함)
  - ✓ 특성 수가 매우 많아지면 정규방정식 계산이 매우 느려짐.

- 일단 A^TA의 역행렬이 구해지면 정규 방정식 계산의 복잡도는 특성수 n과 데이터 샘플수 m에 선형적으로 증가함.
  - ✓ 정규 방정식 계산의 복잡도는 $O(n)+O(m)$.
  - ✓ 따라서 학습된 선형 회귀 모델은 예측이 매우 빠름.

ⓒ dgkim@hanyang.ac.kr

경사 하강법

- 정규 방정식으로 선형 회귀 모델을 훈련할 경우, 특성 수와 훈련 샘플수가 매우 많으면 메모리 저장에 부담이 많아지고, 계산 시간이 느려지므로 이를 극복하기 위해 반복법으로 선형 회귀 모델을 훈련하여야 함.
- 경사 하강법(GDM[Gradient Descent Method])은 여러 종류의 문제에서 최적의 해법을 찾을 수 있는 매우 일반적인 최적화 알고리즘임.

[경사 하강법의 기본 개념]

- 경사 하강법이 기본적인 아이디어는 비용 함수를 최소화하기 위해 반복적으로 파라미터를 조정해가는 것임.

[경사 하강법 계산 과정]

- 파라미터 벡터 θ에 관하여 비용 함수의 현재 그래디언트(gradient)를 계산하고 그 그래디언트가 감소하는 방향으로 진행하여 파라미터 벡터 θ를 개선하는 것임.

- 경사 하강법의 기본 식:

$$\boldsymbol{\theta}^{(update)} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathrm{MSE}(\boldsymbol{\theta})$$

  ▷ η: 학습률 또는 스텝의 크기.

  ▷ 비용 함수 $\mathrm{MSE}(\boldsymbol{\theta})$의 그래디언트:

$$\nabla_{\theta}\mathrm{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \dfrac{\partial}{\partial\theta_1}\mathrm{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \dfrac{\partial}{\partial\theta_n}\mathrm{MSE}(\boldsymbol{\theta}) \\ \dfrac{\partial}{\partial\theta_{n+1}}\mathrm{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m}A^{T}(A\boldsymbol{\theta} - \boldsymbol{y})$$

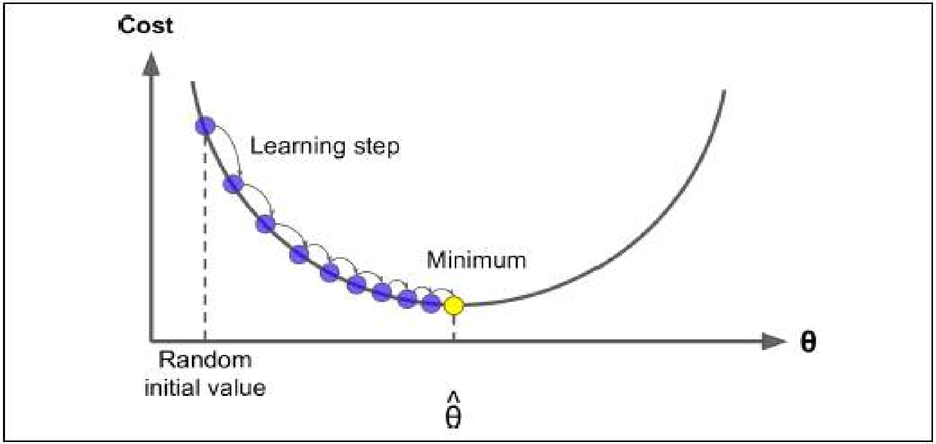
$$\frac{\partial}{\partial\theta_j}\mathrm{MSE}(\boldsymbol{\theta}) = \frac{2}{m}\sum_{i=1}^{m} a_{ij}(\boldsymbol{a}_i\boldsymbol{\theta} - y_i), \quad j = 1, \ldots, n+1$$

$$A = \begin{pmatrix} \boldsymbol{a}_1 \\ \vdots \\ \boldsymbol{a}_m \end{pmatrix} = \begin{pmatrix} x_{10} & \cdots & x_{1p} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_{m0} & \cdots & x_{mp} & 1 \end{pmatrix}$$

✓ 여기서 특성수는 $n = p+1$이고 샘플수는 m.

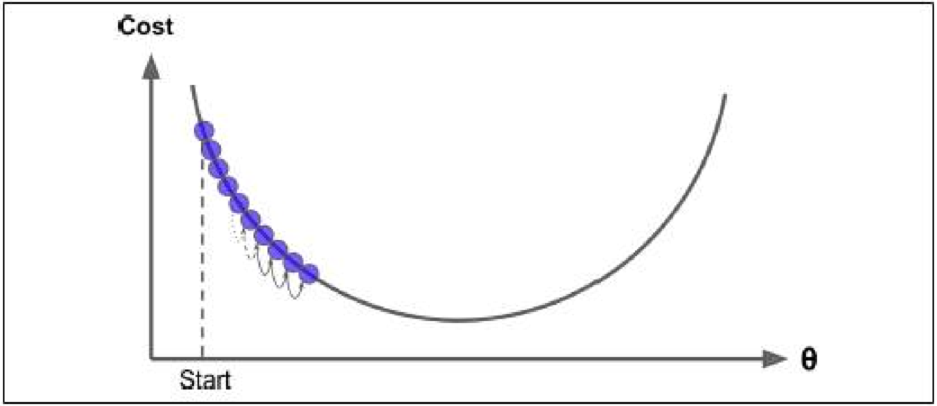
[경사 하강법의 장단점]

- 경사 하강법의 기본 스텝을 반복하여 그래디언트가 0이 되어 비용 함수가 최솟값에 도달하게 되면 알고리즘이 종료됨.
  - ✓ 초기에 파라미터 θ를 임의의 값으로 시작하여(무작위 초기화) 한 번에 조금씩 비용 함수가 감소하는 방향으로 진행함. 이러한 진행과정은 비용 함수가 최솟값에 수렴할 때까지 점진적으로 향상시킴.

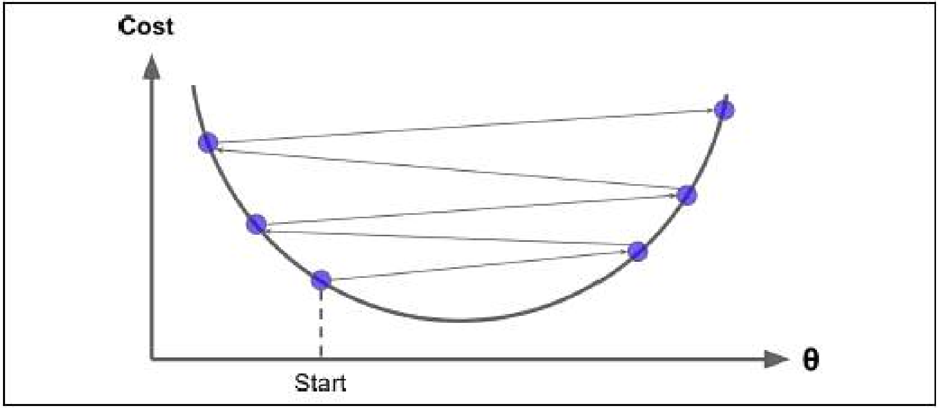


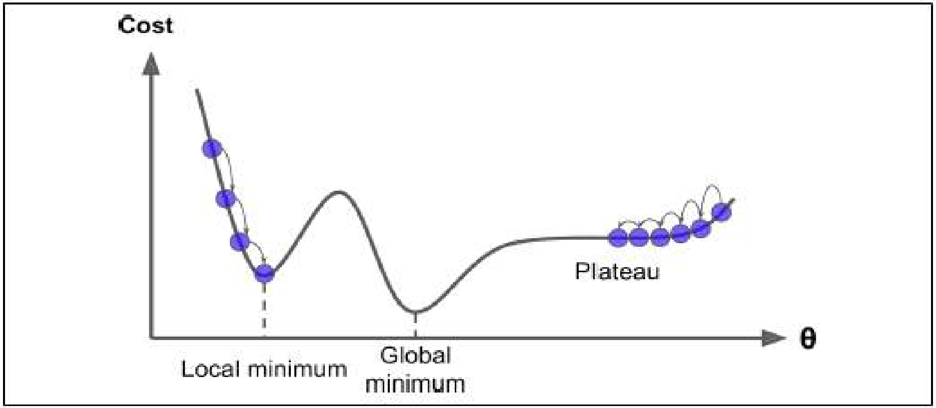
✓ 경사 하강법에서 중요한 하이퍼파라미터는 **스텝의 크기**(step size)로, **학습률**(learning rate)임.

- 학습률이 너무 작으면 알고리즘이 수렴하기 위해 반복을 많이 진행해야 하므로 시간이 많이 소요됨.



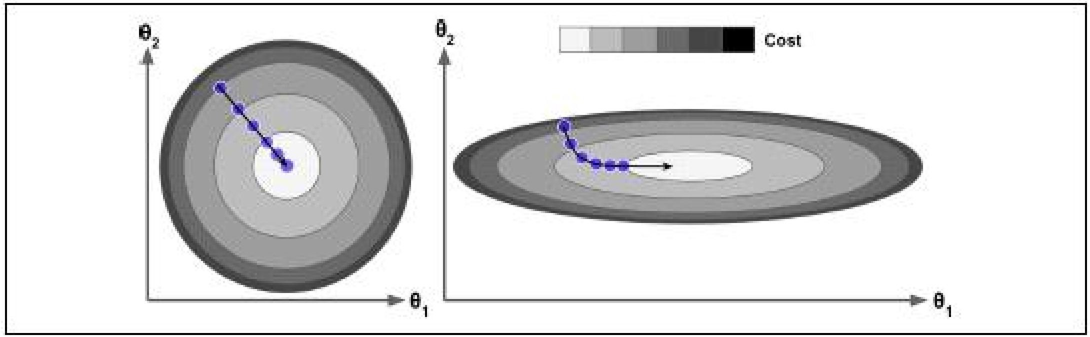
- 학습률이 너무 크면 더 큰 값으로 발산하게 되어 적절한 해를 찾지 못할 수 있음.
- 일반적인 비용 함수인 경우 극솟값(local minimum) 또는 극댓값(local maximum)을 가지므로 경사 하강법은 <u>최솟값(global minimum)으로 수렴하기 어려움</u>.


Cost
θ
Start


Cost
θ
Plateau
Local minimum
Global minimum

- 일반적으로 경사 하강법은 두 가지 문제점을 갖고 있음.
  1) 초기 파라미터 값에 민감함.
  2) 그래디언트의 크기가 작은 지역에서는 진행 속도가 느림.

- 선형 회귀를 위한 MSE 비용 함수는 볼록 함수이기 때문에 경사 하강법이 최솟값에 수렴함. 그러나 특성 간에 스케일의 차이가 많으면, 수렴 속도가 느려질 수 있음.
  - ✓ 왼쪽 그림은 경사 하강법 알고리즘이 최솟값으로 곧장 진행하여 도달하고 있음.
  - ✓ 반면, 오른쪽 그림은 처음에 최솟값 방향으로 진행하다가 평편한 골짜기로 길게 돌아 느리게 도달하고 있음.



- ✓ 경사 하강법을 사용할 때는 반드시 <u>모든 특성이 같은 스케일을 갖도록 조정</u>해야 함.
- ✓ 특히, 모델이 가진 파라미터가 많을수록 파라미터 공간의 차원이 커져 검색이 어려워짐.

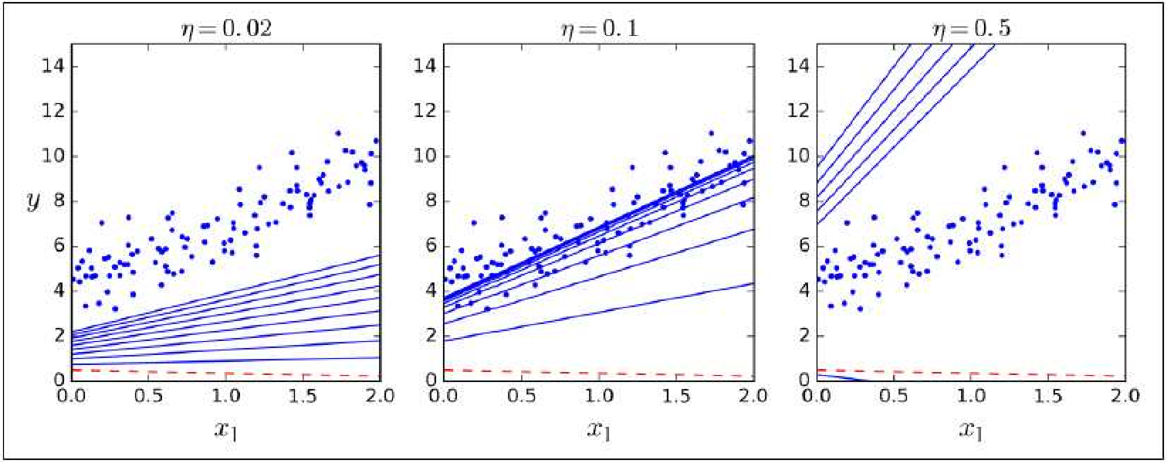
경사 하강법 알고리즘의 종류

1) 배치 경사 하강법 (Batch Gradient Descent)
2) 확률적 경사 하강법 (Stochastic Gradient Descent)
3) 미니배치 경사 하강법 (Mini-batch Gradient Descent)

[배치 경사 하강법]

- 경사 하강법 알고리즘의 매 스텝에서 전체 훈련 세트를 포함하는 디자인 행렬 A에 대해 계산하는 알고리즘을 **배치 경사 하강법(Batch Gradient Descent)**이라고 함.
  - ✓ 경사 하강법은 특성 수에는 민감하지 않으므로, 수십만 개의 특성에서 선형 회귀 훈련을 시키려면 정규 방정식보다 경사 하강법이 효율적임.
  - ✓ 배치 경사 하강법은 매우 큰 훈련 세트에서는 계산 속도가 아주 느려짐.

- 여러 가지 학습률에 따른 배치 경사 하강법



✓ 왼쪽은 학습률이 너무 낮아 알고리즘이 최적점에 수렴하지만, 시간이 오래 걸림.

✓ 가운데는 학습률이 아주 적당하여, 반복 몇 번 만에 최적점에 수렴함.

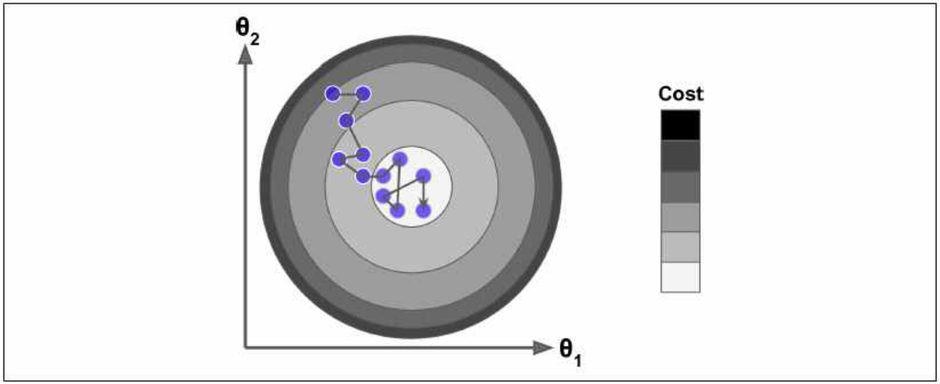
✓ 오른쪽은 학습률이 너무 높아, 알고리듬이 이리저리 널뛰면서 매 단계마다 최적점에서 멀어지고 있음.

- 적절한 학습률은 그리드 탐색을 사용하여 찾을 수 있음.
- 반복 횟수의 지정 하기.
  - ✓ 반복 횟수가 너무 작으면, 최적점에 수렴하기 전에 알고리즘이 멈출 수 있음.
  - ✓ 반복 횟수가 너무 크면, 더 이상 변하지 않는 동안 시간을 낭비할 수 있음.
  - ✓ 간단한 해결책은 반복 횟수를 크게 정하고, 그래디언트의 크기가 허용값 ϵ(tolerance) 이내로 아주 작아지면 경사 하강법이 최솟값에 수렴한 것으로 간주하여 알고리즘을 중단시킴.

[확률적 경사 하강법]

- 배치 경사 하강법의 가장 큰 문제는 매 스텝에서 전체 훈련 세트를 사용해 그래디언트를 계산해야하는 것임. 따라서 훈련 세트가 커지면 매우 느려지게 됨.
- **확률적 경사 하강법** (SGD[SGDStochastic Gradient Descent])은 매 스텝에서 한 개의 훈련 데이터 포인트 샘플을 무작위로 선택하고 그 하나의 샘플에 대한 그래디언트를 계산함.

- SGD알고리즘의 장점
  - ✓ 매 반복에서 매우 적은 데이터만 처리하기 때문에 알고리즘의 수렴 속도가 훨씬 빠름.
  - ✓ 매 반복에서 하나의 샘플만 메모리에 있으면 되므로, 매우 큰 훈련 세트도 훈련시킬 수 있음.
- SGD알고리즘의 단점
  - ✓ 무작위적으로 샘플 하나만을 사용하여 그래디언트를 계산하기 때문에, 배치 경사 하강법 보다 훨씬 불안정하게 (평균적으로) 수렴함.
  - ✓ 무작위성으로 인해 스텝의 크기가 적절하지 못하면, 극솟값(local minimum)에서 탈출할 수 있지만, 최솟값(global minimum)에 다다르지 못 할 경우가 발생함.



- SGD 알고리즘의 학습 스케줄
  - ✓ 초기에는 학습률을 크게 하고, 점차 작게 줄여서 알고리즘이 최솟값에 도달하게 함.
  - ✓ 매 반복에서 학습률을 결정하는 함수를 **학습 스케줄(learning schedule)**이라 함.

- 다음 코드는 'SGDRegressor' 클래스를 사용하여 학습률 0.1로 기본 학습 스케줄에 따라 에포크를 50번 수행하는 것임.

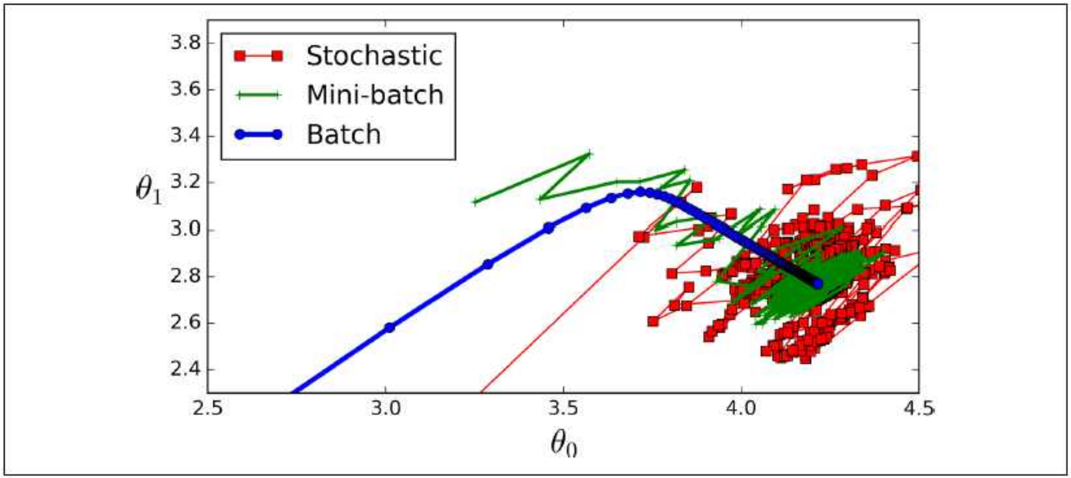
```
from sklearn.linear_model import SGDRegressor

sgd_reg = SGDRegressor(max_iter=50, penalty=None, eta0=0.1)
sgd_reg.fit(X, y.ravel())
```

```
sgd_reg.intercept_, sgd_reg.coef_
```

```
(array([ 4.18380366]), array([ 2.74205299]))
```

미니배치 경사 하강법

- **미니배치 경사 하강법**은 **미니배치(mini-batch)**라 부르는 임의의 작은 샘플 세트에 대해 그래디언트를 계산하여 적용하는 경사 하강법임.
- 미니배치 경사 하강법의 주요 장점은 행렬 연산에 최적화된 하드웨어, 특히 GPU를 사용해서 얻는 성능향상임.
- 다음 그림은 세 가지 경사 하강법 알고리즘이 훈련 과정 동안 파라미터 공간에서 움직인 경로임.



- **지금까지 논의한 선형 회귀 알고리즘의 비교**

알고리즘	$m \gg 1$	$n \gg 1$	하이퍼 파라미터 수	스케일 조정 필요	사이킷런
Normal Eq.	fast	slow	0	No	LinearRegression
BGD	slow	fast	2	Yes	n/a
SGD	fast	fast	2 이상	Yes	SGDRegressor
MBGD	fast	fast	2 이상	Yes	n/a

✓ **m은 훈련 샘플 수이고 n은 특성 수임.**

선형 회귀의 성능

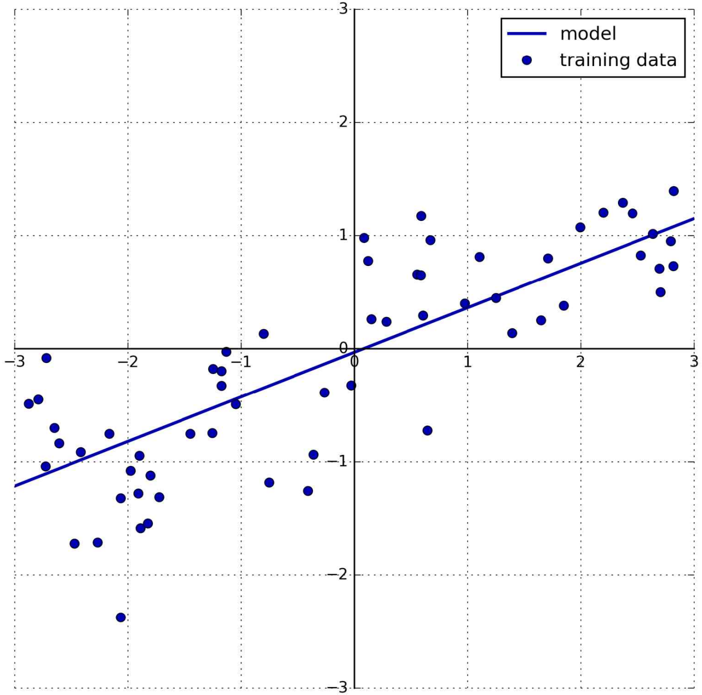
- 다음은 아래 그림의 선형 모델을 만드는 코드임.

```
from sklearn.linear_model import LinearRegression

X, y = mglearn.datasets.make_wave(n_samples=60)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

lr = LinearRegression().fit(X_train, y_train)
```

```
print("lr.coef_:", lr.coef_)
print("lr.intercept_:", lr.intercept_)
```

```
lr.coef_: [0.394]
lr.intercept_: -0.031804343026759746
```

✓ “lr”의 객체 “coef_”와 “intercept_”의 속성에 각각 가중치와 절편(편향)의 정보가 저장되어 있음.



ⓒ dgkim@hanyang.ac.kr

- 훈련 세트와 테스트 세트의 성능 확인.

```python
print("Training set score: {:.2f}".format(lr.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lr.score(X_test, y_test)))
```

```
Training set score: 0.67
Test set score: 0.66
```

✓ 테스트 세트의 점수 R^2값이 0.66이라는 것은 그리 좋은 결과는 아님.

✓ 훈련 세트와 테스트 세트의 점수가 비슷하다는 것은 모델이 과대적합이 아니라 과소적합일 가능성이 높음.

✓ 선형 모델의 경우 저차원 데이터셋에서는 모델이 단순하여 과대적합이 될 수 없음.

✓ 고차원 데이터셋은 특성수가 증가하여 선형 모델의 성능이 매우 높아져 과대적합의 가능성이 높음.

[보스턴 주택가격 데이터셋에 적용한 선형 모델]

- 다음은 선형 모델이 보스턴 주택가격 데이터셋 (복잡한 데이터셋)에서 어떻게 작동하는지 살펴보고자 함.
  - ✓ 이 데이터셋은 1978년 발표된 보스턴 지역의 주택 가격에 영향을 미치는 요소임.
  - ✓ 샘플수가 506개이고, 유도된 특성까지 포함하여 특성수가 104개임.

1	CRIM	자치시별 1인당 범죄율
2	ZN	25,000 평방피트를 초과하는 거주지역의 비율
3	INDUS	비소매 상업지역이 점유하고 있는 토지의 비율
4	CHAS	찰스강에 대한 더미변수(강의 경계에 위치한 경우 1, 아니면 0)
5	NOX	10ppm 당 농축 이산화질소
6	RM	주택 1가구당 평균 방의 개수
7	AGE	1940년 이전에 건축된 소유주택의 비율
8	DIS	5개 보스턴 직업센터까지의 접근성 지수
9	RAD	방사형 도로까지의 접근성 지수
10	TAX	10,000 달러 당 재산세율
11	PTRATIO	자치시별 학생/교사 비율
12	B	1000(Bk-0.63)^2, 이때 Bk는 자치시별 흑인의 비율
13	LSTAT	모집단의 하위계층의 비율(%)

- 먼저 데이터셋을 읽어 들이고 훈련 세트와 테스트 세트로 나누어 선형 모델을 만듦.

```
X, y = mglearn.datasets.load_extended_boston()


X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
lr = LinearRegression().fit(X_train, y_train)
```

```
print("Training set score: {:.2f}".format(lr.score(X_train, y_train)))
print("Test set score: {:.2f}".format(lr.score(X_test, y_test)))
```

```
Training set score: 0.95
Test set score: 0.61
```

✓ 이 경우 이 선형 모델이 과대적합되었다는 것을 알 수 있음.