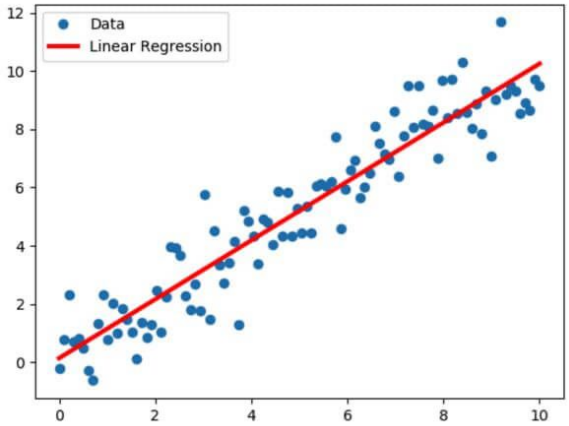
선형회귀

1조 이의찬 정세웅 한형진 황예은

Part 1 선형회귀

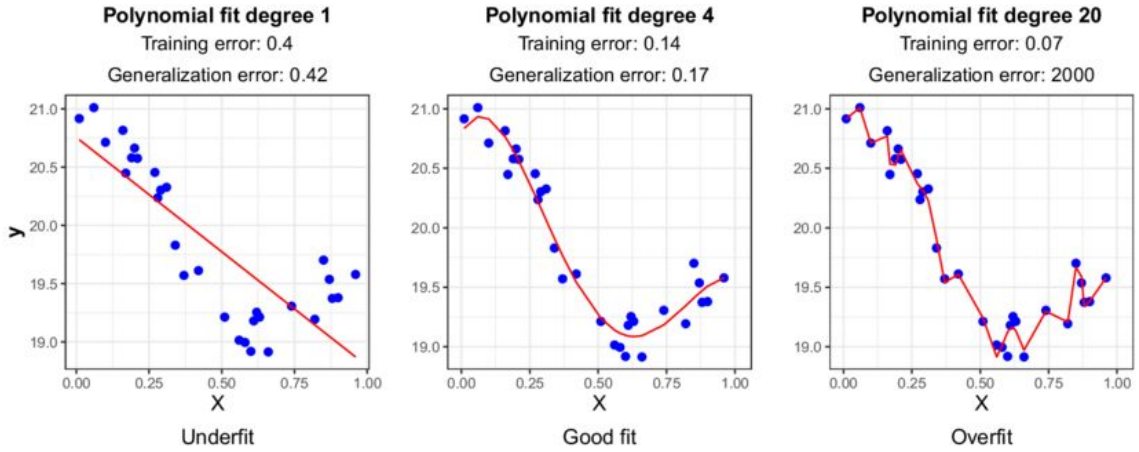
1-1. 정의

- **지도학습** (supervised learning) : 정답 레이블 y가 있는 데이터 x로 학습
  x를 y에 mapping하는 target function을 찾는 문제
  - **분류** (classification) : 예측하는 결과값이 이산적
  - **회귀** (regression) : 예측하는 결과값이 연속적
    - **선형회귀** (linear regression) : target function f(x)가 선형함수인 회귀



1-1. 정의

- **가설** (hypothesis, H(x)) : target function을 근사하는 candidate model

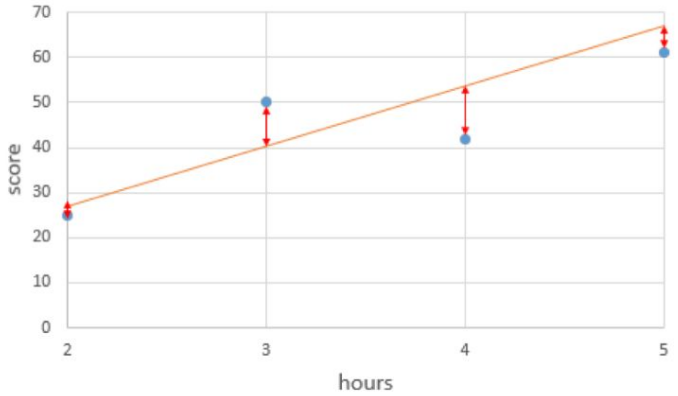


1-1. 정의

- **가설** (hypothesis, H(x)) : target function을 근사하는 candidate model
- **비용 함수** (cost function)

  (= 손실 함수(loss function) = 오차 함수(error function) = 목적 함수(objective function))

  : 가설의 정확도를 측정. 예측값 H(x)와 실제값 y간의 거리.



1-1. 정의

- **가설** (hypothesis, H(x)) : target function을 근사하는 candidate model
- **비용 함수** (cost function)

  (= 손실 함수(loss function) = 오차 함수(error function) = 목적 함수(objective function))

  : 가설의 정확도를 측정. 예측값 H(x)와 실제값 y 간의 거리.
- **최적화** (optimization) : 비용 함수의 결과값을 최소화하는 H(x)의 파라미터를 찾는 것

=> 문제에 따라 적합한 가설, 비용 함수 사용! 선형 회귀는?

1-2. 선형회귀의 가설

H(x) = Wx + b

가중치
(weight)

편향
(bias)

n = feature 수
m = 샘플 수

$$\hat{y} = w_1 \times x_1 + w_1 \times x_1 + \cdots + w_n \times x_n + b$$

상수항 결합 (bias augmentation)

b 1

$$\hat{y} = \begin{pmatrix} w_0 & w_1 & \cdots & w_n \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} = \mathbf{w}^T\mathbf{x}$$

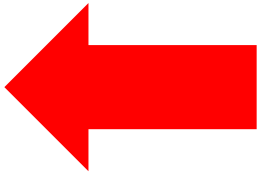
$$\hat{\mathbf{y}} = \begin{pmatrix} x_0^1 & x_1^1 & \cdots & x_n^1 \\ x_0^2 & x_1^2 & \cdots & x_n^2 \\ \vdots & & & \\ x_0^m & x_1^m & \cdots & x_n^m \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \\ \cdots \\ w_n \end{pmatrix} = \begin{pmatrix} \mathbf{x}^1 \\ \mathbf{x}^2 \\ \vdots \\ \mathbf{x}^m \end{pmatrix} \cdot \begin{pmatrix} w_0 \\ w_1 \\ \cdots \\ w_n \end{pmatrix} = \mathbf{X}\mathbf{w}$$

1-3. 선형회귀의 비용 함수

1) **평균 제곱 오차** (Mean Squared Error, MSE, L2 loss)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left\{y^{(i)} - H(x^{(i)})\right\}^2$$

- 제곱으로 인해 이상치에 민감
- convex



대부분 MSE 사용!

2) **평균 절대 오차** (Mean Absolute Error, MAE, L1 loss)

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y^{(i)} - H(x^{(i)})\right|$$

- 이상치에 대해 강건(robust)

3) Huber loss

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

- 오차가 δ 미만이면 제곱, 오차가 δ 이상이면 절대값을 취하는 MSE와 MAE의 절충안

1-3. 선형회귀의 비용 함수

+) **최소제곱법** (ordinary least squares) : 어떤 계의 해방정식을 근사적으로 구하는 방법
잔차제곱합(RSS: Residual Sum of Squares)를 최소화하는 가중치 벡터를 구한다.

→ **sklearn의 LinearRegression이 최소제곱법을 사용**

$$\hat{y} = Xw$$

$$\begin{aligned} \text{RSS} &= e^T e \\ &= (y - Xw)^T (y - Xw) \\ &= y^T y - 2y^T Xw + w^T X^T Xw \end{aligned}$$

$$\frac{d\text{RSS}}{dw} = -2X^T y + 2X^T Xw$$

$$\frac{d\text{RSS}}{dw} = 0$$

$$X^T Xw^* = X^T y$$

normal equation :

$$w^* = (X^T X)^{-1} X^T y$$

예제: 농어의 길이로부터 무게 예측

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

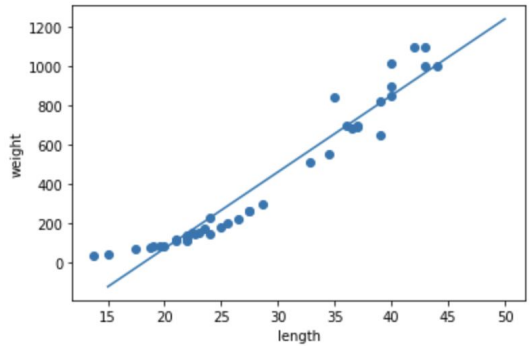
```
# 농어의 길이, 무게 데이터
perch_length = np.array([8.4, 13.7, 15.0, 16.2, 17.4, 18.0, 18.7, 19.0, 19.6, 20.0, 21.0,
       21.0, 21.0, 21.3, 22.0, 22.0, 22.0, 22.0, 22.0, 22.5, 22.5, 22.7,
       23.0, 23.5, 24.0, 24.0, 24.6, 25.0, 25.6, 26.5, 27.3, 27.5, 27.5,
       27.5, 28.0, 28.7, 30.0, 32.8, 34.5, 35.0, 36.5, 36.0, 37.0, 37.0,
       39.0, 39.0, 39.0, 40.0, 40.0, 40.0, 40.0, 42.0, 43.0, 43.0, 43.5,
       44.0])
perch_weight = np.array([5.9, 32.0, 40.0, 51.5, 70.0, 100.0, 78.0, 80.0, 85.0, 85.0, 110.0,
       115.0, 125.0, 130.0, 120.0, 120.0, 130.0, 135.0, 110.0, 130.0,
       150.0, 145.0, 150.0, 170.0, 225.0, 145.0, 188.0, 180.0, 197.0,
       218.0, 300.0, 260.0, 265.0, 250.0, 250.0, 300.0, 320.0, 514.0,
       556.0, 840.0, 685.0, 700.0, 700.0, 690.0, 900.0, 650.0, 820.0,
       850.0, 900.0, 1015.0, 820.0, 1100.0, 1000.0, 1100.0, 1000.0,
       1000.0])
```

```
# 훈련 세트와 테스트 세트로 나눕니다
train_input, test_input, train_target, test_target = train_test_split(
    perch_length, perch_weight, random_state=42)

# sklearn.linear_model.LinearRegression.fit(X, y) : X의 shape (sample 수, feature 수)
train_input = train_input.reshape(-1, 1)
test_input = test_input.reshape(-1, 1)
```

예제: 농어의 길이로부터 무게 예측

```
lr = LinearRegression()
lr.fit(train_input, train_target)
```

```
plt.scatter(train_input, train_target)

plt.plot([15, 50], [15 * lr.coef_ + lr.intercept_, 50 * lr.coef_ + lr.intercept_])
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



lr.coef_ = W
- shape : (feature 수,)
  (feature 수, target 수)

lr.intercept_ = b
- shape : (target tn,)

```
print(lr.score(train_input, train_target))
print(lr.score(test_input, test_target))
```

```
0.939846333997604
0.8247503123313558
```

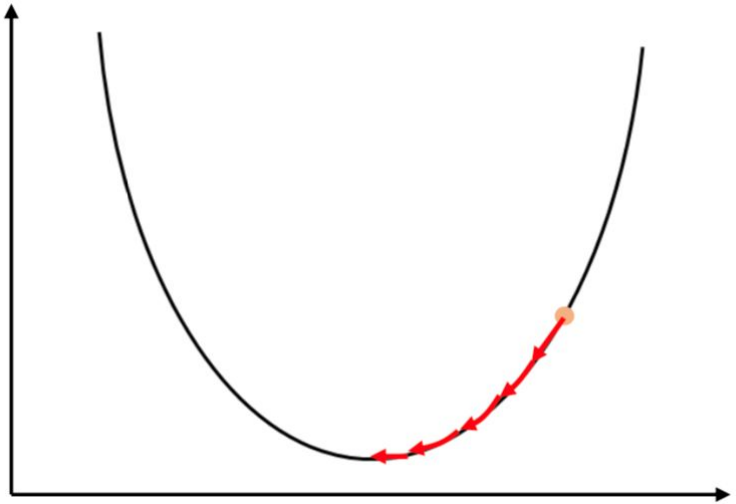
$$R^2 = 1 - \frac{SSE}{SST}$$

$$SSE = \sum_{i=1}^{m}(\hat{y}_i - y_i)^2$$

$$SST = \sum_{i=1}^{m}(y_i - \bar{y}_i)^2$$

lr.score : 결정계수
(coefficient of determination, R^2)
- 추정한 선형 모형이 주어진 자료에 적합한 정도를 재는 척도 [0, 1]

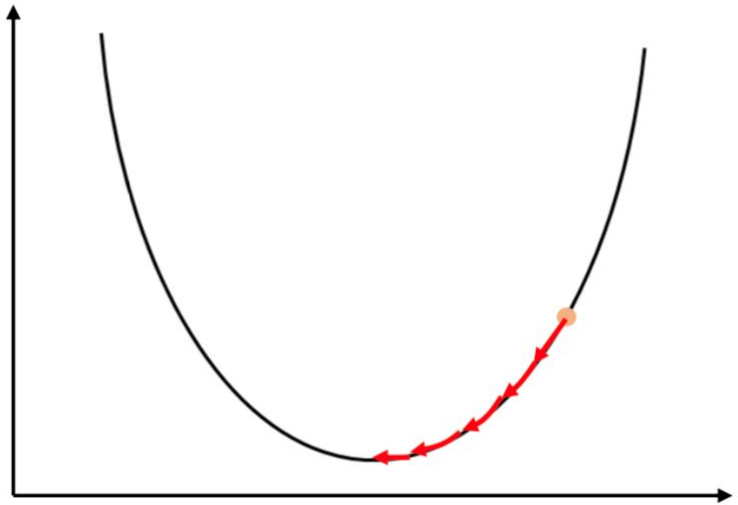
1-4. 최적화 - 경사하강법



- 앞서 정의한 비용 함수의 값을 최소로 하는 **W**와 **b**의 값을 찾을 때 사용되는 옵티마이저 알고리즘 (최적화 알고리즘이라고도 부름)

- 옵타마이저 알고리즘을 통해 적절한 W와 b를 찾는 과정을 머신 러닝에서 학습이라고 부름

1-4. 최적화 - 경사하강법



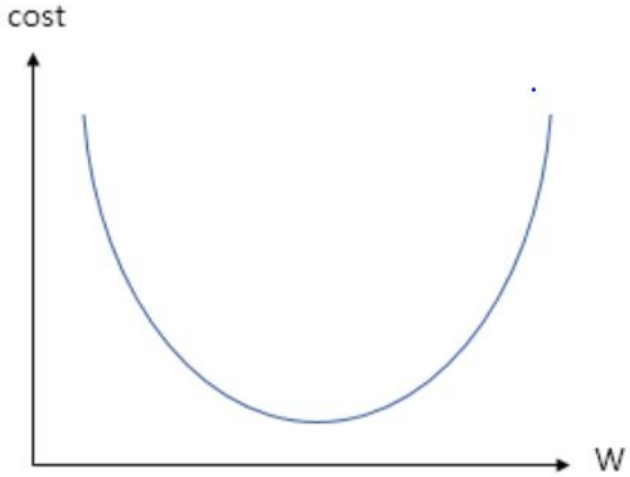
경사하강법

1차 미분계수를 이용해 함수의 최소값을 찾아가는 방법으로, 함수 값이 낮아지는 방향으로 독립 변수 값을 변형시켜가면서 최종적으로 최소 함수 값을 갖도록 하는 독립 변수 값을 찾는 방법

가장 기본적인 옵티마이저 알고리즘인 경사하강법

1-4. 최적화 - 경사하강법

비용함수와 가설 함수의 기울기 ‘W’의 관계로 보는 경사하강 과정

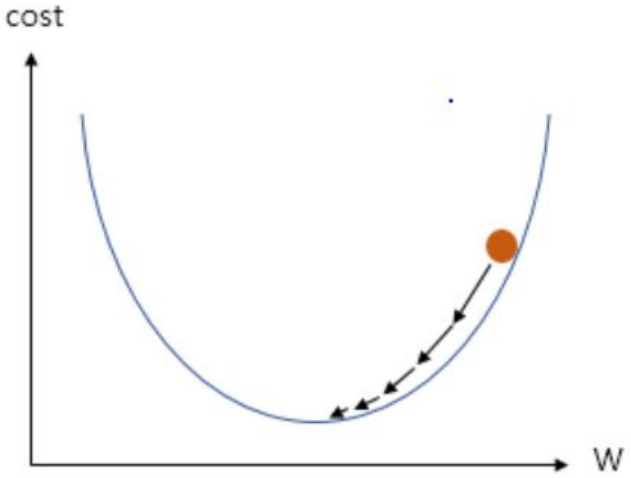


- 기울기 W 가 무한대로 커지면 커질 수록 cost의 값 또한 무한대로 커지고, 반대로 기울기 W가 무한대로 작아져도 cost의 값은 무한대로 커짐.

- 위의 그래프에서 cost가 가장 작을 때는 맨 아래의 볼록한 부분이므로 cost가 가장 최소값을 가지게 하려면 맨 아래의 볼록한 부분의 W 의 값을 찾아야 함.

1-4. 최적화 - 경사하강법

비용함수와 가설 함수의 기울기 'W'의 관계로 보는 경사하강 과정

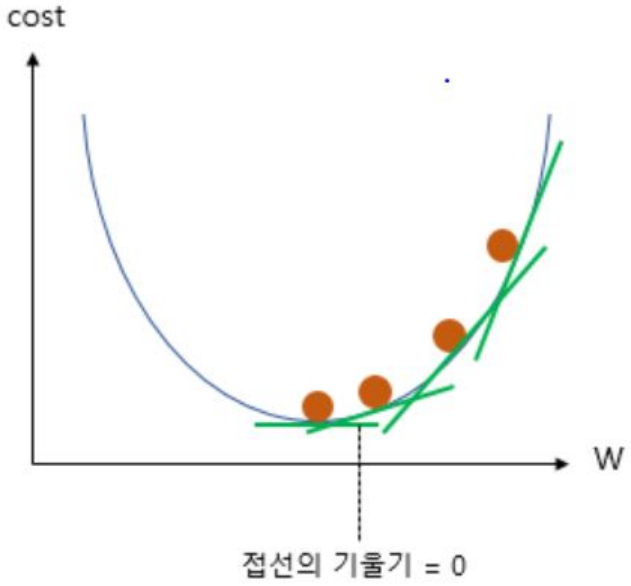


임의의 초기값 W 값이 정해지면, 맨 아래의
볼록한 부분을 향해 점차 W의 값을 수정해감

경사 하강법은 미분에서 배우는
한 점에서의 순간 변화율 또는
접선에서의 기울기의 개념을 사용

1-4. 최적화 - 경사하강법

비용함수와 가설 함수의 기울기 'W'의 관계로 보는 경사하강 과정



초록색 선에서 W는 접선의 기울기를 보여주는데, 그래프의 볼록한 부분으로 갈수록 접선의 기울기가 점차 작아짐

맨 아래의 볼록한 부분에서는 접선의 기울기가 0이 됨

즉, cost가 최소화가 되는 지점은 접선의 기울기가 0이 되는 지점이며, 또한 미분값이 0이 되는 지점

1-4. 최적화 - 경사하강법

경사 하강 과정을 나타내는 공식

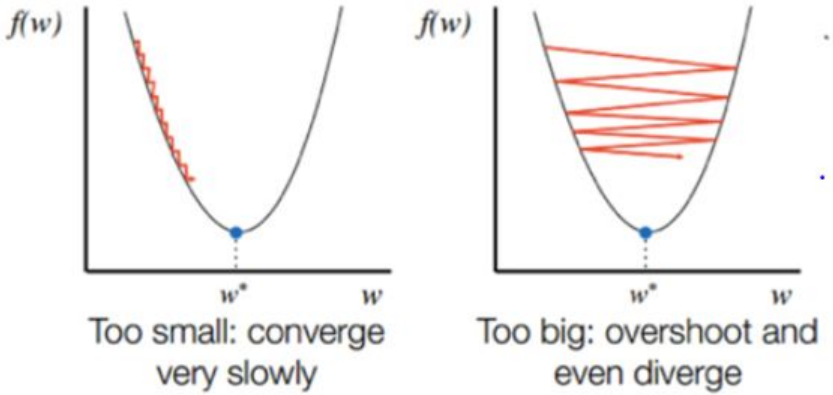
$$W := W - \alpha \frac{\partial}{\partial W} cost(W)$$

해당 공식에서 알파는 학습률을 의미

경사하강법은 즉, 비용 함수(Cost function)를 미분하여 현재 W에서의 접선의 기울기를 구하고, 접선의 기울기가 낮은 방향으로 W의 값을 변경하는 작업을 반복하는 것

학습률 : 경사 하강법에서 가중치(w)와 편향(b)을 반복 학습시킬 때, 한번 반복 학습 시킬 때마다 얼만큼씩 이동시킬 것인지를 정하는 상수 (사람이 임의로 지정)

1-4. 최적화 - 경사하강법



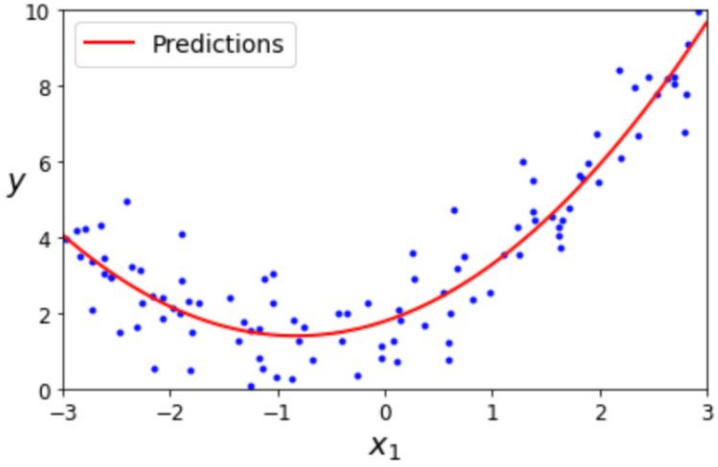
학습률이 너무 작을 경우
- 알고리즘이 수렴하기 위해 반복해야 하는 값이 많으므로 학습 시간이 오래 걸림
- 지역 최소값(local minimum)에 수렴할 수 있음

학습률이 너무 클 경우
- 스텝이 너무 커서 전역 최소값(global minimum)을 가로질러반대편으로 건너뛰어 최소값에서 멀어질 수 있다.

최적의 학습률을 찾는 것이 중요!

Part 2 다항회귀

2-1. 다항 회귀



다항 회귀

회귀식의 독립변수가 2차, 3차 방정식 같은 다항식으로 표현되는 선형 회귀

-> 차원이 높은 다항식인데 어떻게 선형 회귀를 할 수 있지? y = a*x^2 + b * x + c 라는 다항식이 있을 때 x^2=z로 치환하면 y= a * z + b * x + c라는 선형식으로 쓸 수 있다.

비선형 데이터도 선형 회귀를 통해 학습 및 예측 가능

2-1. 다항 회귀

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
```

```
poly_ftr = PolynomialFeatures(degree=3).fit_transform(X)
```

```
model = LinearRegression()
```

사이킷 런에서는 직접적으로 다항회귀를 구현해주는 클래스를 가지고 있지 않다. 대신, PolynomialFeatures라는 피처를 다항식 피처로 변환해주는 클래스를 이용하여 구현한다.

1. 피처를 PolynomialFeature 클래스를 이용하여 차수를 정하고 다항식 피처로 변환한다.

2. 이후에 선형회귀 클래스인 LinearRegression()을 이용해서 학습 - 예측을 수행한다.

2-2. 다항 회귀 구현

```
#라이브러리 임포트
import numpy as np
import numpy.random as rnd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
```

1. 필요한 라이브러리 import

```
#비선형 데이터 생성
m = 100
X = 6 * np.random.rand(m, 1) - 3
y = 0.5 * X**2 + X + 2 +np.random.randn(m,1)
```

2. 비선형 데이터 생성

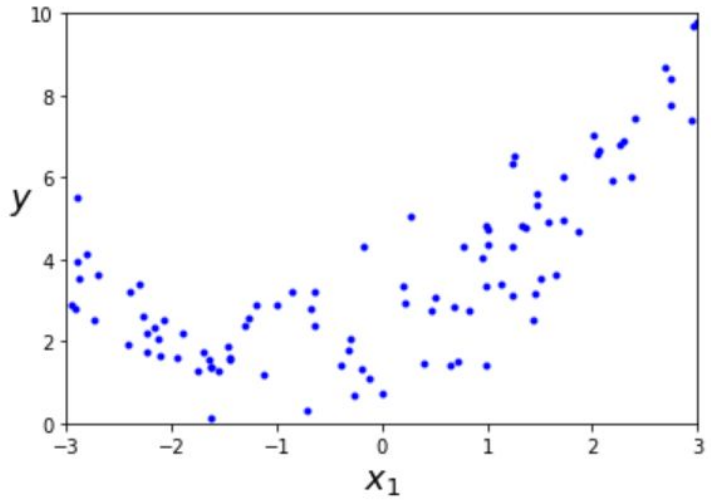
np.random.rand(m,1)은 0 ~ 1 사이의 난수를 생성해 m by 1 matrix를 생성

X = 6 * np.random.rand(m, 1) – 3에서
X의 범위는 – 3 < X < 3

y에 원하는 식으로 표현

2-2. 다항 회귀 구현

```
#데이터 시각화
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()
```



3. 데이터 시각화

앞서 생성한 X와 y를 시각화하면
왼쪽과 같이 비선형 데이터 확인 가능

2-2. 다항 회귀 구현

```
#PolynomialFeature를 이용하여 2차식 피처로 변환

poly_features = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly_features.fit_transform(X)
```

4. 2차식 피처로 변환

PolynomialFeatures와 fit_transform으로
X를 2차식 피처로 변환

```
#선형 회귀 객체 생성
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
```

```
LinearRegression()
```

5. 선형 회귀 객체 생성

LinearRegression을 이용해 객체 생성 및 학습

2-2. 다항 회귀 구현

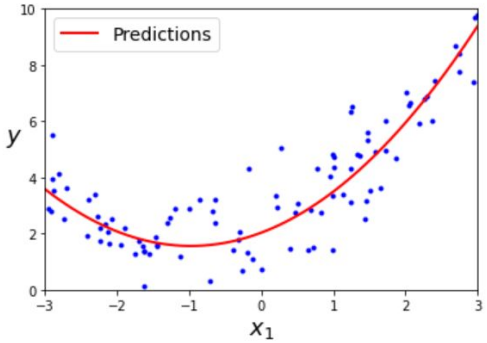
```
#예측
X_new=np.linspace(-3, 3, 100).reshape(100, 1)
X_new_poly = poly_features.transform(X_new)
y_new = lin_reg.predict(X_new_poly)
```

6. 예측

새로 생성한 X_new를 transform 시키고 predict를 통해 출력할 y_new 값을 최종적으로 생성

Reshape(100,1)을 통해
100개의 rows 와 1 columns로

```
#다항 회귀 시각화
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()
```



7. 다항 회귀선 시각화