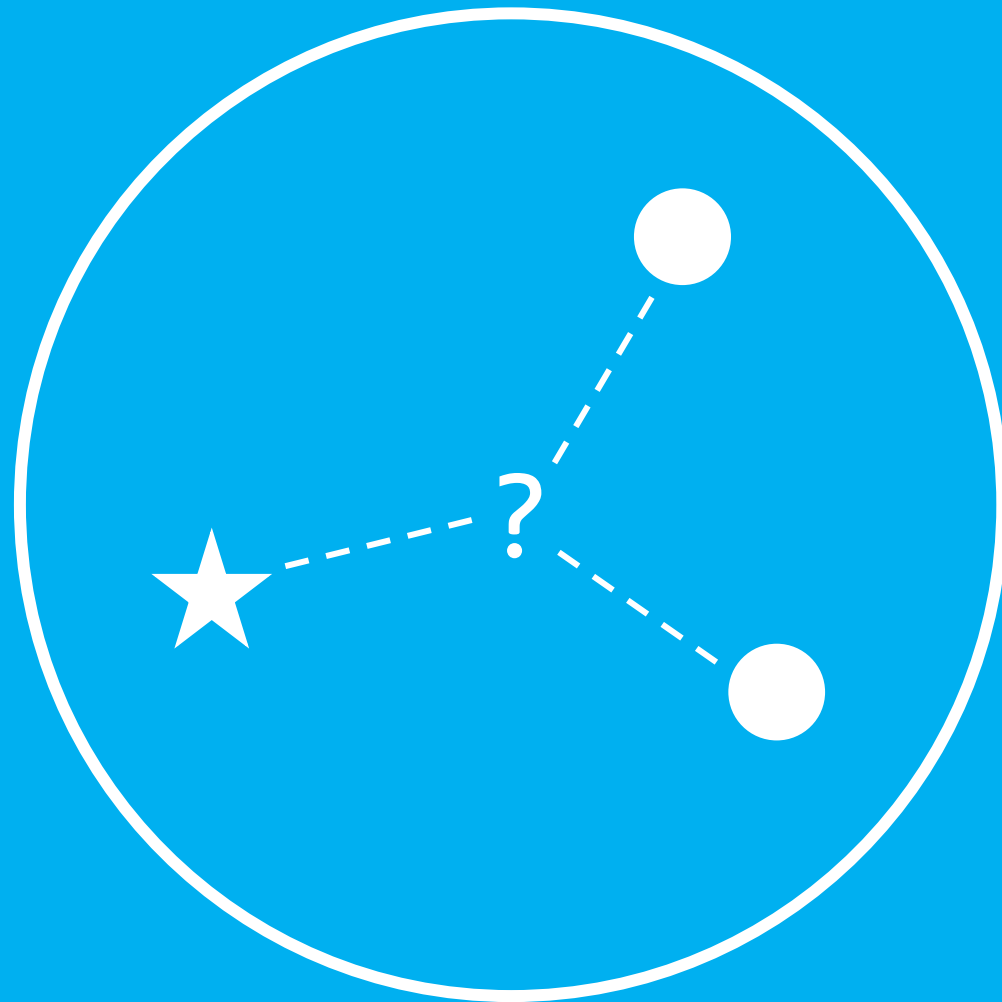


# 3주차 정규세션

K-최근접 이웃 회귀



# 진도 세션 개요

## K-최근접 이웃회귀

### 회귀(Regression)

- 회귀 개념
- 다양한 회귀모델

- KNN 이해하기
- 교과서 예제 진행
- 과대적합 & 과소적합

### 실습

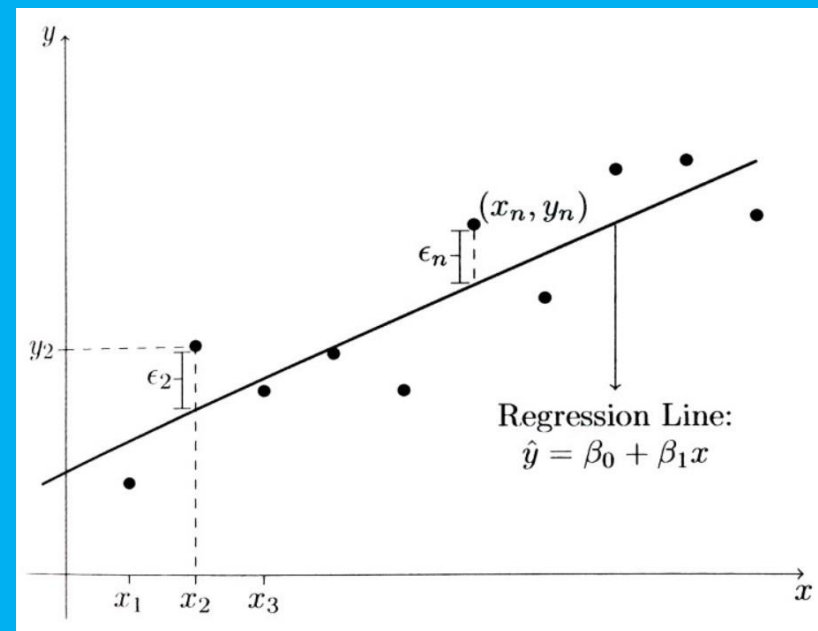
- 간단한 KNN 실습

# K-최근접 이웃 회귀

## 회귀:

[사전적 의미] 하나의 종속 변수와 두 개 이상의 독립 변수 사이에 나타나는 관계를 최소 제곱법으로 추정하는 방법.

교과서: 임의의 어떤 숫자를 예측하는 것.



# K-최근접 이웃 회귀

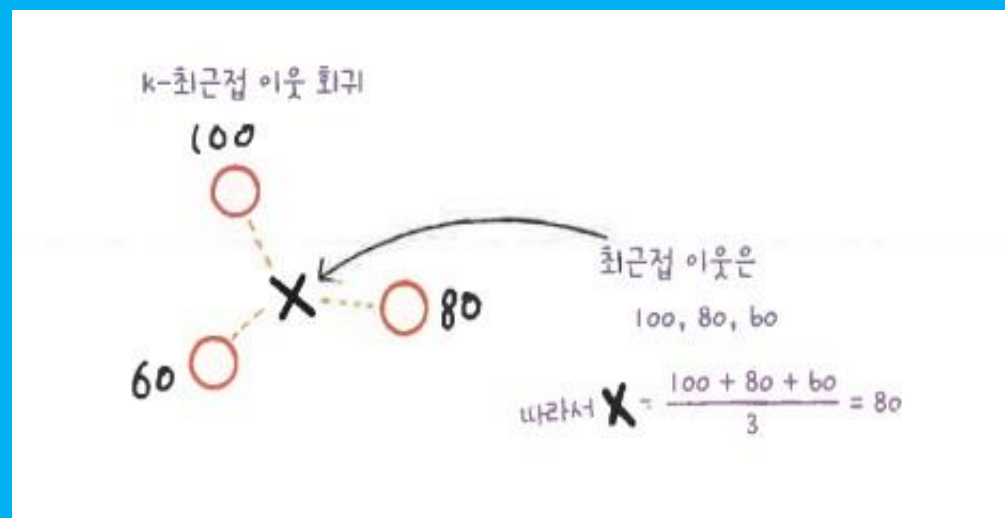
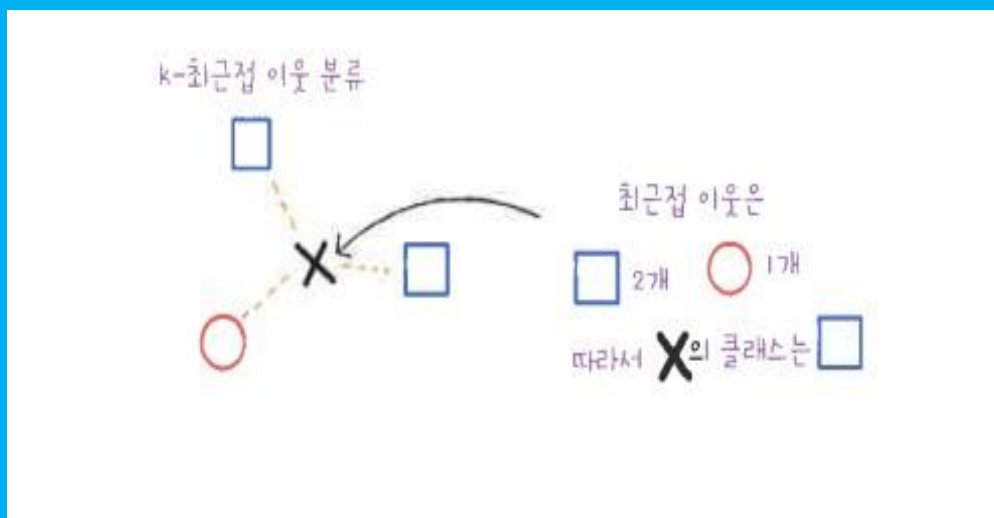
## 회귀와 분류 차이:

분류(classify)

다른 그룹으로 구분 해 놓는 것  
클래스가 target

회귀(regression)

이웃 값을 기준으로 수치를 예측  
임의의 값이 target

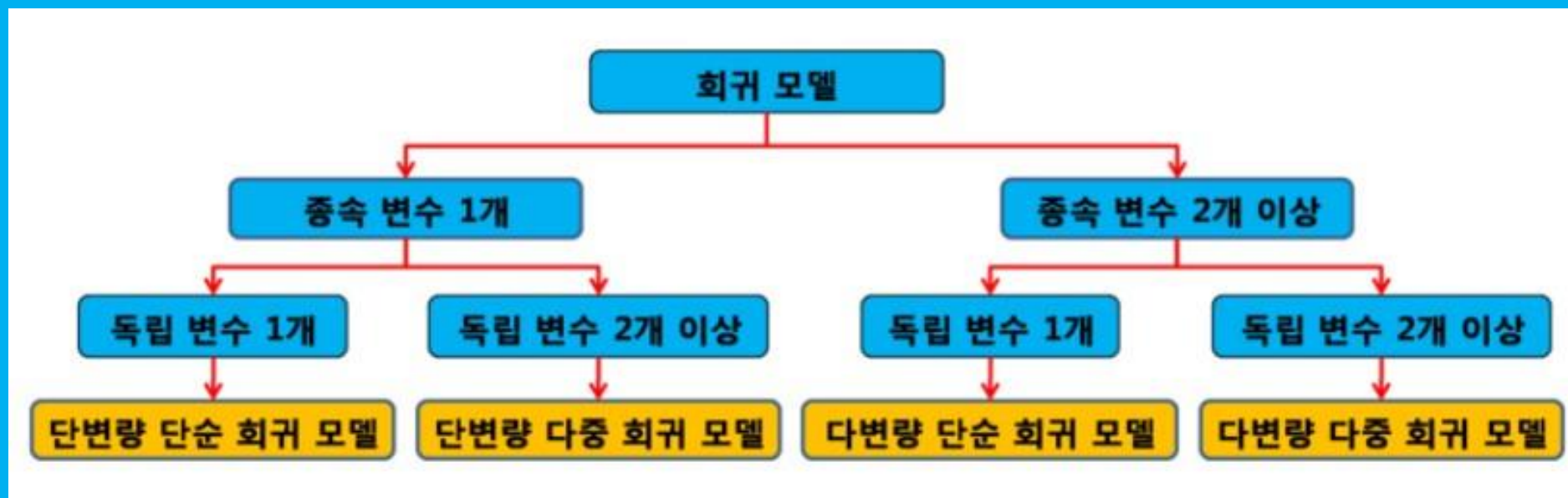


# K-최근접 이웃 회귀

## 여러가지 회귀모델

선형성, 비선형성에 따른 차이

종속 변수의 개수, 독립 변수의 개수에 따라서 종류 상이

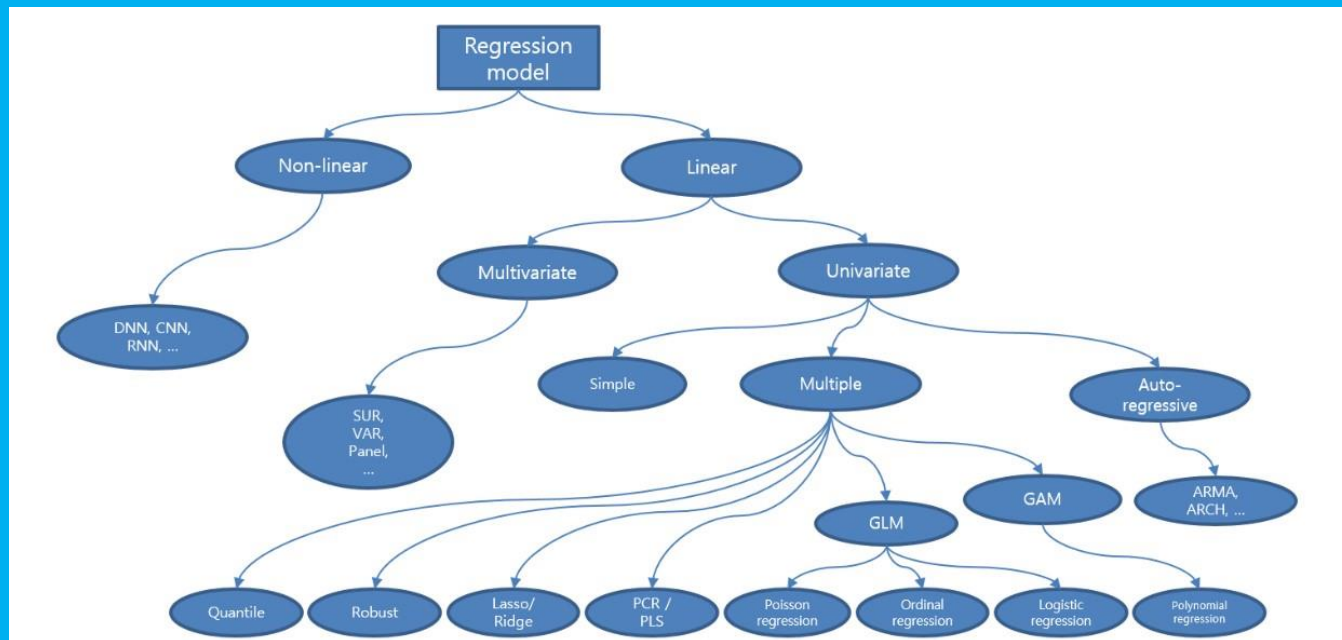


# K-최근접 이웃 회귀

## 여러가지 회귀모델

데이터의 특징, 모델링 목적에 따라 적절한 회귀 모델을 선택  
탐사 분석 과정을 통해 데이터 특성 파악  
만든 모델과 결과가 잘 맞는지 검증

All models are wrong, but some are useful

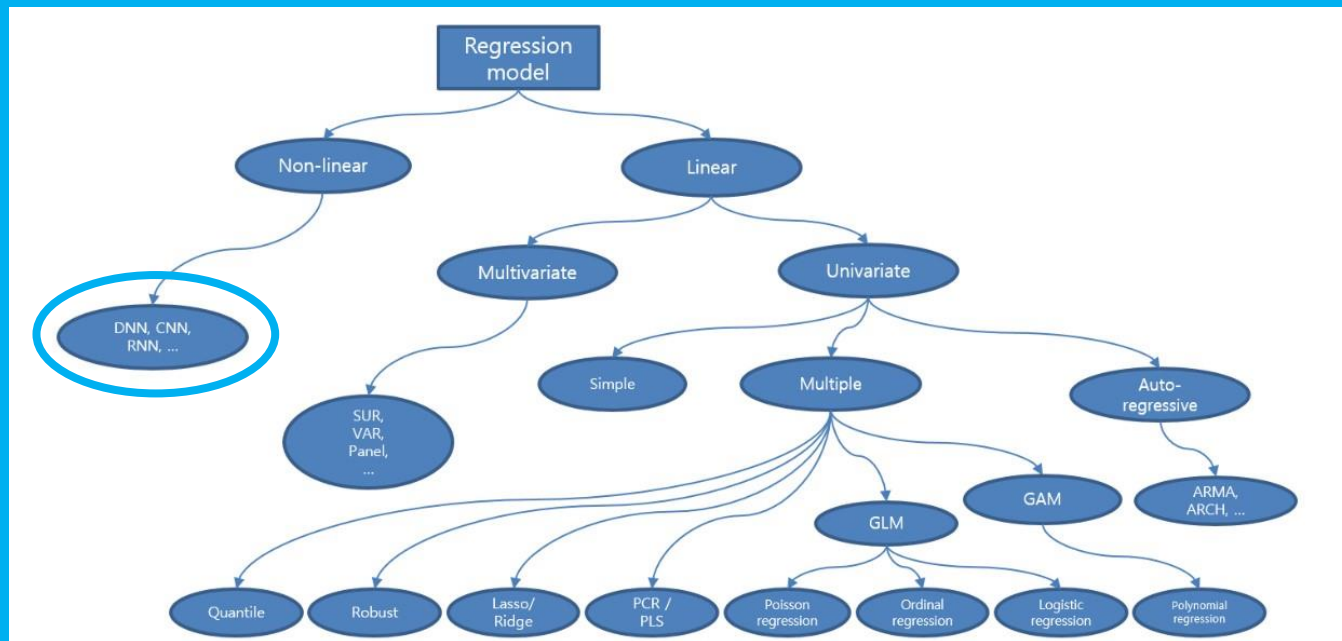
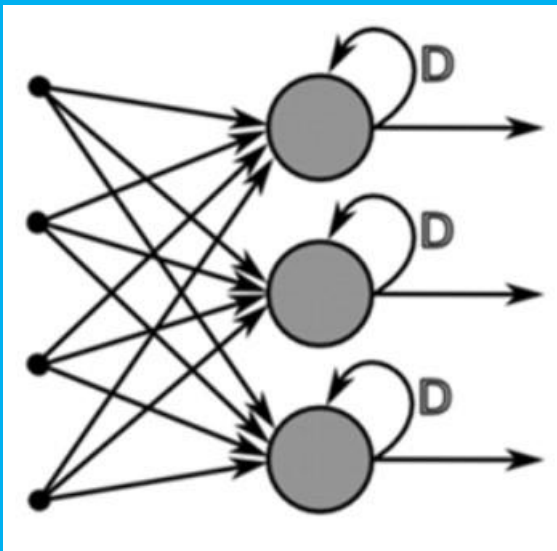


# K-최근접 이웃 회귀

## 여러가지 회귀모델 : 비선형 모델

RNN(Recurrent Neural Network)

과거의 데이터를 기억하고 새로운 데이터 처리에 이를 이용 ex)주식 가격 예측

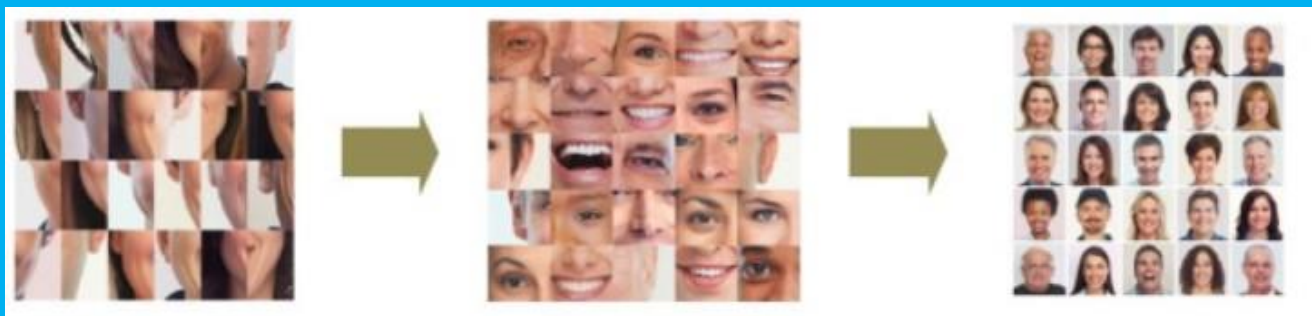


# K-최근접 이웃 회귀

K-최근접 이웃 회귀 = K-Nearest Neighbor Regression

지도학습 기반으로 거리를 기준으로 결과값(예측값)을 도출  
- 주변의 K개의 샘플을 통해 값을 반환

사용 분야: 이미지 처리, 음악 또는 상품 추천 알고리즘





# K-최근접 이웃 회귀

## 가중회귀:

단순히 평균만을 도출해내는 것으로는 유의미한 정보를 얻기 어려움  
따라서 거리가 가까운 데이터에 가중치를 더욱 두는 방안

Ex) 영화 Y의 등급 예측하기

영화 A / [등급 5.0] / 거리 3.2

영화 B / [등급 6.8] / 거리 11.5

영화 C / [등급 9.0] / 거리 1.1

단순 평균 => 6.93 등급

Y는 C와 가장 유사하다고 판단 되기에 C의 값에 가중치를 부여

분자: 점수 / 거리      분모: 1 / 거리

$$\frac{\frac{5.0}{3.2} + \frac{6.8}{11.5} + \frac{9.0}{1.1}}{\frac{1}{3.2} + \frac{1}{11.5} + \frac{1}{1.1}} = 7.9$$

# K-최근접 이웃 회귀

## 데이터 준비:

농어의 길이를 통해 무게를 예측 할 수 있다고 가정

농어의 길이: feature

농어의 무게: target

```
import numpy as np

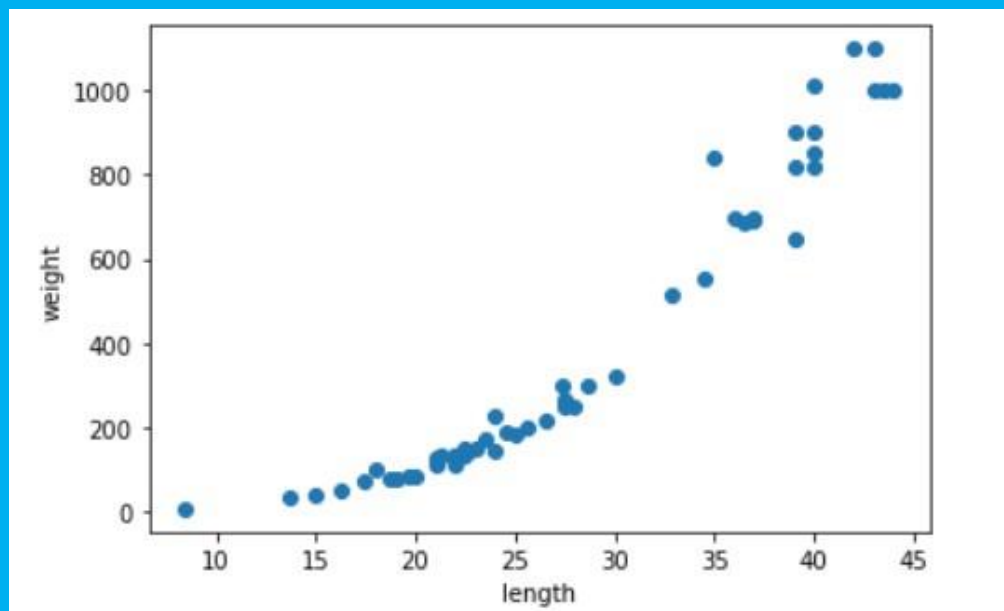
perch_length = np.array([8.4, 13.7, 15.0, 16.2, 17.4, 18.0, 18.7, 19.0, 19.6, 20.0, 21.0,
    21.0, 21.0, 21.3, 22.0, 22.0, 22.0, 22.0, 22.0, 22.5, 22.5, 22.7,
    23.0, 23.5, 24.0, 24.0, 24.6, 25.0, 25.6, 26.5, 27.3, 27.5, 27.5,
    27.5, 28.0, 28.7, 30.0, 32.8, 34.5, 35.0, 36.5, 36.0, 37.0, 37.0,
    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])
```

# K-최근접 이웃 회귀

## 데이터 개요:

농어의 길이와 무게가 서로 어떤 연관성이 있는지 그래프 통해 확인

```
import matplotlib.pyplot as plt
plt.scatter(perch_length, perch_weight)
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



# K-최근접 이웃 회귀

## 훈련 세트와 테스트 세트 나누기:

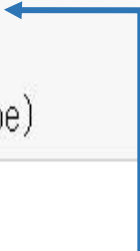
사이킷런을 이용해 세트 만들어 주기

단, 이때 세트는 2차원 데이터를 인식하기 때문에 reshape 함수 이용

```
from sklearn.model_selection import train_test_split
train_input, test_input, train_target, test_target = train_test_split(perch_length, perch_weight, random_state = 42)

# data를 2차원으로 변환
train_input = train_input.reshape(-1,1)
test_input = test_input.reshape(-1,1)
print(train_input.shape, test_input.shape)
```

(42, 1) (14, 1)



함수 reshape(-1, 1)에서 -1은 모든 원소의 개수를 의미

# K-최근접 이웃 회귀

## 훈련 세트와 테스트 세트 나누기:

random\_state는 똑같은 값이 나올 수 있도록 지정해주는 값  
무작위로 추출한 여러가지 경우의 수 중 하나를 지정해주는 것 => seed값

```
from sklearn.model_selection import train_test_split
train_input, test_input, train_target, test_target = train_test_split(perch_length, perch_weight, random_state = 42)

# data를 2차원으로 변환
train_input = train_input.reshape(-1,1)
test_input = test_input.reshape(-1,1)
print(train_input.shape, test_input.shape)
```

```
(42, 1) (14, 1)
```

# K-최근접 이웃 회귀

## 결정계수(R):

사이킷런의 KNeighborsRegressor을 사용  
Fit 함수를 통해 데이터를 학습 시키고 결과값 score을 확인(1보다 작게 나온다)

```
from sklearn.neighbors import KNeighborsRegressor  
knr = KNeighborsRegressor()  
knr.fit(train_input, train_target)  
knr.score(test_input, test_target)
```

0.992809406101064

$$R^2 = 1 - \frac{(\text{타깃} - \text{예측})^2 \text{의 합}}{(\text{타깃} - \text{평균})^2 \text{의 합}}$$

회귀에서는 정확한 수치를 맞추기는 힘들  
예측하는 값과 타깃 모두 임의의 수치이기 때문

# K-최근접 이웃 회귀

## 절댓값 오차:

Score 함수 이외에도 `mean_absolute_error`을 통해 직관적으로 오차 범위를 파악 할 수 있다

```
from sklearn.metrics import mean_absolute_error

test_prediction = knr.predict(test_input)
mae = mean_absolute_error(test_target, test_prediction)
print(mae)
```

19.157142857142862

이는 대략적으로 예측값이 19g 정도 타깃값과 다르다는 것을 의미

# K-최근접 이웃 회귀

데이터 예측:

Predict 함수를 이용해 길이를 입력하여 예상되는 무게를 출력

```
print(knr.predict([[38]]))
```

```
[720.]
```

38cm 에 해당하는 무게는 720g으로 출력



# K-최근접 이웃 회귀

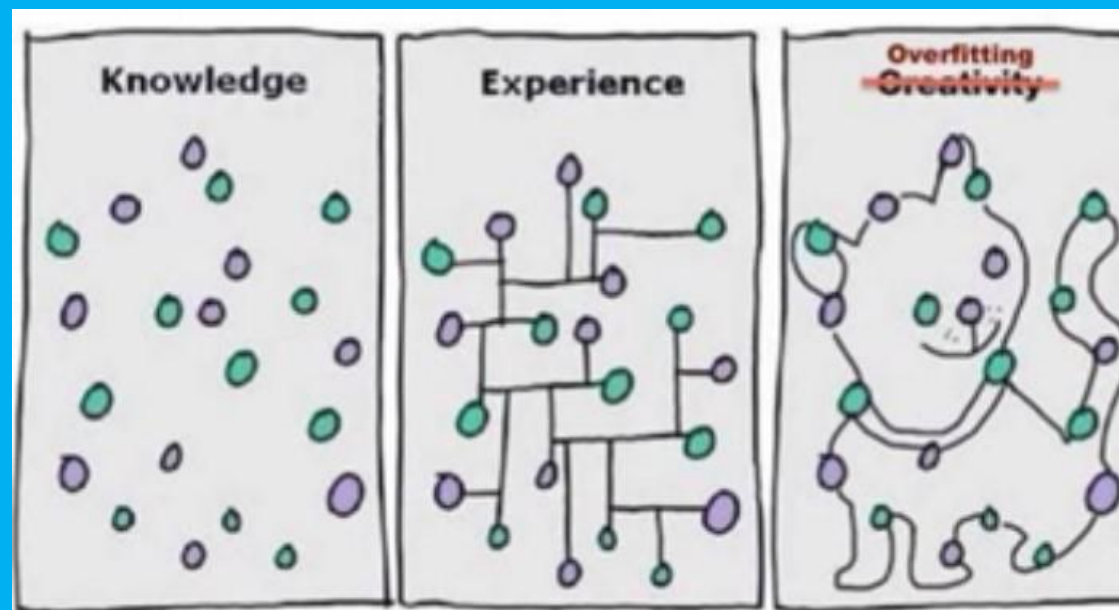
## 과대적합 vs 과소적합:

과대적합(overfitting):

학습 데이터에 지나치게 최적화되어 발생됨  
새로운 데이터에 정확한 분류나 예측을  
진행하지 못함

과소적합(underfitting):

머신러닝 모델이 충분히 복잡하지 않아 학습  
데이터의 구조와 패턴을 제대로 반영하지 못함



# K-최근접 이웃 회귀

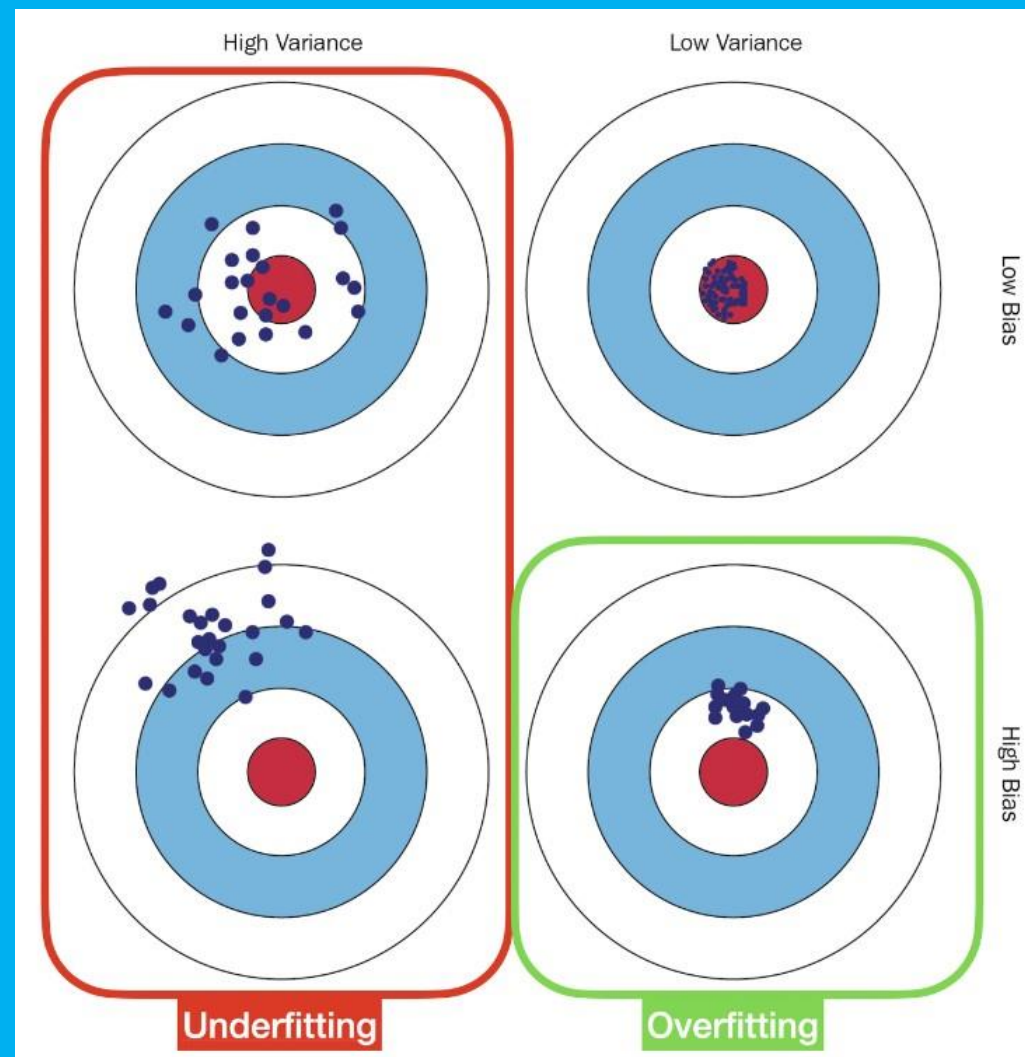
## 과대적합 vs 과소적합:

과대적합(overfitting):

학습 데이터에 지나치게 최적화되어 발생됨  
새로운 데이터에 정확한 분류나 예측을  
진행하지 못함

과소적합(underfitting):

머신러닝 모델이 충분히 복잡하지 않아 학습  
데이터의 구조와 패턴을 제대로 반영하지 못함



# K-최근접 이웃 회귀

## 과대적합 vs 과소적합:

```
knr.score(test_input, test_target)
```

```
0.992809406101064
```

>

```
print(knr.score(train_input, train_target))
```

```
0.9698823289099254
```

테스트 세트가 오히려 훈련 세트보다 점수가 더 높게 나옴  
주로 훈련 세트로 모델을 만들기 때문에 훈련 세트에서 더  
높은 점수가 나오는 것이 일반적

위와 같은 경우는 과소적합 : [테스트 세트] > [훈련 세트]

- 모델이 너무 단순
- 데이터가 적음

# K-최근접 이웃 회귀

## 과소적합 해결:

모델을 좀 더 복잡하게 만들기

- k 값(이웃의 개수)을 조절
- 국지적인 패턴에 민감

```
knr.n_neighbors = 3
knr.fit(train_input, train_target)
print(knr.score(train_input, train_target))
print(knr.score(test_input, test_target))
```

```
0.9804899950518966
0.9746459963987609
```

```
knr3 = KNeighborsRegressor(n_neighbors=3)
```

\* 이러한 방식으로도 k값 조정 가능

K-최근접 이웃 알고리즘 default 값은 5 -> 3으로 낮춤

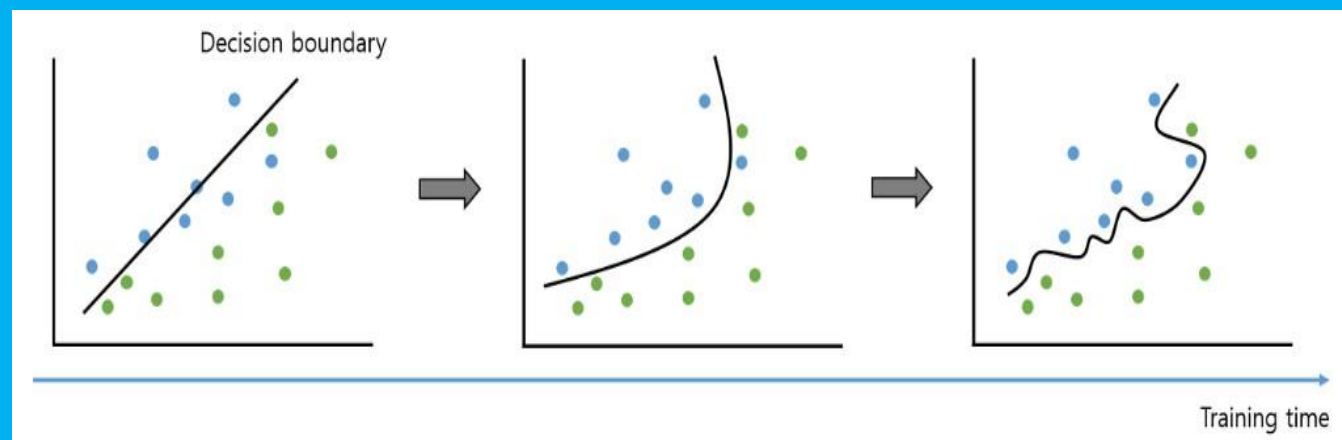
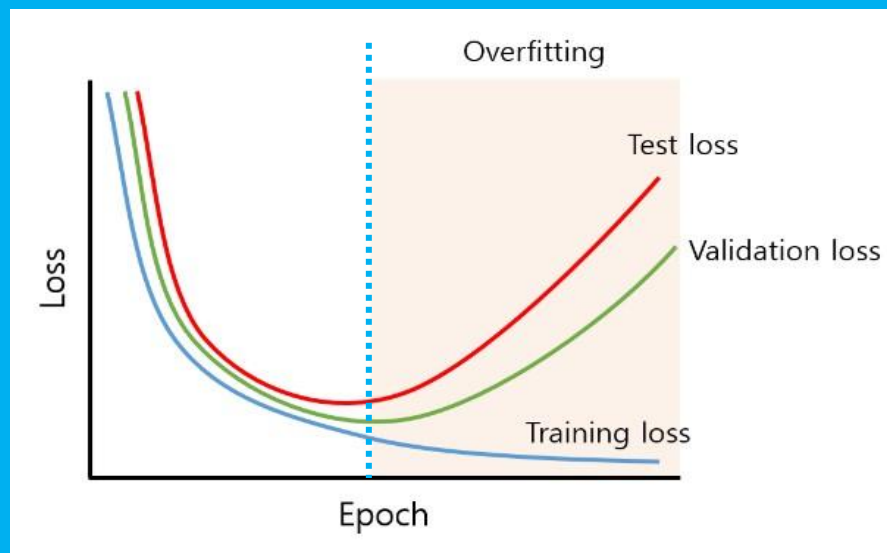
# K-최근접 이웃 회귀

## 과대적합 해결:

학습 조기 종료(Early Stopping)

훈련된 모델이 validation loss가 증가 할 때 학습 종료(점선 구간)

\* Validation = 검증용 데이터셋, 가장 좋은 모델을 고르기 위한 데이터셋



# K-최근접 이웃 회귀

## K-NN 모델의 한계:

입력된 수치들 보다 훨씬 크거나 작은 수치를 예측해야 한다면?  
주변에 outlier이 많다면?

