K최근접이웃알고리즘

숭실대학교 Soongsil University

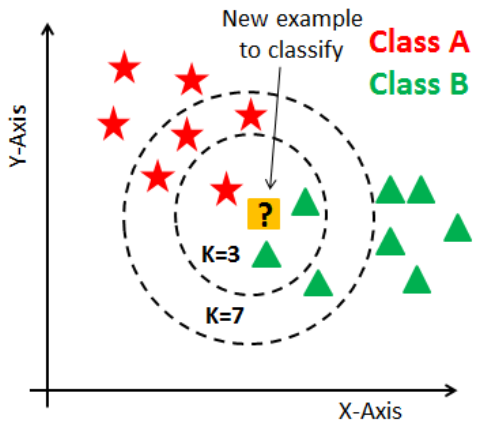
neon7624@ssu.ac.kr

목차

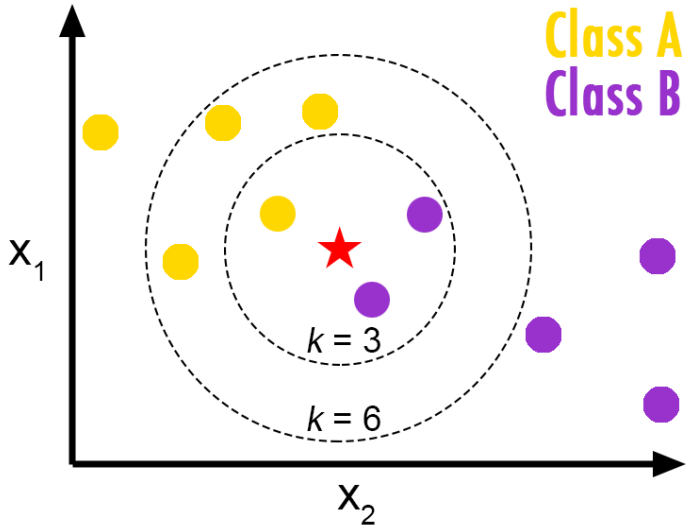
생선 분류 머신러닝

- **가장 간단한 머신러닝 알고리즘 중 하나인 K-최근접 이웃 알고리즘을 사용하여 2개의 종류를 분류하는 머신러닝 모델을 훈련한다.**

- **K-최근접 이웃 알고리즘이란,**
  - ✓ 새로운 데이터에 근접한 임의의 K개수 만큼의 데이터를 통해 어떤 데이터로 분류할 것인지를 추정하는 알고리즘이다.



출처: 데이터 캠프



K-최근접 이웃 알고리즘을 이용한 분류

- **다음 코드를 활용하여 연어를 예측 해보기**

```
if 25 <= salmon_length <= 45 :
    print('연어')
```

- **저 크기의 생선은 무조건 연어인가?**
  - ✓ 머신러닝으로는 어떻게 해결할 수 있을까?
  - ✓ 누구도 알려주지 않는 기준?
  - ✓ 어떻게?
- **즉, 누군가 알려주지 않아도 머신러닝은 "25~45cm 길이의 생선은 연어이다"라는 기준을 찾는다.**
  - ✓ 2개의 클래스(class)로 분류(classification)하는 것을 이진 분류(binary classification)라고 한다.

연어 데이터 입력하기

- 연어의 길이

```
salmon_length = [25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7, 30.0, 30.0, 30.7, 31.0, 31.0, 31.5, 32.0, 32.0, 32.0, 33.0, 33.0, 33.5, 33.5, 34.0, 34.0, 34.5, 35.0, 35.0, 35.0, 35.0, 36.0, 36.0, 37.0, 38.5, 38.5, 39.5, 41.0, 41.0, 41.5, 41.5, 41.5, 42.0, 42.0]
```

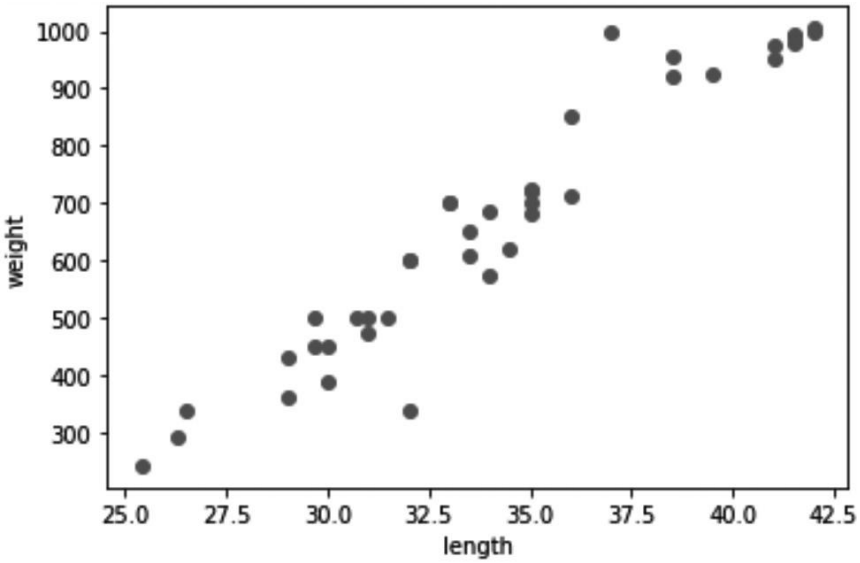
- 연어의 무게

```
salmon_weight = [242.0, 290.0, 340.0, 363.0, 430.0, 450.0, 500.0, 390.0, 450.0, 500.0, 475.0, 500.0, 500.0, 340.0, 600.0, 600.0, 700.0, 700.0, 610.0, 650.0, 575.0, 685.0, 620.0, 680.0, 700.0, 725.0, 720.0, 714.0, 850.0, 1000.0, 920.0, 955.0, 925.0, 975.0, 950.0, 980.0, 995.0, 990.0, 1000.0, 1005.0]
```

산점도

- **연어 데이터를 활용하여 산점도 그래프 만들기**

```
import matplotlib.pyplot as plt
# matplotlib의 pyplot 함수를 plt로 줄여서 사용
plt.scatter(salmon_length, salmon_weight)
# 산점도 그래프의 연어 길이와 무게를 x, y 값으로 사용
plt.xlabel('length') # x축은 길이
plt.ylabel('weight') # y축은 무게
plt.show( )
```



빙어 데이터 준비하기

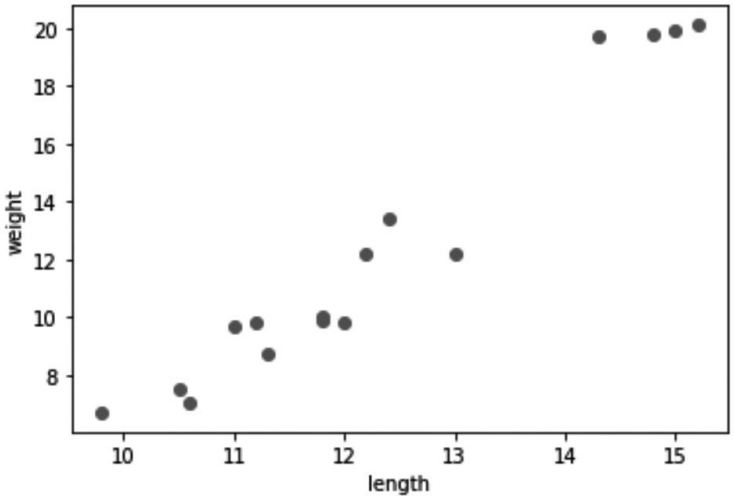
- 빙어의 길이

smelt_length = [9.8, 10.5, 10.6, 11.0, 11.2, 11.3, 11.8, 11.8, 12.0, 12.2, 12.4, 13.0, 14.3, 14.8, 15.0, 15.2]

- 빙어의 무게

smelt_weight = [6.7, 7.5, 7.0, 9.7, 9.8, 8.7, 10.0, 9.9, 9.8, 12.2, 13.4, 12.2, 19.7, 19.8, 19.9, 20.1]

```
plt.scatter(smelt_length, smelt_weight)
# 산점도 그래프의 빙어 길이와 무게를 x, y값으로 사용
plt.xlabel('length') # x축은 길이
plt.ylabel('weight') # y축은 무게
plt.show( )
```



연어와 빙어 데이터 합치기

```
fish_length = salmon_length+smelt_length
fish_weight = salmon_weight+smelt_weight
print(fish_length)
print(fish_weight)
```

```
fish_length =
[25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7, 30.0, 30.0, 30.7, 31.0, 31.0, 31.5, 32.0, 32.0,
32.0, 33.0, 33.0, 33.5, 33.5, 34.0, 34.0, 34.5, 35.0, 35.0, 35.0, 35.0, 36.0, 36.0, 37.0,
38.5, 38.5, 39.5, 41.0, 41.0, 41.5, 41.5, 41.5, 42.0, 42.0, 9.8, 10.5, 10.6, 11.0, 11.2,
11.3, 11.8, 11.8, 12.0, 12.2, 12.4, 13.0, 14.3, 14.8, 15.0, 15.2]

fish_weight =
[242.0, 290.0, 340.0, 363.0, 430.0, 450.0, 500.0, 390.0, 450.0, 500.0, 475.0, 500.0,
500.0, 340.0, 600.0, 600.0, 700.0, 700.0, 610.0, 650.0, 575.0, 685.0, 620.0, 680.0,
700.0, 725.0, 720.0, 714.0, 850.0, 1000.0, 920.0, 955.0, 925.0, 975.0, 950.0, 980.0,
995.0, 990.0, 1000.0, 1005.0, 6.7, 7.5, 7.0, 9.7, 9.8, 8.7, 10.0, 9.9, 9.8, 12.2, 13.4,
12.2, 19.7, 19.8, 19.9, 20.1]
```

넘파이로 2차원 리스트 만들기

```
import numpy as np
fish_data = np.column_stack((fish_length, fish_weight))
print(fish_data)
```

```
[[ 25.4 242. ]  [ 31. 500. ]   [ 34.5 620. ]  [ 41. 975. ]   [ 11.2 9.8]
 [ 26.3 290. ]  [ 31.5 500. ]  [ 35. 680. ]   [ 41. 950. ]   [ 11.3 8.7]
 [ 26.5 340. ]  [ 32. 340. ]   [ 35. 700. ]   [ 41.5 980. ]  [ 11.8 10. ]
 [ 29. 363. ]   [ 32. 600. ]   [ 35. 725. ]   [ 41.5 995. ]  [ 11.8 9.9]
 [ 29. 430. ]   [ 32. 600. ]   [ 35. 720. ]   [ 41.5 990. ]  [ 12. 9.8]
 [ 29.7 450. ]  [ 33. 700. ]   [ 36. 714. ]   [ 42. 1000. ]  [ 12.2 12.2]
 [ 29.7 500. ]  [ 33. 700. ]   [ 36. 850. ]   [ 42. 1005. ]  [ 12.4 13.4]
 [ 30. 390. ]   [ 33.5 610. ]  [ 37. 1000. ]  [ 9.8 6.7]     [ 13. 12.2]
 [ 30. 450. ]   [ 33.5 650. ]  [ 38.5 920. ]  [ 10.5 7.5]    [ 14.3 19.7]
 [ 30.7 500. ]  [ 34. 575. ]   [ 38.5 955. ]  [ 10.6 7. ]    [ 14.8 19.8]
 [ 31. 475. ]   [ 34. 685. ]   [ 39.5 925. ]  [ 11. 9.7]     [ 15. 19.9]
                                                             [ 15.2 20.1]]
```

- numpy라는 수학 계산 라이브러리를 사용한다. as 명령어로 함수 이름을 단순하게 바꿀 수 있다. 여기서는 numpy를 np로 줄여서 사용한다.
- **column_stack( )** 함수는 전달받은 리스트를 일렬로 세운 다음 차례대로 연결한다. 다음은 출력되는 56개의 데이터의 순서를 나열한 것이다.

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
result = np.column_stack((a, b))
print(result)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

정답 데이터 생성

- **ones( ), zeros( ) 함수는 각각 원하는 개수의 1과 0을 채운 배열을 만들어 준다.**

```
fish_target = np.concatenate((np.ones(40), np.zeros(16)))
print(fish_target)
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0.
0. 0. 0. 0. 0. 0. 0. 0.]
```

훈련 세트와 테스트 세트 나누기

- **train_test_split( ) 함수는 비율에 맞게 훈련용 세트와 테스트 세트로 나눈다.**
- **test_size는 기본적으로 25%를 테스트 세트로 분류한다.**
- **stratify 매개변수에 타깃 데이터를 전달하면 클래스 비율에 맞게 나눈다.**
- **shape는 입력 데이터의 크기를 출력한다.**

```
from sklearn.model_selection import train_test_split
train_input, test_input, train_target, test_target = train_test_
split(fish_data, fish_target, stratify=fish_target, random_state=42, test_size=0.25)
print(train_input.shape, test_input.shape)
print(train_target.shape, test_target.shape)
```

train_input / test_input	(42, 2) (14, 2)
train_target / test_target	(42,) (14,)

모델 생성

- 파이썬 패키지 전체를 임포트하지 않고 특정 클래스만 임포트하려면 from~import 구문 사용한다.
- kn 변수에 KNeighborsClassifier( )로 생성된 객체를 할당한다.

```
from sklearn.neighbors import KNeighborsClassifier
kn = KNeighborsClassifier( )
```

모델 훈련

- 연어를 찾을 기준을 학습한다.
- fit 메서드가 이런 역할을 수행한다.

```
kn.fit(train_input, train_target)
```

모델 평가

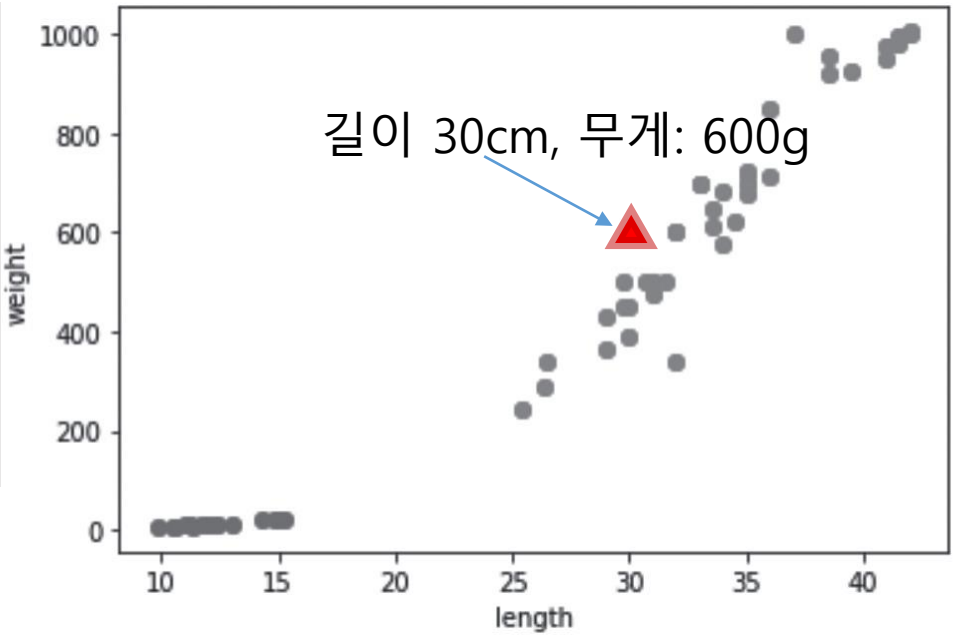
- 모델을 평가하는 메서드는 score( ) 메서드이며, 0에서 1 사이의 값을 반환한다.

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

새로운 데이터의 산점도

- 모델을 평가하는 메서드는 score( ) 메서드이며, 0에서1 사이의 값을 반환한다.

```
plt.scatter(salmon_length, salmon_weight)
plt.scatter(smelt_length, smelt_weight)
plt.scatter(30, 600, marker='^')
plt.xlabel('length')
plt.ylabel('weight')
plt.show( )
```



새로운 데이터로 예측된 결과

- **새로운 데이터를 예측, predict 메서드가 이런 역할을 수행한다.**

```
kn.predict([[30, 600]]) #새로운 데이터의 길이와 무게 값
```

```
array([1.]) #1은 연어(정답 값) 0은 빙어를 의미함
```

핵심 포인트

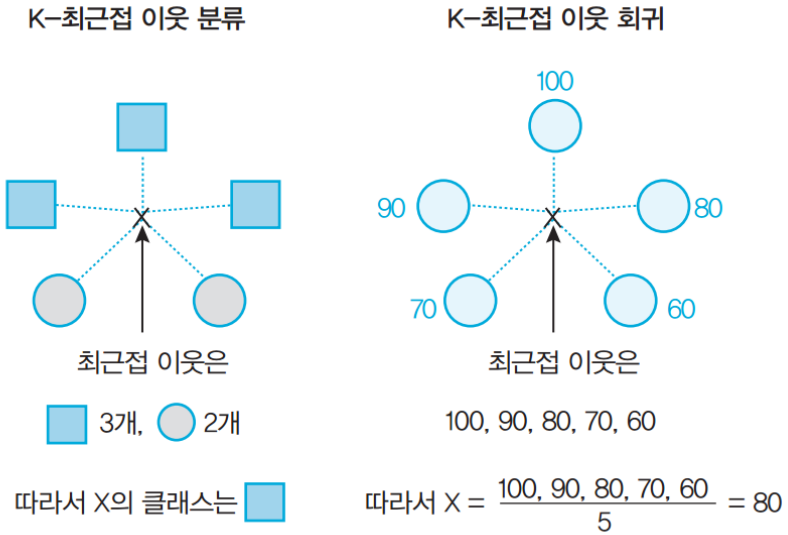
- 특성은 데이터를 표현하는 하나의 성질
- 머신러닝 알고리즘이 데이터에서 규칙을 찾는 과정을 훈련이라고 하고, 사이킷런에서는 fit()메서드가 하는 역할
- 머신러닝 프로그램에서는 알고리즘이 구현된 객체를 모델이라고 함
- 정확도는 정확한 답을 몇 개 맞혔는지를 백분율로 나타낸 값으로 사이킷런에서는 0~1 사이의 값으로 출력
- 정확도 = (정확히 맞힌 개수) / (전체 데이터 개수)

확인 문제

1. 데이터를 표현하는 하나의 성질로써, 예를 들어 국가 데이터의 경우 인구수, GDP, 면적 등이 하나의 국가를 나타낸다. 머신러닝에서 이런 성질을 무엇인가?
   ① 특성
   ② 특질
   ③ 개성
   ④ 요소
2. 가장 가까운 이웃을 참고하여 정답을 예측하는 알고리즘이 구현된 사이킷런 클래스는 무엇인가?
   ① SGDClassifier
   ② LinearRegression
   ③ RandomForestClassifier
   ④ KneighborsClassifier
3. 사이킷런 모델을 훈련할 때 사용하는 메서드는 어떤 것인가?
   ① Predict()
   ② fit()
   ③ score()
   ④ transform()

k-최근접 이웃 알고리즘 – 회귀(수치예측)

- **앞에서 연어와 빙어를 '구분'하는 머신러닝 모델을 성공적으로 개발하였다.**
  - ✓ 이번에는 광어의 길이에 따라 무게를 '예측'하는 인공지능 모델을 만들어 볼 것이다.
  - ✓ 인공지능 모델을 훈련하고, 정확도를 측정한 다음 모델을 이용하여 새로운 데이터에 대한 예측 결과를 살펴본다.
  - ✓ 회귀는 정해진 클래스가 없고 임의의 수치를 출력한다.



데이터 준비

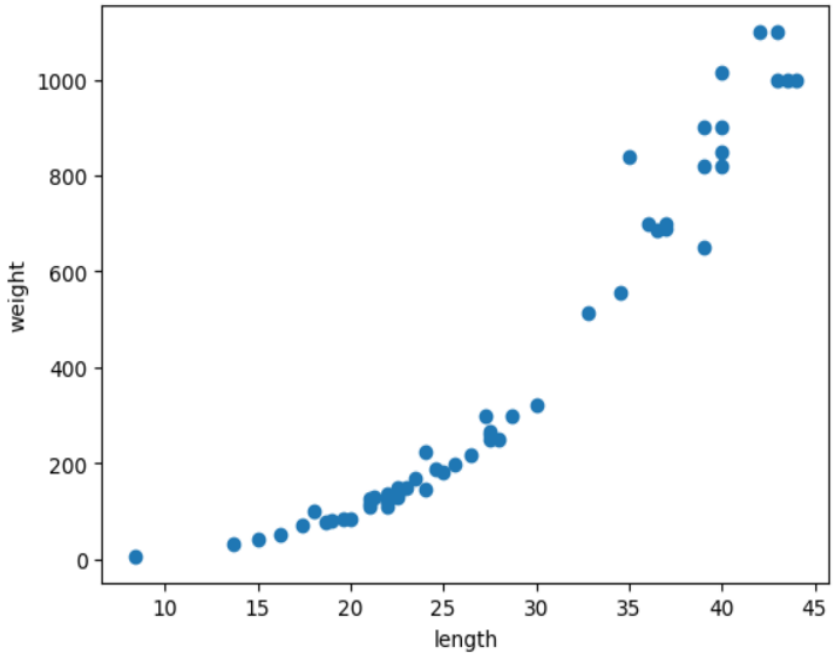
- **광어 데이터 56마리의 길이와 무게를 넘파이를 이용하여 리스트로 만든다.**

```
import numpy as np

flat_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]
)

flat_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])
```

산점도

- 광어 데이터 56마리의 길이와 무게를 이용하여 산점도를 표시

```
import matplotlib.pyplot as plt

plt.scatter(flat_length, flat_weight)
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



훈련 세트와 테스트 세트 준비

- **train_test_split( ) 함수는 비율에 맞게 훈련 세트와 테스트 세트로 나눈다.**
- **test_size는 기본적으로 25%를 테스트 세트로 분류한다.**
- **현재 1차원 배열(42,) (14,), 사이킷런에 사용할 데이터는 2차원 배열이어야 한다.**
- **reshape(a,b) 메서드를 통해 axb 크기로 변경한다.**

```
from sklearn.model_selection import train_test_split

train_input, test_input, train_target, test_target = train_test_split(flat_length,

flat_weight, random_state=42, test_size=0.25)

print(train_input.shape, test_input.shape) # (42,) (14,)
```

2차원 배열 변경

```
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)
```

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

reshape(-1,1)

- 열 자리의 1은 1열로 만들라는 것
- 행 자리의 -1은 남은 배열의 길이와 남은 차원으로부터 추정해서 알아서 지정하라는 의미

모델 생성

- K-최근접 이웃 회귀 알고리즘을 구현한 클래스이다.
- knr 변수에 객체를 먼저 생성한다.

```
from sklearn.neighbors import KNeighborsRegressor
knr = KNeighborsRegressor( )
```

n_neighbor = 5 (기본값)

모델 훈련

- **k-최근접 이웃 회귀 모델을 훈련한다.**

```
knr.fit(train_input, train_target)
```

모델 평가

- **k-최근접 이웃 회귀 모델을 훈련한다.**
- **회귀에서는 이 점수를 결정 계수 (coefficient of determination) 또는 R2라고 한다.**

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

0.992809406101064

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

과대적합과 과소적합

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

```
0.9698823289099254
```

- 훈련 세트의 점수가 테스트 세트의 점수보다 낮게 나왔다. 상식적으로는 훈련 세트의 점수가 더 높게 나와야 한다. (왜냐하면 훈련 세트로 훈련을 시켰기 때문)
- 테스트 세트의 결정계수(0.992809406101064)가 훈련 세트의 결정계수(0.9698823289099254)보다 높을 경우 '과소적합'되었다고 표현한다.
- 과대 적합 : 모델의 훈련 세트 성능이 테스트 성능보다 훨씬 높을 때, 모델이 훈련세트에 너무 집착해서 데이터에 내재된 거시적인 패턴을 감지하지 못함을 의미한다.
- 과소 적합 : 훈련 세트와 테스트 세트 성능이 모두 동일하게 낮거나 테스트 세트 성능이 오히려 더 높을 때를 말한다.

이웃개수 줄이기

- **과소적합 문제를 해결하기 위해 이웃의 개수를 3으로 설정한다.**

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

0.9804899950518966

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

0.9746459963987609

- 다시 훈련을 시키고 훈련 세트와 테스트 세트의 점수를 각각 확인 해보니 0.98과 0.97로 나왔다.
- 예상대로 테스트 세트의 점수가 훈련 세트보다 낮아졌으므로 과소적합 문제를 해결했다.
- 두 점수의 차이가 크지도 않으므로 과대적합도 아니다.

새로운 데이터로 예측된 결과

- 길이에 대한 무게를 예측한다.
- 새로운 데이터를 예측, predict 메서드가 이런 역할을 수행한다.

```
knr.predict([[40]])
```

```
array([921.66666667])
```

핵심 포인트

- 회귀는 임의 수치를 예측하는 문제이다. 따라서 타깃값도 임의의 수치가 된다.
- K-최근접 이웃 회귀는 K-최근접 이웃 알고리즘을 사용해 회귀 문제를 푼다. 가장 가까운 이웃 샘플을 찾고 이 샘플들의 타깃값을 평균하여 예측으로 삼는다.
- 결정계수는 대표적인 회귀 문제의 성능 측정 도구이다. 1에 가까울수록 좋고, 0에 가깝다면 성능이 나쁜 모델이다.
- 과대적합은 모델의 훈련 세트 성능이 테스트 세트 성능보다 훨씬 높은 때 일어난다. 모델이 훈련 세트에 너무 집착해서 데이터에 내재된 거시적인 패턴을 감지하지 못한다.
- 과소적합은 이와 반대이다. 이런 경우 더 복잡한 모델을 사용해 훈련 세트에 잘 맞는 모델을 만들어야 한다.

핵심 패키지와 함수

- **scikit-learn**
  - ✓ KNeighborsRegressor는 k-최접근 이웃 회귀 모델을 만드는 사이킷런 클래스이다. n_neighbors 매개변수로 이웃의 개수를 지정한다. 기본값은 5이다. 다른 매개변수는 KNeighborsClassifier 클래스와 거의 동일한다.
  - ✓ mean_absolute_error()는 회귀 모델의 평균 절대값 오차를 계산한다. 첫 번째 매개변수는 타깃, 두 번째 매개변수는 예측값을 전달한다. 이와 비슷한 함수로는 평균 제곱 오차를 계산하는 mean_squared_error()가 있다.
- **numpy**
  - ✓reshape()는 배열의 크기를 바꾸는 메서드이다. 바꾸고자 하는 배열의 크기를 매개변수로 전달한다. 바꾸기 전후의 배열 원소 개수는 동일해야 한다.

확인 문제

1. K-최근접 이웃 회귀에서는 새로운 샘플에 대한 예측을 어떻게 하는가?
   ① 이웃 샘플 클래스 중 다수인 클래스
   ② 이웃 샘플의 타깃값의 평균
   ③ 이웃 샘플 중 가장 높은 타깃값
   ④ 이웃 샘플 중 가장 낮은 타깃값

확인 문제

2. 과대적합과 과소적합에 대한 이해를 돕기 위해 복잡한 모델과 단순한 모델을 만들었다. 앞서 만들 회귀 모델의 K 값을 1,5,10으로 바꿔가면 훈련해 보자. 그 다음 농어의 길이를 5에서 45까지 바꿔가면 예측을 만들어 그래프로 나타내 보자. N이 커짐에 따라 모델이 단순해지는 것을 볼 수 있는가?

```
# k-최근접 이웃 회귀 객체를 만듭니다
knr = KNeighborsRegressor()
# 5에서 45까지 x 좌표를 만듭니다
x = np.arange(5, 46).reshape(-1, 1)

# n = 1, 5, 10일 때 예측 결과를 그래프로 그립니다.
for n in [1, 5, 10]:
    # 모델 훈련
    knr.n_neighbors =
    knr.fit(train_input, train_target)
    # 지정한 범위 x에 대한 예측 구하기
    prediction =
    # 훈련 세트와 예측 결과 그래프 그리기
    plt.scatter(train_input, train_target)
    plt.plot(x, prediction)
    plt.title('n_neighbors = {}'.format(n))
    plt.xlabel('length')
    plt.ylabel('weight')
    plt.show()
```