### ∨ 마켓과 머신러닝



#### ∨ 생선 분류 문제

- 도미를 잘 아는 김팀장이 생선 길이가 30cm 이상이면 도미라는 규칙을 알려주어 이를 바탕으로 파이썬 프로그램을 작성
- 하지만 30cm보다 큰 생선이 무조건 도미라고 말할 수 없음
- 머신러닝은 누구도 알려주지 않는 기준을 찾아서 일을 하여 규칙을 찾으므로 이러한 문제를 머신러닝으로 해결할 수 있음
- 일반적인 프로그램 누군가가 사전에 규칙을 정해서 프로그램을 작성
- 머신러닝 프로그램 스스로 프로그램을 만들어서 규칙을 찾게끔 함

#### ∨ 도미 데이터 준비하기(35마리)

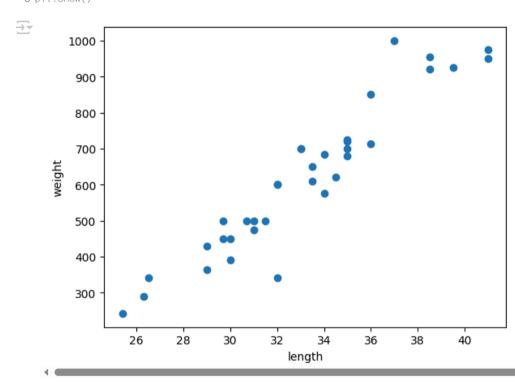
1 bream\_length = [25.4, 26.3, 26.5, 29.0, 29.0, 29.7, 29.7, 30.0, 30.0, 30.0, 31.0, 31.0, 31.5, 32.0, 32.0, 32.0, 33.0, 33.5, 33.5, 34.0, 34.0, 34.5, 35.0, 35.0, 35.0, 35.0, 20.0, 20.0, 20.0, 340.0, 363.0, 450.0, 500.0, 450.0, 500.0, 450.0, 500.0, 450.0, 500.0, 450.0, 500.0, 450.0, 500.0, 450.0, 500.0, 450.0, 500.0, 500.0, 450.0, 500.0, 500.0, 450.0, 500.0,

1 print(len(bream\_length)) #변수 위에 마우스를 놓아도 표시 2 print(len(bream\_weight))

**→** 35 35

1 #두 특성을 숫자로 보는 것보다 그래프로 표현하면 데이터를 잘 이해할 수 있으므로 2 #길이를 X축으로 하고 무게를 Y축으로 하여 산점도(scatter plot)를 만들어 각 도미를 그래프에 점으로 표시

```
3
4 import matplotlib.pyplot as plt
5 plt.scatter(bream_length, bream_weight)
6 plt.xlabel('length')
7 plt.ylabel('weight')
8 plt.show()
```



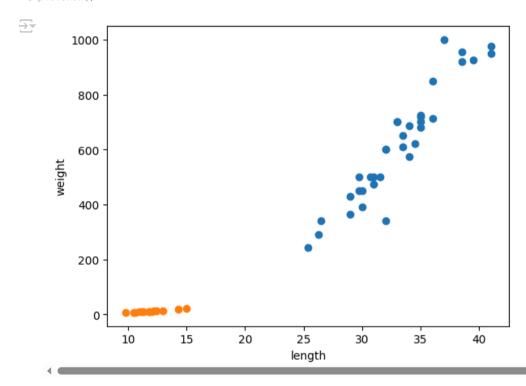
# < <u>빙어 데이터 준비하기(14마리)</u>

```
1 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, 15.0]
2 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, 19.9]

1 print(len(smelt_length))
2 print(len(smelt_weight))

14
14
```

```
1 plt.scatter(bream_length, bream_weight)
2 plt.scatter(smelt_length, smelt_weight)
3 plt.xlabel('length')
4 plt.ylabel('weight')
5 plt.show()
```



## 첫 번째 머신러닝 프로그램

```
1 #생선을 분류하기 위해 k-최근접 이웃 알고리즘을 사용하려면 도미와 빙어 데이터를 하나의 데이터로 합침 2 #사이킷런 머신러닝 패키지를 사용하기 위해서는 샘플 하나의 데이터가 하나의 리스트에 담기도록 변경해야 함 3 4 length = bream_length + smelt_length 5 weight = bream_weight + smelt_weight 6 7 #zip() 함수와 리스트 내포 구문을 사용해 각 특성의 리스트를 세로 방향으로 늘어뜨린 2차원 리스트로 만들어 줌 8 fish_data = [[I, w] for I, w in zip(length, weight)] 9 print(fish_data)
```

₹ [[25.4, 242.0], [26.3, 290.0], [26.5, 340.0], [29.0, 363.0], [29.0, 430.0], [29.7, 450.0], [29.7, 500.0], [30.0, 390.0], [30.0, 450.0], [30.7, 500.0], [31.0, 475.0], [31.0, 500.0]

1 #머신러닝 알고리즘이 생선의 길이와 무게를 보고 도미와 빙어를 구분하는 2 #규칙을 찾기를 원하므로 적어도 어떤 생선이 도미인지 빙어인지를 3 #알려주어야 하며 이를 지도 학습이라고 함 4 5 #이를 위해 도미와 빙어를 숫자 1과 0으로 표현하며 2개를 구분하는 경우 6 #찾으려는 대상을 1로 놓고 그 외는 0으로 놓아야 하므로 도미를 찾는 7 #대상으로 정의했기 때문에 도미를 1로 놓고 빙어를 0으로 놓음 8 9 fish\_target = [1]\*35 + [0]\*14 10 print(fish target)

#### ∨ k-최근접 이웃 알고리즘

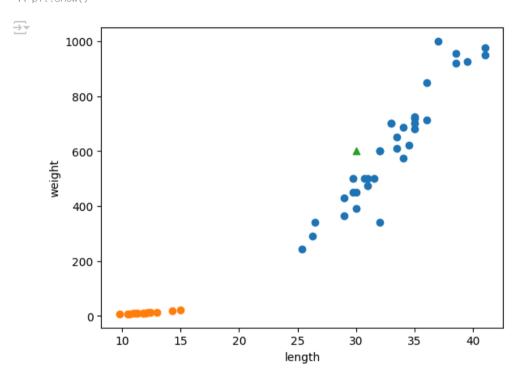
1 #scikit-learn은 knn 분류기를 KNeighborsClassifier 클래스로 제공
2 from sklearn.neighbors import KNeighborsClassifier
3
4 #클래스 객체를 만들고 fish\_data와 fish\_target을 전달해 도미를 찾기 위한 기준을 학습시키는 훈련을 함
5 #이를 위해 fit() 메소드를 사용해 fish\_data와 fish\_target을 순서대로 전달하고 주어진 데이터로 알고리즘을 훈련
6 kn = KNeighborsClassifier()
7 kn.fit(fish\_data, fish\_target)
8
9 #객체 또는 모델인 kn이 얼마나 잘 훈련되었는지 평가하기 위해 score() 메소드를 사용해 모델을 평가
10 #0에서 1사이의 값을 반환하며 1은 모든 데이터를 정확히 맞혔다는 것을 의미하며 이 값을 정확도라고 함
11 kn.score(fish\_data, fish\_target)

# ₹ 1.0

## 새로운 생선(30,600) 예측하기

1 #새로운 데이터가 있을 때 직관적으로 이 삼각형은 주변에 다른 도미 데이터가 많기 때문에 2 #도미라고 판단할 것이며 k-최근접 이웃 알고리즘도 삼각형 주위에 도미 데이터가 많으므로 삼각형을 도미라고 판단할 것임

```
3
4 plt.scatter(bream_length, bream_weight)
5 plt.scatter(smelt_length, smelt_weight)
6
7 plt.scatter(30, 600, marker='^') #파이썬 마커 - https://wikidocs.net/92083
8
9 plt.xlabel('length')
10 plt.ylabel('weight')
11 plt.show()
```



```
1 #predict() 메소드를 사용해 리스트의 리스트를 전달해 새로운 데이터의 정답을 예측 2 #반환되는 값이 1이면 도미이며, 0이면 빙어이므로 새로 들어온 데이터인 삼각형은 도미임 3 4 kn.predict([[30, 600]])

→ array([1])
```

1 #속성 -X에 전달한 fish\_data를 가지고 있음 2 3 print(kn.\_fit\_X)

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

```
1 #속성 -y에 전달한 fish_target를 가지고 있음
2 print(kn. v)
3
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
1 #무조건 도미의 문제
     #k-최근접 이웃 알고리즘은 새로운 데이터에 대해 예측할 때 가장 가까운
     #직선거리에 어떤 데이터가 있는지 살피기만 하면 됨
     #k는 주위에서 바라볼 이웃의 개수를 뜻하고, 기본값은 5이므로 5개의 주변 샘플을 보고 판단함
6 #기준(기본값)은 n_neighbors 매개변수로 바꿀 수 있음
7 #기준값을 49로 바꾸어 가장 가까운 데이터 49개를 사용하는 k-최근접 이웃 모델에
8 #fish_data를 적용하면 fish_data에 있는 모든 생선을 사용하여 예측하므로
9 #fish data의 데이터 49개 중에 도미가 35개로 다수를 차지하므로 어떤 데이터를 넣어도 무조건 도미로 예측하게 됨
10
11 kn49 = KNeighborsClassifier(n_neighbors=49)
12 kn49.fit(fish_data, fish_target)
13
14 #이러한 kn49 모델은 도미만 올바르게 맞히기 때문에 정확도가 낮아지게 됨 - k-최근접 이웃 알고리즘 단점!!!!
15 kn49.score(fish_data, fish_target)
0.7142857142857143
1 print (35/49)
0.7142857142857143
```

#### ∨ 확인 문제

```
1 #n_neighbors의 기본값인 5부터 49까지 바꾸어 가며 점수가 1.0 아래로 내려가기 시작하는 이웃의 개수를 찾아보자!!!
2 #k-최근접 이웃 알고리즘의 훈련은 데이터를 저장하는 것이 전부이기 때문에
3 #객체를 매번 새로 만들거나 fit() 메소드를 통해 훈련할 필요가 없음
4
5 kn = KNeighborsClassifier()
6 kn.fit(fish_data, fish_target)
7
8 for n in range(5, 50):
```

```
9 kn.n_neighbors = n #최근접 이웃 개수 설정
10 score = kn.score(fish_data, fish_target) # 점수 계산
11
12 if score < 1: # 100% 정확도에 미치지 못하는 이웃 개수 출력
13 print(n, score)
14 break
18 0.9795918367346939
```

# ∨ <<<참조자료 사이트>>>