인공지능과 머신러닝, 딥러닝

인공지능이란

- 사람처럼 학습하고 추론할 수 있는 지능을 가진 컴퓨터 시스템
- 강인공지능(인공일반지능)과 약인공지능으로 구분됨

머신러닝이란

- 규칙을 일일이 프로그래밍하지 않아도 자동으로 데이터에서 규칙을 학습하는 알고리즘을 연구하는 분야
- 통계학과 깊은 관련이 있음
- 하지만 최근 머신러닝의 발전은 통계나 수학 이론보다 경험을 바탕으로 발전하는 경우도 많음. 이는 컴퓨터 과학 분야가 이런 발전을 주도함
- 대표적인 라이브러리로 사이킷런이 있음

딥러닝이란

- 머신러닝 알고리즘 중에 인공 신경망을 기반으로 한 방법들을 통칭하여 딥러닝이라고 함
- 대표적인 라이브러리로 텐서플로와 파이토치가 있음

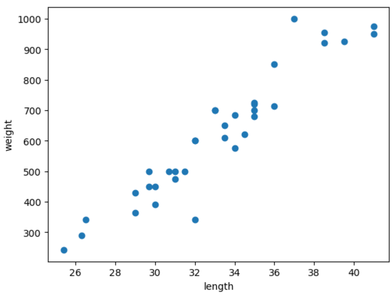
마켓과 머신러닝

생선 분류 문제

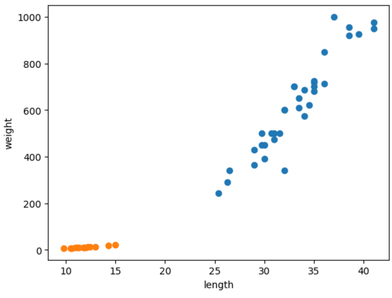
- 이진 분류: 2개의 클래스(종류) 중 하나를 고르는 문제
- 특성: 데이터의 특징
- 선형적: 산점도 그래프가 일직선에 가까운 형태로 나타나는 경우

In [1]:
```
# 생선의 기리
bream_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]

# 생선의 무게
bream_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]
```

In [2]:
```
import matplotlib.pyplot as plt

plt.scatter(bream_length, bream_weight)
plt.xlabel("length")
plt.ylabel("weight")
plt.show()
```



In [3]:
```
# 빙어 데이터 준비하기

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]
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]

plt.scatter(bream_length,bream_weight)
plt.scatter(smelt_length,smelt_weight)
plt.xlabel("length")
plt.ylabel("weight")
plt.show()
```



첫 번째 머신러닝 프로그램

In [4]:
```
length=bream_length+smelt_length
weight=bream_weight+smelt_length
```

In [5]:
```
# 사이킷런은 2차원 리스트가 필요함
fish_data=[[l,w] for l,w in zip(length,weight)]
```

In [6]:
```
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], [31.5, 500.0], [32.0, 340.0], [32.0, 600.0], [32.0, 600.0], [33.0, 700.0], [33.0, 700.0], [33.5, 610.0], [33.5, 650.0], [34.0, 575.0], [34.0, 685.0], [34.5, 620.0], [35.0, 680.0], [35.0, 700.0], [35.0, 725.0], [35.0, 720.0], [36.0, 714.0], [36.0, 850.0], [37.0, 1000.0], [38.5, 920.0], [38.5, 955.0], [39.5, 925.0], [41.0, 975.0], [41.0, 950.0], [9.8, 9.8], [10.5, 10.5], [10.6, 10.6], [11.0, 11.0], [11.2, 11.2], [11.3, 11.3], [11.8, 11.8], [11.8, 11.8], [12.0, 12.0], [12.2, 12.2], [12.4, 12.4], [13.0, 13.0], [14.3, 14.3], [15.0, 15.0]]
```

In [7]:
```
fish_target=[1]*35 +[0]*14
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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

In [8]:
```
from sklearn.neighbors import KNeighborsClassifier
kn= KNeighborsClassifier()
kn.fit(fish_data,fish_target) # fit() 메서드는 주어진 데이터로 알고리즘을 훈련함
kn.score(fish_data,fish_target) # score() 메서드는 모델을 평가함(정확도)
```

Out[8]:
```
1.0
```

k-최근접 이웃 알고리즘

- 주위의 다른 데이터를 보고 다수를 차지하는 것을 정답으로 사용함
- 데이터가 아주 많은 경우에는 사용하기 어려움. 메모리가 많이 필요하고 직선거리를 계산하는 데도 많은 시간이 필요하기 때문
- 정확도: (정확히 맞힌 개수) / (전제 데이터 개수)

In [9]:
```
kn.predict([[300,600]]) # predict() 메서드는 새로운 데이터의 정답을 예측함
```

Out[9]:
```
array([1])
```

In [10]:
```
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. ]
 [  31.   475. ]
 [  31.   500. ]
 [  31.5  500. ]
 [  32.   340. ]
 [  32.   600. ]
 [  32.   600. ]
 [  33.   700. ]
 [  33.   700. ]
 [  33.5  610. ]
 [  33.5  650. ]
 [  34.   575. ]
 [  34.   685. ]
 [  34.5  620. ]
 [  35.   680. ]
 [  35.   700. ]
 [  35.   725. ]
 [  35.   720. ]
 [  36.   714. ]
 [  36.   850. ]
 [  37.  1000. ]
 [  38.5  920. ]
 [  38.5  955. ]
 [  39.5  925. ]
 [  41.   975. ]
 [  41.   950. ]
 [   9.8    9.8]
 [  10.5   10.5]
 [  10.6   10.6]
 [  11.    11. ]
 [  11.2   11.2]
 [  11.3   11.3]
 [  11.8   11.8]
 [  11.8   11.8]
 [  12.    12. ]
 [  12.2   12.2]
 [  12.4   12.4]
 [  13.    13. ]
 [  14.3   14.3]
 [  15.    15. ]]
```

In [11]:
```
print(kn._y)
```

```
[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]
```

In [12]:
```
kn49=KNeighborsClassifier(n_neighbors=49) # 참고 데이터를 49개로 한 kn49모델
```

In [13]:
```
# 가장 가까운 데이터 49개를 사용하는 knn 모델에서는 어떤 데이터에서든 무조건 도미로 예측할 것임
# 원래는 5개의 데이터로 다수결의 원칙을 따라 데이터를 예측함
kn49.fit(fish_data,fish_target)
kn49.score(fish_data,fish_target)
```

Out[13]:
```
0.7142857142857143
```