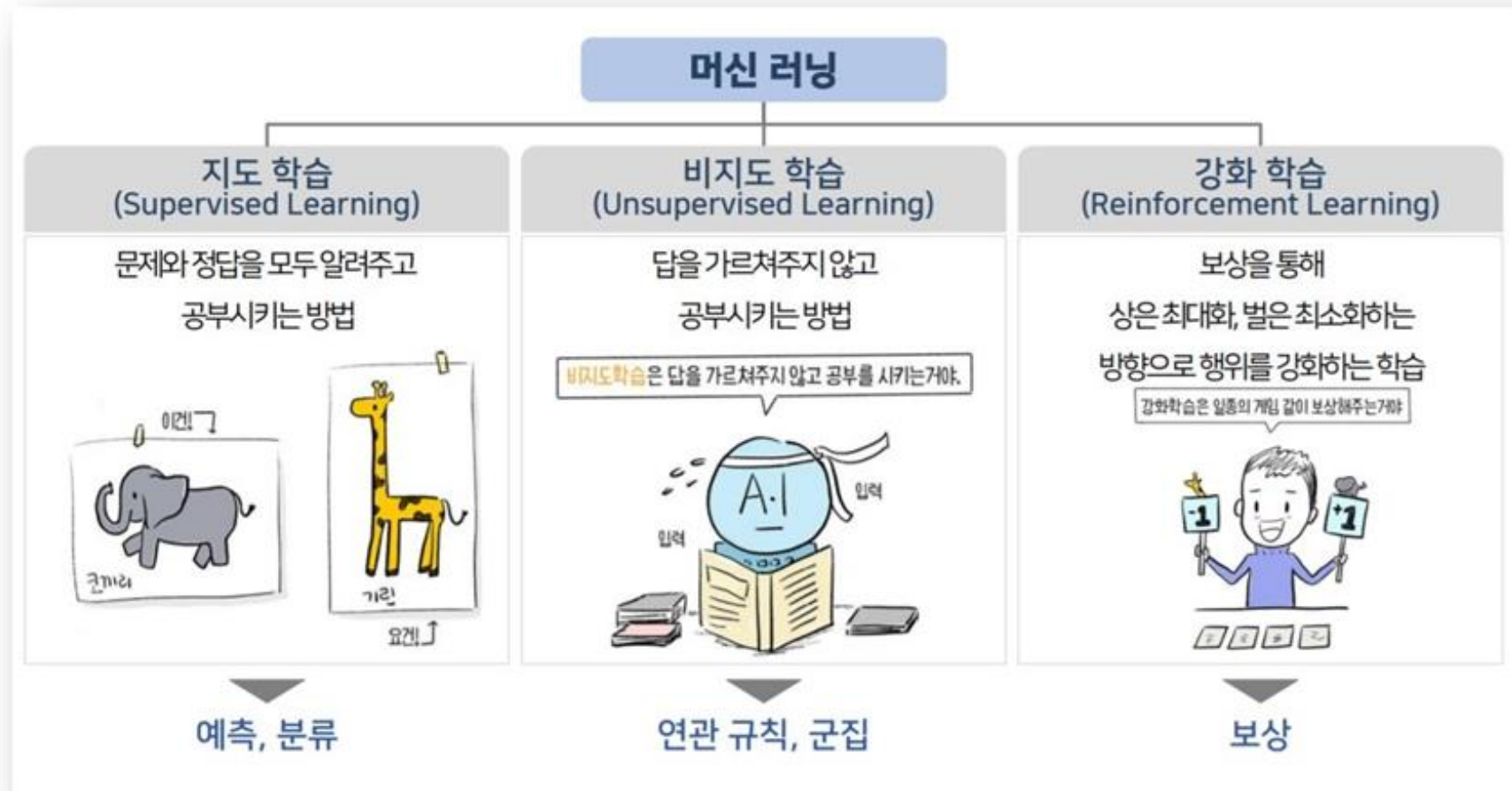
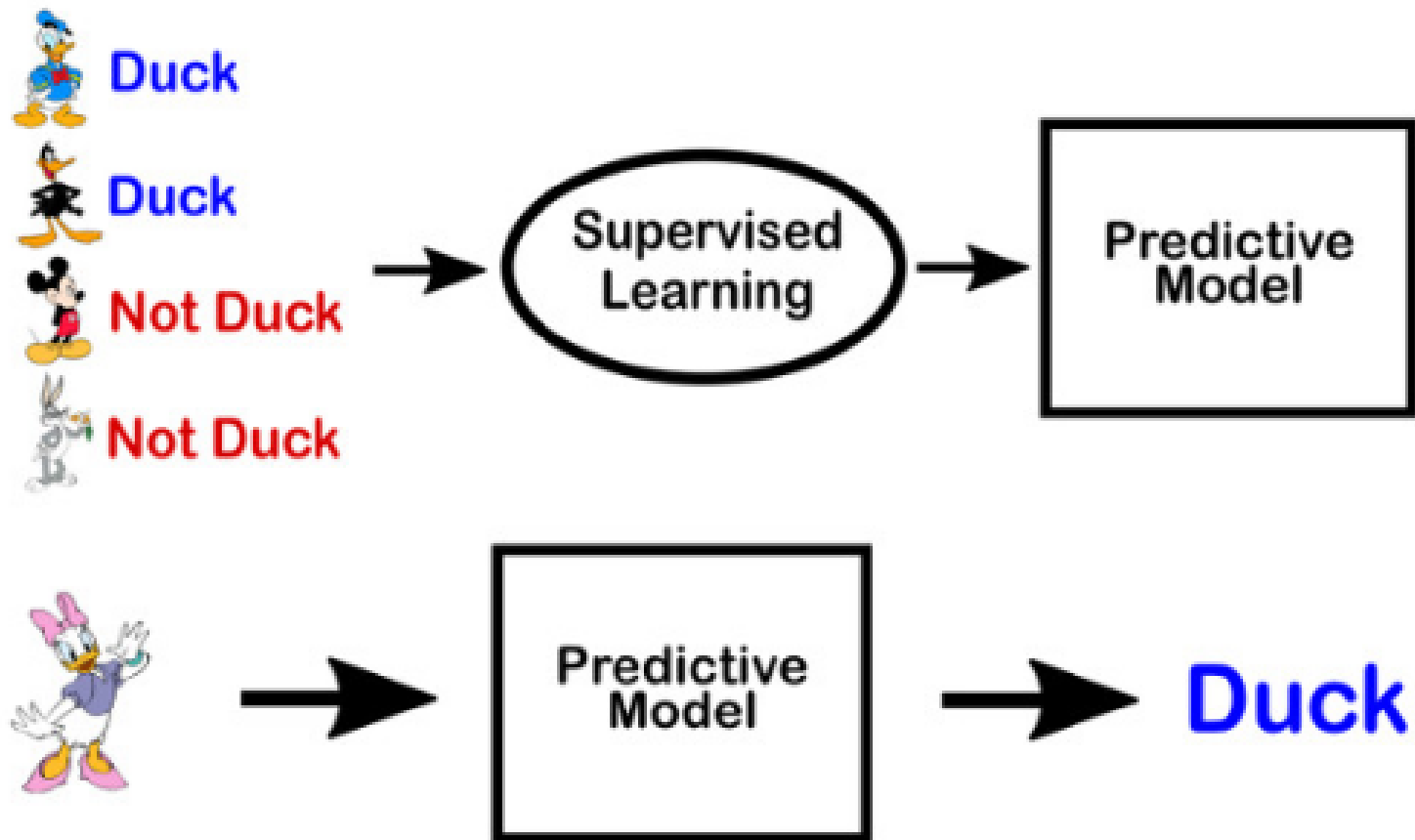


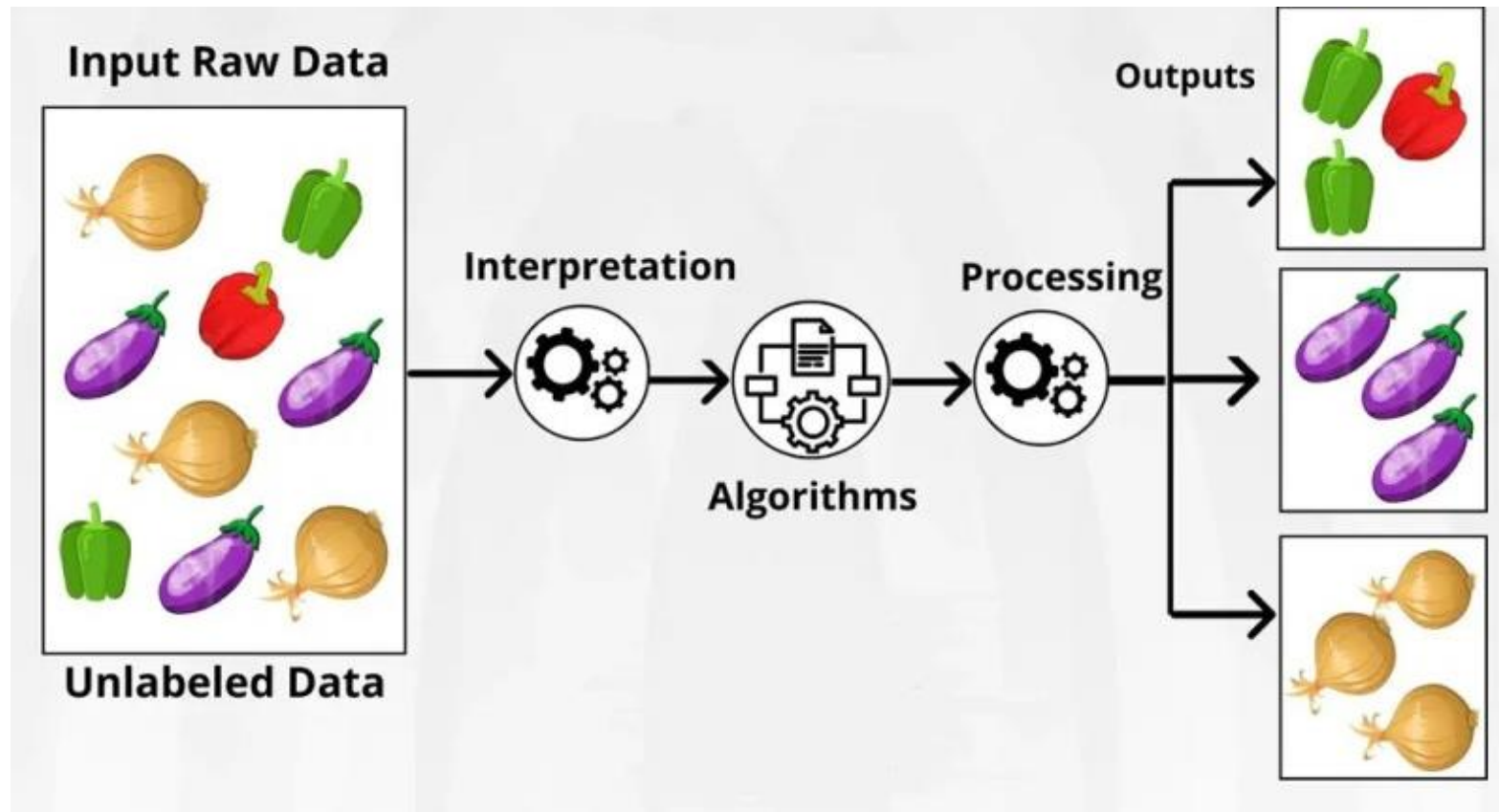
# 머신러닝 문제 종류



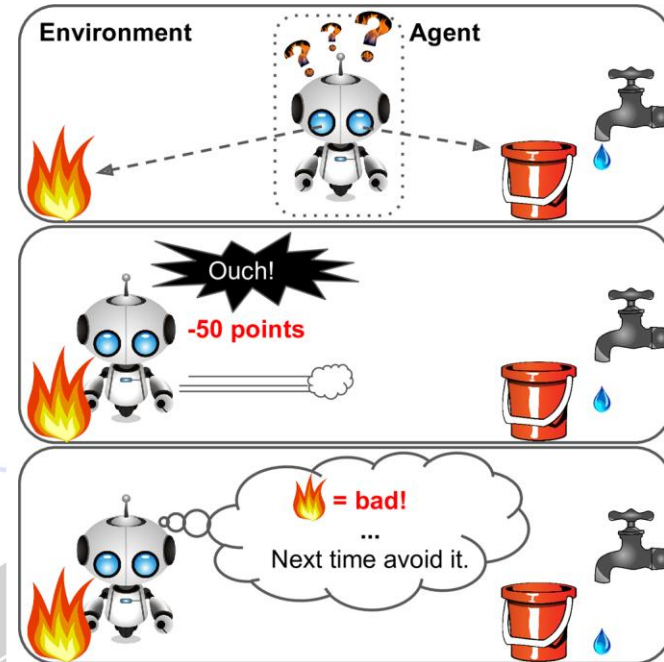
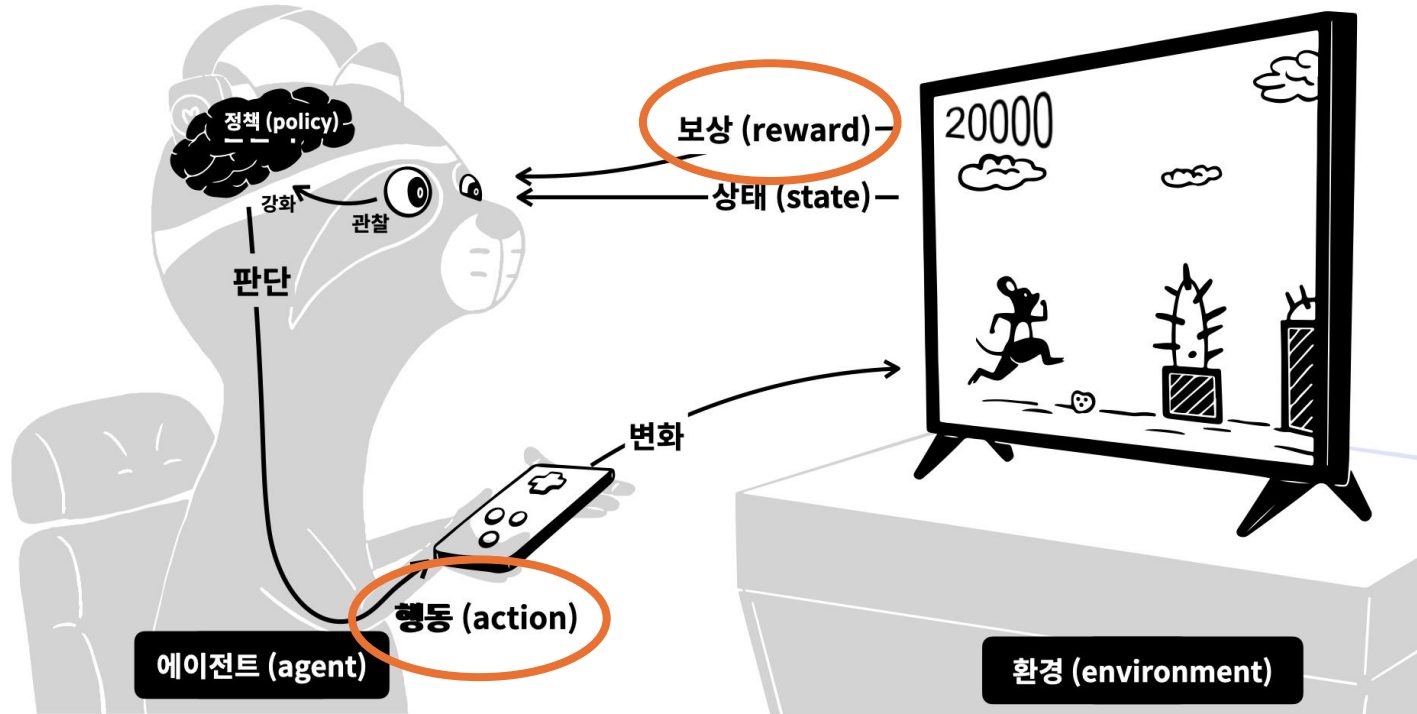
# 지도학습



# 비지도 학습



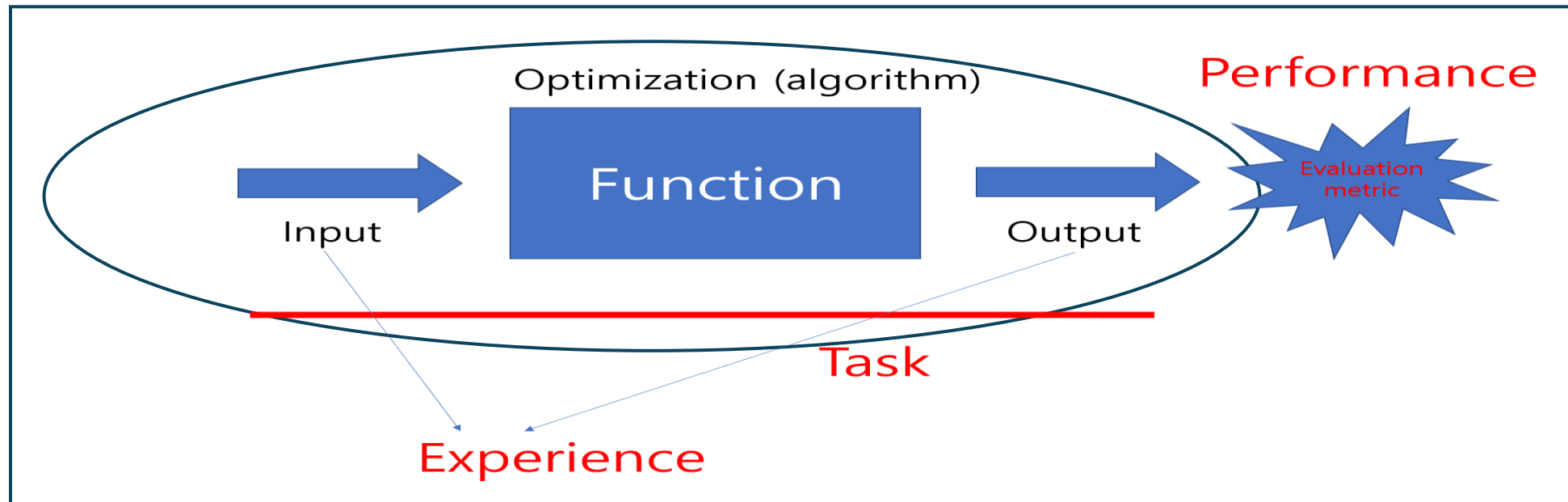
# 강화 학습



- 1 Observe
- 2 Select action using policy
- 3 Action!
- 4 Get reward or penalty
- 5 Update policy (learning step)
- 6 Iterate until an optimal policy is found

# AI Goals

## Learning

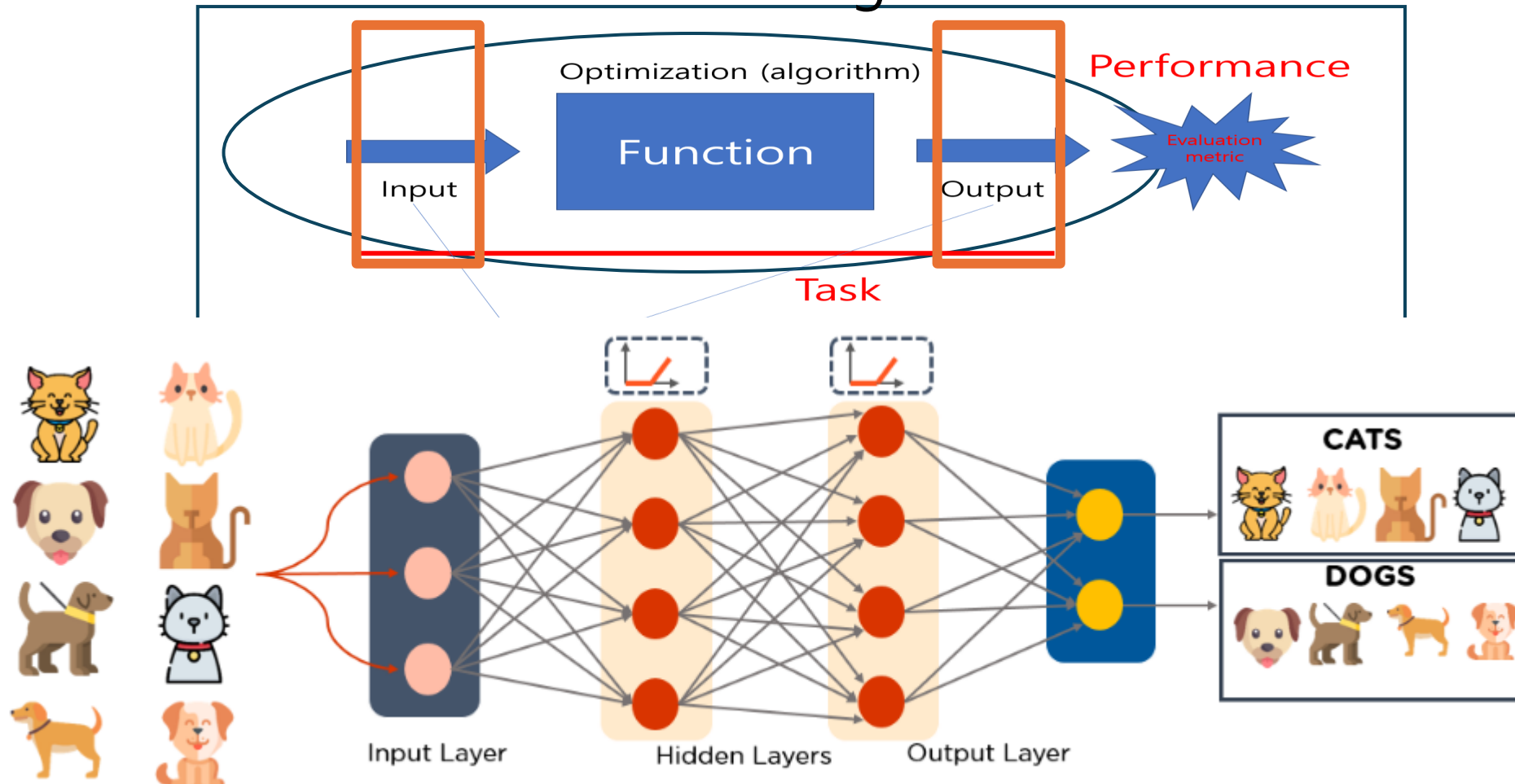


## Goal

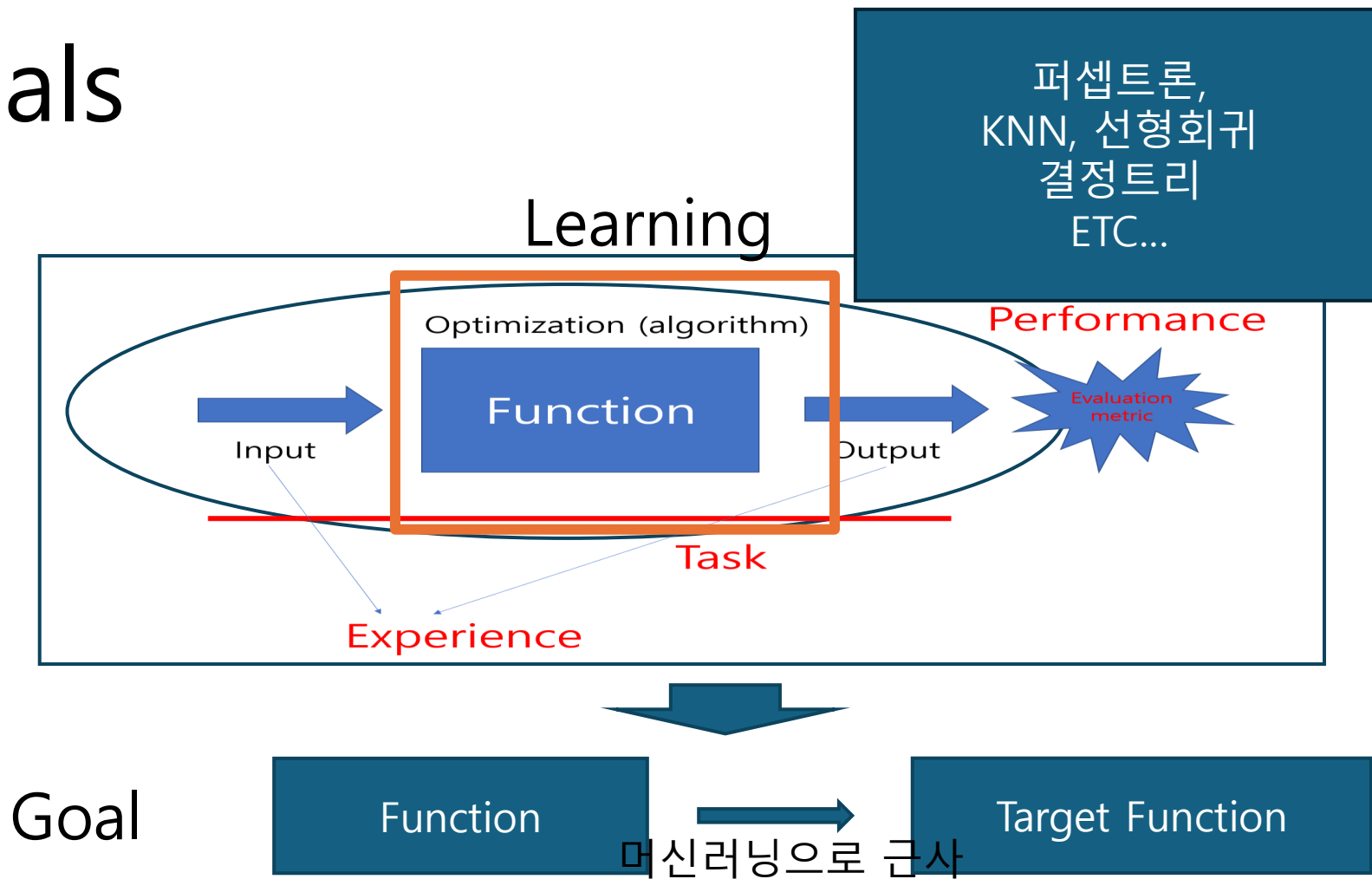


# AI Goals

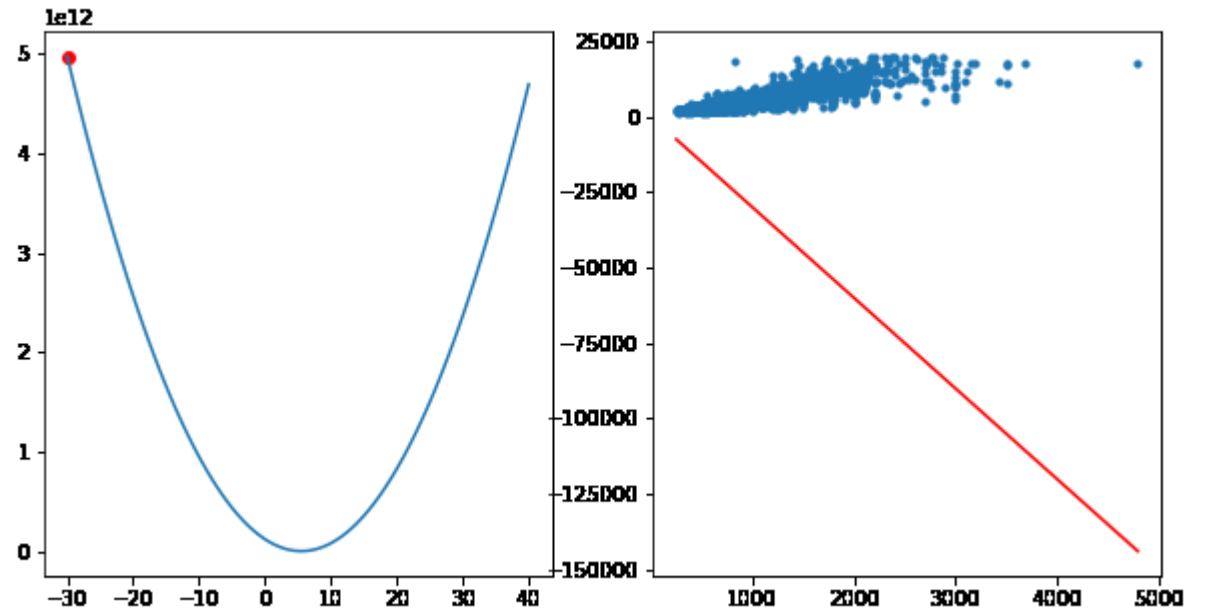
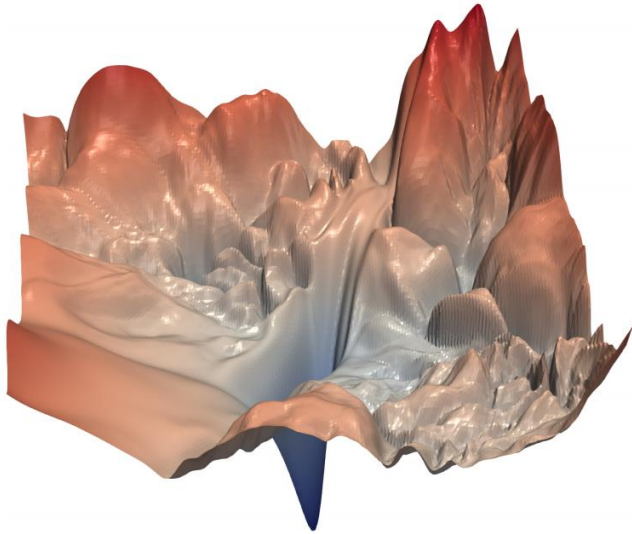
## Learning



# AI Goals



# AI Goals



Experience

Goal

Function

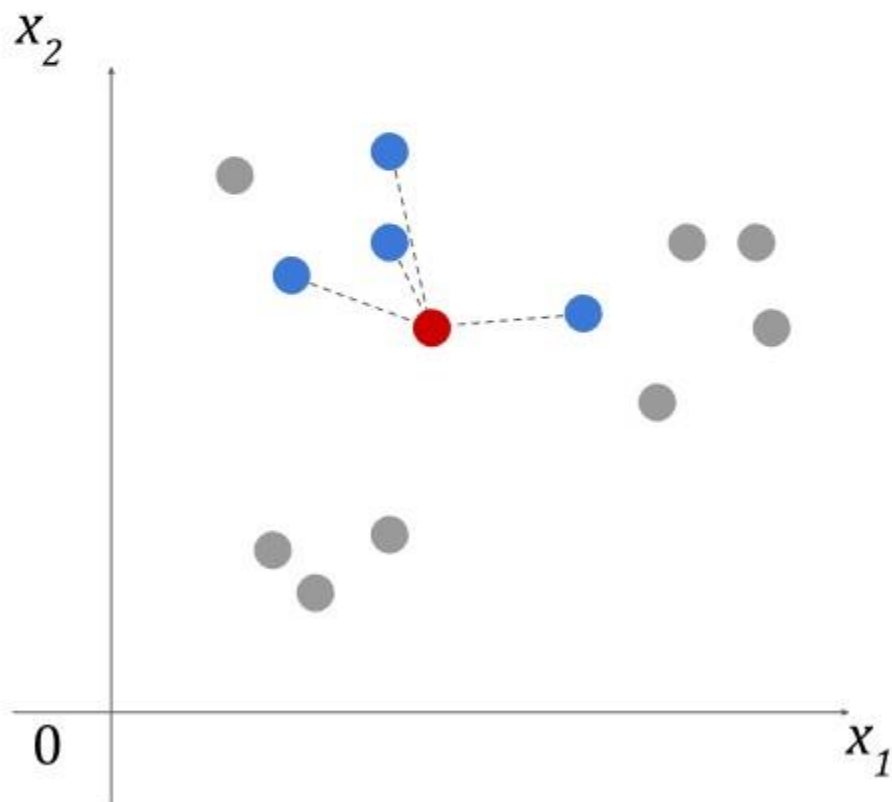
Target Function

머신러닝으로 근사



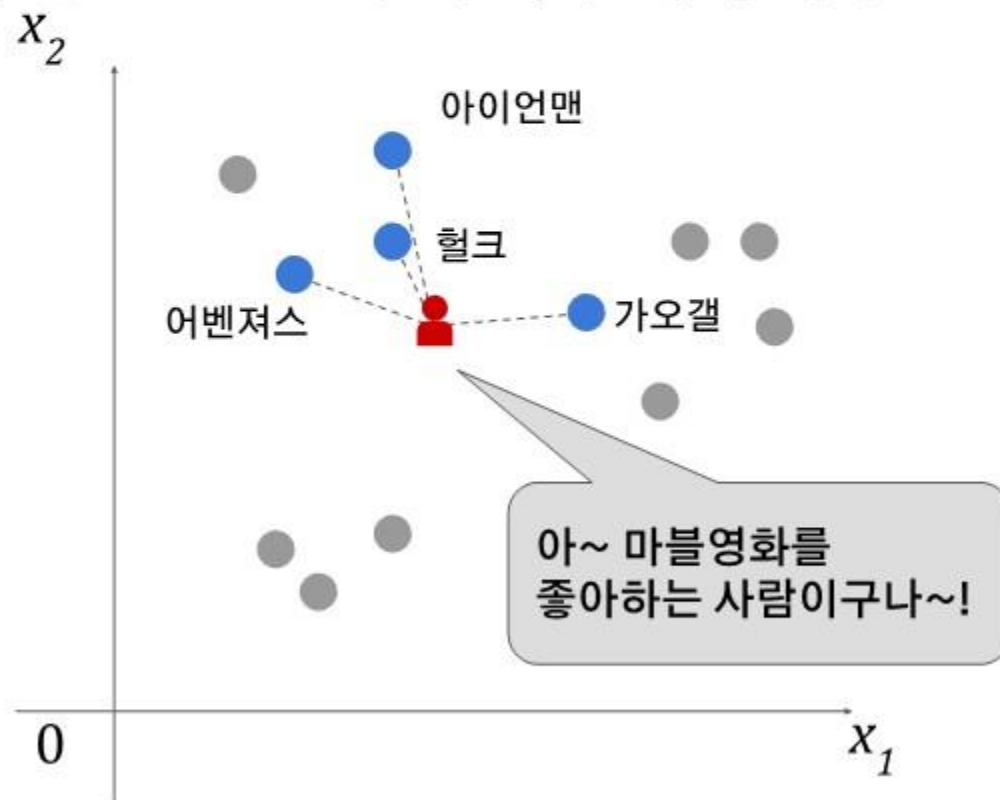
# k-Nearest Neighbors

- kNN = 가장 가까운 k개의 점



# k-Nearest Neighbors

- 어떤 점의 특성을 알고 싶다면?
  - 가까이 있는 점으로부터 특성 파악 가능

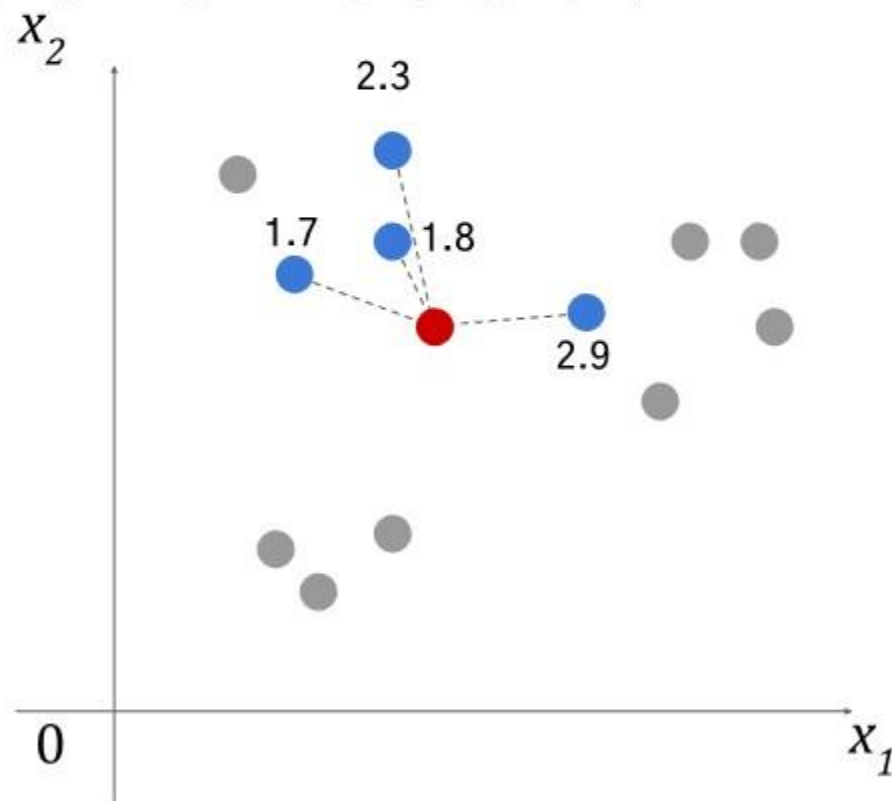


# kNN의 특징

- 데이터 기반 분석 방법
- 데이터 분포를 가정하지 않음
- 회귀문제, 분류문제 등에 적용 가능

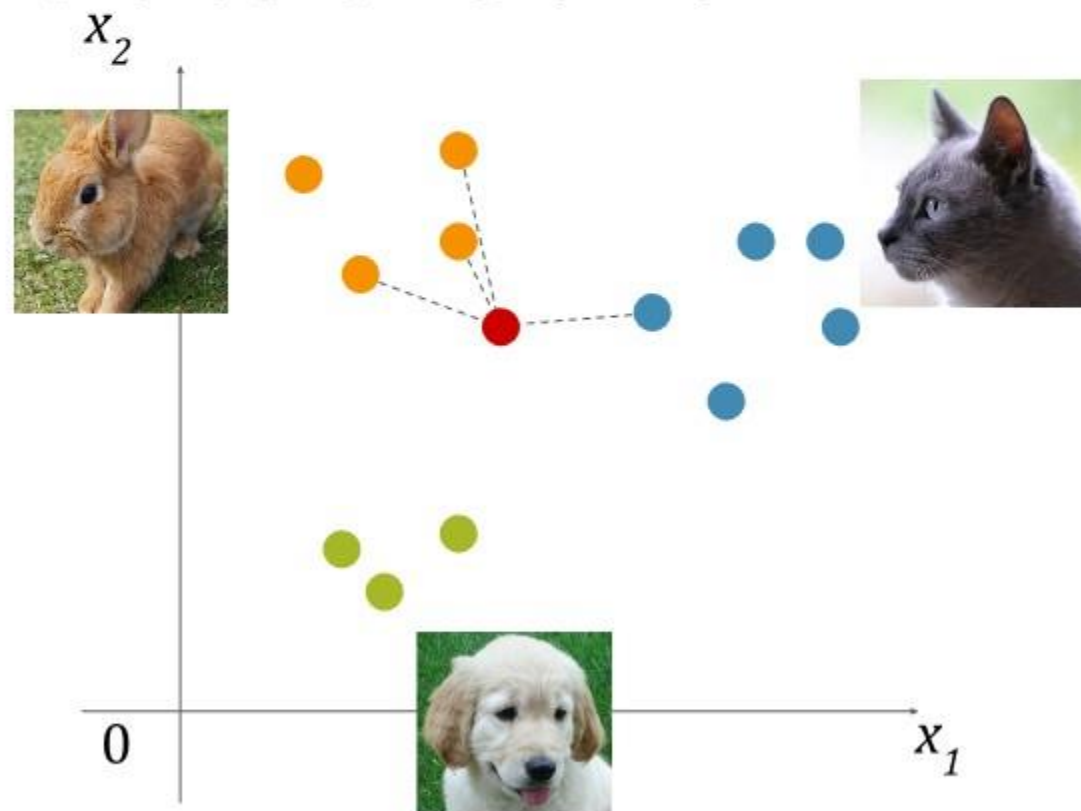
# k-Nearest Neighbors

- kNN을 회귀문제에 적용 가능
  - kNN의 값을 평균 내어 값 예측



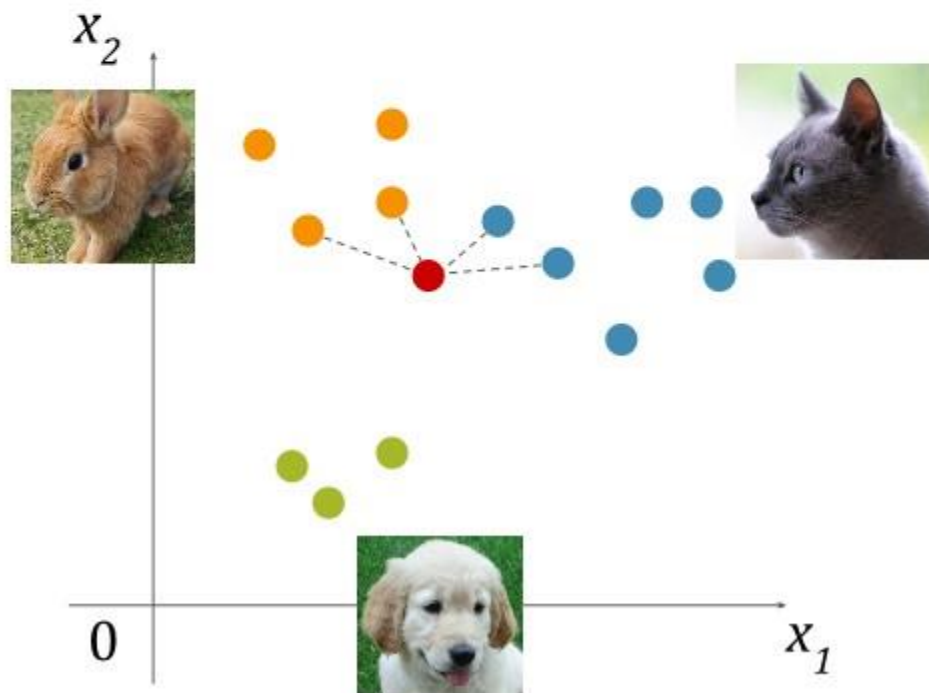
# k-Nearest Neighbors

- kNN을 분류문제에 적용 가능
  - kNN 중에 가장 많은 항목 선택



# k-Nearest Neighbors

- 1등이 여럿일 경우엔?
  - 방법1. 거리에 따라 가중치 주기
  - 방법2. 단독 1등이 나올 때까지 k를 하나씩 줄이기



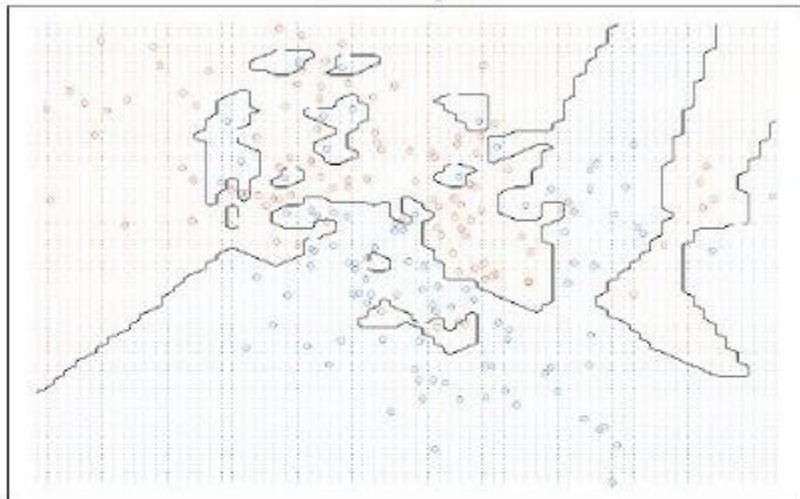
# 적절한 k값은?

- 데이터마다 적절한 k값이 다름
  - k가 낮다 → 불안정한 결과 ~ 오버피팅
  - k가 높다 → 지나친 일반화 ~ 언더피팅

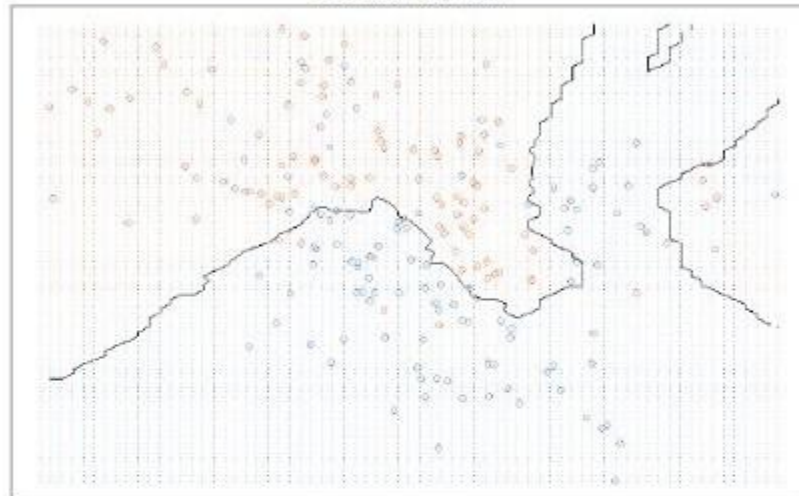


# 적절한 k값은?

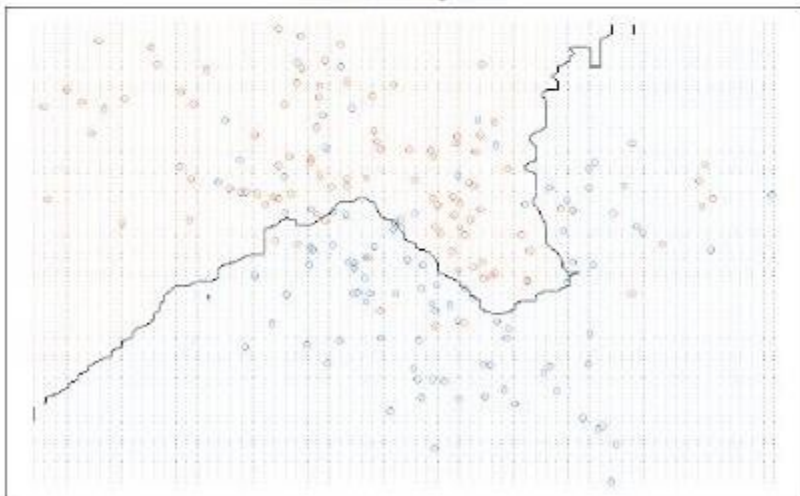
1-nearest neighbour



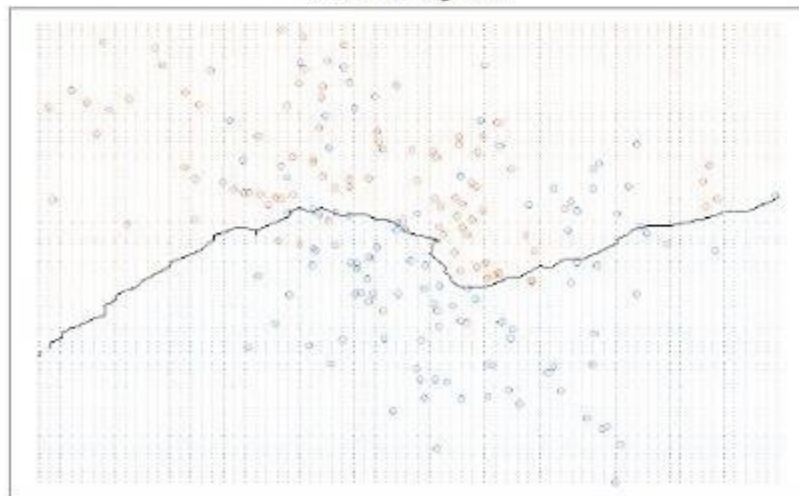
10-nearest neighbour



20-nearest neighbour



50-nearest neighbour





## 적절한 k값은?

- k를 선택하는 방법: 가장 좋은 성능을 내는 값으로 선택
  - k의 값을 1부터 증가시켜가며 각 점들에 대해 knn으로 분류해보고 오류 계산
  - 가장 오류가 적은 k값을 선택

# 거리?

$$X = (x_1, x_2, \dots, x_n)$$

$$Y = (y_1, y_2, \dots, y_n)$$

- Euclidean Distance (L2)

$$\sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

- Manhattan Distance (L1)

$$\sum_{i=1}^n |x_i - y_i|$$

- Cosine Distance

$$1 - \frac{X \cdot Y}{|X||Y|}$$



# 거리?

- Hamming Distance (곰집합)

$$|\{i \in \{1, 2, \dots, n\} | x_i \neq y_i\}| \quad \begin{array}{l} X = (x_1, x_2, \dots, x_n) \\ Y = (y_1, y_2, \dots, y_n) \end{array}$$

- 자카드거리 (집합)

$$1 - \frac{|X \cap Y|}{|X \cup Y|} \quad \begin{array}{l} X = \{x_1, x_2, \dots, x_n\} \\ Y = \{y_1, y_2, \dots, y_m\} \end{array}$$

# kNN 장단점

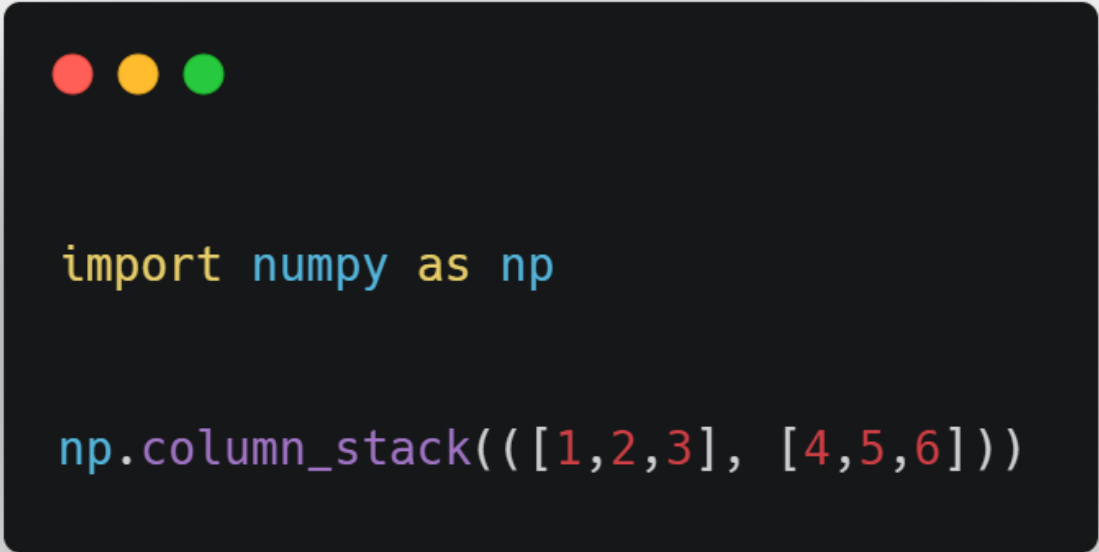
- 장점
  - 쉽고 이해하기 직관적
  - 사전 학습이 필요 없다
  - 어떤 분포든 상관 없음 (비모수 방식)
  - 데이터가 많을 경우 정확도 up! up!
- 단점
  - 데이터가 많을 경우 연산량 up! up!
    - 차원 축소 등으로 계산량 감소
    - 인덱싱으로 탐색 속도 향상  
R-Tree, KD-Tree, KNN-Graph,  
LSH(Locality Sensitive Hashing), etc.
  - 차원의 저주
    - 데이터의 차원이 증가함에 따라 정확도 급하락

# 학습 데이터 준비

```
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, 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]  
  
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, 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]
```

[http://bit.ly/bream\\_smelt](http://bit.ly/bream_smelt)


# 데이터 합치기



```
import numpy as np

np.column_stack(([1,2,3], [4,5,6]))
```

# 각 독립변수 하나의 변수(X)로 합치기



```
fish_data = np.column_stack((fish_length, fish_weight))  
print(fish_data[:5])
```

# Y의 변수의 클래스 지정



```
# [[1] * 35 + [0] * 14]  
fish_target = np.concatenate((np.ones(35), np.zeros(14)))
```




# 훈련세트와 테스트 데이터 나누기

```
from sklearn.model_selection import train_test_split


# random_state = 42를 지정한 이유는 똑같은 시드 = 똑같은 결과로 나옴.
train_input, test_input, train_target, test_target = train_test_split(
    fish_data, fish_target, random_state=42)
```

# 데이터 잘 나누어졌는지 길이 확인



```
print(train_input.shape, test_input.shape)  
print(train_target.shape, test_target.shape)
```

# 데이터 비율 균일하게



```
train_input, test_input, train_target, test_target = train_test_split(
    fish_data, fish_target, stratify=fish_target, random_state=42)

print(test_target)
```

# KNN 모델 학습 및 결과 확인



```
from sklearn.neighbors import KNeighborsClassifier

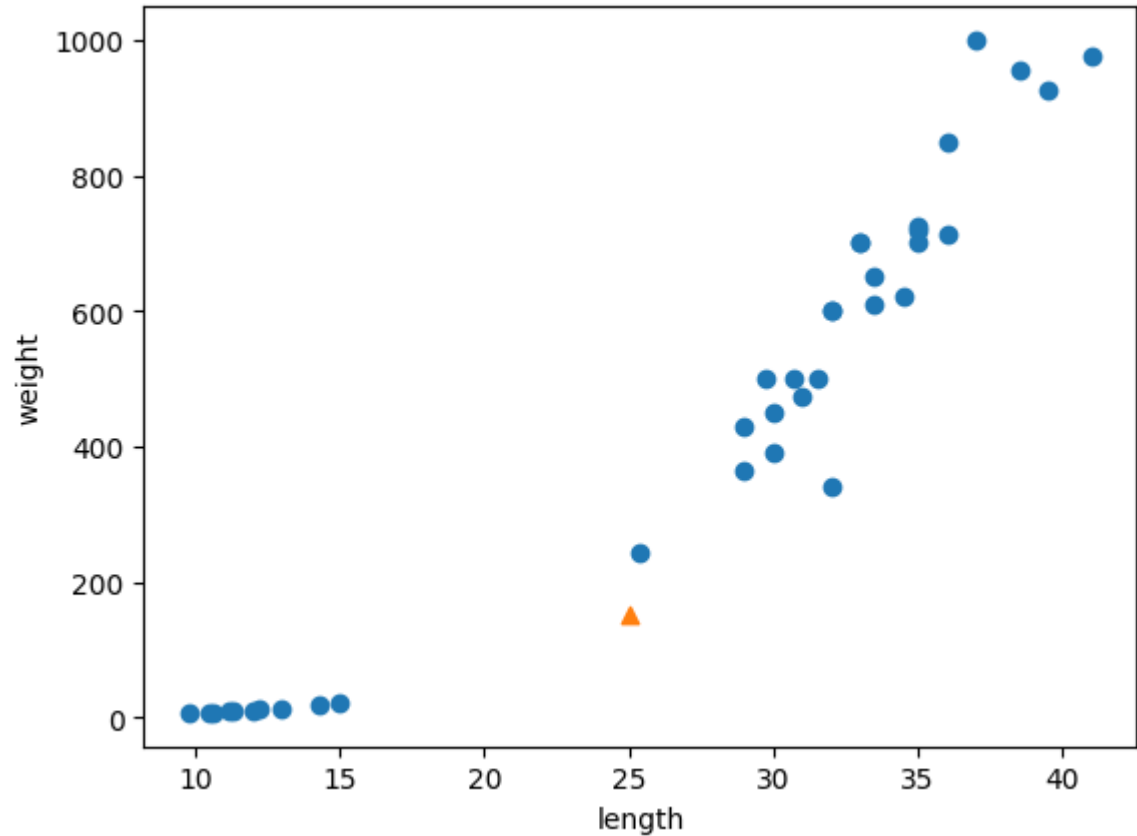
kn = KNeighborsClassifier()
kn.fit(train_input, train_target)
kn.score(test_input, test_target)

print(kn.predict([[25, 150]]))
```

# 차트 그래프 출력

```
import matplotlib.pyplot as plt

plt.scatter(train_input[:,0], train_input[:,1])
plt.scatter(25, 150, marker='^')
plt.xlabel('length')
plt.ylabel('weight')
plt.show()
```



# 25, 150 데이터에서 가장 가까운 데이터

```
distances, indexes = kn.kneighbors([[25, 150]])

plt.scatter(train_input[:,0], train_input[:,1])
plt.scatter(25, 150, marker='^')
plt.scatter(train_input[indexes,0], train_input[indexes,1], marker='D')
plt.xlabel('length')
plt.ylabel('weight')
plt.show()

print(train_target[indexes])
```