



---

---

# 머신러닝 기초

---

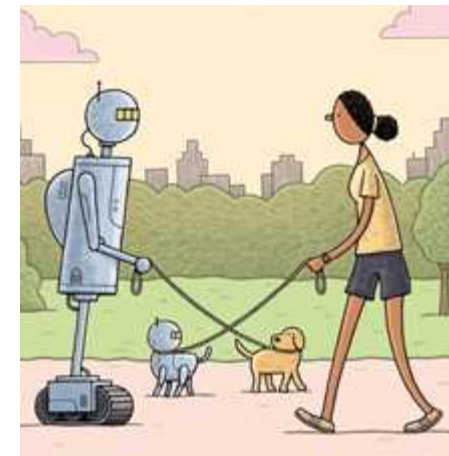
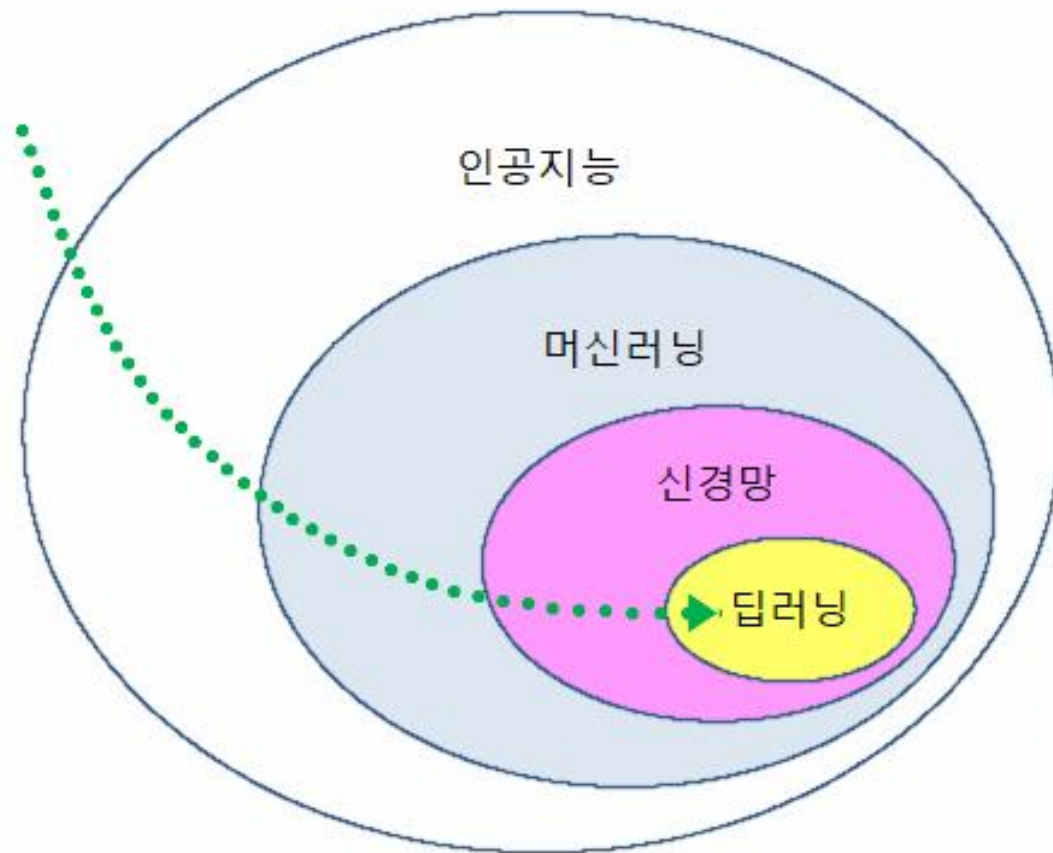
---

2021.3.22  
강사 김현호



# 머신러닝 개념

# 인공지능과 머신러닝

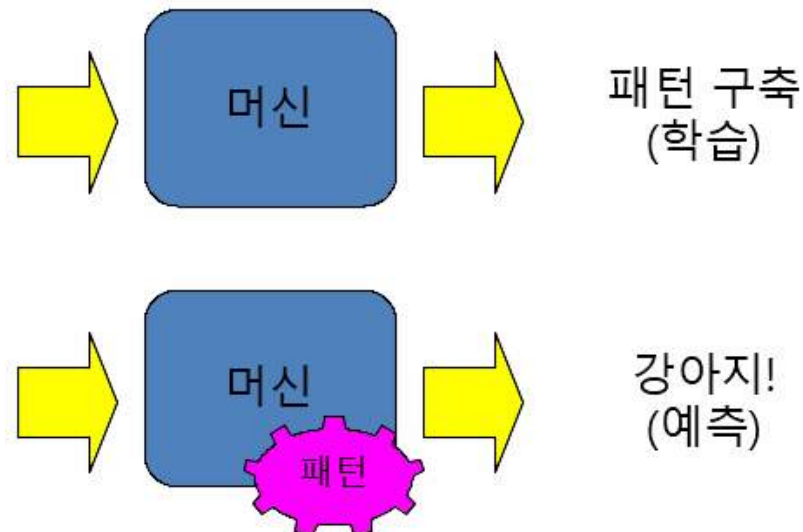


# 머신러닝과 딥러닝

- 머신러닝에서 가장 중요한 문제해결 방법이 딥러닝(신경망) 이다
  - 즉, 머신러닝에는 수많은 다른 알고리즘들이 존재한다 => kNN, SVM, 결정트리 등
  - 이 중, 신경망에 기반한 딥러닝이 머신러닝의 왕자 자리를 차지했다
- 다르게 보면, 딥러닝의 기본틀은 머신러닝 개념이다
  - 딥러닝을 철저히 이해하기 위해서는 머신러닝의 다양한 방법론들을 알 필요가 있다
  - 반대로, 먼저 딥러닝에 익숙해지고 차차 머신러닝 개념을 이해해 나갈수도 있다
  - 딥러닝을 배울 때, 항상 머신러닝 개념을 염두에 두자

# 머신러닝 정의

- 가지고 있는 데이터에서 패턴을 찾아 새로운 데이터를 잘 예측하려는 컴퓨터 알고리즘
  - 어린 아이가 많은 경험을 쌓아 현명한 판단을 할수 있는 어른으로 성장해 나가는 것과 비슷하다
- 컴퓨터(Machine)가 데이터로 부터 배워(Learning) 나가는 것



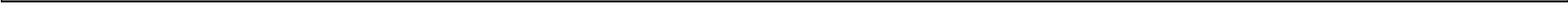
# 데이터는 무엇인가?

- 수이다
  - 생물과 인간은 감
  - 컴퓨터는 숫자
  - 하지만, 감이라는 것도 신경으로 전달되는 전류량일 뿐이다
- 모든 데이터는 숫자로 변환하여 처리한다
  - 키, 몸무게, 국민소득 => 원래 숫자
  - 사진 => 픽셀값 (0~255 범위의 밝기)
  - 문장 => 단어목록 번호로 변환한 숫자들의 연속
  - 개인가 고양이인가? => 개는 0, 고양이는 1
  - 어디로 갈 것인가? => 집 21.3%, PC방 35.2%, 호프집 43.5%

# 머신러닝 구분

- 회귀/분류
  - 회귀(Regression) => 값을 예측 (내일 주가가 얼마일까?)
  - 분류(Classification) => 종류를 예측 (내일은 오를까 내릴까?)
- 지도학습/비지도학습
  - 지도학습(Supervised) => 정답을 보여주고 학습시키는 방법  
=> 과일을 보여주며 이건 사과, 저건 바나나 하며 알려줌
  - 비지도학습(Non-Supervised) => 정답을 알려주지 않고 학습  
=> 아무 말 없이 계속 과일만 보여줌
- 가장 흔한 문제는 **분류**이다
  - 회귀는 주로 경제문제에 적용된다 (내년도 경제성장률, 내일 주가, 소득과 행복과의 관계 등)
  - 비지도학습 문제는 드물다 => 중요하지 않아서가 아니라 어렵기 때문

\_\_\_\_\_





# 파이썬 머신러닝 라이브러리

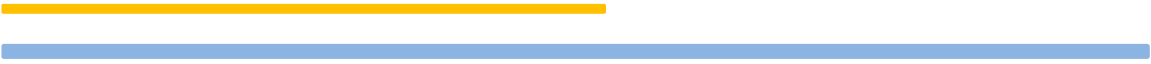
---

- Scikit-Learn
  - 신경망과 딥러닝을 제외한 모든 머신러닝 알고리즘
  - <https://scikit-learn.org>
  - pip install sklearn
- Tensorflow/Keras
  - <http://tensorflow.org>
  - Tensorflow2 부터 keras 는 tensorflow에 합쳐짐
  - pip install tensorflow

# 개발 환경

---

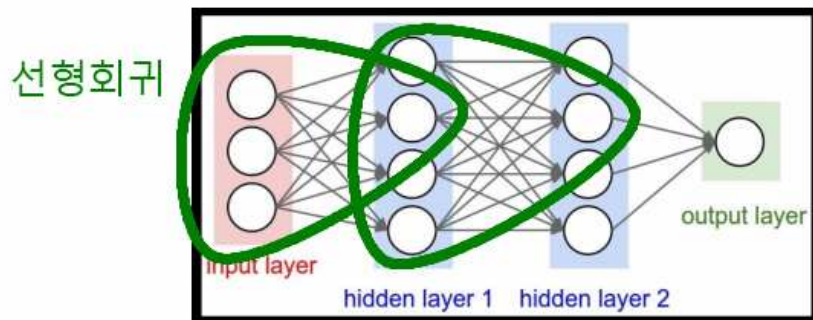
- 개발 폴더 생성 (예: c:\Wpython)
- 아나콘다 설치
  - sklearn 과 jupyter 포함함
  - <https://repo.anaconda.com/archive> 에서 다운로드
  - [https://repo.anaconda.com/archive/Anaconda3-2019.10-Windows-x86\\_64.exe](https://repo.anaconda.com/archive/Anaconda3-2019.10-Windows-x86_64.exe) 추천
- jupyter notebook 또는 jupyter-lab
  - 아나콘다프롬프트 띄우고, 개발폴더로 이동후 “jupyter notebook” 실행
  - 크롬 브라우저에서 코딩
- 텍스트 에디터 (notepad++ 등)



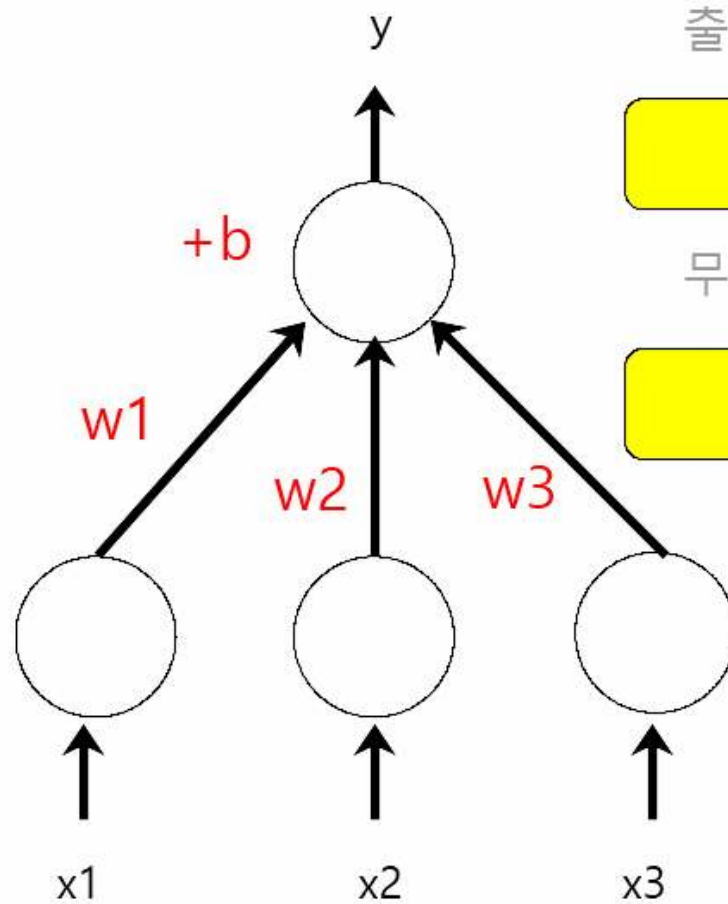
회귀, 분류

# 기본중에 기본 선형회귀

- 선형회귀(Linear Regression) 은 머신러닝의 가장 기본이 되는 알고리즘이다
  - 가장 단순하면서도 활용도가 크다
  - 사례) 키와 몸무게 데이터로 학습한다 => 새로운 학생의 키로 몸무게를 예측한다 (몸무게는 실수값임)
- 게다가, 선형회귀는 신경망의 기본 뼈대가 된다
  - 신경망은 선형회귀 라는 블록으로 쌓아올린 빌딩이다



# 선형회귀 구조



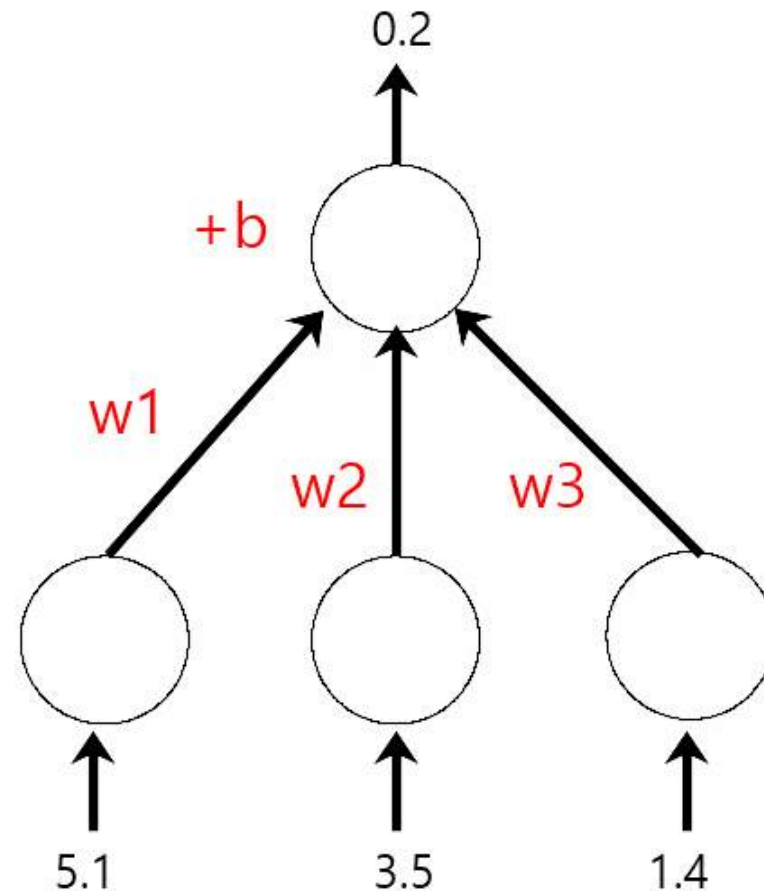
출력값은?

무엇을 찾는가?

# 선형회귀 데이터 예시

	<b>x</b>	<b>y</b>
	속성	목표값
샘플	5.1, 3.5, 1.4	0.2
	4.9, 3.0, 1.4	0.4
	4.7, 3.2, 1.3	0.3
	4.6, 3.1, 1.5	0.2
	5.0, 3.6, 1.4	0.3

$$\begin{aligned}5.1*w1+3.5*w2+1.4*w3+b &= 0.2 \\4.9*w1+3.0*w2+1.4*w3+b &= 0.4 \\4.7*w1+3.2*w2+1.3*w3+b &= 0.3 \\4.6*w1+3.1*w2+1.5*w3+b &= 0.2 \\5.0*w1+3.6*w2+1.4*w3+b &= 0.3\end{aligned}$$



# 풀이의 시작 - 오류계산

- 알고싶은 것은 =>  $w_1, w_2, w_3, b$
- $w_1=w_2=w_3=b=1$  로 값을 대강 주고 시작

계산값	목표값	오류
$5.1*w_1+3.5*w_2+1.4*w_3+b = 11.0$	0.2	$(10.8)**2$
$4.9*w_1+3.0*w_2+1.4*w_3+b = 10.3$	0.4	$(9.9)**2$
$4.7*w_1+3.2*w_2+1.3*w_3+b = 10.2$	0.3	$(9.9)**2$
$4.6*w_1+3.1*w_2+1.5*w_3+b = 10.2$	0.2	$(10.0)**2$
$5.0*w_1+3.6*w_2+1.4*w_3+b = 11.0$	0.3	$(10.7)**2$

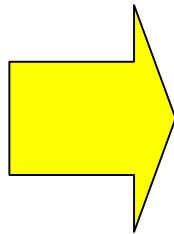
$$\text{오류평균} = 1/5 * [(10.8)**2+(9.9)**2+(9.9)**2+(10.0)**2+(10.7)**2]$$



**어떻게 오차를 줄일 것인가?**

# 풀이의 목표 - 오차줄이기

- 대강 봐도 가중치의 값이 너무 컸다!
- $w_1=w_2=w_3=b=0.9$  로 바꾸어 보자
  - 실제로는 가중치 4개를 다른 값으로 바꾸어 나간다
  - 0.9 로 바꿀것인가, 아님 0.8로 바꿀 것인가 (아주 큰 이슈임)
- 이렇게 계속 가중치를 바꾸어 가다보면 언젠가는 좋은 결과를 얻을 것이다
  - 하지만 너무 대강대강이고 규칙이 없다
  - 가중치 조정값을 결정하는 좀더 체계적인 방법이 없을까?

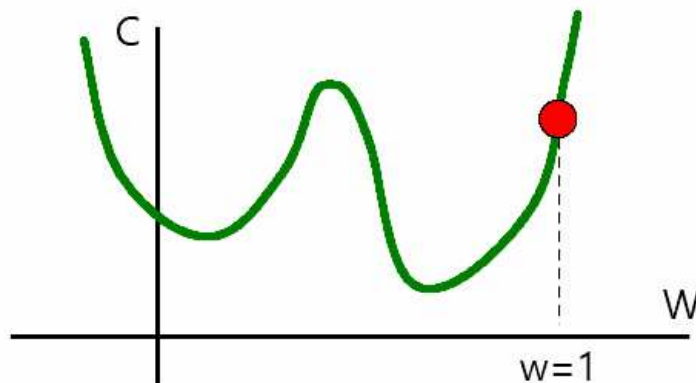


경사하강법



# 풀이의 단서 - 오차함수

- 오차함수(Cost Function)
  - 앞에서  $x$ 는 정해져 있고,  $w$  값이 바뀔에 따라 오류가 커졌다 작아졌다 함을 알 수 있다
  - 이렇게  $w$ 에 따라 오차가 변하므로 오차는  $w$ 의 함수라고 부른다 => 오차함수  $C = C(w)$
  - 앞에서  $C$  값은 예측값과 목표값의 차이를 제공한 후 평균했다 => MSE
  - 아래는 오차함수의 사례이다.  $w$ 를 어떻게 바꿀까?

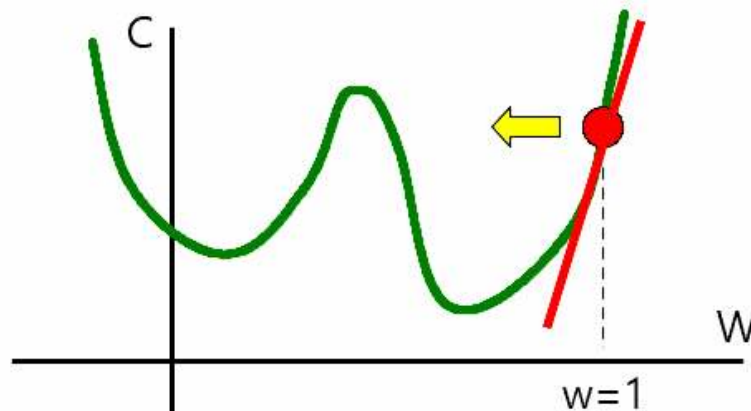


$w$ 를 줄이면 되겠다.  
그런데 얼마나 줄일까?

**혹시? 기울기?**

# 풀이의 해법 - 경사하강법

- 경사하강법 (Gradient Descent)
  - .가중치를 초기화한다 ( $w_1=w_2=w_3=b=1$ )
  - .비용함수를 정의한다  $C(w)=MSE=\text{sum}[(\text{예측값}-\text{목표값})^2]/5$
  - . $w$  값의 변경에 따른  $C$ 의 변화량, 즉 기울기를 구한다
  - .기울기에 비례하여  $w$  값을 바꾸어준다
    - => 기울기가 0.3이면,  $w_1=1$  에서  $w_1=1-0.3=0.7$
    - => 실제로는  $w_1 = 1 - \text{학습률} \times 0.3$  (학습률은 경우에 맞게 여러분이 정한다)
  - .3번으로 돌아간다 ( $w$ 값이 바뀐 상태임)



기울기가 양수면 가중치를 줄여주고  
기울기가 음수면 가중치를 늘려준다

# 풀이의 어려움 - 기울기 구하기

- 결국, 경사하강법을 적용하면 아무리 어려운 문제라도 풀수 있다
  - 단, 기울기를 구할수 있다면 말이다
  - 경사하강법에서의 기울기를 쉽게 구해주는 툴  
=> tensorflow/keras, pytorch 등등임
- 선형회귀에서는 기울기값이 수학적으로 풀려져 있다
  - MSE(Mean Squared Error) 로 비용함수를 정의할때
  - 기울기 = -입력값\*(목표값-예측값)
  - $w \rightarrow w - \text{학습률} * \text{기울기}$

# 참고 - 선형회귀 경사하강법 풀이

$$\hat{y}_i = \sum_j (w_j \cdot x_{ij}) + b$$

$$\begin{aligned} Loss_i &= (y_i - \hat{y}_i)^2 \\ &= (y_i - \sum_j (w_j \cdot x_{ij}) - b)^2 \end{aligned}$$

$$Loss = \frac{1}{N} \sum_i (y_i - \sum_j (w_j \cdot x_{ij}) - b)^2$$

$$\frac{\partial Loss}{\partial w_j} = -\frac{2}{N} \sum_i (y_i - \hat{y}_i) \cdot x_{ij}$$

$$\frac{\partial Loss}{\partial b} = -\frac{2}{N} \sum_i (y_i - \hat{y}_i)$$

$$w_j \rightarrow w_j - lr \cdot \frac{\partial Loss}{\partial w_j} = w_j + lr \cdot \frac{2}{N} \sum_i (y_i - \hat{y}_i) \cdot x_{ij}$$

$$b \rightarrow b - lr \cdot \frac{\partial Loss}{\partial b} = b + lr \cdot \frac{2}{N} \sum_i (y_i - \hat{y}_i)$$

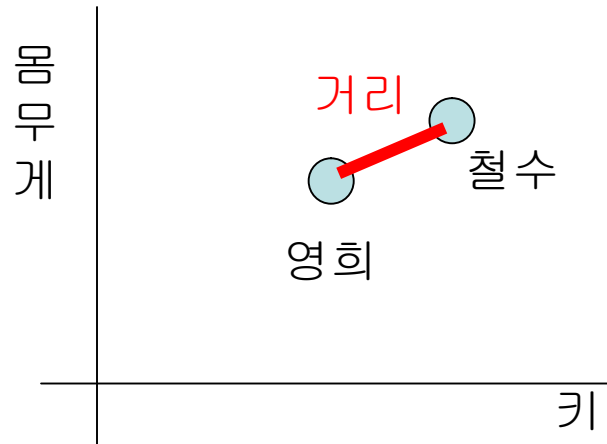
# 실습 - 선형회귀

---

- iris 데이터를 불러온다
- 앞의 세가지 속성으로 네번째 속성값을 선형회귀로 예측해 보자
  - sklearn 의 LinearRegression 사용
  - 경사하강법 직접 구현
  - keras 적용
- petal length 로 petal width를 예측하는 문제를 풀어보자
  - 결과를 matplotlib 그래프로 그려보자
- 선형회귀에서는 무엇을 예측하는가?

# 데이터와 거리(distance)

- 영희와 철수 사이의 거리는?
  - 영희는 키가 165, 몸무게가 45
  - 철수는 키가 175, 몸무게가 65

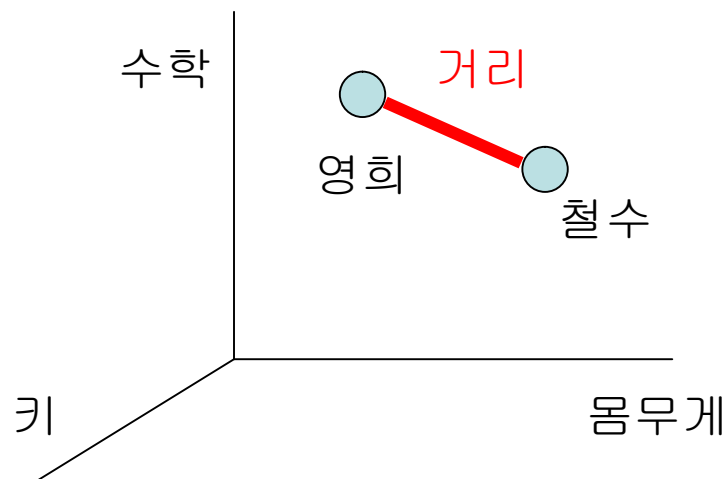


$$\begin{aligned}\text{거리} &= \sqrt{10^2 + 20^2} \\ &= \sqrt{500}\end{aligned}$$

- 각 속성을 좌표축(x축, y축 등)으로 생각하자
  - 각 샘플을 공간상의 점으로 찍을 수 있다
  - 그러면, 샘플들 간의 거리를 구할 수 있다
- 거리는 두 샘플이 얼마나 가까운지, 얼마나 비슷한지를 수치로 나타낼 수 있다.
  - 머신러닝은 모든것을 수치화할 수 있어야 한다

# 속성이 3개일때의 거리

- 영희와 철수 사이의 거리는?
  - 영희는 키가 165, 몸무게가 45, 수학점수 91
  - 철수는 키가 175, 몸무게가 65, 수학점수 86



$$\begin{aligned}\text{거리} &= \sqrt{10^2 + 20^2 + 5^2} \\ &= \sqrt{525}\end{aligned}$$

- 속성이 3개이면 3차원 공간이 됨을 알수 있다
  - 속성의 갯수가 곧 공간의 차원이 된다
  - 아이리스 데이터의 경우 몇 차원일까?

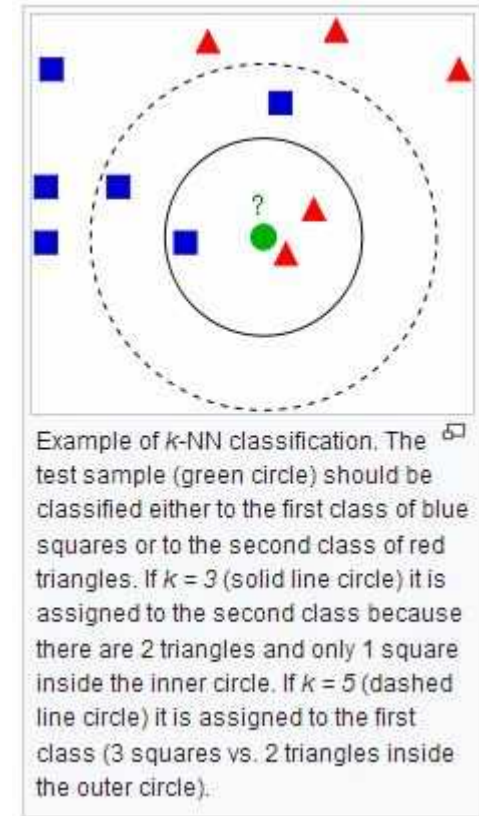
# 실습 - 거리

- $x_1$ 의 좌표가 (2,5),  $x_2$ 의 좌표가 (5,2) 일때, 두 점 사이의 거리를 numpy 를 이용해 구해보자.
- $x_3$ 의 좌표가 (3,7)일때, 3점을 3x2 어레이로 만들고 이들 사이의 거리를 나타내는 3x3 거리표를 만들어보자.
- 위의 세점을 그래프로 표시하자.
- iris 데이터는 속성이 4개이다. 샘플들 간의 거리를 모두 측정하면 150x150의 행렬을 만들수 있다. 이 결과를 plt.imshow() 함수를 이용해 표시해 보자.
- iris 첫번째 샘플과 다른 모든 샘플들 간의 거리를 구하면 150개의 값이 나온다(자기 자신 포함). 이 결과를 plt.plot() 함수로 그려보자.



# 직관적인 kNN

- kNN: k Nearest Neighbours
  - 가장 가까운 샘플 k 개를 고려한다는 의미임
  - 어떤 사람의 **가장 친한 친구** 5명을 알면 그 사람을 파악할 수 있을까?
  - 어느정도는 가능할 것이다. 이것이 kNN의 핵심 아이디어이다.
- 옆의 그림에서 녹색점은 빨간쪽인가 파란쪽인가?
  - 가장 가까운 친구 3명을 고려하면, 빨간쪽
  - 5명을 고려하면 파란쪽
  - **k 값**을 어떻게 잡는지가 아주 중요함을 알수 있다
- kNN은 **분류(classification)** 알고리즘이다
  - 이것인지, 저것인지, 아니면 다른것인지 판별해 준다
  - 앞에서 배운 **거리**를 이용해 가까운 친구를 찾는다
  - 그러므로 모든 점들 간의 거리를 구해야 한다
  - 샘플 수가 많을 수록 계산량이 엄청나게 증가한다



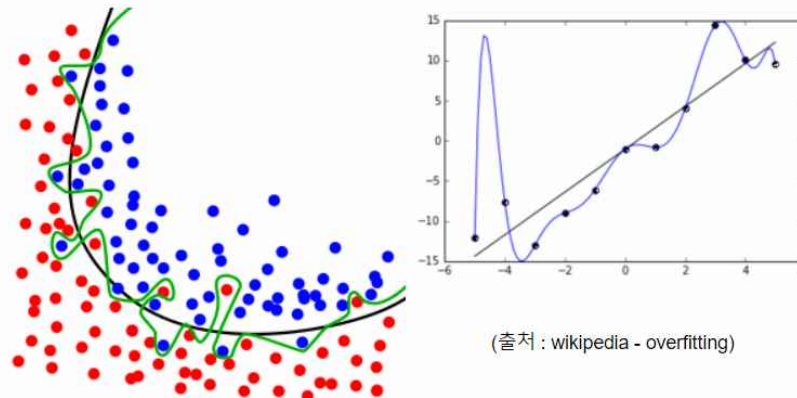
# 학습/테스트 데이터 분리

- 머신러닝은 목적은 **미래를 예측**하려는 데 있다
  - 과거 데이터로 새로운 데이터를 예측한다
- 하지만, 모든 데이터는 **과거 데이터**이다
  - 만들어진 모델을 검증하려면 미래를 기다릴 수 밖에 없다
  - 미래에 데이터를 확보하더라도 바로 과거 데이터가 되어버린다
- 최대한 좋은 모델을 만들기 위해 가진 **데이터의 일부**를 테스트 데이터로 떼어 놓는다
  - 데이터는 돈이다. 아주 아깝지만 어쩔 수 없다
  - 1000개의 데이터가 있으면, 750개는 학습데이터로 250개는 검증용 테스트 데이터로 분리해 보자
- sklearn 의 **train\_test\_split()** 함수를 이용
  - 디폴트로, 랜덤하게 데이터를 분리한다
  - fit() 으로 학습할 때는 X\_train, y\_train 을, predict() 로 예측할 때는 X\_test, y\_test 를 사용한다

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target)
```

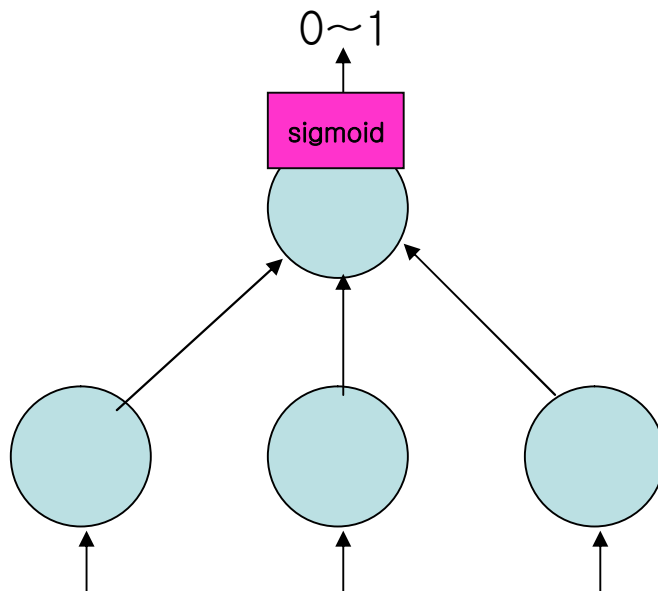
# 과적합 이슈

- 지나치게 과거데이터에 꼭맞게 만들어진 모델을 **과대적합(overfitting)** 되었다고 한다
  - 사람 얼굴을 판별하는 모델이 있다고 하자. 과대적합된 모델은 내일 아침에 출근하는 나의 얼굴을 알아보지 못할 것이다.
- 너무 대강 만들어진 모델을 **과소적합** 되었다고 한다
  - 나의 과거나 다른과목 성적을 보지않고, 학급 평균으로 나의 수학점수를 예측하는 모델이 있다고 하자. 너무 과소적합되었다.
  - 즉, 주어진 데이터를 너무 사용하지 않았다
- kNN 에서  $k=1$  인 경우가 과대적합,  $k=\text{샘플수}$  인 경우가 과소적합의 사례이다
- 아래 그림에서 왼쪽은 분류, 오른쪽은 회귀 모델이다. 과대/과소적합을 구분해보자.
- 머신러닝에서 과적합 이슈는 가장 어려운 문제 중에 하나이다
  - 신경망에서는 훈련횟수와 망구조를 단순화/복잡화 함으로서 과적합을 피하려고 노력한다

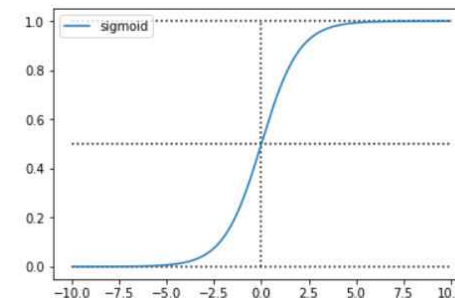


# 분류의 기초 - 로지스틱 회귀

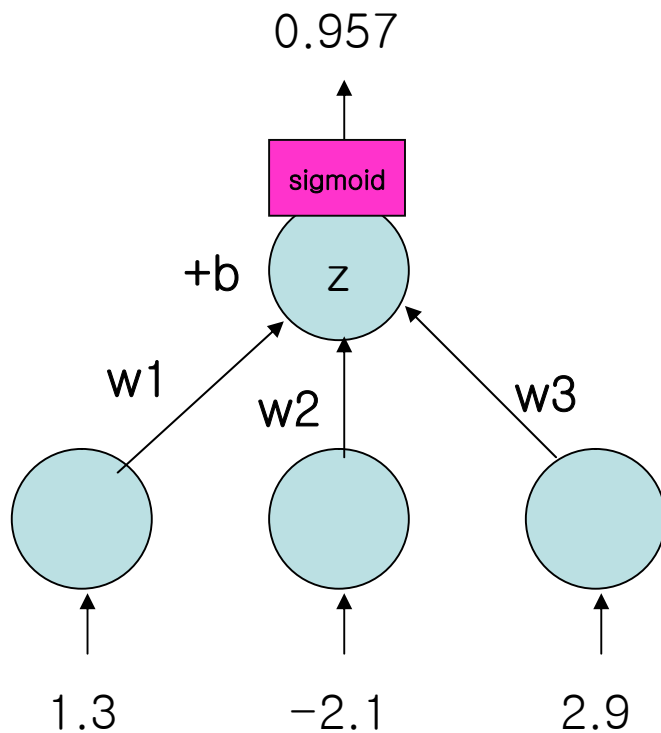
- 로지스틱 회귀(Logistic Regression)
  - 이름은 회귀이지만 **분류** 알고리즘이다
  - 선형회귀와 함께 **신경망의 기본 뼈대**를 이룬다
  - 선형회귀와 다른 점은, 출력값을 **0~1 사이의 확률값**으로 바꿔주는 것 뿐이다 (**sigmoid** 함수 적용)
  - 출력값이 0.5 보다 큰지 작은지로 두가지 클래스를 구분한다



$$\text{sigmoid}(t) = \frac{1}{1 + e^{-t}}$$



# 로지스틱 회귀 계산



$$z = 1.3*w1 - 2.1*w2 + 2.9*w3 + b$$
$$y = \text{sigmoid}(z)$$
$$= 1/(1 + \exp(-z))$$

$w_1=w_2=w_3=b=1$  인 경우,

$$z = 3.1$$
$$y = 1/(1+\exp(-3.1))$$
$$= 0.957$$

- 로지스틱회귀는 선형회귀의 출력값을 0~1 사이의 값으로 바꾸어 준다.
- 신경망에서는 이런 함수를 활성화함수라고 한다 (activation function)
- 0~1의 의미 ➔ 확률
- 예를들어, 0에 가까우면 강아지, 1에 가까우면 고양이로 판단한다
- 로지스틱회귀의 목표도 결국  $w$ 와  $b$ 의 가중치를 알아내는 것이다

# 로지스틱회귀에 경사하강법 적용하기

- 비용함수 : 크로스 엔트로피 (Cross Entropy)

$$cross\_entropy = -y \cdot \log(\hat{y}) - (1 - y) \cdot \log(1 - \hat{y})$$

- y 는 0 아니면 1이다. 그러므로 두 항목중 하나만 살아남는다
- 예를 들어, 목표값이 1이고 출력값이 0.9 라면  
→  $cross\_entropy = -\log(0.9) = 0.105$  (항상 양수가 된다)

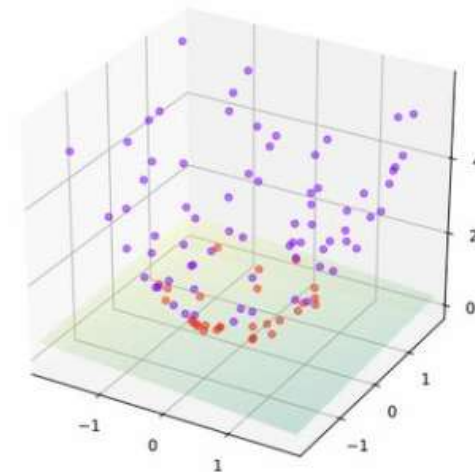
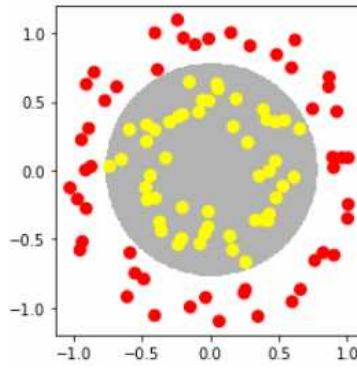
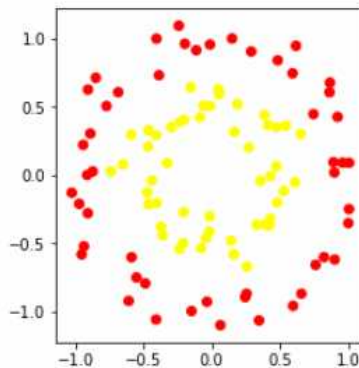
- 기울기 계산

$$기울기_j = -\frac{1}{N} \sum_i x_{ij} \cdot (y_i - \hat{y}_i)$$

- 신기하게도 선형회귀와 같은 결과가 나온다
- 시그모이드함수 + 크로스엔트로피 → 선형회귀 결과

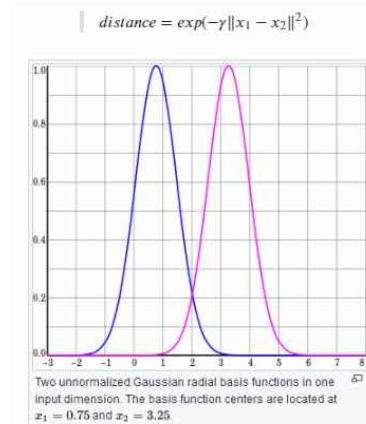
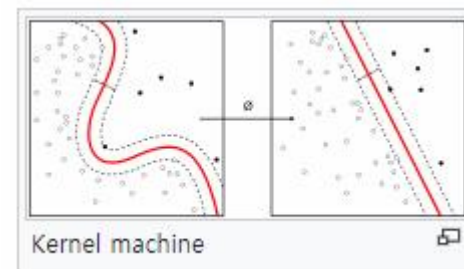
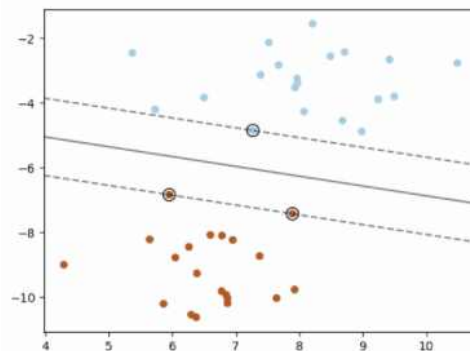
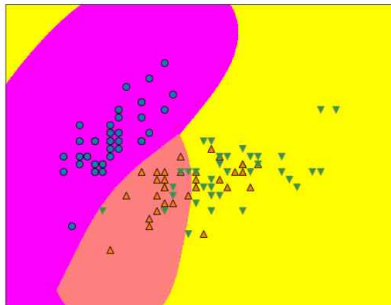
# 후덜덜 커널(kernel) 기법

- 커널 기법 : 속성을 늘려주거나 변환하는 수학적 방법
  - iris 데이터는  $150 \times 4$  이다. 이것을  $150 \times 8$  과 같이 속성을 늘려준다
  - 속성이  $x, y$  좌표일 때, 반지름과 각도로 변환한다
- 해결 사례
  - 원점을 중심으로 데이터가 방사형으로 분포했을 경우,  $(x, y)$  좌표를 반지름  $r$  ( $x^2 + y^2$ ) 으로 변환하면 아래와 같이 분류가 가능하다
  - 기존 속성  $(x, y)$  를  $(x, y, x^2, y^2)$  으로 속성을 늘리면 4차원 공간이 되면서, 고차원에서 분류가 쉽게 된다 ➔ 다항식 커널기법



# 수학적 머신러닝의 왕자 SVM

- SVM : Support Vector Machine
  - **커널 기법**을 극한으로 적용한 **분류** 알고리즘이다
  - 두 클래스 사이를 **매끄러운 곡선**으로 경계를 나누어준다
  - 서포트벡터 : 경계선에서 가장 가까운 샘플(점)들
    - ➔ 서포트벡터만 경계를 판단하는데 영향을 준다
  - 아래 그림과 같이 RBF 라는 종모양 함수를 적용하여 **속성을 늘림**
    - ➔ 각 샘플당 하나씩 종모양함수를 적용해 속성을 늘려줌
  - 결론적으로, 샘플당 속성을 하나씩 늘려줘서 **고차원 공간**에서 분류
  - 아주 수학적인 이론이므로, 관심있는 사람만 [en.wikipedia.org](http://en.wikipedia.org) 참고





# 꼭 잊어버리는 정규화

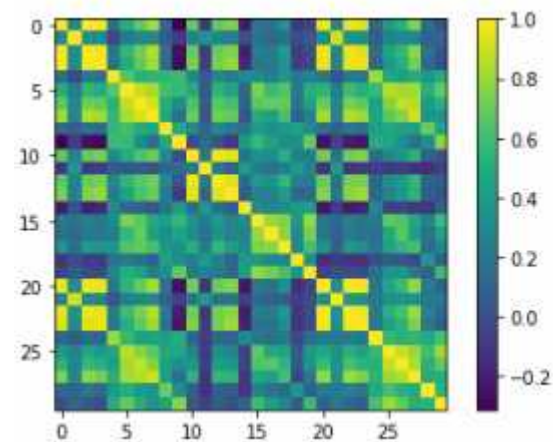
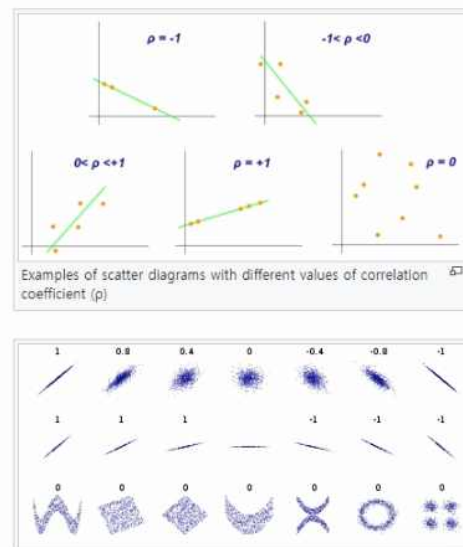
- 정규화 : 속성들의 값을 비슷하게 맞춰줌
  - 데이터를 사전에 가공하는 전처리(preprocessing) 방법중 하나
  - 대표적인 것이 평균을 0으로, 표준편차를 1로 맞추어 주는 것임
- 필요성
  - 키와 몸무게 데이터가 있다. (165, 53) 과 같을 것인데, 과연 단위가 무엇인가? 몸무게를 그램으로 하면 어떻게 될까? (165, 53000) 키를 키로미터 단위로 하면? (0.00165, 53)
  - 숫자의 절대값은 크게 의미가 없다. 무슨 단위(scale)를 적용하는 지에 따라 크게 변하기 때문이다
  - 그리고 속성간에 값의 크기가 다르면 수학적인 분석에 문제가 발생할 수 있다
- 대표적인 정규화 방법
  - 평균을 0, 표준편차를 1로 바꾼다 (표준정규화)
  - 최소값을 0, 최대값을 1로 바꾼다 (최대최소법)

# 두 속성의 관계 - 상관계수

- 상관계수 : Pearson correlation coefficient
  - 두 속성의 얼마나 가까운지를 수치로 나타냄 (sepal\_length vs sepal\_width)
  - -1~1 의 값을 가짐 (0 이면 상관관계가 전혀 없음)
  - 정확히 비례하면 1, 반비례면 -1

$$correlation = \frac{1}{N} \sum_i \frac{(x_i - \bar{x})(y_i - \bar{y})}{\sigma_x \sigma_y}$$

- 적용사례
  - 유방암 데이터와 같이 속성이 많은 경우, 하나의 그림으로 상관관계 표시



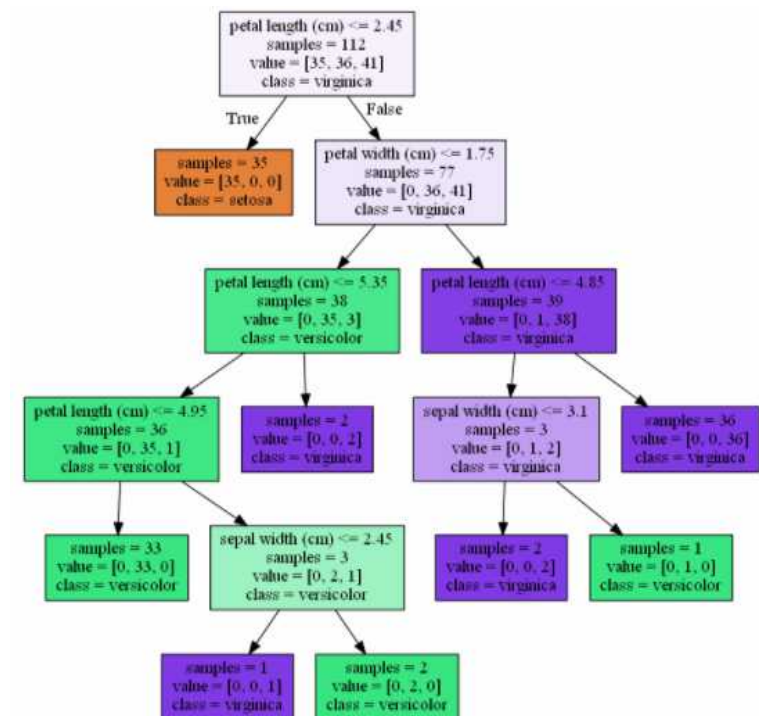
# 스무고개 - 결정 트리

- 결정 트리 : Decision Tree

- 스무고개 놀이 같이 한번에 하나의 질문을 던지고 왼쪽/오른쪽으로 가지를 쳐 나가서 분류하는 알고리즘
- 속성들 중에서 하나의 속성을 결정하고 기준값을 정한다
- 기준에 맞으면 왼쪽, 아니면 오른쪽으로 가지치기
- “엔트로피” 라는 개념을 이용함  
(엔트로피는 혼잡도 또는 복잡도임)  
→ 엔트로피 최소화가 목표

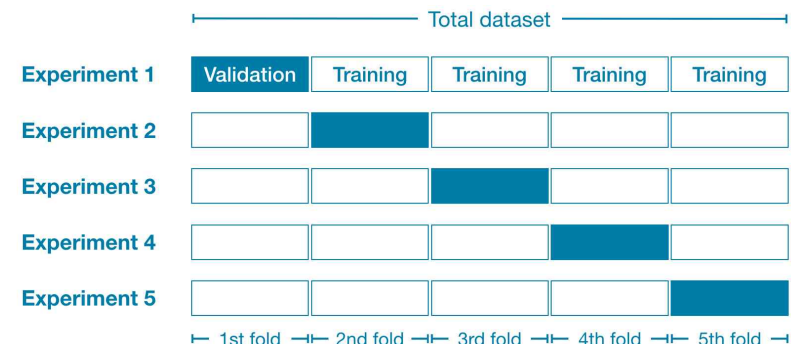
- 결정트리의 특징

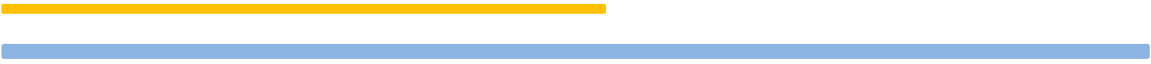
- 개념은 간단하나, 어떤 속성을 정할지 기준값을 얼마로 할지 결정이 아주 어려움
- 나무를 작게 만들수도 최대한 크게 만들수도 있는 유연성이 있다
- 이런 유연성 덕분에 나무를 하나가 아닌 여러개로 만들어 성능좋은 모델을 만듦  
(그래디언트 부스팅 - 최고성능의 모델)
- 정규화 전처리가 필요없다



# 공신력있는 교차검증

- 교차검증 : Cross Validation
  - 이전까지 생성된 모델의 **검증**을 위해 학습/테스트 데이터를 분리했다
  - 하지만 어떤 샘플이 학습데이터로 뽑히는가에 따라 결과가 달라졌다
  - 교차검증은 **원본데이터를 N개로 나누어** 그 중 하나를 테스트 데이터로 사용한다  
➔ 서브 모델을 N개 만들어 검증하게 됨
  - 교차검증을 사용하면 완벽하지는 않지만 어느정도 모델의 능력에 대해 공신력있게 평가할 수 있게 된다
- 교차검증 도구들
  - `cross_val_score()` : 섞지 않고 원본비율은 유지
  - KFold : 섞지 않고 원본비율도 유지하지 않음
  - StratifiedKFold : 섞지 않고 원본비율 유지
  - LeaveOneOut : 샘플 한개만 테스트데이터로 놓음 (샘플 갯수만큼 서브모델 생성)

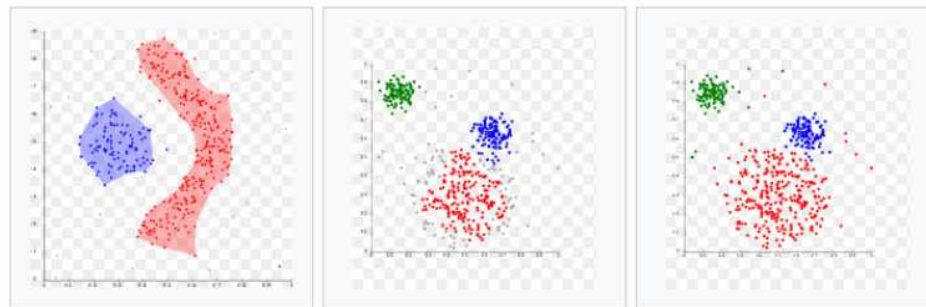




# 비지도 학습

# 비지도 학습 개념

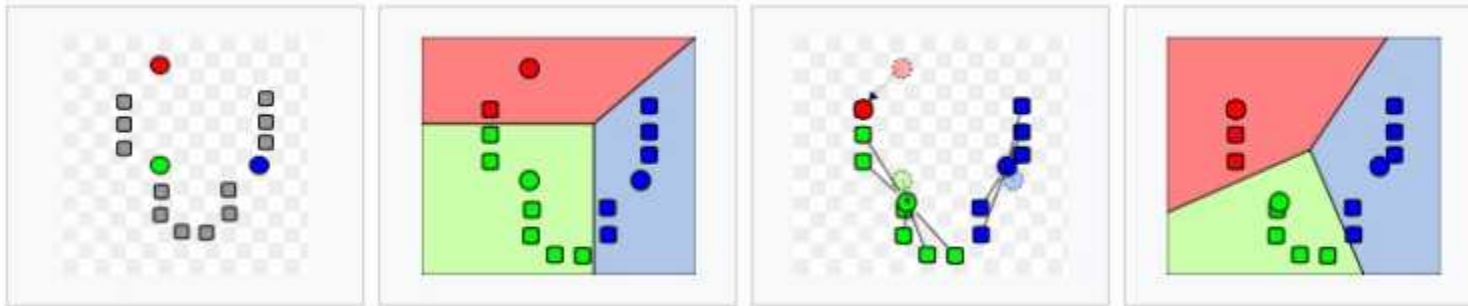
- 앞에서 배운 회귀와 분류는 지도학습(supervised learning) 이었다
  - 항상 샘플들 마다 이름표 또는 꼬리표가 붙어 있었다
  - 그러므로, 목표값을 잘 맞추도록 모델을 잘 만들면 되었다
- 하지만, 비지도학습은 **목표값이 없이 데이터만 있는** 문제이다
  - 어떻게 보면 답이 없는 문제일 수 있다
  - 기본적인 목표는 **비슷한 그룹**끼리 잘 나누려는 데 있다
  - 기타, 목표값을 이용하지 않고 데이터 자체의 속성을 분석하는 것도 비지도학습으로 부르기도 한다
- 사례
  - 100명의 학생들이 있을 때 어떻게 나눌 것인가? ➔ 남/녀, 성적순, 성격
  - 동물뼈가 대량으로 출토되었다. 어떻게 종류를 구분할 것인가?
  - 아이에게 과일을 많이 주었다. 아이는 과일들을 어떻게 구분하고 있을까? ➔ 맛있는것/맛없는것? 색깔별? 모양별?
  - 사람들은 어떻게 iris 꽃을 세가지 품종으로 구분했을까?



# 편짓기 기술 - 군집(Clustering)

- K-means 알고리즘

- 몇 그룹으로 나눌지를 사전에 지정 (K값)
- 처음 아무곳에나 K개의 중심점을 찍고, 차츰차츰 위치를 바꾸어나간다



- DBSCAN 알고리즘

- 뱀꼬리잡기 게임과 같이 가까이 있는 점들을 이어나간다
- 태평양에 흩어진 섬들을 군도로 묶어 주는 것과 비슷하다

