



## 6장 다양한 머신러닝 기법들 : 다항 회귀, 결정 트리, SVM

## 이장에서 배울 것들

- 직선이나 평면이 아닌 회귀 함수가 가능한가.
- 데이터의 특징에 따라 분류나 의사결정의 기준을 정할 수 있는가.
- 지금의 딥러닝 이전에 가장 강력한 기계 학습 방법은 어떤 것들이 있었나.
- 다양한 기계 학습 방법들은 어떤 장단점이 있나.

# 6.1 다항 회귀

- 선형회귀는 입력과 출력이 선형관계를 갖는 것으로 가정
- 실제 데이터는 아래와 같이 비선형 방정식을 따를 수도 있으며 이것을 **다항 회귀**polynomial regression라고 함

$$f(x) = y = \theta_2 x^2 + \theta_1 x + \theta_0$$

- 다변량(multivariate) 회귀를 할 때 다항(polynomial) 회귀를 이용하면 두 개의 특징  $x_1, x_2$ 를 가진 2차 다항식은 아래와 같은 함수가 될 것

$$f(x) = y = \theta_5 x_1^2 + \theta_4 x_2^2 + \theta_3 x_1 x_2 + \theta_2 x_1 + \theta_1 x_2 + \theta_0$$

- 함수는  $(x_1 + x_2 + 1)^2$ 로 얻어지는 다항식의 계수를 변경하는 것과 같음
- $n$ 개의 특징을 가진  $X = (x_1, x_2, \dots, x_n)$ 를 입력으로 하는  $d$ 차 다항식 다변량 회귀라는 것은 입력을 그대로 사용하지 않고  $(x_1 + x_2 + \dots + x_n + 1)^d$ 에서 나타나는 항들을 사용하는 것
  - 데이터 정제로 간주할 수 있는데 사이킷런의 preprocessing 서브 모듈에 있는 PolynomialFeatures 클래스를 사용함
  - 클래스를 생성할 때 차수를 지정하고, 입력을 fit\_transform() 메소드에 넘기면 다항 회귀에 필요한 형태로 변환하며 이것을 다항 특징 변환(polynomial feature transformation)이라고 함



```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

data_loc = 'https://github.com/dknife/ML/raw/main/data/'
life = pd.read_csv(data_loc + 'life_expectancy.csv')
```

```
life.head()

life = life[['Life expectancy', 'Alcohol', 'Percentage expenditure',
             'Polio', 'BMI', 'GDP', 'Thinness 1-19 years']]
life.dropna(inplace = True)
X = life[['Alcohol', 'Percentage expenditure', 'Polio',
          'BMI', 'GDP', 'Thinness 1-19 years']]
y = life['Life expectancy']
```

- preprocessing 서브 모듈의 PolynomialFeatures 클래스를 활용해 입력 데이터를 다항 회귀에 사용



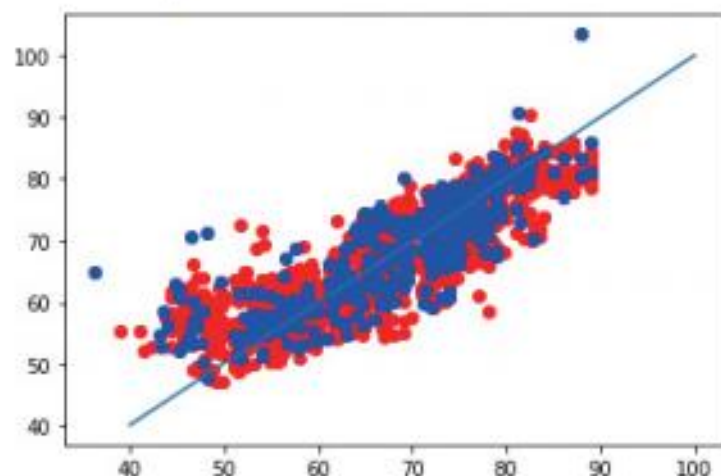
```
from sklearn.preprocessing import PolynomialFeatures  
poly_feature = PolynomialFeatures(degree = 3)  
X = poly_feature.fit_transform(X)
```



```
X_train,X_test,y_train,y_test = train_test_split(X, y, test_size = 0.2)
lin_model = LinearRegression()
lin_model.fit(X_train, y_train)


y_hat_train = lin_model.predict(X_train)
y_hat_test = lin_model.predict(X_test)
plt.scatter(y_train, y_hat_train, color='r')
plt.scatter(y_test, y_hat_test, color='b')
plt.plot([40, 100], [40, 100])
print('Mean squared error:', mean_squared_error(y_test, y_hat_test))
```

Mean squared error: 26.166291726402985



## 6.2 다항 회귀의 문제점 - 과적합, 그리고 폭발적인 복잡도 증가

- 다항 회귀를 통해 더 좋은 예측 성능을 가진 회귀 함수를 찾을 수 있음
- 선형 회귀를 사용한 것보다 언제나 다항 회귀를 사용하는 것이 옳은 것이 라고 생각할 수 있지만 실제로는 그렇지 않음
- 찾은 회귀 함수가 학습을 수행한 훈련용 데이터에 대해서는 어느 정도 오차를 가지는지 확인
  - 22 정도의 값으로 검증용 데이터에 비해서 크게 낮은 값을 가짐

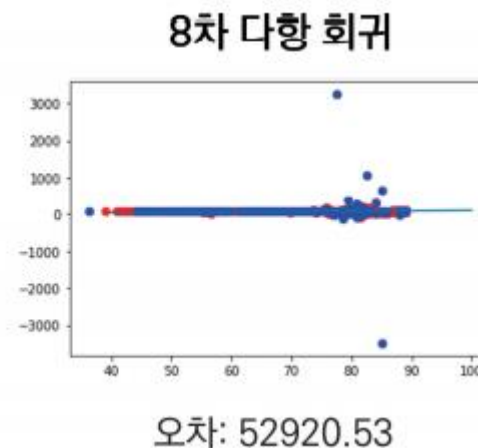
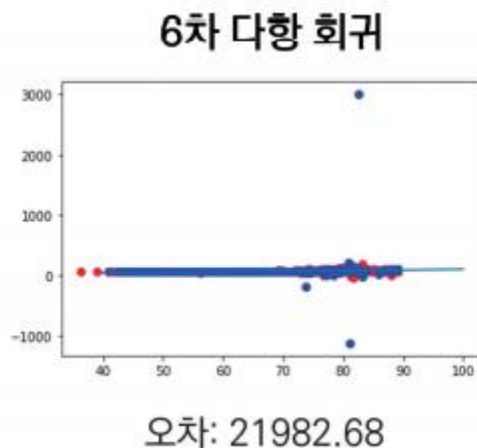
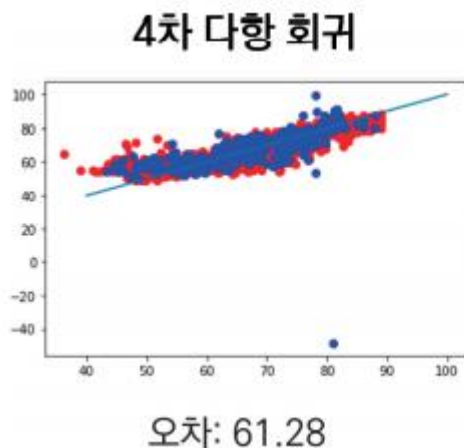


```
print('Mean squared error:', mean_squared_error(y_train, y_hat_train))
```

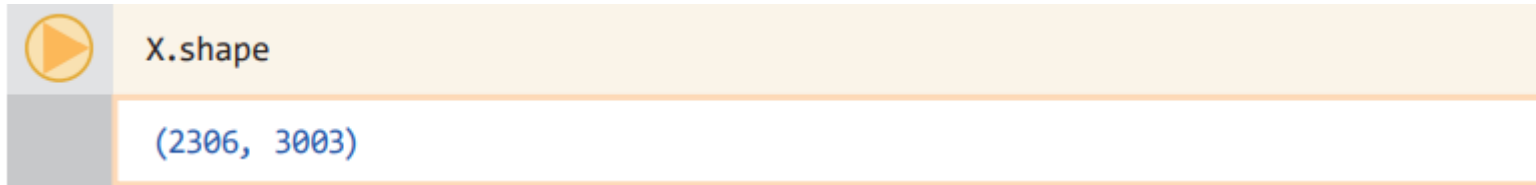
```
Mean squared error: 22.33210976632871
```



- 훈련용 데이터를 잘 설명하도록 함수의 모양을 바꾸어 놓아도 검증용 데이터에 대한 예측 능력이 그만큼 좋아지지 않을 수 있다는 것
- 훈련용 데이터에만 지나치게 맞춰진 **과적합**overfitting으로, 일반화 능력이 떨어지고 새로운 데이터에 대해서는 잘 동작하지 않기 때문



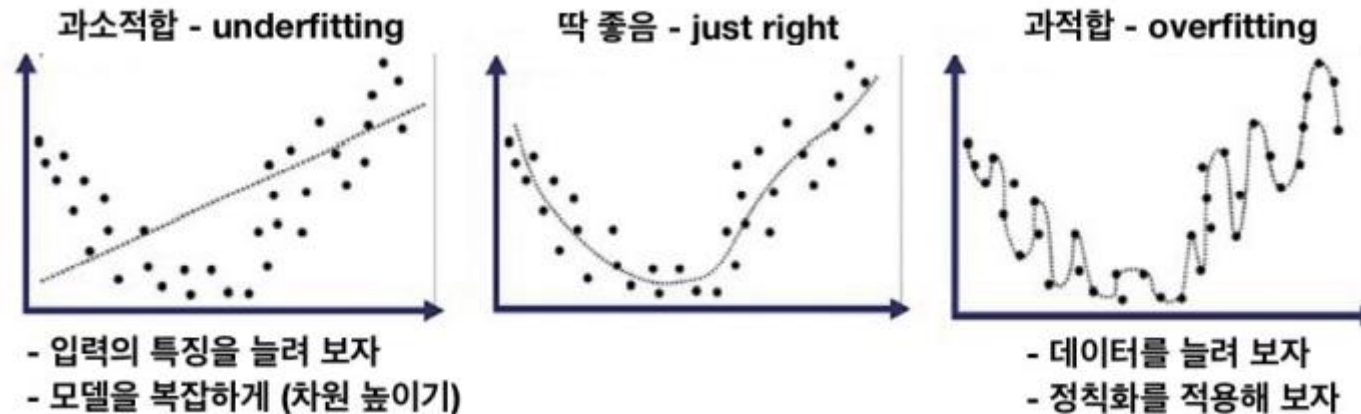
- 다른 문제점은 다항식의 차수가 높아질수록 입력의 크기가 폭발적으로 증가한다는 것
- 기대 수명 예측에서 선형 회귀에는 6개의 입력 속성이 사용되었지만, 8차 다항식을 사용하면 입력의 크기가 매우 커짐



- 차수를 높이는 것은 과적합의 문제와 함께 급격히 커지는 계산 복잡도의 문제를 갖게 됨

## 6.3 과적합과 과소적합 - 공짜 점심은 없다

- **과적합** *overfitting*이란 학습에 사용된 데이터에 대해서는 좋은 성능을 보이지만, 새로운 데이터에 대해 적용하면 성능이 잘 나오지 않는, 즉 일반화 능력이 떨어지는 모델이 만들어지는 것
- 학습 데이터를 설명하는 함수를 제대로 찾지 못하여 학습 데이터와 다른 데이터 모두에 대해 예측을 제대로 하지 못하는 모델을 만드는 일은 **과소적합** *underfitting*이라고 함



- 과소적합은 모델이 지나치게 단순하거나 예측을 제대로 할 수 없는 특징들만 제공된 경우에 발생하므로 입력 데이터의 특징을 늘려 새로 학습하거나, 학습 모델이 가진 차수<sup>degree</sup> 등의 복잡도를 높이는 방법으로 문제를 해결함
- 과적합은 이와 달리 특징을 줄이거나, 데이터를 늘리고, 모델을 더 단순하게 만들어 피해야 함
  - 과적합과 관련한 월퍼트<sup>Wolpert</sup>의 유명한 수학 정리 "공짜 점심은 없다"
  - 잘 학습된 모델이라도 이 모델을 무력화할 데이터가 존재한다는 것

- 과적합의 원인
  - 특이한 경우만 추출되었거나, 지나치게 적은 데이터 표본
  - 주어진 데이터를 빠짐 없이 설명하려는 욕심에 너무 복잡한 모델을 사용하는 것
- 모델의 학습을 오히려 방해하는 방식으로 일반화 능력을 높이고 그것을 **정칙화** regularization 기법이라고 함



# LAB<sup>6-1</sup> 다항 회귀의 회귀 함수를 그려 보자

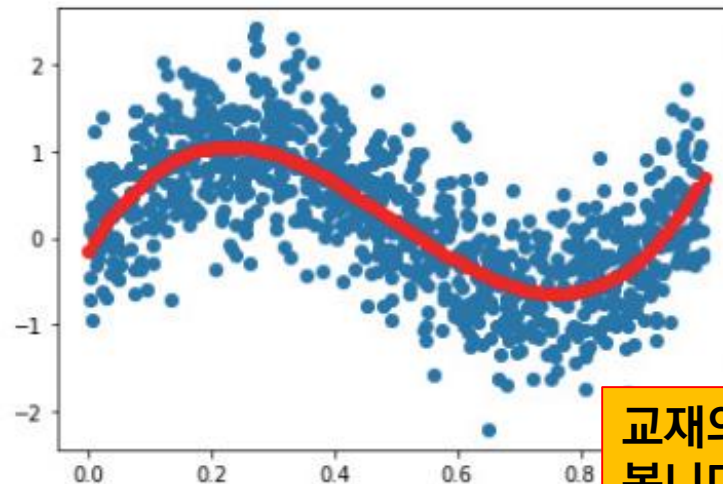
## 실습 목표

아래의 위치에는 직선으로 표현하기 어려운 데이터가 있다.

<https://github.com/dknife/ML/raw/main/data/nonlinear.csv>

이 데이터를 화면에 그려보고 이 데이터를 설명하는 회귀 함수를 다항 회귀를 이용하여 찾아 보자. 그리고 회귀 함수를 데이터의 독립 변수 범위 내에서 가시화해 보라.

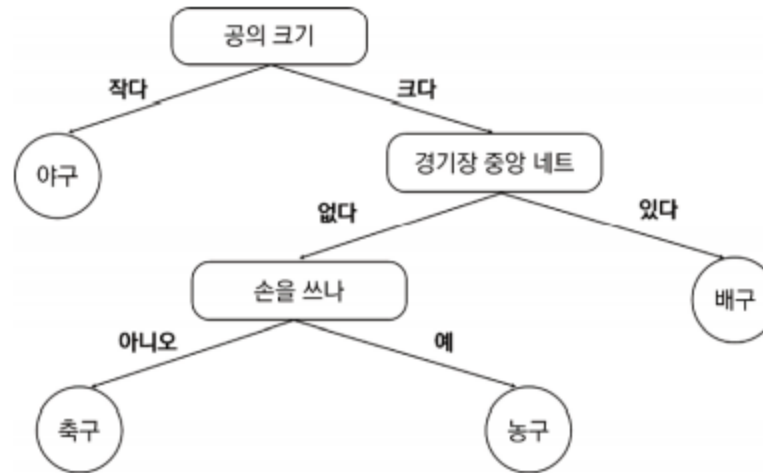
## 원하는 결과



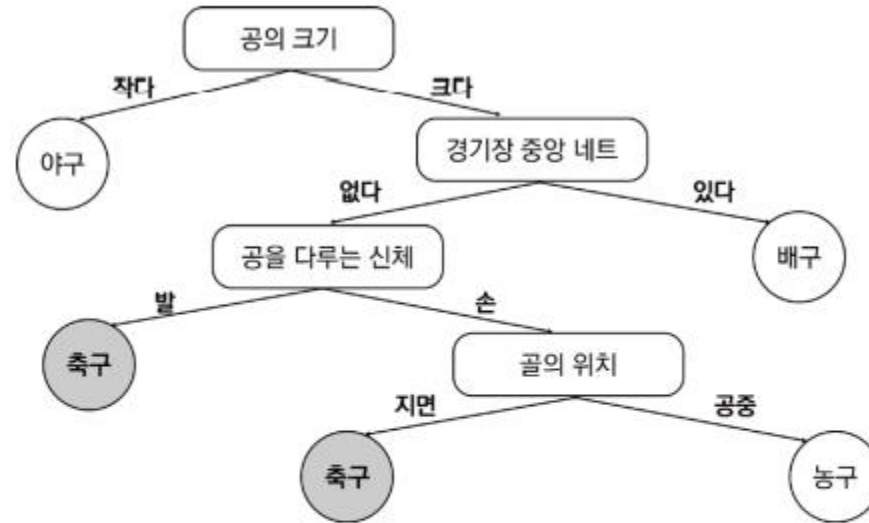
교재의 해답 코드를 하나하나 단계별로 수행해 봅니다

## 6.4 결정 트리를 이용한 분류

- 결정 트리<sup>decision tree</sup>는 귀납 추론을 위해 자주 사용되는 실용적인 방법
- 데이터들을 트리 구조의 루트<sup>root</sup>에서 시작하여 차례로 중간 노드<sup>internal node</sup>들을 거쳐 단말 노드<sup>leaf node</sup>에 배정하는 기능을 수행



- 축구는 하나가 아니라 두 개의 단말 노드에 나타나게 되며, 축구 종목의 판정은 논리 곱의 논리합  
disjunction of conjunction으로 표현



축구로 판단되는 조건

$$(ball = 크다 \wedge net = 없다 \wedge hand = X)$$

$$\vee (ball = 크다 \wedge net = 없다 \wedge hand = 0 \wedge goal = 지면)$$



- 기계학습이라는 것은 데이터를 경험으로 삼아 알고리즘을 찾는 일
- 기계 학습의 결정 트리에서 다루는 것은 데이터가 주어졌을 때, 데이터의 특징들을 보고 앞에서 본 것과 같은 트리를 찾아내는 일

*(ball, net, hand, goal)* – 종목

---

*(작다, 없다, 0, 없다)* – 야구

*(크다, 없다, 0, 지면)* – 축구

*(크다, 없다, X, 지면)* – 축구

*(크다, 있다, 0, 없다)* – 배구

*(크다, 없다, 0, 공중)* – 농구

## 6.5 어떤 속성이 가장 중요한가?

- 데이터는 여러 가지 속성 중에서 어떤 것이 가장 중요한 것인지 판단하기 위해 **정보 이득** information gain이라는 개념이 사용
- 특정한 속성이 원하는 분류 방식에 부합하게 데이터를 나누는지를 측정할 수 있는 척도

- 이를 이해하기 위해서는 우선 엔트로피<sup>entropy</sup>라는 개념을 살펴볼 필요
- 데이터를 두 부류로 나눌 수 있다고 하자. 한 부류를 양성, 다른 부류를 음성이라고 하면 각각의 데이터는 양성 혹은 음성 중에 하나가 될 것
- 전체 데이터 표본  $S$ 에서 차지하는 양성 데이터의 비율을  $p^+$ , 음성인 데이터 비율을  $p^-$ 라고 하면 이 데이터 표본의 엔트로피는 다음과 같음
  - $\lg$  는 밑이 2인 로그, 즉  $\log_2$ 를 의미

$$H(S) = -p^+ \lg p^+ - p^- \lg p^-$$

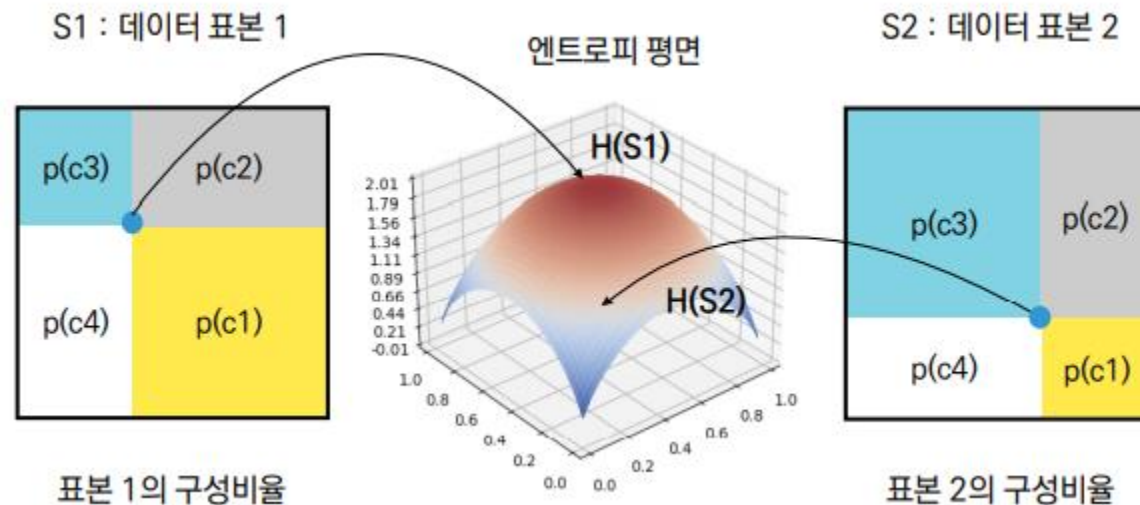
- 엔트로피는 원래 정보량을 측정하기 위해 고안된 것으로, 이 값이 크면 많은 정보가 담겨 있다는 것

- 가장 엔트로피가 높은 경우는 두 부류가 균등하게 섞여 있는 경우로 각각의 비율이 1/2인 경우

$$H(p^+ = 0.5, p^- = 0.5) = -\frac{1}{2} \lg 2^{-1} - \frac{1}{2} \lg 2^{-1} = 1$$

- 만약 데이터 표본 S에 속한 데이터들이 두 개의 부류가 아니고 C라는 클래스 집합의 각 클래스로 나뉘어질 수 있다고 하면, 이 데이터 표본 S의 엔트로피는 다음과 같이 구함

$$H(S) = \sum_{c \in C} p(c) \lg p(c)$$



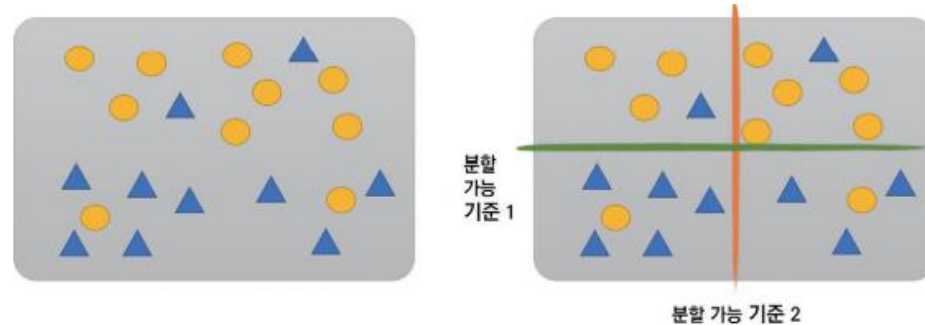
- 정보 이득이라는 것은 특정한 속성에 따라 데이터를 나누었을 때 줄어드는 엔트로피로 정의
- 어떤 속성  $A$ 가 가지는 모든 값들의 집합을  $A$ 라고 할 때, 데이터 표본  $S$ 를 나누어서 얻는 정보 이득  $G(S, A)$ 은 다음과 같음

$$G(S, A) = H(S) - \sum_{a \in A} \frac{|S_a|}{|S|} H(S_a)$$

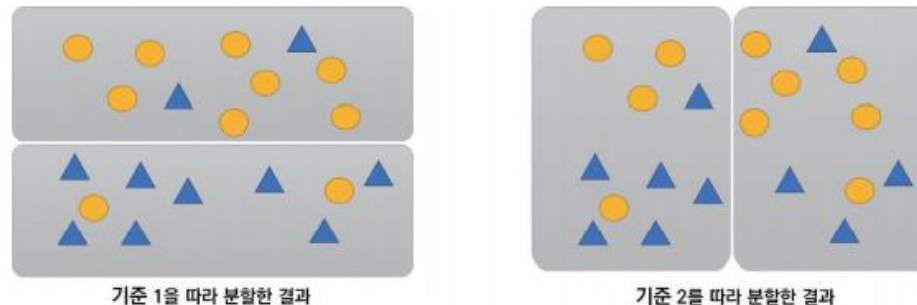
- $S_a$ 는  $A$  속성의 값이  $a$ 인  $S$ 의 부분집합
- 우리가 가진 데이터 표본  $S$ 에 있는 데이터를 쪼개려고 할 때 가장 중요한 기준으로 삼을 수 있는 속성은 이 정보 이득이 가장 큰 속성
- 이것을 찾아 결정 트리를 만들어 나가는 방법이 ID3 알고리즘

## 6.6 결정 트리 분할의 기준 - 정보량과 불순도

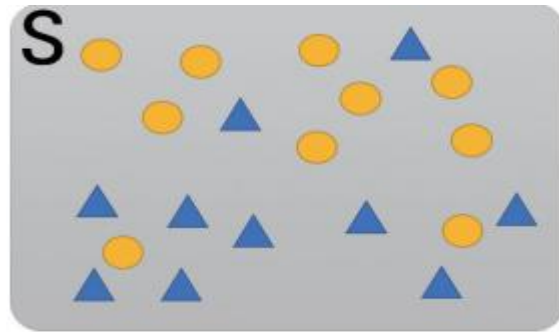
- 결정 트리를 만들어 나가는 것은 현재 하나의 묶음로 되어 있는 그룹을 더 세분화된 그룹으로 나눌 것인지, 나눈다면 어떤 기준으로 나눌 것인지를 판단하는 일
- 대표적인 결정 트리 알고리즘인 ID3는 정보 이득, CART 알고리즘은 불순도 개념을 사용



각각의 기준을 적용하면 아래 그림과 같이 서로 다른 방식으로 분할된다.



- 엔트로피 개념을 통해 섞여 있는 데이터들의 정보량이 얼마인지를 측정
- 원래의 노드 S에 속한 20개의 데이터는 10개의 삼각형과 10개의 동그라미를 가지고 있다.
- 묶음의 정보량은 엔트로피를 이용하여 다음 그림과 같이 계산



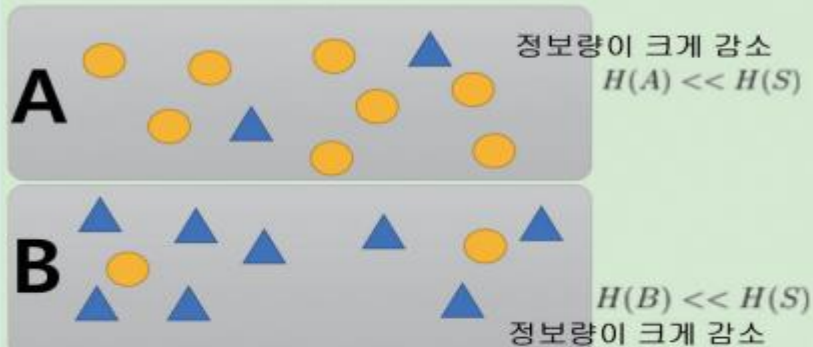
정보량 = 엔트로피

$$H(S) = - \left( \frac{1}{2} \lg \frac{1}{2} + \frac{1}{2} \lg \frac{1}{2} \right) = 1$$

- 좋은 분할이라면 데이터가 더 잘 나누어져야 함
  - 분할된 노드 각각에 한 종류의 데이터 비율이 높아져서 정보량이 줄어들어야 한다는 것.
  - 원래의 정보량에서 분할 후의 정보량을 뺀 값이 커질수록 좋다는 것이며 이것을 **정보이득** *information gain*이라고 정의

정보량의 감소(정보이득)를 이용하여 좋은 분할을 찾는 방법: 정보이득 = 원래의 정보량 - 분할 후의 정보량

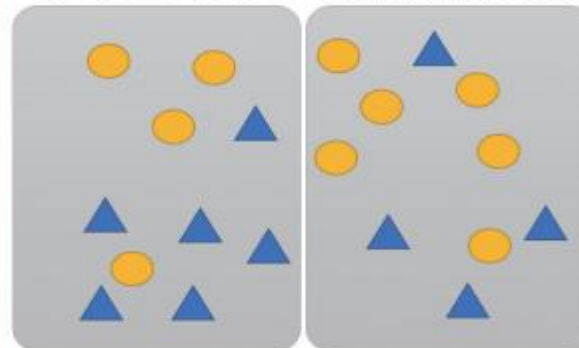
정보량은 같은 종류의 데이터가 많을수록 감소



기준 1을 따라 분할한 결과

정보량이 조금 감소

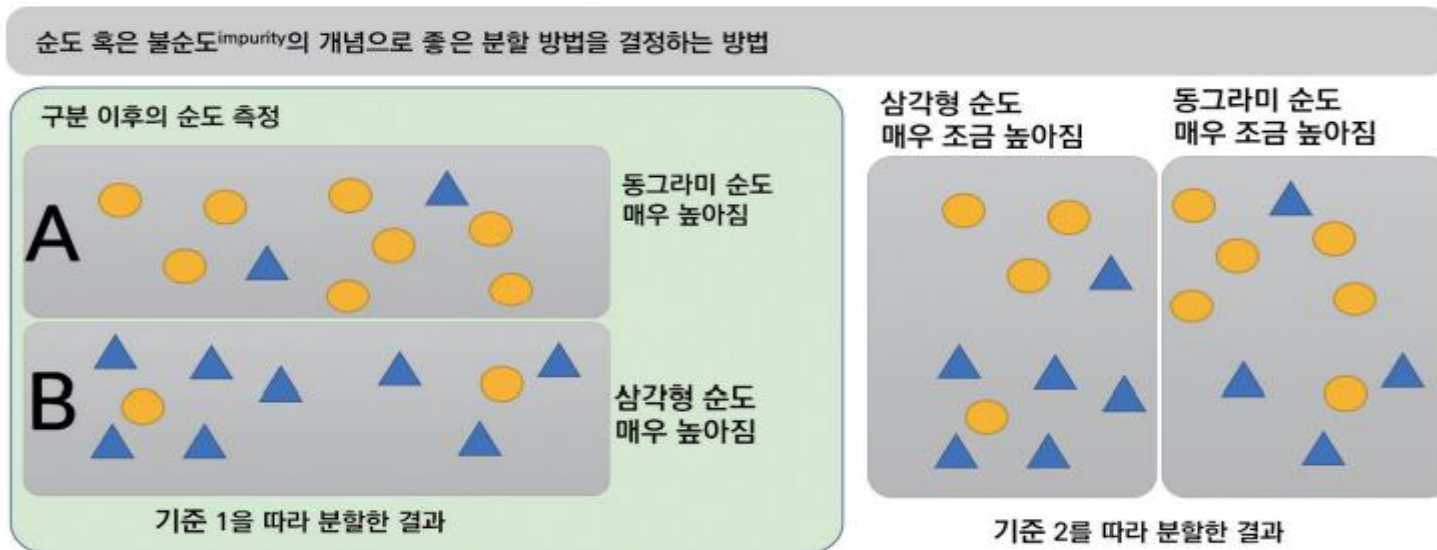
정보량이 조금 감소



기준 2를 따라 분할한 결과



- 엔트로피가 아닌 데이터의 순도<sup>purity</sup> 혹은 불순도<sup>impurity</sup> 개념으로 설명할 수도 있음
- 두 가지 분할 결과를 살펴보면 기준 1의 경우 그룹 A와 B에서 각각 동그라미와 삼각형의 순도가 매우 높아짐



## 6.7 결정 트리를 손으로 만들어 보자 - ID3 알고리즘

- 데이터가 주어졌을 때 ID3 알고리즘이 어떻게 결정 트리를 만들어 내는지 정보 이득을 직접 계산하며 살펴 보자

데이터 표본 S	
특성	-(ball, net, hand, goal) - 종목
-----	
데이터 1	-(작다, 없다, 0, 없다) - 야구
데이터 2	-(크다, 없다, 0, 지면) - 축구
데이터 3	-(크다, 없다, X, 지면) - 축구
데이터 4	-(크다, 있다, 0, 없다) - 배구
데이터 5	-(크다, 없다, 0, 공중) - 농구

- 야구, 배구, 농구의 비율은 1/5이고, 축구는 2/5
  - 데이터 표본의 엔트로피를 측정하면 다음과 같음

$$H(S) = -\frac{2}{5} \lg \frac{2}{5} - 3 \cdot \frac{1}{5} \lg \frac{1}{5}$$

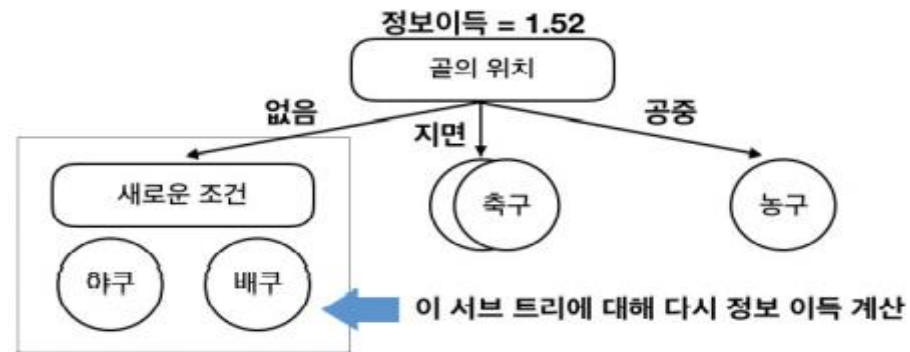
- 각 속성별 정보이득을 계산하여 어떤 속성이 가장 데이터를 잘 나누는지 확인
  - $G(S, \text{net})$ 과  $G(S, \text{ball})$ 은 같은 값

$$G(S, \text{ball}) = H(S) - \frac{4}{5} \left( -\frac{2}{4} \lg \frac{2}{4} - 2 \cdot \frac{1}{4} \lg \frac{1}{4} \right) = 1.92 - 1.20 = 0.72$$

$$G(S, \text{hand}) = H(S) - \frac{4}{5} \left( -4 \cdot \frac{1}{4} \lg \frac{1}{4} \right) = 1.92 - 1.60 = 0.32$$

$$G(S, \text{goal}) = H(S) - \frac{2}{5} \left( -2 \cdot \frac{1}{2} \lg \frac{1}{2} \right) = 1.92 - 0.4 = 1.52$$

- 중요한 속성은 goal이라고 할 수 있음
- 결정 트리의 루트 노드는 속성 goal에 따라 데이터를 구분
  - 아직 더 세분화가 필요한 서브 트리에 대해서 재귀적으로 정보 이득을 계산

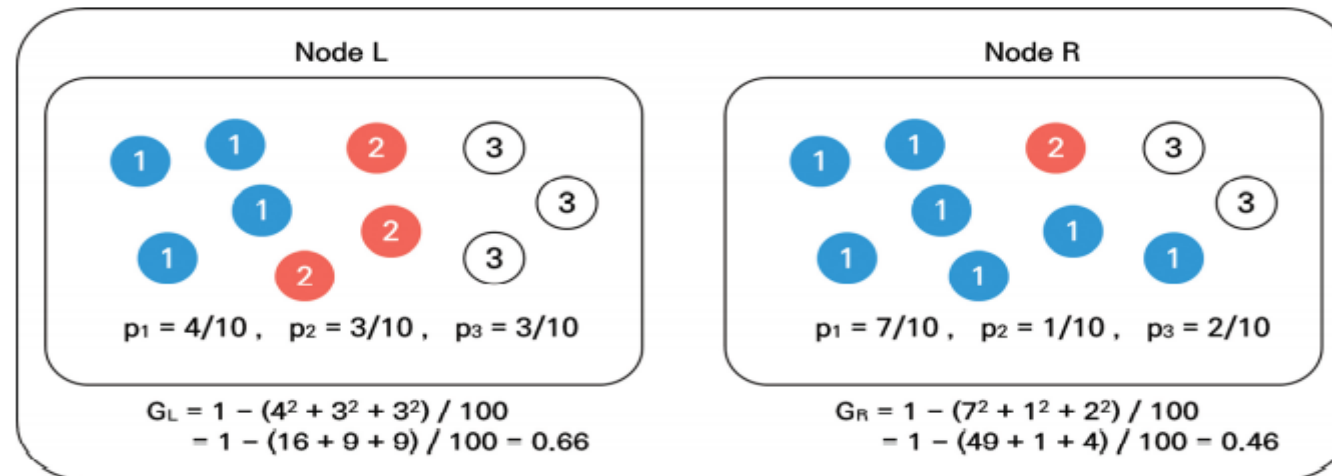


## 6.8 지니 불순도를 이용한 효율적인 평가 – CART 알고리즘

- ID3 알고리즘은 엔트로피에 기반한 정보 이득 개념을 사용하는데, 이 개념은 다소 복잡한 수식과 계산을 요구하기에 이를 피하기 위해 많이 사용하는 척도가 지니 불순도
- 지니 불순도는 엔트로피의 대체 척도라고 볼 수 있는데, 하나의 그룹 내에 섞여 있는  $n$ 개 종류의 레이블이 있고, 레이블이  $i$ 인 객체 비율이  $p_i$  라고 할 때 다음과 같은 값을 가짐

$$G = 1 - \sum_{i=1}^n p_i^2$$

- 어떤 그룹의 데이터를 두 개로 쪼갤 때 이 불순도를 최대한 낮추는 속성과 해당 속성의 값을 찾아 트리를 만들어 나가면 ID3처럼 결정 트리를 만들 수 있음



- 지니 불순도를 이용하여 결정 트리를 만드는 방법이 **CART 알고리즘**
  - CART는 **분류와 회귀 트리** classification and regression tree의 약자
  - 현재 데이터가 섞여 있는 노드를 두 개의 노드로 나눌 때 어떤 속성  $A$ 와 해당 속성에 어떤 값  $a$ 를 기준으로 쪼갤 것인지를 찾는 문제
- 쪼개어서 얻는 두 노드  $L$ 과  $R$ 이라고 하고, 각각의 원소 개수를  $m_L, m_R$ 이라고 하고 쪼개어지기 전 노드의 원소 개수는  $m$
- 알고리즘이 하는 일은 불순도  $G_L$ 과  $G_R$ 로 결정되는 다음 비용함수  $J(A, a)$ 가 최소가 되게 하는  $A$ 와  $a$ 를 선택하는 것

$$\operatorname{argmin}_{A,a} J(A, a) = \operatorname{argmin}_{A,a} \left( \frac{m_L}{m} G_L + \frac{m_R}{m} G_R \right)$$

## 6.9 사이킷런의 결정 트리로 붓꽃 분류하기

- 사이킷런 패키지는 우리가 살펴본 결정 트리를 손쉽게 활용할 수 있는 방법을 제공
  - 분류classification를 학습하면서 다루었던 붓꽃iris 데이터를 이용하여 결정 트리를 만들어 보자.

```
▶ from sklearn.datasets import load_iris  
iris = load_iris()  
X, y = iris.data, iris.target
```

```
[ ] from sklearn.tree import DecisionTreeClassifier  
dec_tree = DecisionTreeClassifier(max_depth=3)  
dec_tree.fit(X, y)
```

```
▼ DecisionTreeClassifier  
DecisionTreeClassifier(max_depth=3)
```



- 사이킷런의 tree 서브 모듈에는 export\_graphviz라는 함수가 있는데, 이를 dec\_tree의 out\_file 인자에 지정된 파일 이름으로 트리 정보를 가지는 dot 파일을 내보낼 수 있음
  - 노드를 쪼갤 때 사용되는 속성의 이름을 feature\_names 인자에 알려줌
  - iris 데이터에는 feature\_names로 저장

```
▶ from sklearn.tree import export_graphviz
  export_graphviz(
    dec_tree,
    out_file=("./dec_tree_for_iris.dot"),
    feature_names=iris.feature_names,
  )

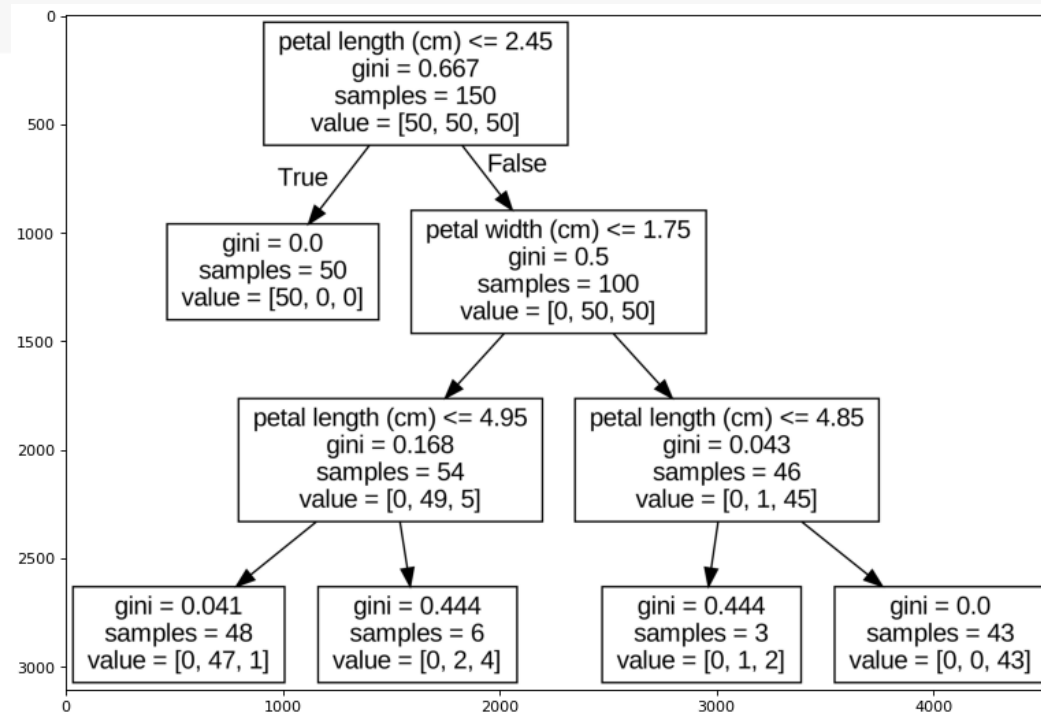
  # Convert to png using system command (requires Graphviz)
  from subprocess import call
  call(['dot', '-Tpng', 'dec_tree_for_iris.dot', '-o', 'dec_tree_for_iris.png', '-Gdpi=600'])
  !ls
```

```
➞ dec_tree_for_iris.dot  dec_tree_for_iris.png  sample_data
```

- dot 파일을 jpg나 png 파일로 변환할 수 있음



```
import matplotlib.pyplot as plt
dec_tree_img = plt.imread('./dec_tree_for_iris.png')
plt.figure(num=None, figsize=(12, 8), dpi=80,
           facecolor='w', edgecolor='k')
plt.imshow(dec_tree_img)
```

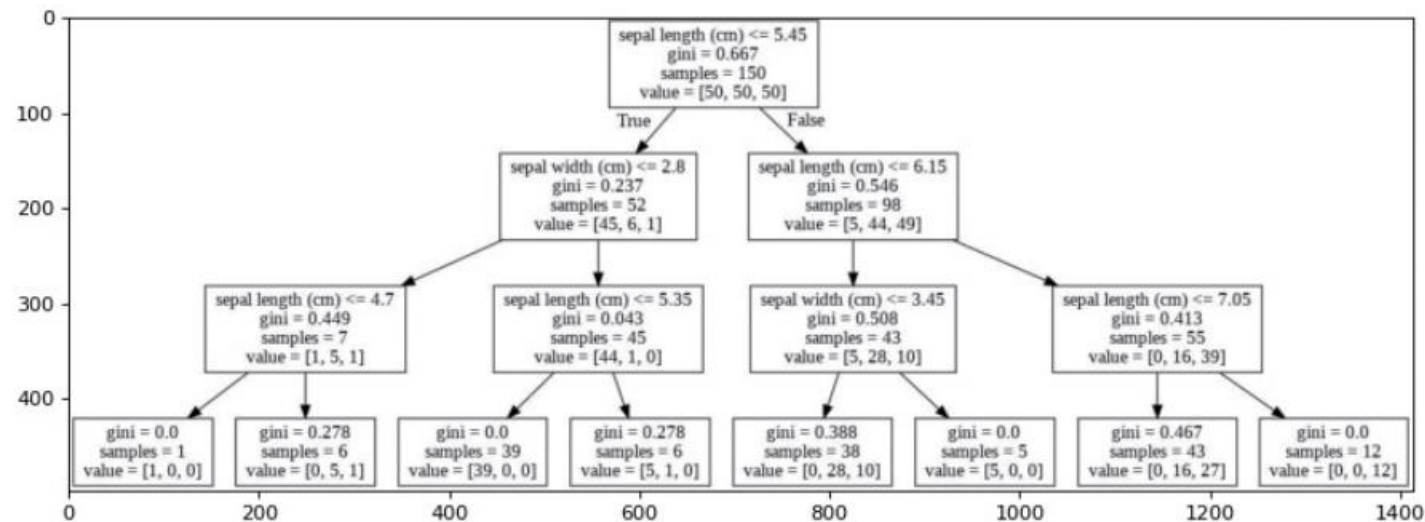


# LAB<sup>6-2</sup> 꽃받침의 너비와 길이로 결정트리를 만들자

## 실습 목표

붓꽃 데이터를 그대로 사용하여 결정 트리를 만들었더니, 꽃잎<sup>petal</sup>의 길이와 너비만이 분류 기준으로 사용되고 있다. 꽃받침<sup>sepal</sup>의 너비와 길이만 가지고 결정 트리를 만들어 보고, 전체 속성을 사용했을 때 왜 이 기준이 선택되지 않았는지 생각해 보자.

## 원하는 결과



교재의 해답 코드를 하나하나 단계별로 수행해 봅니다

## LAB<sup>6-2</sup> 꽃받침의 너비와 길이로 결정트리를 만들자

```
[25] iris = load_iris()
X, y = iris.data, iris.target
X = X[:, 0:2]
dec_tree = DecisionTreeClassifier(max_depth=3)
dec_tree.fit(X, y)

export_graphviz(
    dec_tree,
    out_file='./dec_tree_for_iris.dot',
    feature_names=iris.feature_names[0:2],
)

# Convert to png using system command (requires Graphviz)
from subprocess import call
call(['dot', '-Tpng', 'dec_tree_for_iris.dot', '-o', 'dec_tree_for_iris.png', '-Gdpi=600'])
!ls

dec_tree_img = plt.imread('./dec_tree_for_iris.png')
plt.figure(num=None, figsize=(12, 8), dpi=80,
           facecolor='w', edgecolor='r')
plt.imshow(dec_tree_img)
```

# LAB<sup>6-3</sup> 엔트로피를 이용하여 결정 트리 만들기

## 실습 목표

CART 알고리즘은 노드를 분할할 때 지니 불순도에 기반한 비용함수 뿐만 아니라 엔트로피를 이용한 비용함수를 사용할 수 있다고 하였다. 엔트로피를 사용하고, 모든 속성을 다 썼을 때에 어떤 결정 트리가 생성되는지 확인해 보자.

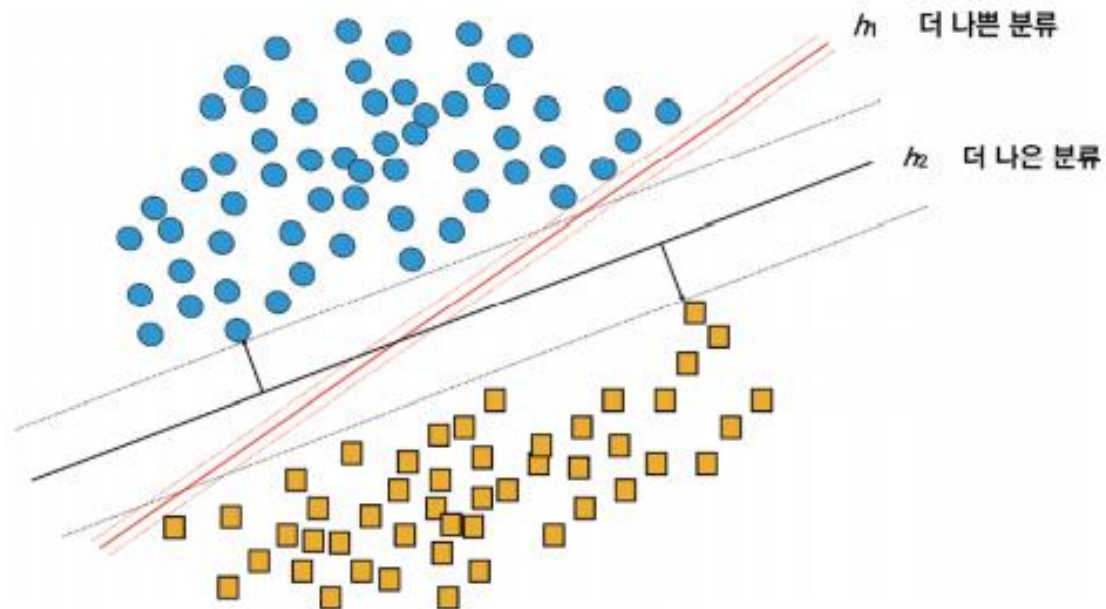


엔트로피를 이용한 노드 분할을 하기 위해서는 결정 트리를 만들 때 생성자에 `criterion = 'entropy'` 인자를 지정하면 된다.

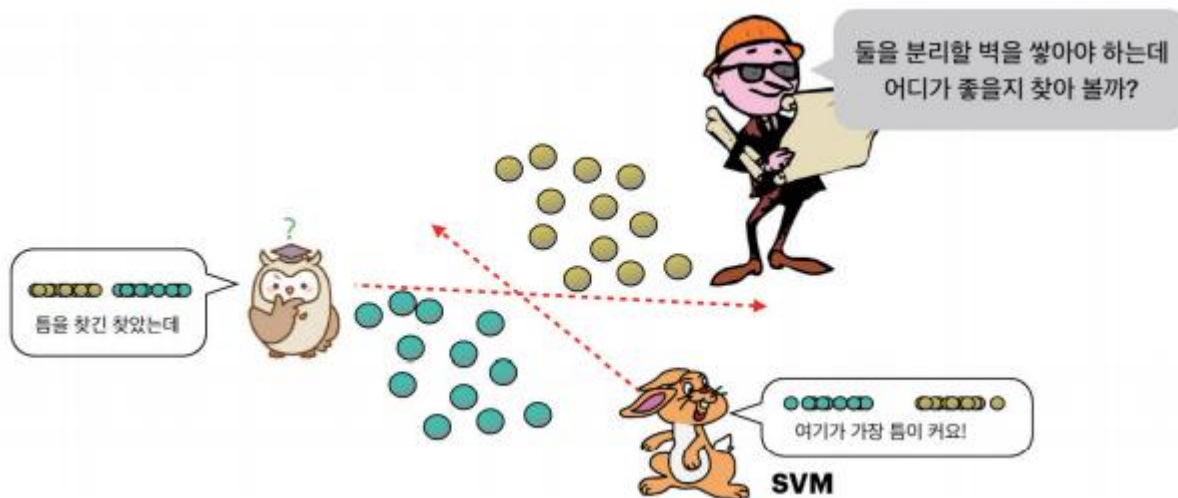
교재의 해답 코드를 하나하나 단계별로 수행해 봅니다

## 6.10 SVM – 서포트 벡터 머신의 소개

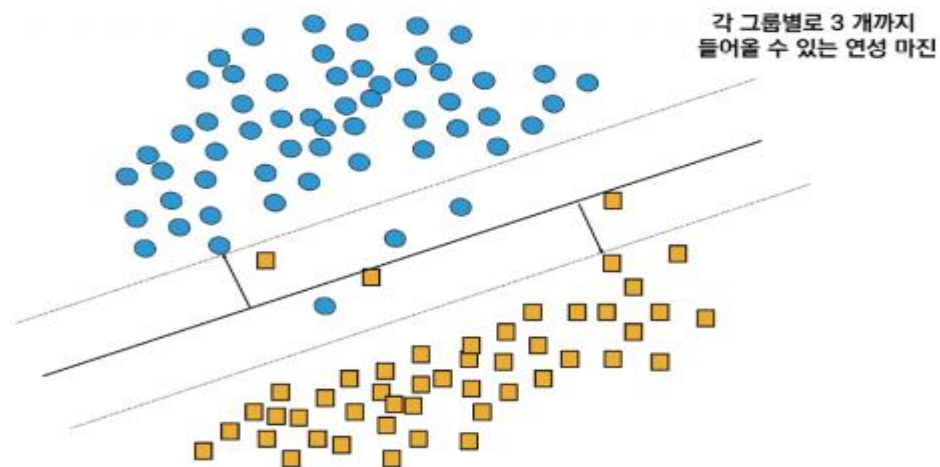
- 서포트 벡터 머신(support vector machine)은 줄여서 SVM
- 인공 신경망이 딥러닝(deep learning)을 통해 인공지능 분야의 중심으로 떠오르기 전에 가장 각광받던 학습 방법 의 하나
- 파란색 원과 노란색 사각형은 서로 다른 그룹에 속한 데이터들이다. 이들을 구분하는 초평면(hyperplane)은 여러 개 존재
  - 여기서는  $h_1$  과  $h_2$  라는 두 개의 직선으로 표현
- 좋은 분리 평면은 새로운 데이터가 들어왔을 때에도 판정을 잘 할 수 있는 평면
- 초평면을 화살표로 표시된 법선(normal) 벡터 방향으로 움직였을 때 데이터에 닿지 않는 폭이 넓을 수록 좋을 것



- SVM은 두 데이터 그룹을 나누는 초평면을 찾으면서 이 폭이 가장 넓은 것을 찾는 방법
  - 이 폭을 **마진**margin이라고 부름
  - 그림에서 볼 수 있는 것처럼 어떠한 데이터도 이 마진 내에 들어오지 않을 경우 마진을 **하드 마진**hard margin이라고 부름



- 마진 안에 아무런 데이터도 들어오지 않도록 하는 것이 불가능하거나 어떤 데이터는 잡음에 가까워 무시하는 것이 좋을 수도 있음
- 일부 데이터가 마진 내에 들어오도록 허용하면서 분리 평면을 찾을 수 있을 경우 **소프트 마진** soft margin 이라고 부름



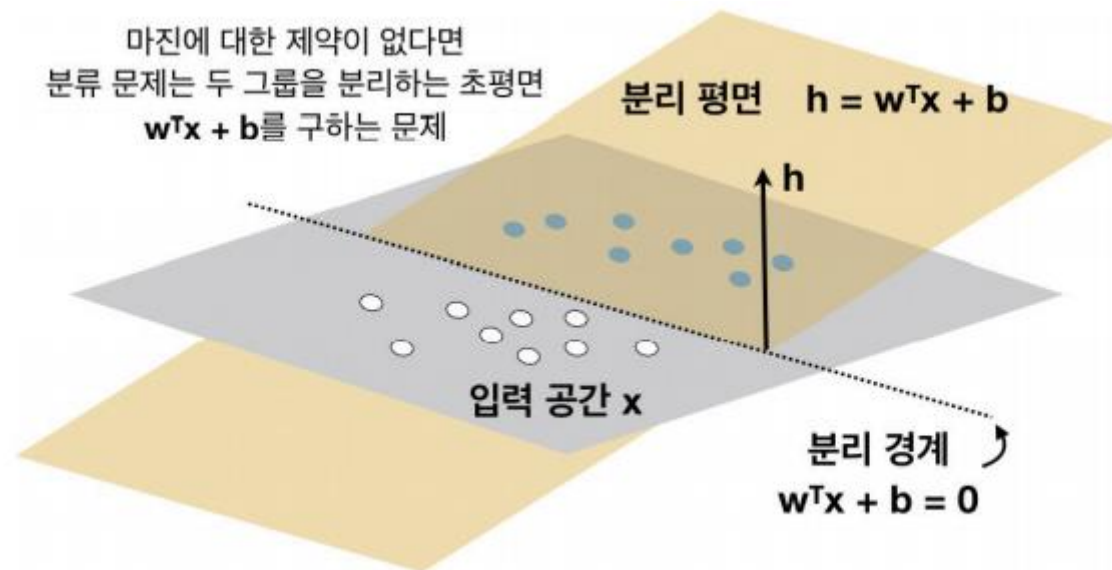
- 하드 마진이든 소프트 마진이든 마진을 최대한 넓게 만들려고 하기 때문에 마진의 양쪽에는 서로 다른 그룹에 속하는 데이터들이 하나씩 닿아 있으며 이것을 **서포트 벡터** support vector 라고 함



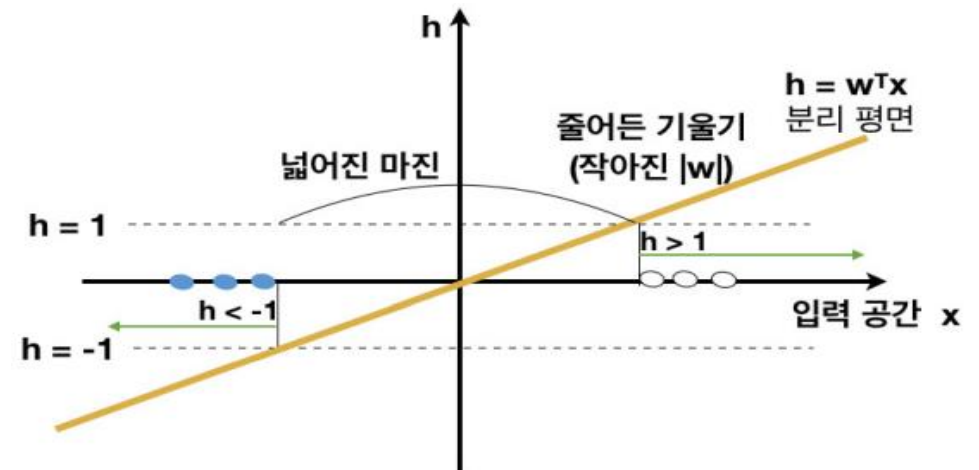
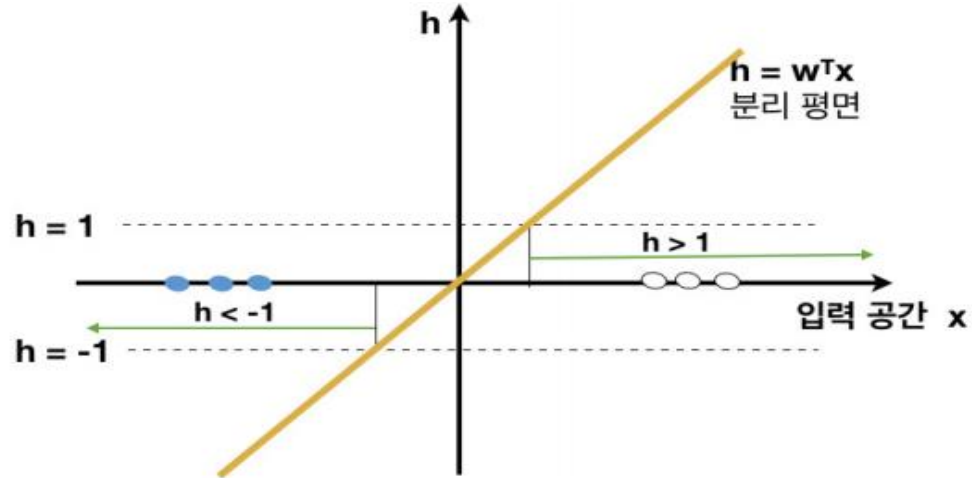
- 하드 마진을 사용할 경우에는 분류가 안될 수도 있고, 잡음에 민감할 수밖에 없으므로 소프트 마진을 사용하는 것이 바람직
  - 잡음에 민감하다는 것은 데이터에 과적합된다는 의미
  - 소프트 마진을 사용하는 것도 모델 정착화의 일종
- 소프트 마진을 사용할 때는 마진 내에 들어갈 수 있는 데이터의 수를 제어하며 이 값을 제어하는 변수를 **슬랙**slack

## 6.11 하드 마진 서포트 벡터 머신의 구현

- 서포트 벡터 머신의 기본적인 동작은 아래 그림과 같이 설명
- 레이블이 부여된 데이터 입력이 존재하는 공간이 회색 초평면으로 나타나 있고, 그 위에 흰색 레이블과 푸른색 레이블을 가진 데이터가 놓여 있음
- 이 공간에 존재하는 독립변수에 의해 결정되는 종속 변수  $h$ 를  $w^T x + b$ 라고 정의하면,  $h = w^T x + b$ 은 노란색으로 비스듬하게 표시된 초평면



- **마진margin**에 대한 제약 조건이 없다면, 서포트 벡터 머신은 흰색 레이블의 데이터들은  $h$  가 음수, 푸른색 레이블의 데이터들은  $h$  가 양수가 되게 하는  $w$  와  $b$  를 찾으면 됨
- 답이 되는 평면이 하나가 아니며 이런 경우에는 가장 "좋은" 평면을 찾아야함
  - 서포트 벡터 머신에서는 마진의 넓이가 큰 값이 될수록 좋은 답
- 평면을 수직선에 가깝게 눕혀보자. 직선으로 생각하면 기울기에 해당하는  $w$  벡터 가 0에 가까워지는 것
  - 마진이 점점 넓어지는 것을 확인할 수 있음



- 다음과 같은 제약 조건을 가진 최적화 문제가 된다.

---

### 하드 마진 SVM의 최적화 문제

---

최적화:  $\operatorname{argmin}_{\mathbf{w}, b} \frac{1}{2} \mathbf{w}^T \mathbf{w}$

제약조건:  $i = 1, 2, \dots, m$  인  $m$  개 데이터 인스턴스  $\mathbf{x}^{(i)}$  모두에  $|\mathbf{w}^T \mathbf{x}^{(i)} + b| \geq 1$

---

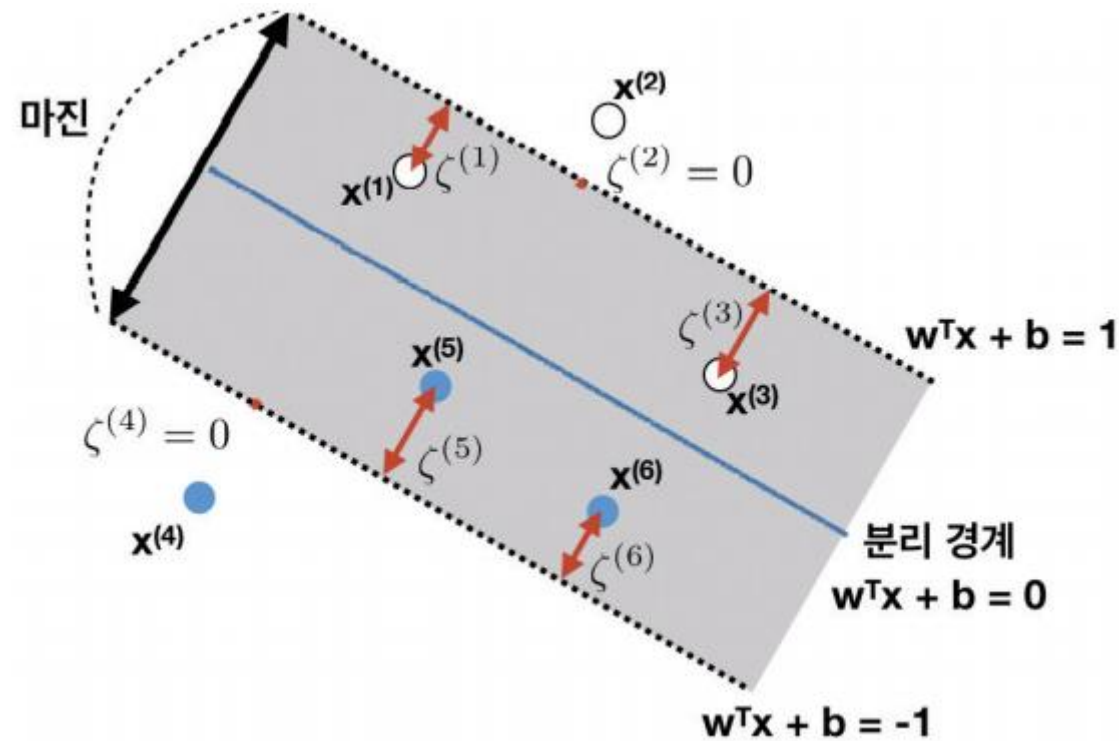
## 6.12 소프트 마진 서포트 벡터 머신의 구현

- 소프트 마진에는 **슬랙**slack 변수가 사용됨
- 이 슬랙 변수는 각 데이터 인스턴스마다 정의되므로  $m$ 개의 데이터 인스턴스가 있으면 각각에 대해  $\zeta^{(i)}$ 가 존재
- 슬랙 변수가 하는 일은 각각의 데이터가  $[-1, 1]$  사이의 범위를 갖는 마진 안으로 들어갈 수 있는 정도를 의미
  - 하드 마진은  $\zeta^{(i)} = 0$
- 슬랙 변수에 의해 각각의 데이터 인스턴스에 대해 제약 조건이 다음과 같이 변경

$i = 1, 2, \dots, m$  인  $m$  개 데이터 인스턴스  $\mathbf{x}^{(i)}$  모두에 대해

$$|\mathbf{w}^T \mathbf{x}^{(i)} + b| \geq 1 - \zeta^{(i)}$$

- 그림으로 나타내면 아래와 같다. 각각의 데이터 인스턴스는 마진 밖이나 안에 존재할 수 있는데, 마진 내에 많이 파고 들어올수록 슬랙 변수의 값이 커지는 것
  - 좋은 분류를 위해서는 슬랙 변수를 최소로 만드는 최적화 문제를 품



- 수식으로 표현하자면 원래의 최적화 문제였던  $w^T x$ 를 최소로 만드는 문제와 함께  $\sum_{i=1}^m \zeta^{(i)}$ 를 최소화하는 문제를 동시에 풀어야 함

---

### 소프트 마진 SVM의 최적화 문제

---

$$\text{최적화: } \operatorname{argmin}_{w, b} \frac{1}{2} w^T w + C \sum_{i=1}^m \zeta^{(i)}$$

$$\text{제약조건: } m \text{ 개 데이터 인스턴스 } x^{(i)} \text{ 모두에 대해 } |w^T x^{(i)} + b| \geq 1 - \zeta^{(i)}$$

---

- 최적화를 위한 **목적 함수** objective function가 두 개의 항으로 구성되어 있으며, 앞의 항은 하드 마진과 동일
- 두 번째 항은 슬랙 변수를 최소화하기 위한 것인데 제어 변수 C가 곱해져 있음
  - 0이라면 슬랙 변수에 대한 최적화가 전혀 이루어지지 않을 것이며, 이 값을 크게 두면 슬랙 변수를 최소화하는 일에 비중이 더 커지게 될 것

- 하드 마진과 소프트 마진 서포트 벡터 머신이 풀어야 하는 최적화 문제는 잘 알려진 **2차 계획법**quadratic programming을 통해 풀 수 있음
- **선형 계획법**linear programming은 다양한 선형 제약 조건을 만족하면서 선형인 **목적 함수**objective function를 최적화하는 파라미터를 찾는 것
- 제약 조건이나 목적함수가 선형이 아닌 경우의 최적화는 **비선형 최적화**nonlinear programming라고 함
- 비선형 최적화 중에서 제약 조건은 선형이면서 목적함수가 2차 형식인 경우의 해를 구하는 방법을 2차 계획법
  - 서포트 벡터 머신의 최적화가 대표적



## 6.13 사이킷런을 이용한 서포트 벡터 머신 사용하기

- 사이킷런을 이용하여 실제로 서포트 벡터 머신을 이용하여 데이터를 구분

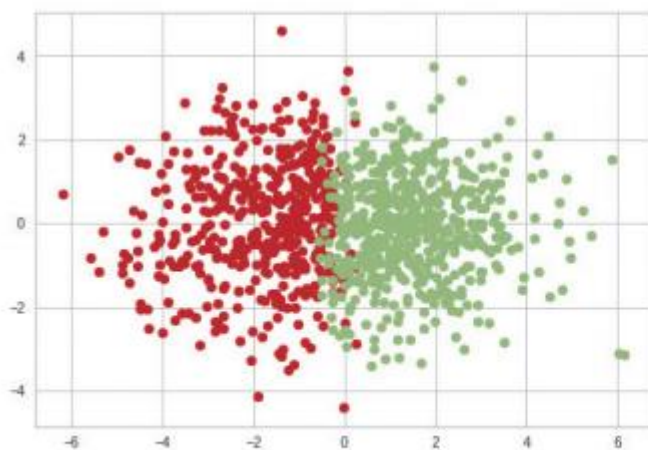


```
import pandas as pd
import numpy as np
data_loc = 'https://github.com/dknife/ML/raw/main/data/'
df = pd.read_csv(data_loc + 'two_classes.csv')
df.tail(5)
```

	x1	x2	y
995	2.664896	-1.955326	0
996	-2.019928	0.334542	1
997	-4.634470	0.300158	1
998	1.426275	-2.765590	0
999	1.988053	1.466494	0



```
df_positive = df[df['y']>0]      # y가 1인 데이터만 추출  
df_negative = df[df['y']==0]     # y가 0인 데이터만 추출  
import matplotlib.pyplot as plt  
plt.scatter(df_positive['x1'], df_positive['x2'], color='r')  
plt.scatter(df_negative['x1'], df_negative['x2'], color='g')
```



- 슬랙 변수 최적화의 가중치가 될 C 키워드 매개변수를 지정하고, 손실 함수를 loss 키워드 매개변수에 지정
  - SVM에서 사용하는 표준적인 손실함수는  $\max(0, 1 - h)$ 의 **경첩<sup>hinge</sup> 손실 함수**
  - 평균 제곱 오차를 쓰기 위해 'mse' 등을 지정하면 오류 ('hinge' 혹은 'squared\_hinge'만 가능)

```
from sklearn.svm import LinearSVC
X = df[['x1', 'x2']].to_numpy()      # x1, x2를 입력 벡터로 한다
y = df['y']                          # y열의 값이 레이블
svm_simple = LinearSVC(C=1, loss='hinge') # SVM 클래스 생성
svm_simple.fit(X, y)                 # 입력과 레이블로 SVM 학습 실시

LinearSVC(C=1, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='hinge', max_iter=1000, multi_class='ovr',
          penalty='l2', random_state=None, tol=0.0001, verbose=0)
```

- 학습이 끝나면 회귀 분석에서 사용했던 방법처럼, predict() 함수를 이용하여 입력을 넣고, 레이블을 예측

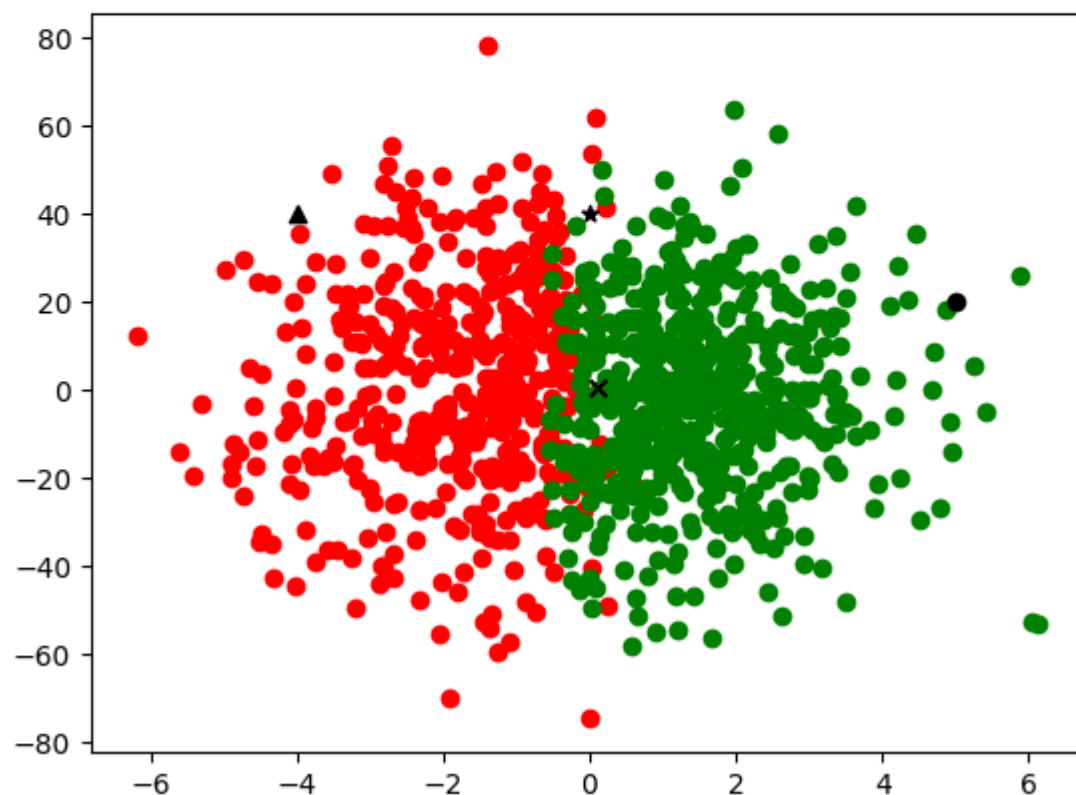
```
svm_simple.predict([[0.12, 0.56], [-4, 40], [0, 40], [5, 20]])

array([0, 1, 0, 0])
```

- 제대로 예측한 것인지 확인해 보자


```
plt.scatter(df_positive['x1'], df_positive['x2'], color='r')  
plt.scatter(df_negative['x1'], df_negative['x2'], color='g')  
plt.scatter(X_test[0, 0], X_test[0,1], marker='x', color='k')  
plt.scatter(X_test[1, 0], X_test[1,1], marker='^', color='k')  
plt.scatter(X_test[2, 0], X_test[2,1], marker='*', color='k')  
plt.scatter(X_test[3, 0], X_test[3,1], marker='o', color='k')
```

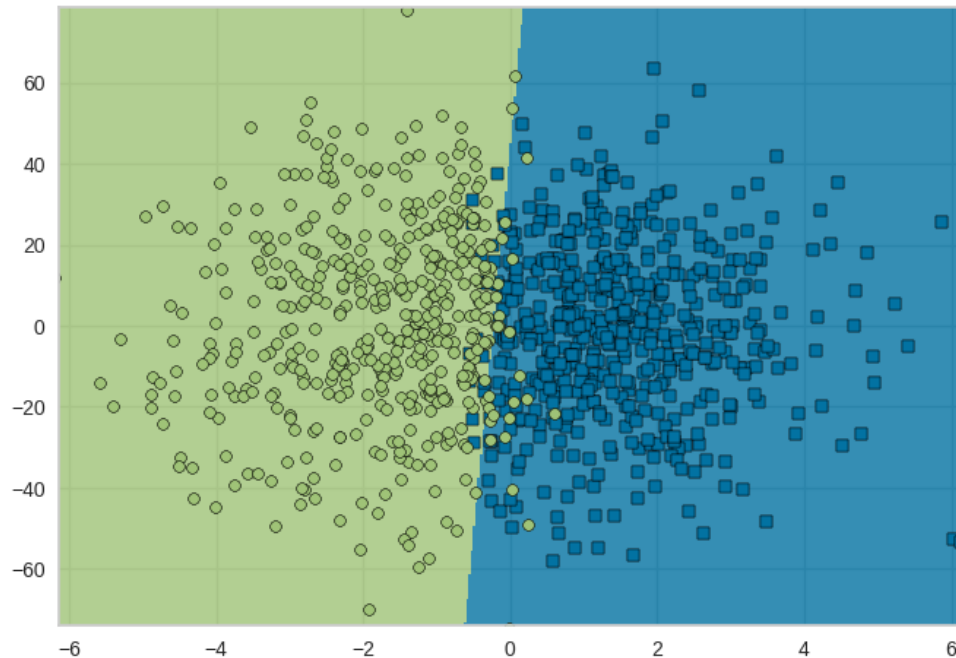
<matplotlib.collections.PathCollection at 0x7e2c46295db0>



- SVM 분류기의 분리 경계를 확인하고 싶으면 다음과 같이 학습된 `svm_simple`로 `DecisionViz` 클래스를 만들고 `fit()`과 `draw()`를 차례로 부르면 됨

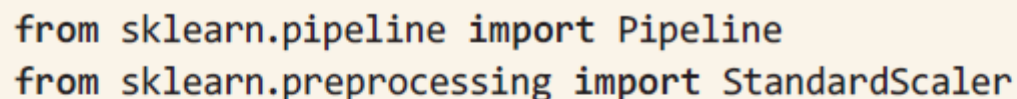
```
from yellowbrick.contrib.classifier import DecisionViz  
viz = DecisionViz(svm_simple, title="linear SVM")  
viz.fit(X, y)  
viz.draw(X, y)
```

 /usr/local/lib/python3.10/dist-packages/sklearn/svm/\_base.py:1244: ConvergenceWarning: Liblinear failed to converge, increase the number of iterations.  
warnings.warn(  
 )



## 6.14 파이프라인을 이용한 데이터 정제

- SVM은 데이터의 크기에 민감한 특성을 갖고 있음
- 데이터 정제를 위해 정규화normalization 혹은 표준화standardization을 수행함
- 사이킷런 패키지는 이를 위해 파이프라인pipeline이라는 편리한 방법을 제공

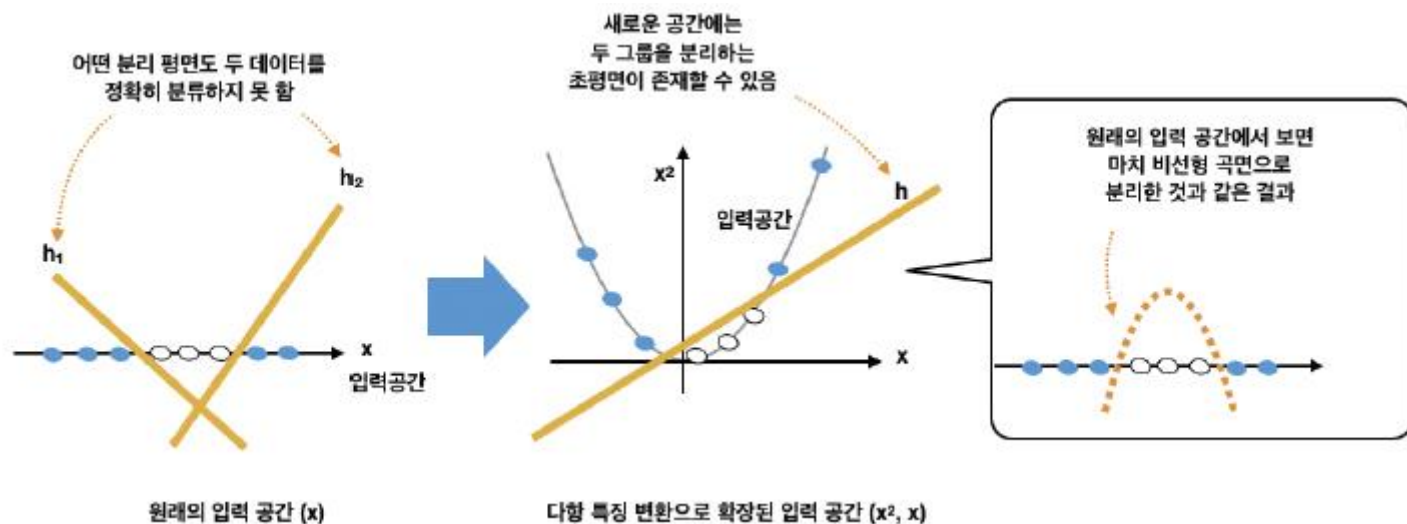


```
svm_std = Pipeline([                                # 파이프라인으로 SVM 객체 구성
    ("std", StandardScaler()),                      # 데이터 표준화 단계 포함
    ("lsvm", LinearSVC(C=1, loss='hinge')),         # 선형 SVM 분류기 포함
])
svm_std.fit(X, y)
```

[illegible]

## 6.15 다항 특징 변환을 통한 비선형 서포트 벡터 머신의 구현

- 다항 회귀에 사용했던 PolynomialFeatures는 데이터 정제 단계이므로 학습 알고리즘에 관계 없이 쓸 수 있음
- 다항 특징 변환을 하면 새로운 특징  $x^2$ 이 추가되어 데이터는 화살표 오른쪽처럼 2차원 평면에 존재하고, 분리 평면도 존재
  - 원래의 입력 공간에서 보면 비선형 곡면을 통한 분류와 같은 효과

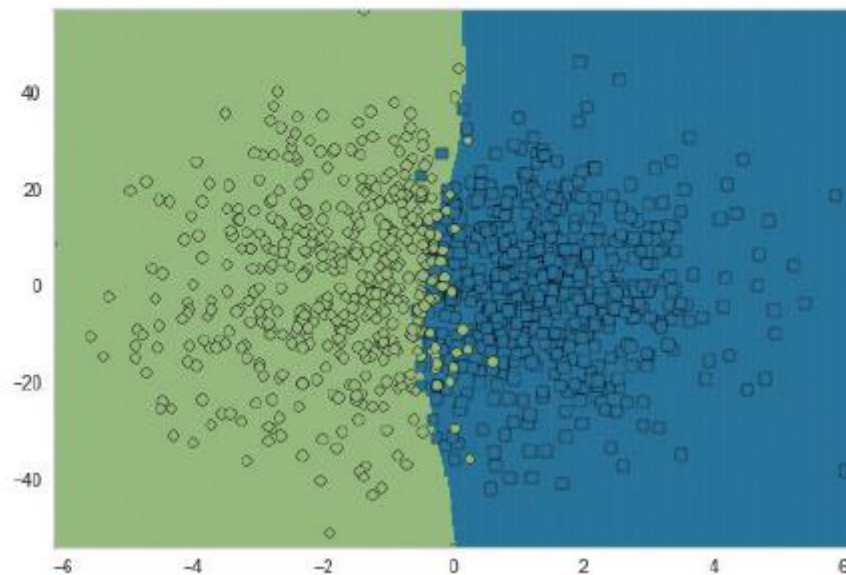




- 데이터 정제를 포함한 학습모델을 만들기 위해 사용했던 **파이프라인**pipeline에 입력 데이터의 다항 특징 변환을 포함하는 것



```
from sklearn.preprocessing import PolynomialFeatures
svm_poly = Pipeline([
    ("std", StandardScaler()),
    ("poly_inputs", PolynomialFeatures(degree=5)),
    ("lsmv", LinearSVC(C=0.01, loss="hinge"))
])
svm_poly.fit(X, y)
viz = DecisionViz(svm_poly, title="polynomial feature SVM")
viz.fit(X, y)
viz.draw(X, y)
```



# LAB<sup>6-4</sup> 비선형 SVM을 이용한 데이터 분류

## 실습 목표

아래의 URL로 접근하면 선형 분리가 불가능하도록 섞여있는 두 종류의 데이터 인스턴스들이 있다.

[https://github.com/dknife/ML/raw/main/data/twisted\\_data.csv](https://github.com/dknife/ML/raw/main/data/twisted_data.csv)

이 데이터 인스턴스들을 분류할 수 있도록 SVM을 학습시켜 보라.



## 힌트

원래의 입력 공간에서는 선형 분리가 불가능한 경우에도 입력에 대해 다항 변환을 수행하면 더 높은 차원의 공간에서 선형 분리가 가능할 수 있다. 따라서 적절한 차수의 다항 특징 변환을 적용하자.

교재의 해답 코드를 하나하나 단계별로 수행해  
봅니다

## 6.16 커널 트릭을 이용한 비선형 서포트 벡터 머신

- 다항 특징 변환을 사용할 경우 차수가 높아지면 입력되는 특징 벡터의 차원이 급격히 높아지는 문제
- SVM은 이러한 문제를 피하고 효율적인 비선형 분류를 하기 위해 **커널 트릭** kernel trick이라는 것을 사용할 수 있음
- 다항 특징 변환으로 얻는 고차원 데이터를 사용하는 것이 아니라 원래의 입력들 사이의 내적을 구한 뒤에 그 결과만 제공해도 되는 함수: 커널 함수
  - 변환된 벡터의 내적  $\phi(x^{(i)})^T \phi(x^{(j)})$ 를 원래의 벡터만으로 구할 수 있는 함수
    - $K(x^{(i)}, x^{(j)})$  : **커널 함수** kernel function

# LAB<sup>6-5</sup> 커널 트릭을 이용한 비선형 SVM

## 실습 목표

선형 분리가 불가능하도록 섞여있는 두 그룹의 데이터가 아래에서 있다.

[https://github.com/dknife/ML/raw/main/data/twisted\\_data.csv](https://github.com/dknife/ML/raw/main/data/twisted_data.csv)

비선형 SVM으로 이 데이터들을 분류하되 다항 특징 변환을 사용하지 않고 해결하라.



## 힌트

앞에서 설명한 커널 트릭을 사용한다. 커널 트릭은 학습 알고리즘 내에서 이루어지는 것이므로 데이터 정제 과정에서 특징 벡터를 변환하는 일은 하지 않는다. 따라서 PolynomialFeatures를 사용하지 않는다. SVM을 수행하는 클래스를 생성할 때, 사용할 커널을 지정하면 된다.

교재의 해답 코드를 하나하나 단계별로 수행해  
봅니다

## LAB<sup>6-5</sup> 커널 트릭을 이용한 비선형 SVM



```
import pandas as pd  
import numpy as np
```

```
data_loc = https://github.com/dknife/ML/raw/main/data/  
df = pd.read_csv(data_loc + 'twisted_data.csv')  
print(df.tail(5))
```

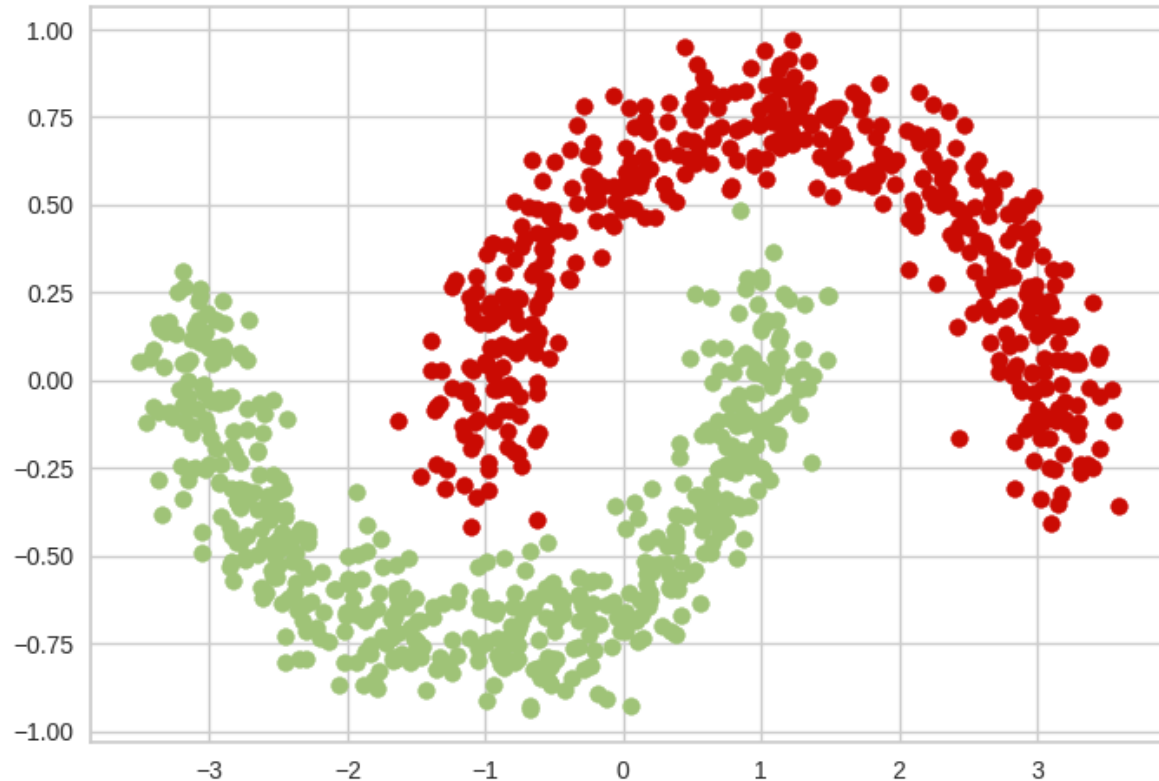


	x1	x2	y
995	1.112475	-0.180790	0
996	-2.544258	-0.559755	0
997	-1.044663	0.164435	1
998	-1.475026	-0.683385	0
999	1.860988	0.728986	1

# LAB<sup>6-5</sup> 커널 트릭을 이용한 비선형 SVM

```
df_positive = df[df['y']>0]
df_negative = df[df['y']==0]
import matplotlib.pyplot as plt
plt.scatter(df_positive['x1'], df_positive['x2'], color='r')
plt.scatter(df_negative['x1'], df_negative['x2'], color='g')
```

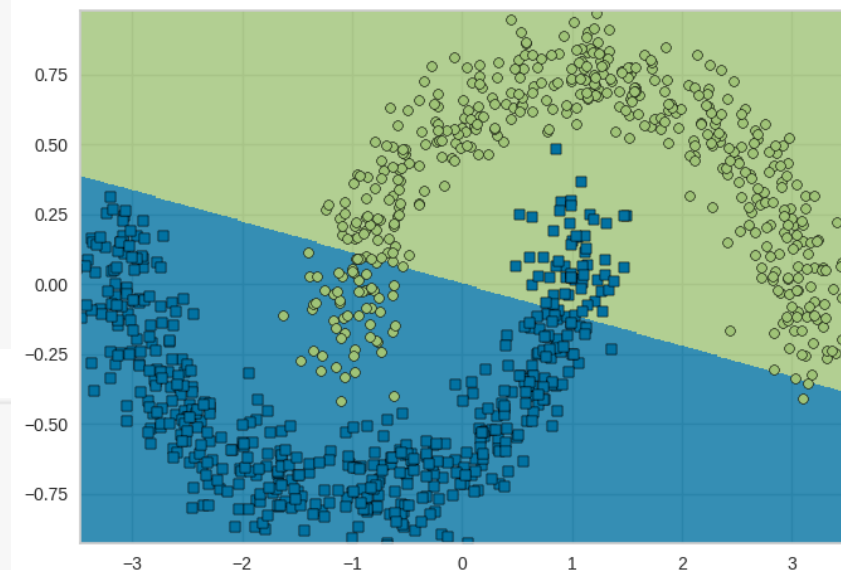
<matplotlib.collections.PathCollection at 0x7e2c45e05330>



# LAB<sup>6-5</sup> 커널 트릭을 이용한 비선형 SVM

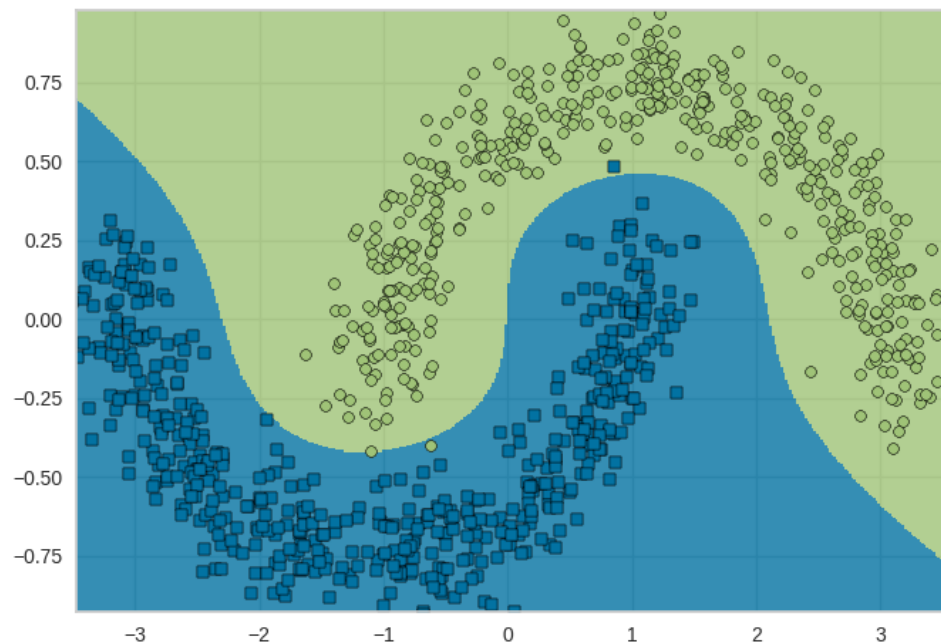
```
[37] from sklearn.svm import LinearSVC
      from sklearn.pipeline import Pipeline
      from sklearn.preprocessing import StandardScaler
      from sklearn.preprocessing import PolynomialFeatures
      from yellowbrick.contrib.classifier import DecisionViz
      X = df[['x1', 'x2']].to_numpy()
      y = df['y']
```

```
▶ polynomial_svm_clf = Pipeline([
    ("scaler", StandardScaler()),
    ("poly_features", PolynomialFeatures(degree=1)),
    ("svm_clf", LinearSVC(C=1, loss="hinge"))
])
polynomial_svm_clf.fit(X, y)
viz = DecisionViz(polynomial_svm_clf, title="polynomial feature SVM")
viz.fit(X, y)
viz.draw(X, y)
```



# LAB<sup>6-5</sup> 커널 트릭을 이용한 비선형 SVM

```
▶ polynomial_svm_clf = Pipeline([  
    ("scaler", StandardScaler()),  
    ("poly_features", PolynomialFeatures(degree=3)),  
    ("svm_clf", LinearSVC(C=1, loss="hinge"))  
])  
polynomial_svm_clf.fit(X, y)  
viz = DecisionViz(polynomial_svm_clf, title="polynomial feature SVM")  
viz.fit(X, y)  
viz.draw(X, y)
```





# Mini-project – 얼굴 인식을 SVM으로

```
import matplotlib.pyplot as plt
import numpy as np

from skimage.io import imread
from skimage.transform import resize
from skimage.feature import hog
```



```
[ ] url = 'https://github.com/dknife/ML/raw/main/data/Proj2/faces/'

face_images = []

for i in range(15):
    file = url + 'img{0:02d}.jpg'.format(i+1)
    img = imread(file)
    img = resize(img, (64,64))
    face_images.append(img)
```

```
def plot_images(nRow, nCol, img):
    fig = plt.figure()
    fig, ax = plt.subplots(nRow, nCol, figsize = (nCol,nRow))
    for i in range(nRow):
        for j in range(nCol):
            if nRow <= 1: axis = ax[j]
            else: axis = ax[i, j]
            axis.get_xaxis().set_visible(False)
            axis.get_yaxis().set_visible(False)
            axis.imshow(img[i*nCol+j])
```

```
plot_images(3,5, face_images)
```

# Mini-project - 얼굴 인식을 SVM으로

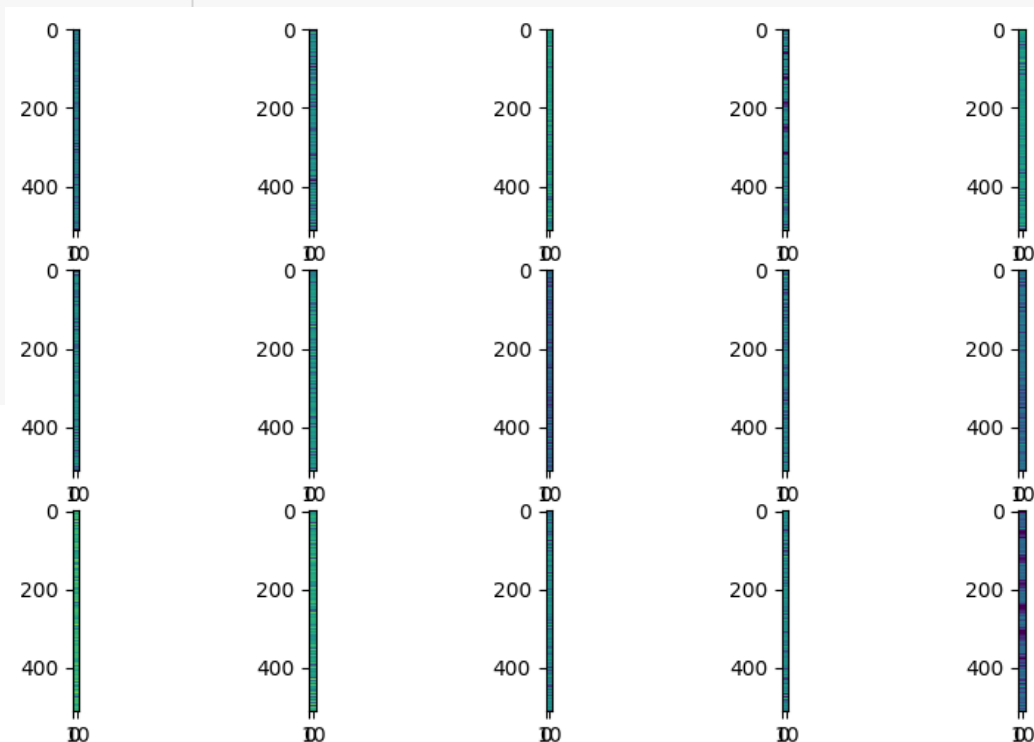
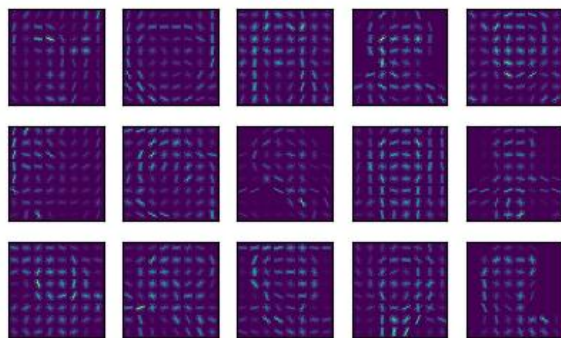
```
face_hogs = []
face_features = []

for i in range(15):
    hog_desc, hog_image = hog(face_images[i], orientations=8, pixels_per_cell=(8, 8), cells_per_block=(1, 1), visualize=True, channel_axis=-1)
    face_hogs.append(hog_image)
    face_features.append(hog_desc)

plot_images(3, 5, face_hogs)

print(face_features[0].shape)

fig = plt.figure()
fig, ax = plt.subplots(3, 5, figsize = (10, 6))
for i in range(3):
    for j in range(5):
        ax[i, j].imshow(resize(face_features[i*5+j], (512, 16)))
```



# Mini-project – 얼굴 인식을 SVM으로

```
url = 'https://github.com/dknife/ML/raw/main/data/Proj2/animals/'
```

```
animal_images = []
```

```
for i in range(15):  
    file = url + 'img{0:02d}.jpg'.format(i+1)  
    img = imread(file)  
    img = resize(img, (64,64))  
    animal_images.append(img)
```

```
plot_images(3, 5, animal_images)
```

<Figure size 640x480 with 0 Axes>

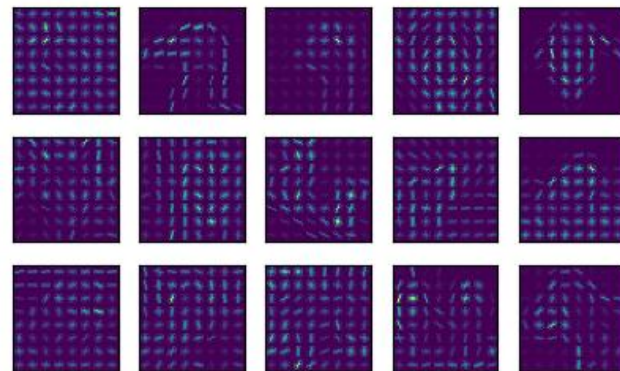


```
animal_hogs = []  
animal_features = []
```

```
for i in range(15):  
    hog_desc, hog_image = hog(animal_images[i], orientations=8, pixels_per_cell=(8, 8), cells_per_block=(1, 1), visualize=True, channel_axis=-1)  
    animal_hogs.append(hog_image)  
    animal_features.append(hog_desc)
```

```
plot_images(3, 5, animal_hogs)
```

```
fig = plt.figure()  
fig, ax = plt.subplots(3,5, figsize = (10,6))  
for i in range(3):  
    for j in range(5):  
        ax[i, j].imshow(resize(animal_features[i*5+j], (512,16)))
```

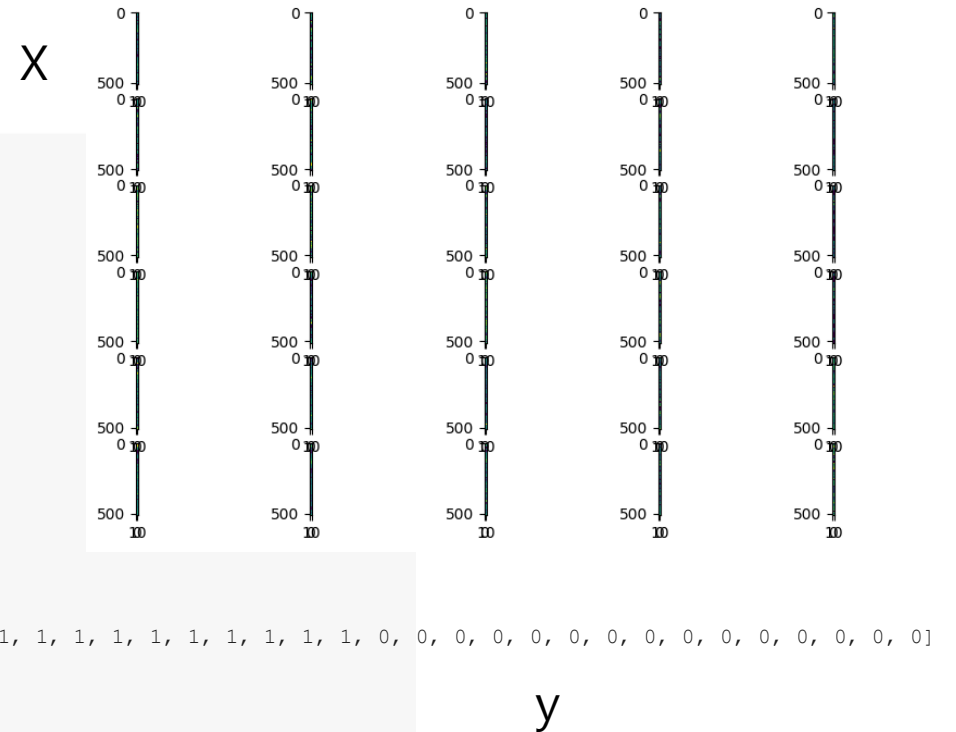


# Mini-project – 얼굴 인식을 SVM으로

```
▶ X, y = [], []

for feature in face_features:
    X.append(feature)
    y.append(1)
for feature in animal_features:
    X.append(feature)
    y.append(0)

fig = plt.figure()
fig, ax = plt.subplots(6,5, figsize = (10,6))
for i in range(6):
    for j in range(5):
        ax[i, j].imshow(resize(X[i*5+j], (512,16)), interpolation='nearest')
print(y)
```





# Mini-project – 얼굴 인식을 SVM으로



```
url = 'https://github.com/dknife/ML/raw/main/data/Proj2/test_data/'  
  
test_images = []  
  
for i in range(10):  
    file = url + 'img{0:02d}.jpg'.format(i+1)  
    img = imread(file)  
    img = resize(img, (64,64))  
    test_images.append(img)  
  
plot_images(2, 5, test_images)
```



<Figure size 640x480 with 0 Axes>

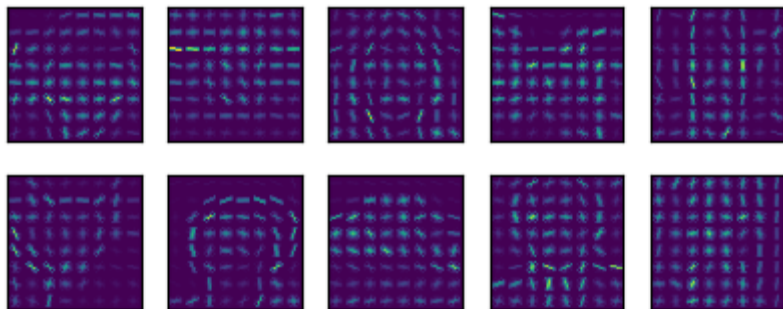


# Mini-project – 얼굴 인식을 SVM으로

```
test_hogs = []
test_features = []
for i in range(10):
    hog_desc, hog_image = hog(test_images[i], orientations=8, pixels_per_cell=(8, 8), cells_per_block=(1, 1), visualize=True, multichannel=True)
    test_hogs.append(hog_image)
    test_features.append(hog_desc)

plot_images(2, 5, test_hogs)

fig = plt.figure()
fig, ax = plt.subplots(2, 5, figsize = (10, 4))
for i in range(2):
    for j in range(5):
        ax[i, j].imshow(resize(test_features[i+5*j], (512, 16)), interpolation='nearest')
```



# Mini-project – 얼굴 인식을 SVM으로

```
[ ] test_result = polynomial_svm_clf.predict(test_features)
    print(test_result)
```

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

```
[ ] fig = plt.figure()
    fig, ax = plt.subplots(2,5, figsize = (10,4))
    for i in range(2):
        for j in range(5):
            ax[i, j].get_xaxis().set_visible(False)
            ax[i, j].get_yaxis().set_visible(False)
            if test_result[i+5+j] == 1:
                ax[i, j].imshow(test_images[i+5+j], interpolation='nearest')
```

<Figure size 640x480 with 0 Axes>





# 핵심 정리

- 다항 회귀는 비선형 회귀의 일종으로 입력에 대해 다항 특징 변환을 적용하여 데이터의 차원을 높이고 2 차 이상의 고차항 등이 나타나게 하여 곡면으로 회귀가 이루어지게 한다.
- 다항 특징 변환의 차수를 지나치게 높이면 입력 데이터의 차원이 급격히 증가하고 회귀 함수가 매우 복잡해져서 효율성과 예측 정확성 모두 나빠질 수 있다.
- 복잡한 모델일수록 과적합의 가능성이 높아지며, 일반화 역량이 나빠질 수 있다.
- 과적합은 모델의 단순화, 훈련 데이터의 증가를 통해 회피할 수 있다.
- 모델이 데이터를 잘 표현하지 못하는 과소적합은 모델을 더 복잡하게 할 필요가 있다.
- 결정 트리는 엔트로피에 기반한 정보 이득을 높이는 방향으로 데이터를 그룹화한다.
- 정보 이득을 계산하는 것보다 쉬운 방법으로 지니 불순도를 사용할 수 있다.
- 정보 이득에 기반한 결정트리와 지니 불순도를 이용한 결정 트리는 비슷한 결과를 만들어낸다.
- 서포트 벡터 머신은 데이터를 정의 그룹과 부의 그룹 두 가지로 구분하는 이진 분류기이다.

# 핵심 정리

- 서포트 벡터 머신은 데이터를 쪼갤 때 **가장 큰 마진**을 얻을 수 있는 방법을 찾는다.
- 마진 내에 데이터가 침범하지 못 하게 하면 **하드 마진**이라고 하고, 일부 데이터가 마진 내에 들어올 수 있는 경우를 **소프트 마진**이라고 한다.
- **서포트 벡터 머신**은 기본적으로 **선형 분리**를 실시한다.
- 서포트 벡터 머신에도 **다항 특징 변환** 기법을 데이터 정제 단계에서 실시하여 **비선형 분리**를 할 수 있다.
- 다항 특징 변환이 지나치게 입력의 차원을 높이는 문제를 해결하기 위해 **커널 트릭**을 사용할 수 있다.
- 기계 학습에서 **커널**이라는 것은 변환된 두 벡터의 내적을, 실제로 각 벡터에 대한 변환을 수행하지 않고, 원래의 벡터만으로 구할 수 있는 함수이다.