6. 로지스틱 회귀와 서포트 벡터 머신

2022.10.31

Chung-Ang University
AI/ML Innovation Research Center
Hyun-soon Lee

<로지스틱 회귀>

목차

01

로지스틱 회귀란?

01 로지스틱 회귀란?

- 분류 문제 : 몇 가지 이산적 값 중 하나를 선택하는 모델. '분류 모델(classification model)'이라고 부름
- 샘플이 특정 클래스에 속할 확률 추정하는 데 널리 사용됨 [1]
  - 양성 클래스(positive class): 레이블 1인 경우, 추정 확률이 50%가 넘는 경우
  - 음성 클래스(negative class): 레이블이 0인 경우

- 입력 특성의 가중치 합을 계산하고 편향을 더함.
- 선형 회귀처럼 바로 결과를 출력하지 않고 결과값의 로지스틱(logistic)을 출력함

[1] 핸즈온 머신러닝 2판, 오렐리앙 제롱, 한빛미디어

01 로지스틱 회귀란?

- [표 9-1]은 GRE와 GPA 데이터를 통해 대학의 합격 여부(Admit 열)를 나타냄

표 9-1 GRE와 GPA 정보를 활용하여 합격 여부를 나타내는 데이터

Number	Admit	GRE	GPA	Number	Admit	GRE	GPA
1	0	380	3.61	16	0	660	3.34
2	1	660	3.67	17	1	740	4.00
3	1	800	4.00	18	0	560	3.19
4	1	640	3.19	19	0	380	2.94
5	0	520	2.93	20	0	400	3.65
6	1	760	3.00	21	0	600	2.82
7	1	560	2.98	22	1	620	3.18
8	0	400	3.08	23	0	560	3.32
9	1	540	3.39	24	0	640	3.67
10	0	700	3.92	25	1	680	3.85
11	0	800	4.00	26	0	580	4.00
12	0	440	3.22	27	0	600	3.59
13	1	760	4.00	28	0	740	3.62
14	0	700	3.08	29	0	620	3.30
15	1	700	4.00	30	0	580	3.69

01 로지스틱 회귀란?

- GRE와 GPA 정보를 산점도로 표현
  - 합격자는 파란색, 불합격자는 빨간색으로 나타냄

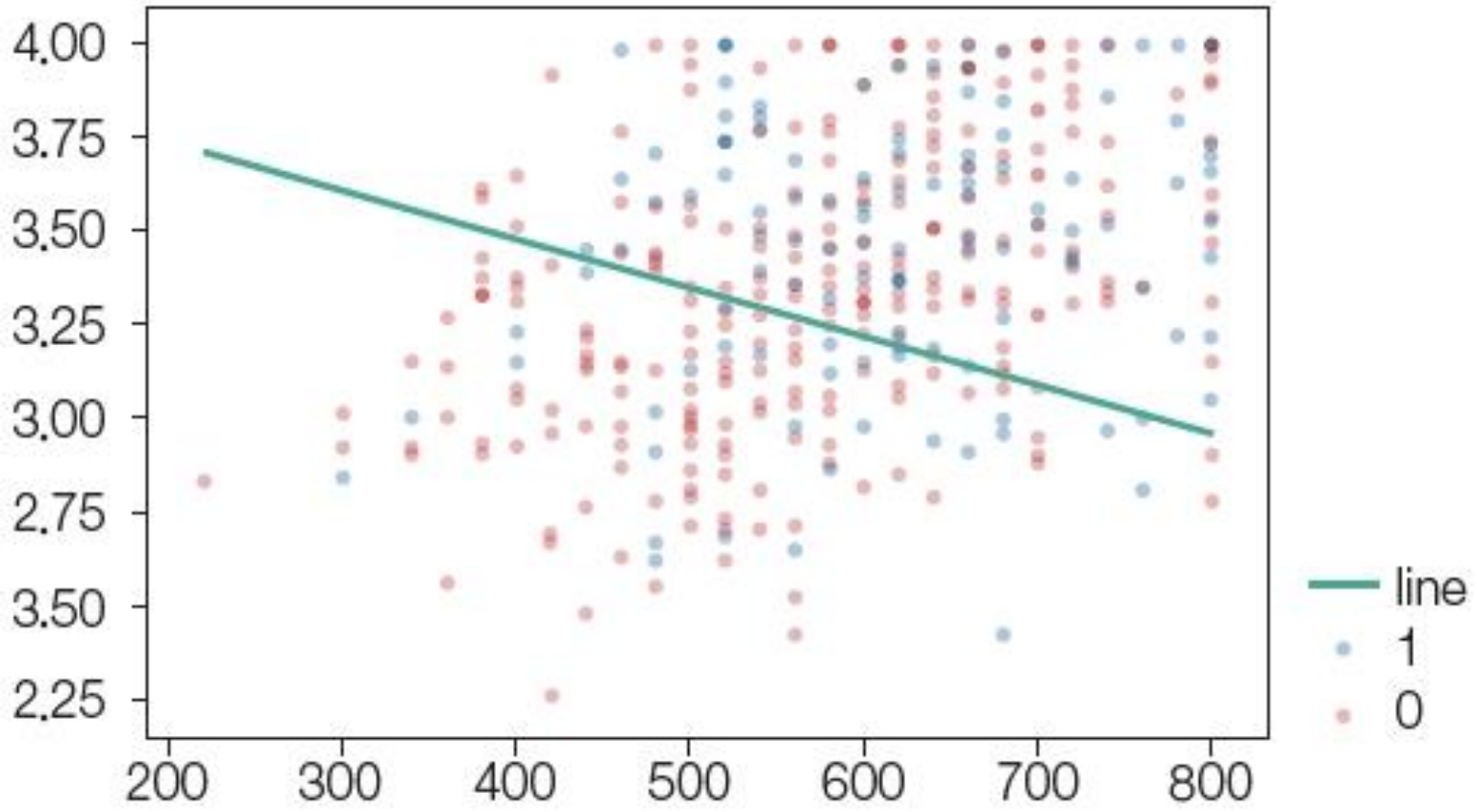


그림 9-1 산점도 위에 선형회귀 모델 표현

01 로지스틱 회귀란?

- 초록색 선을 추가해 선 상단은 합격, 선 하단은 불합격
  - 아래 수식으로 기존 선형회귀 모델을 적용

$$f(x) = 4 - 0.0013 \times GRE - GPA$$

$$Admit = \begin{cases} 1 & \text{if } f(x) \geq 0 \\ 0 & otherwise \end{cases}$$

  - 문제점:

❶ $f(x)$ 의 값이 1 이상이나 0 이하로 나올 수 있음

❷ 각 피쳐들이 Y에 영향을 주는 것을 해석하는 문제

❸ 사건의 발생 여부는 이산적인데 실제 $f(x)$ 수식은 연속적

01 로지스틱 회귀란?

1. 로지스틱 회귀의 개념

- 이진 분류(binary classification) 문제를 확률로 표현
- 어떤 사건이 일어날 확률을 P(X)로 나타내고 일어나지 않을 확률을 1 - P(x)로 나타냄 ($0 \leq P(X) \leq 1$)
- 오즈비(odds ratio) : 어떤 사건이 일어날 확률과 일어나지 않을 확률의 비율

$$\frac{P(X)}{1-P(X)}$$

01 로지스틱 회귀란?

1. 로지스틱 회귀의 개념

- 확률이 올라갈수록 오즈비도 급속히 상승

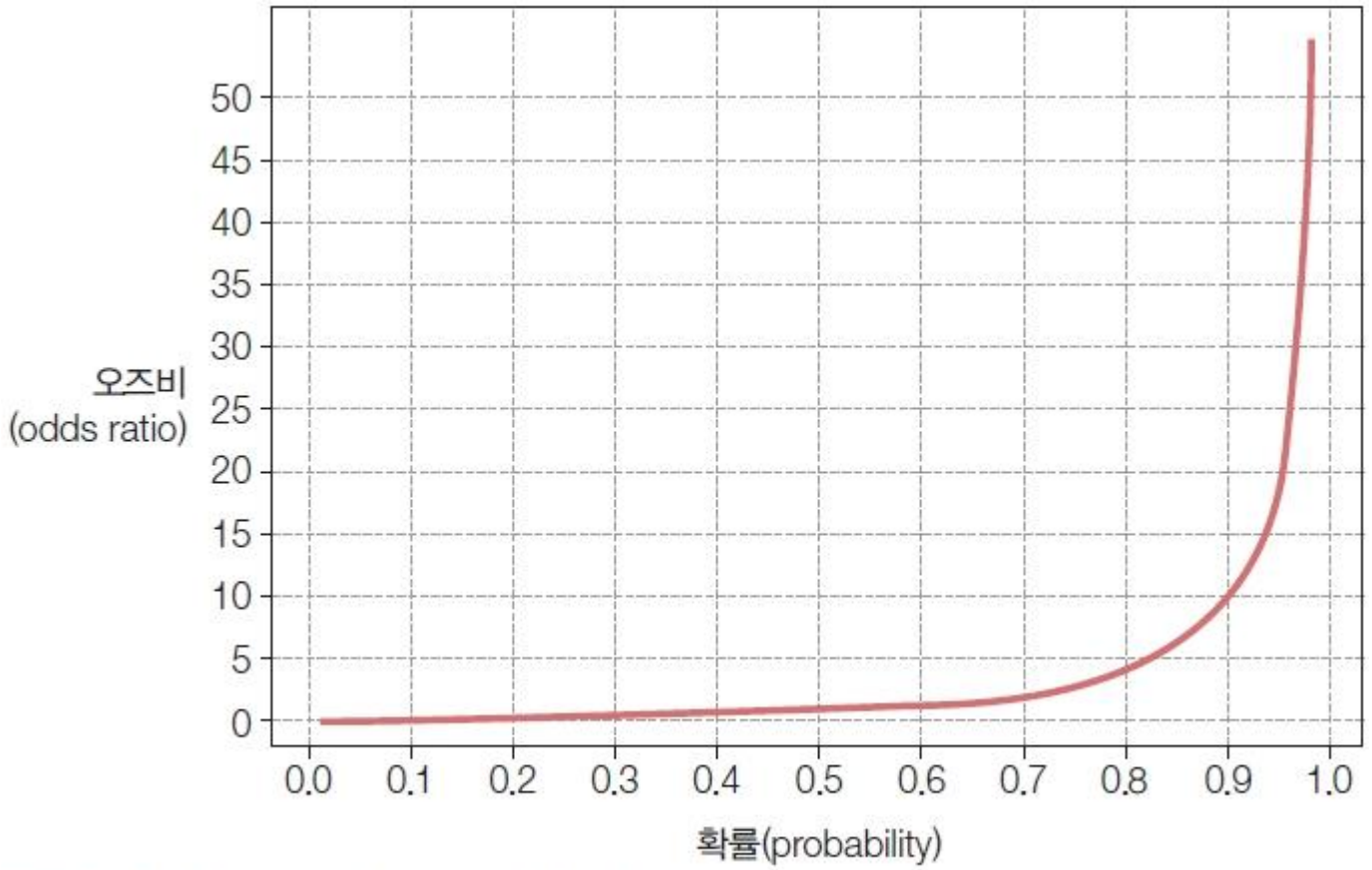


그림 9-2 확률이 올라가면서 오즈비가 상승하는 그래프

01 로지스틱 회귀란?

1. 로지스틱 회귀의 개념

- 로짓(logit) 함수 : 오즈비에 상용로그를 붙인 수식

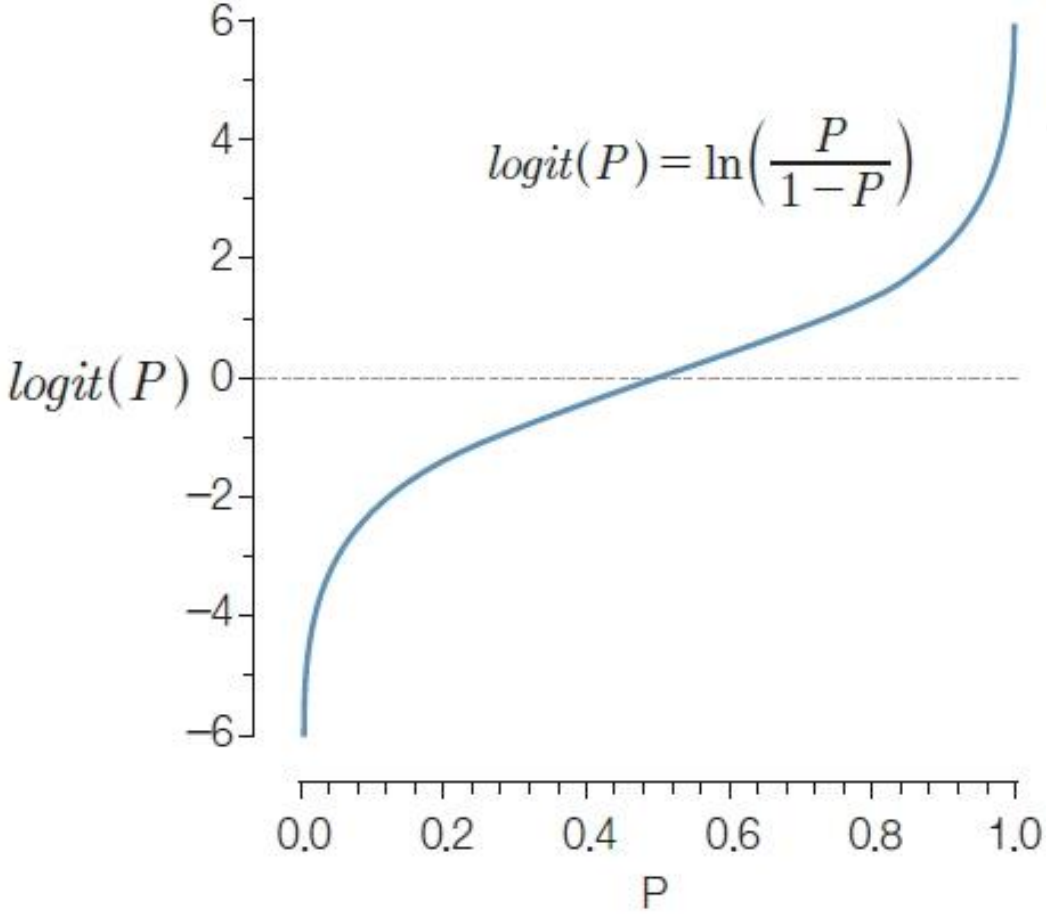


그림 9-3 로짓 함수 그래프

$$logit(p(y=1|x)) = \log_e\left(\frac{p}{1-p}\right)$$
$$= \log_e(p) - \log_e(1-p)$$
$$= -\log_e\left(\frac{1}{p} - 1\right)$$

01 로지스틱 회귀란?

1. 로지스틱 회귀의 개념

- X 값으로 확률을 넣으면 logit(P) 꼴로 나타남
- 확률을 구하려면 기존 함수의 역함수를 취하여 연산

$$f(z) = y = -\log_e\left(\frac{1}{z} - 1\right)$$

$$z = -\log_e\left(\frac{1}{y} - 1\right)$$

$$e^{-z} = \frac{1-y}{y}$$

$$y \times e^{-z} + y = 1$$

$$y(e^{-z} + 1) = 1$$

$$y = \frac{1}{1+e^{-z}}$$

01 로지스틱 회귀란?

1. 로지스틱 회귀의 개념

- 로지스틱 함수(logistic function) : 로짓 함수의 역함수
  - 그래프가 S자 커브 형태인 0과 1사이의 값을 출력하는 시그모이드 함수 (sigmoid function)

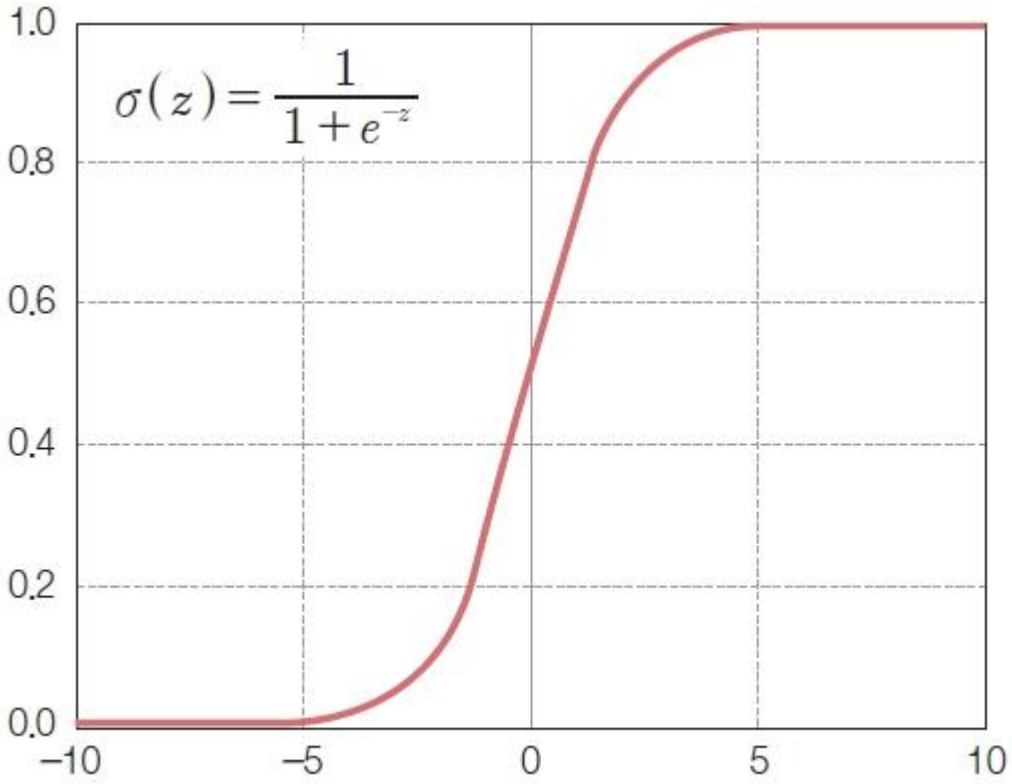


그림 9-4 시그모이드 함수

01 로지스틱 회귀란?

1. 로지스틱 회귀의 개념

- 로지스틱 회귀(Logistic Regression) : 종속변수가 이분형일 때 수행할 수 있는, 예측 분석을 위한 회귀분석 기법
- 시그모이드 함수 수식
  - y 값을 확률 p로 표현
  - z 값은 선형회귀와 같이 가중치와 피쳐의 선형 결합(linear combination)으로 표현 가능

$$p = \sigma(z) = \frac{1}{1+e^{-z}},\ \frac{p}{1-p} = \frac{\frac{1}{1+e^{-z}}}{\frac{e^{-z}}{1+e^{-z}}} = \frac{1}{e^{-z}} = e^{z}$$

$$\log_e \frac{p}{1-p} = z$$

$$\log_e \frac{p}{1-p} = z = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n$$

- 적절한 z값을 찾으면, 이 값으로 시그모이드 함수를 사용한 0에서 1까지의 확률값을 구할 수 있음.

2. 로지스틱 회귀의 기본 함수

2.1 가설함수

- 가설함수(hypothesis function)

$$h_\theta(x) = g(z) = \frac{1}{1+e^{-z}}$$

  - z는 가중치 값과 피쳐 값의 선형 결합, 찾아야하는 것은 가중치 값
  - 가중치 값을 찾는 학습을 위해 경사하강법 알고리즘 사용

$$z = w_0x_0 + w_1x_1 + \cdots + w_nx_n = \theta^T X$$

2. 로지스틱 회귀의 기본 함수

2.2 비용함수

- 먼저 비용함수를 정의하고 예측값과 실제값 간의 차이를 최소화하는 방향으로 학습
- 실제값이 1일 때와 실제값이 0일 때 각각 다르게 비용함수를 정의

$$Cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

2. 로지스틱 회귀의 기본 함수

2.2 비용함수

- (a)는 y = 1일 때, (b)는 y = 0일 때 비용함수 그래프(0 ≤ h ≤ 1)
  - (a)에서 h 값이 1에 가까워질수록 비용함수가 0에 가까워짐
    (b)에서 h 값이 0에 가까워질수록 비용함수가 0에 가까워짐

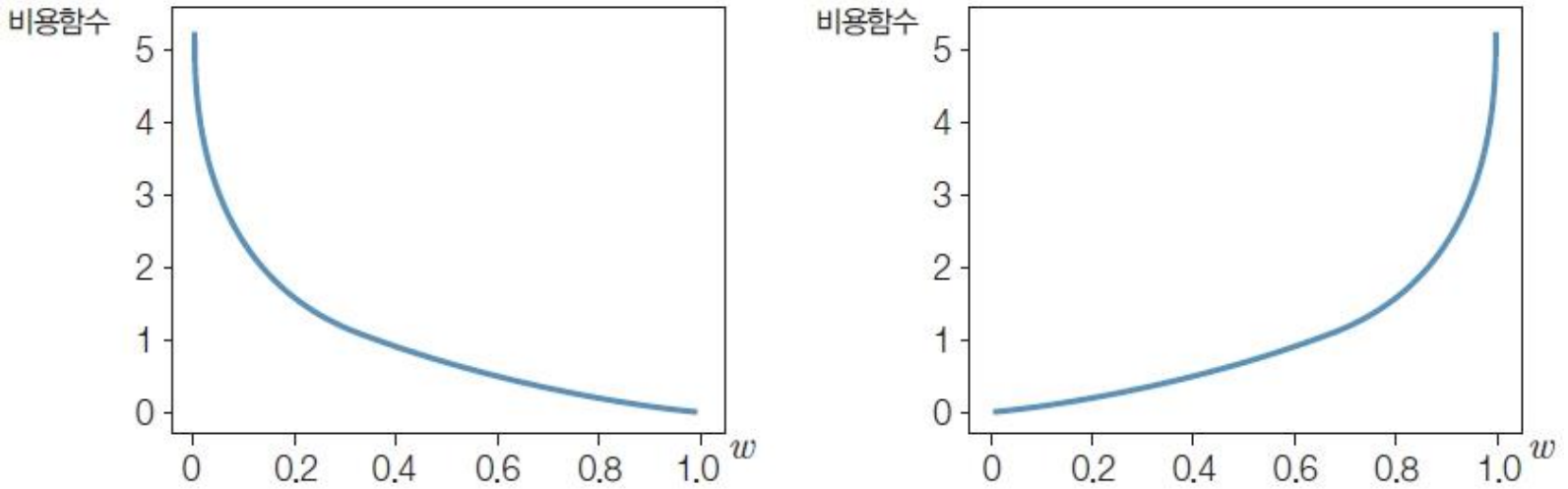


그림 9-5 비용함수 그래프

2. 로지스틱 회귀의 기본 함수

2.2 비용함수

- 두 경우의 비용함수를 하나로 통합
- y값이 1일 경우, 오른쪽 항목이 사라짐
- y값이 0인 경우, 왼쪽 항목이 사라짐.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$

$$= -\frac{1}{m}\sum_{i=1}^{m}[\underbrace{y^{(i)}\log h_\theta(x^{(i)})}_{\text{왼쪽 항목}} + \underbrace{(1-y^{(i)})\log(1-h_\theta(x^{(i)}))}_{\text{오른쪽 항목}}]$$

2. 로지스틱 회귀의 기본 함수

2.3 비용함수의 미분과 가중치 업데이트

- θ의 최적값을 구하기 위해 J 값을 θ에 대해 미분
  - θ는 z값 안에 있는 w_j의 집합

$$\frac{\partial}{\partial \partial_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_j^i$$

- 가중치 값 업데이트
  - 선형회귀와 동일하게 모든 θ에 대해 동시에 가중치가 업데이트됨

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \partial} J(\theta)$$

$$= \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_j^i$$

02

분류 문제의 성능지표

02 분류 문제의 성능지표

- 혼동행렬(confusion matrix)
  - 분류 문제의 성능 측정을 위해 먼저 이해해야 할 것
  - 예측 값이 실제 값 대비 얼마나 잘 맞는지 2 X 2 행렬로 표현하는 기법(그림 9-7)
- 실제값과 예측값의 조합으로 발생 가능한 4가지 경우
  - True Positive(TP) : 예측값과 실제값이 모두 1로 동일할 때, 즉 모델의 예측값이 정답이고 예측 대상이 1일 때
  - True Negative(TN) : 예측값과 실제값이 모두 0으로 동일할 때, 즉 모델의 예측값이 정답이고 예측 대상이 0일 때
  - False Negative(FN) : 실제값은 1이지만 예측값이 0으로, 모델의 예측값이 오답이고 예측값이 0을 예측할 때
  - False Positive(FP) : 실제값은 0이지만 예측값이 1로, 모델의 예측값이 오답이고 예측값이 1을 예측할 때

그림 9-7 혼동행렬

02 분류 문제의 성능지표

- 사이킷런으로 혼동행렬표(confusion matrix) 나타내기
  - In [1] - y_true: 실제 값, y_pred: 예측 값
  - In [2] - 넘파이 배열: **True Negative, False Positive, False Negative, True Positive**

In [1]:	**from sklearn.metrics import confusion_matrix** **y_true = [1, 0, 1, 1, 0, 1]** **y_pred = [0, 0, 1, 1, 0, 1]** **confusion_matrix(y_true, y_pred)**
Out [1]:	**array([[2, 0],** **[1, 3]], dtype=int64)**
In [2]:	**tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()** **(tn, fp, fn, tp)**
Out [2]:	**(2, 0, 1, 3)**

02 분류 문제의 성능지표

3. 혼동행렬표를 사용한 지표

3.1 정확도

- 정확도(accuracy) : 전체 데이터 개수 대비 정답을 맞춘 데이터의 개수

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

그림 9-7 정확도 측정

02 분류 문제의 성능지표

3. 혼동행렬표를 사용한 지표

3.1 정확도

- 사이킷런으로 정확도 구하기

In [3]:	import numpy as np from sklearn.metrics import accuracy_score y_pred = np.array([0, 1, 1, 0]) y_true = np.array([0, 1, 0, 0]) sum(y_true == y_pred) / len(y_true)
Out [3]:	0.75
In [4]:	accuracy_score(y_true, y_pred)
Out [4]:	0.75

02 분류 문제의 성능지표

3. 혼동행렬표를 사용한 지표

3.2 정밀도, 민감도, F1 스코어

- 정밀도와 민감도는 불균일한 데이터셋을 다룰 때 유용
  - 데이터에서 1과 0의 비율이 7:3 또는 3:7 이상 차이나는 상태
- 정밀도(precision) : 모델이 1이라고 예측했을 때 얼마나 잘 맞을지에 대한 비율

$$PRECISION(PPV) = \frac{TP}{TP + FP}$$

그림 9-8 정밀도 측정

02 분류 문제의 성능지표

3. 혼동행렬표를 사용한 지표

3.2 정밀도, 민감도, F1 스코어

- 민감도(recall) : 실제 1인 값을 가진 데이터를 모델이 얼마나 1이라고 잘 예측했는지에 대한 비율
  - 반환율 또는 재현율이라고도 부름

$$RECALL(TPR) = \frac{TP}{TP + FN}$$

그림 9-9 민감도 측정

02 분류 문제의 성능지표

3. 혼동행렬표를 사용한 지표

3.2 정밀도, 민감도, F1 스코어

- F1 스코어(F1 score) : 정밀도와 민감도의 조화평균 값, 해당 모델이 얼마나 성능이 좋은지 확인할 경우 사용

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall}$$

In [5]:
```
import numpy as np
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

y_pred = np.array([0, 1, 1, 0, 1, 1, 1, 0])
y_true = np.array([0, 1, 0, 0, 0, 0, 1, 1])
```

02 분류 문제의 성능지표

3. 혼동행렬표를 사용한 지표

3.2 정밀도, 민감도, F1 스코어

In [6]:	precision_score(y_true, y_pred) # 정밀도
Out [6]:	0.4
In [7]:	recall_score(y_true, y_pred) # 민감도
Out [7]:	0.6666666666666666
In [8]:	f1_score(y_true, y_pred) # F1 스코어
Out [8]:	0.5

03

로지스틱 회귀 구현하기

03 로지스틱 회귀 구현하기

1. 로지스틱 회귀 구현을 위한 함수

1.1 시그모이드 함수

$$h_\theta(x) = g(z) = \frac{1}{1+e^{-z}}$$

```
def sigmoid(z):
    return 1 / (1 + np.exp(z))
```

03 로지스틱 회귀 구현하기

1. 로지스틱 회귀 구현을 위한 함수

1.2 가설함수

- 시그모이드 함수의 z 값은 실제로는 가중치와 피쳐의 선형 결합이므로 피쳐 값들을 x 벡터로, 가중치 값들은 θ로 입력해줌

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

```
def hypothesis_function(x, theta):
    z = (np.dot(-x,theta))
    return sigmoid(z)
```

03 로지스틱 회귀 구현하기

1. 로지스틱 회귀 구현을 위한 함수

1.3 비용함수

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$

$$= -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log h_\theta(x^{(i)}) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

```
def compute_cost(x, y, theta):
    m = y.shape[0]
    J = (-1.0 / m) * (
        y.T.dot(np.log(hypothesis_function(x,theta))) + ₩
        (1-y).T.dot(np.log(1- hypothesis_function(x,theta))))

    return J
```

03 로지스틱 회귀 구현하기

1. 로지스틱 회귀 구현을 위한 함수

1.4 경사하강법 : 가중치 업데이트

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_j^i$$

(1/2)

```
def minimize_gradient(x, y, theta, iterations=100000,
alpha=0.01):
    m = y.size
    cost_history = []
    theta_history = []

    for _ in range(iterations):
        original_theta = theta
        for i in range(theta.size):
            partial_marginal = x[:, i].reshape(x.shape[0], 1)
            delta = hypothesis_function(x, original_theta) - y
            grad_i = delta.T.dot(partial_marginal)
```

03 로지스틱 회귀 구현하기

1. 로지스틱 회귀 구현을 위한 함수

1.4 경사하강법 : 가중치 업데이트

(2/2)

```
theta[i] = theta[i] - (alpha * grad_i)

        if (_ % 100) == 0:
            theta_history.append(theta)
            cost_history.append(compute_cost(x, y, theta))
    return theta, np.array(cost_history),np.array(theta_history)
```

03 로지스틱 회귀 구현하기

2. 사이킷런을 사용하여 학습하기

2.1 데이터셋 준비

- 앞의 코드들과 동일하게 사이킷런에서 로지스틱 회귀 사용가능
  - 인터넷 사용자가 초보자(newbie)인지 아닌지 구별하는 와튼대학교의 'uva.txt'데이터를 사용

In [1]:

```
import pandas as pd
data_url= "http://www-stat.wharton.upenn.edu/~waterman/DataSets/uva.txt"
df = pd.read_table(data_url)
df[:5]
```

Out [1]:

	who	Newbie	Age	Gender	Household Income	Sexual Preference	Country	Education Attainment	Major Occupation	Marital Status	Years on Internet
0	id74364	0	54.0	Male	$50-74	Gay male	Ontario	Some College	Computer	Other	4-6 yr
1	id84505	0	39.0	Female	Over $100	Heterosexual	Sweden	Professional	Other	Other	1-3 yr
2	id84509	1	49.0	Female	$40-49	Heterosexual	Washington	Some College	Management	Other	Under 6 mo
3	id87028	1	22.0	Female	$40-49	Heterosexual	Florida	Some College	Computer	Married	6-12 mo
4	id76087	0	20.0	Male	$30-39	Bisexual	New Jersey	Some College	Education	Single	1-3 yr

03 로지스틱 회귀 구현하기

2. 사이킷런을 사용하여 학습하기

2.2 데이터 전처리

- 필요없는 데이터 드롭: who, Country, Years on Internet 열 제거

In [2]:

```
df.pop('who')
df.pop('Country')
df.pop('Years on Internet')

df.dtypes
```

Out [2]:

```
Newbie                 int64
Age                  float64
Gender                object
Household Income      object
Sexual Preference     object
Education Attainment  object
Major Occupation      object
Marital Status        object
dtype: object
```

03 로지스틱 회귀 구현하기

2. 사이킷런을 사용하여 학습하기

2.2 데이터 전처리

- 데이터 타입 변환하고 결측값 확인하여 채우기
  - astype 함수: category 타입의 데이터들에 대해서 원핫인코딩(one-hot encoding) 형태로 바꾸기 위해 데이터 타입을 변환
  - category 타입에 해당하는 열들의 이름을 정리하기 위하여

In [3]:
```
category_cols = ["Gender", 'Household Income',
                 'Sexual Preference', 'Education Attainment',
                 'Major Occupation', "Marital Status"]

for col in category_cols:
    df[col] = df[col].astype('category')

df.dtypes
```

03 로지스틱 회귀 구현하기

2. 사이킷런을 사용하여 학습하기

2.2 데이터 전처리

Out [3]:

```
Newbie                  int64
Age                   float64
Gender                 category
Household Income       category
Sexual Preference      category
Education Attainment   category
Major Occupation       category
Marital Status         category
dtype: object
```

03 로지스틱 회귀 구현하기

2. 사이킷런을 사용하여 학습하기

2.2 데이터 전처리

- 판다스의 get_dummies 함수: 데이터들을 원핫인코딩(one-hot encoding) 형태로 변환하기 위해

  => 열의 형태가 38개로 증가하는 것을 확인할 수 있음.

- 원핫인코딩(one-hot encoding): scikit-learn에서 제공하는 머신러닝 알고리즘은 문자열 값을 입력 값으로 허락하지 않기 때문에 모든 문자열 값들을 숫자형으로 인코딩하는 전처리 작업, 범주형 데이터의 개수만큼 가변수를 생성하여 존재 유무를 1 또는 0으로 표현하는 기법

  => 3주차 데이터 전처리 강의 참조

In [4]:	**df_onehot = pd.get_dummies(df)** **df_onehot.shape**
Out [4]:	**(19583, 38)**

03 로지스틱 회귀 구현하기

2. 사이킷런을 사용하여 학습하기

2.2 데이터 전처리

- Isnull 함수: 데이터 결측값 확인, 각 열별 데이터 중 isnull 함수 결과의 합을 표현

  => Age에만 결측값이 존재하는 것을 확인

In [5]:	df_onehot.isnull().sum()	
Out [5]:	Newbie	0
	Age	561
	Gender_Female	0
	Gender_Male	0
	Household Income_$10-19	0 (...이하 생략)

03 로지스틱 회귀 구현하기

2. 사이킷런을 사용하여 학습하기

2.2 데이터 전처리

- loc 함수: 결측값을 채우는 가장 간단한 방법

  => Age의 평균값으로 결측값을 채움

In [6]:	df_onehot.loc[pd.isnull(df_onehot['Age']), "Age"] =df_onehot['Age'].mean()

03 로지스틱 회귀 구현하기

2. 사이킷런을 사용하여 학습하기

2.3 데이터 분리

- 데이터를 x데이터와 y데이터로 나눈 후, 이를 다시 훈련(train)과 테스트(test) 형태로 분류해야 함.

In [7]:	x_data = df_onehot.iloc[:, 1:].values y_data = df_onehot.iloc[:, 0].values.reshape(-1, 1) y_data.shape, x_data.shape
Out [7]:	((19583, 1), (19583, 37))

03 로지스틱 회귀 구현하기

2. 사이킷런을 사용하여 학습하기

2.3 데이터 분리

- X_data에 대해서 MinMaxScaler 함수를 사용하여 전체 데이터에 대한 스케일링을 실시

In [8]:	from sklearn import preprocessing # Min-Max Standardzation min_max_scaler = preprocessing.MinMaxScaler() x_data = min_max_scaler.fit_transform(x_data)

03 로지스틱 회귀 구현하기

2. 사이킷런을 사용하여 학습하기

2.3 데이터 분리

- train_test spilt 함수를 적용하여 생성된 데이터를 학습 데이터셋과 테스트 데이터셋으로 분리함

In [9]:	from sklearn.model_selection import train_test_split X_train, X_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.33, random_state=42) X_train.shape, X_test.shape
Out [9]:	((13120, 37), (6463, 37))

03 로지스틱 회귀 구현하기

2. 사이킷런을 사용하여 학습하기

2.3 데이터 분리

- LogisticRegression 클래스를 사용하여 학습된 모델을 생성함.
  - fit_intercept 매개변수: 절편을 생성

In [10]:	**from sklearn.linear_model import LogisticRegression** **logreg = LogisticRegression(fit_intercept=True)** **logreg.fit(X_train, y_train.flatten())**
Out [10]:	**LogisticRegression()**

03 로지스틱 회귀 구현하기

2. 사이킷런을 사용하여 학습하기

2.3 데이터 분리

- penalty 매개변수: regularization을 위하여
- Max_iter, tol 매개변수: 경사하강법이 가지는 매개변수

In [11]:	LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False)
Out [11]:	LogisticRegression(multi_class='warn', solver='warn')

03 로지스틱 회귀 구현하기

2. 사이킷런을 사용하여 학습하기

2.4 값 예측하기와 성능 측정하기

- 생성된 모델을 사용하여 실제값을 예측하기 위하여 predict 함수를 사용함.
  - X_test에 존재하는 값 5개까지 모두 0으로 예측됨

In [12]:	logreg.predict(X_test[:5])
Out [12]:	array([0, 0, 0, 0, 0], dtype=int64)

03 로지스틱 회귀 구현하기

2. 사이킷런을 사용하여 학습하기

2.4 값 예측하기와 성능 측정하기

- 로지스틱 회귀의 가설함수가 확률을 예측함
- 각 값들의 예측값에 대한 확률도 계산가능함
  - predict_proba: 0 일때의 확률과 1일 때의 확률을 n X 2형태의 행렬 형태로 나타냄.

In [13]:	logreg.predict_proba(X_test[:5])
Out [13]:	array([[0.56843258, 0.43156742], [0.91112572, 0.08887428], [0.79481085, 0.20518915], [0.85841562, 0.14158438], [0.62764603, 0.37235397]])

03 로지스틱 회귀 구현하기

2. 사이킷런을 사용하여 학습하기

2.4 값 예측하기와 성능 측정하기

- 실제 성능을 측정하는 코드
  - confusion_matrix와 accuracy_score와 같은 함수를 사용함

In [14]:	from sklearn.metrics import confusion_matrix from sklearn.metrics import accuracy_score y_true = y_test.copy() y_pred = logreg.predict(X_test) confusion_matrix(y_true, y_pred)
Out [14]:	array([[4487, 275], [1350, 351]], dtype=int64)
In [15]:	accuracy_score(y_true, y_pred)
Out [15]:	0.7485687761101656

04

다중클래스 분류와 소프트맥스 분류

04 다중클래스 분류와 소프트맥스 분류

1. 다중클래스 분류의 개념

- 다중클래스 분류(multi-class classification) : 2개 이상의 클래스를 가진 y 값에 대한 분류

1.1 다중클래스와 다중레이블

표 10-1 분류 작업에서 다중클래스와 다중레이블의 차이점

분류	다중클래스(multi-class) 분류	다중레이블(multi-label) 분류
작업	2개 이상의 클래스를 가진 분류 작업	상호 배타적이지 않은 속성 예측
중복 선택	중복 선택 불가능 → [1 0 0] 가능, [1 1 0] 불가	중복 선택 가능 → [1 1 0] 가능
예	과일 사진 분류 : 오렌지, 사과, 배	신문기사 분류 : 운동선수-연예인 결혼 기사 → 스포츠/연예 면

04 다중클래스 분류와 소프트맥스 분류

1. 다중클래스 분류의 개념

1.2 분류 접근

- One-vs-All : m개의 클래스가 존재할 때 각 클래스마다 분류기(classifier)를 생성하여 분류
  - One-vs-Rest라고도 부름
  - 대표적으로 소프트맥스 분류(softmax classification)
- One-vs-One : m개의 클래스가 있다면, 이 클래스의 분류기를 하나의 클래스로 하고 나머지 클래스의 분류기들을 만들어 최종적으로 각 분류기들의 결과를 투표로 결정
  - 총 $\frac{m(m-1)}{2}$ 개만큼의 분류기를 생성
  - 분류기가 많아질수록 정확도 높아지지만 비용도 증가

2. 소프트맥스 분류

2.1 소프트맥스 함수

- 시그모이드 함수로 다중클래스 분류 문제 다룰 수 있음
  - 각각의 클래스에 속하는지 속하지 않는지 이진분류기 m개를 생성한 후, 가장 높은 확률이 나오는 클래스를 선택
  - 분류기 번호 m에 대해 $h_m(x;\theta)$로 표현
  - 그러나 $h_m(x;\theta)$ 확률의 합이 1 이상이 된다는 문제 발생
  - 문제 해결 방법은 모든 클래스들의 발생 확률을 1로 정규화

04 다중클래스 분류와 소프트맥스 분류

2. 소프트맥스 분류

2.1 소프트맥스 함수

- 소프트맥스 함수(softmax function) : 다중클래스 분류에서 여러 선형회귀의 출력 결과를 정규화하여 합이 1이 되도록 만드는 함수

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \qquad \text{for } j = 1, 2, 3, \cdots, K$$

$$\sum_{j=1}^{K} \sigma(z)_j = \sum_{j=1}^{K} P_j = 1$$

표 10-2 소프트맥스 함수 값 정리

z_j	e^{z_j}	$\frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$
2	7.389	0.609
1	2.718	0.224
−1	0.367	0.030
0.5	1.648	0.135

2. 소프트맥스 분류

2.1 소프트맥스 함수

In [1]:
```
import numpy as np

def softmax(values):
    array_values = np.exp(values)
    return array_values / np.sum(array_values)

values = [2, 1, 5, 0.5]
y = softmax(values) # array([0.04613281, 0.01697131,
0.92660226, 0.01029362])
y.sum()
```

Out [1]:
```
1.0
```

05

다중클래스 분류를 코드로 구현하기

05 다중클래스 분류를 코드로 구현하기

1. mnist 데이터셋의 이해

- 손글씨를 숫자로 인식하는 이미지 분류 문제

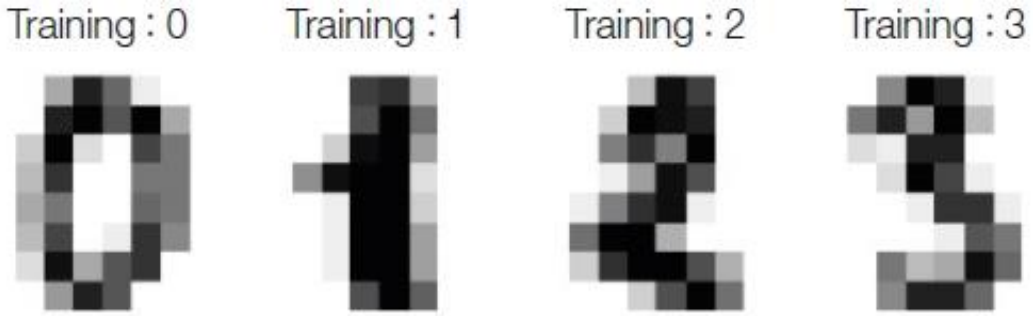


그림 10-1 사이킷런의 mnist 데이터셋 예제

- 컴퓨터는 이미지를 일종의 숫자로 변환하여 인식
  - 이미지를 일종의 점(dot)으로 생각하면 m×n만큼의 공간이 존재하고, 그 공간 안에서 색깔이 진할수록 높은 값, 색깔이 옅을수록 낮은 값을 가짐

05 다중클래스 분류를 코드로 구현하기

2. 데이터 불러오기

- datasets 모듈을 호출
- load_digits 함수로 딕셔너리 타입 데이터를 불러온다

In [1]:	from sklearn import datasets digit_dataset = datasets.load_digits() digit_dataset.keys()
Out [1]:	dict_keys(['data', 'target', 'frame', 'feature_names', 'target_names', 'images', 'DESCR'])

05 다중클래스 분류를 코드로 구현하기

2. 데이터 불러오기

- Out [2]
  - 1797: 데이터의 개수
  - 8, 8: 가로와 세로 각각 8칸씩 총 64칸 존재, 그 값이 모두 채워져 있음
- In [3]
  - 하나의 데이터만 확인
  - target 데이터의 0번째 값은 0임

In [2]:	**digit_dataset["images"].shape**
Out [2]:	**(1797, 8, 8)**
In [3]:	**digit_dataset["target"][0]**
Out [3]:	**0**

05 다중클래스 분류를 코드로 구현하기

2. 데이터 불러오기

- images 데이터의 0번째 값을 출력하면 배열인 array 형태로 나타남
  - 각 숫자가 클수록 실제 검정색에 가까운 값이 출력

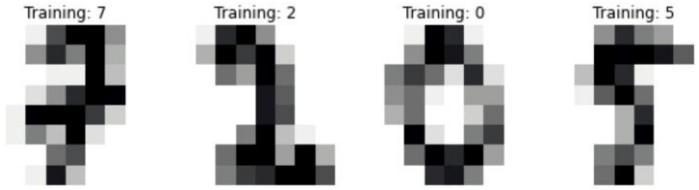
In [4]:	digit_dataset["images"][0]
Out [4]:	array([[0., 0., 5., 13., 9., 1., 0., 0.], [0., 0., 13., 15., 10., 15., 5., 0.], [0., 3., 15., 2., 0., 11., 8., 0.], [0., 4., 12., 0., 0., 8., 8., 0.], [0., 5., 8., 0., 0., 9., 8., 0.], [0., 4., 11., 0., 1., 12., 7., 0.], [0., 2., 14., 5., 10., 12., 0., 0.], [0., 0., 6., 13., 10., 0., 0., 0.]])

05 다중클래스 분류를 코드로 구현하기

2. 데이터 불러오기

In [5]:
```
import matplotlib.pyplot as plt
from random import randint
_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3)) # (1) subplots 4개 생성
for ax in axes: # (2) 각 subplot에 들어갈 숫자를 위해 for문으로 값 생성
    num = randint(1, 1000) # (3) 1~1000 사이의 숫자를 랜덤하게 선택
    image = digit_dataset["images"][num]
    label = digit_dataset["target"][num]
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title('Training: %i' % label)
```

Out [5]:



[TIP] 결과값에 색을 지정하는 요소(property)인 plt.cm.gray_r을 변경하면 좀 더 다양한 형태로 값 표현이 가능하다

05 다중클래스 분류를 코드로 구현하기

2. 데이터 불러오기

- 데이터가 8×8 행렬이므로 2D 이미지로 표현되었지만 다음 코드와 같이 총 64개의 피쳐(feature)를 가진 하나의 데이터로 받을 수 있음

In [6]:	**digit_dataset["data"][0].shape**
Out [6]:	**(64,)**

05 다중클래스 분류를 코드로 구현하기

3. 데이터 분류하기

- 데이터를 훈련 데이터셋과 테스트 데이터셋으로 구분

In [7]:
```
from sklearn.model_selection import train_test_split

X = digit_dataset["data"] # (1) data에 있는 값들 X에 할당
y = digit_dataset["target"] # (2) target에 있는 값들 y에 할당
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

05 다중클래스 분류를 코드로 구현하기

4. 모델 생성하기

- ovr : 클래스 모드를 모두 이진모델로 만들어 학습
- multinomial : 소프트맥스 함수를 사용하여 계산하는 방식. 경사하강법의 매개변수 solver를 sag으로 변경

In [8]:	from sklearn.linear_model import LogisticRegression logreg_ovr = LogisticRegression(multi_class="ovr") logreg_softmax = LogisticRegression(multi_class="multinomial", solver="sag") logreg_ovr.fit(X_train, y_train) logreg_softmax.fit(X_train, y_train)
Out [8]:	LogisticRegression(multi_class='multinomial', solver='sag')

05 다중클래스 분류를 코드로 구현하기

5. 성능 측정하기

- 일반적으로 다중클래스 분류도 기존 혼동행렬을 사용
- 각 클래스 대비 예측한 값을 행렬 형태로 표현

In [9]:
```
from sklearn.metrics import confusion_matrix
y_pred = logreg_ovr.predict(X_test).copy()
y_true = y_test.copy()
confusion_matrix(y_true, y_pred)
```

Out [9]:
```
array([[47, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [ 0, 49, 1, 0, 0, 0, 0, 0, 0, 0],
       [ 0, 0, 49, 2, 0, 0, 0, 0, 0, 0],
       [ 0, 0, 0, 37, 0, 1, 0, 0, 1, 0],
       [ 0, 0, 0, 0, 41, 0, 0, 0, 0, 0],
       [ 0, 1, 0, 0, 1, 41, 1, 0, 0, 1],
       [ 0, 0, 0, 0, 2, 0, 36, 0, 0, 0],
       [ 0, 0, 0, 0, 1, 0, 0, 41, 0, 1],
       [ 0, 0, 0, 0, 0, 0, 0, 0, 44, 0],
       [ 1, 0, 0, 0, 0, 0, 0, 0, 1, 50]], dtype=int64)
```

05 다중클래스 분류를 코드로 구현하기

5. 성능 측정하기

- 라벨별로 분류 성능을 수치화하여 표시

In [10]:

```
from sklearn.metrics import classification_report
print(classification_report(y_true, y_pred))
```

Out [10]:

```
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        47
           1       0.98      0.98      0.98        50
           2       0.98      0.96      0.97        51
           3       0.95      0.95      0.95        39
           4       0.91      1.00      0.95        41
           5       0.98      0.91      0.94        45
           6       0.97      0.95      0.96        38
           7       1.00      0.95      0.98        43
           8       0.96      1.00      0.98        44
           9       0.96      0.96      0.96        52

    accuracy                           0.97       450
   macro avg       0.97      0.97      0.97       450
weighted avg       0.97      0.97      0.97       450
```

05 다중클래스 분류를 코드로 구현하기

5. 성능 측정하기

- **micro**를 선택하면 전체 평균값
  - 각 라벨별로 False Positive와 True Positive 값을 모두 더해서 True Positive 값으로 나눈 값
- **macro**를 선택하면 각 라벨별 결과의 합에 대한 평균을 나타냄
  - classificatrion_report의 각 라벨별 평균, 즉 avg값

In [11]:	**result = confusion_matrix(y_true, y_pred)** **result.diagonal().sum() / result.sum(axis=0).sum()**
Out [11]:	**0.953333333333333**

05 다중클래스 분류를 코드로 구현하기

5. 성능 측정하기

- 각 라벨별의 데이터 개수의 차이가 난다면 micro 선택 결과 중요
- 그렇치 않은 경우는 macro로 선택하여 라벨의 평균적인 성능을 나타냄

In [12]:	from sklearn.metrics import precision_score precision_score(y_true, y_pred, average="micro")
Out [12]:	0.9666666666666667
In [13]:	precision_score(y_true, y_pred, average="macro")
Out [13]:	0.9666219376328072
In [14]:	precision_score(y_true, y_pred, average=None)
Out [14]:	array([0.97916667, 0.98 , 0.98 , 0.94871795, 0.91111111, 0.97619048, 0.97297297, 1. , 0.95652174, 0.96153846])

<서포트 벡터 머신>

01

서포트 벡터 머신

목차

01 서포트 벡터 머신이란?[2]

1. 서포트 벡터 머신의 소개

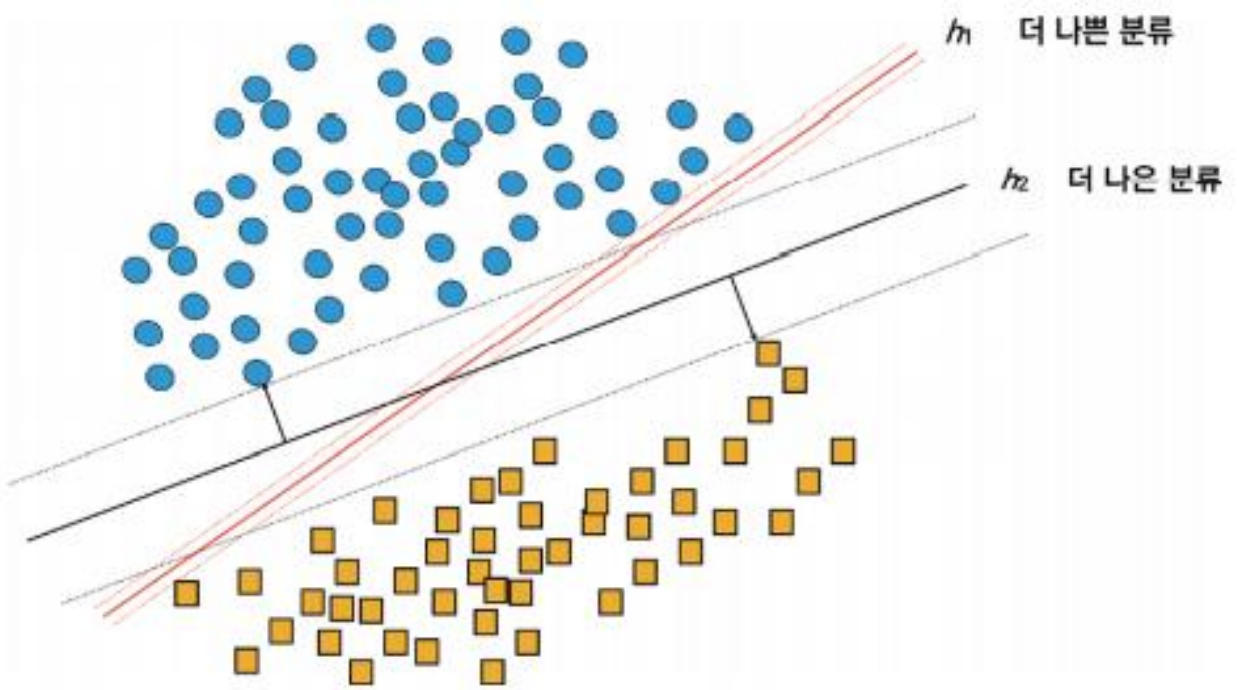
- 서포트 벡터 머신(support vector machine, SVM)
  - 인공 신경망이 딥러닝(deep learning)을 통해 인공지능 분야의 중심으로 떠오르기 전에 가장 각광받던 학습 방법 중의 하나

[2] 으뜸 머신러닝, 강영민, 박동규, 김성수, 생능출판

01 서포트 벡터 머신이란?

1. 서포트 벡터 머신의 소개

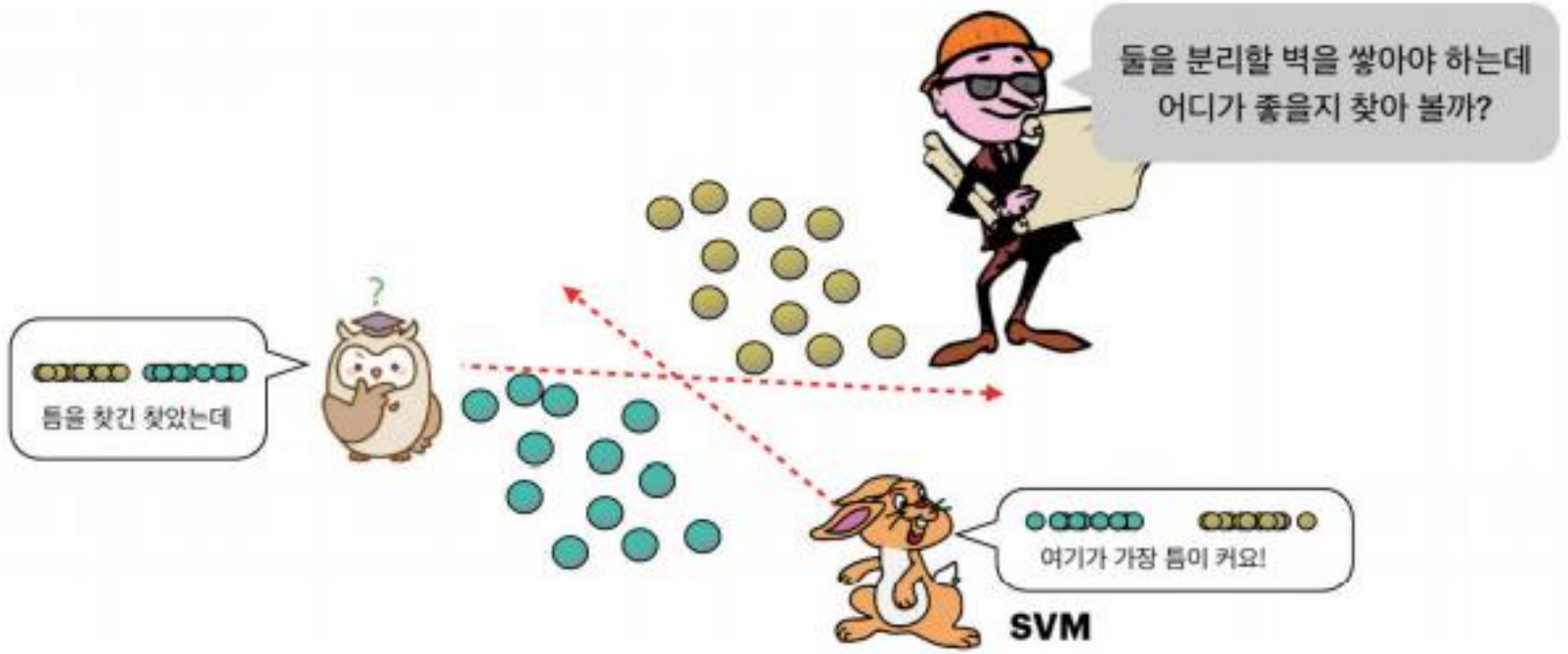
- 파란색 원과 노란색 사각형은 서로 다른 그룹에 속한 데이터들임. 이들을 구분하는 초평면(hyperplane)은 여러 개 존재
  - 여기서는 h1 과 h2 라는 두 개의 직선으로 표현, h2가 더 나은 분류
- 좋은 분리 평면은 새로운 데이터가 들어왔을 때에도 판정을 잘 할 수 있는 평면
- 초평면을 화살표로 표시된 법선(normal) 벡터 방향으로 움직였을 때 데이터에 닿지 않는 폭이 넓을 수록 좋음



01 서포트 벡터 머신이란?

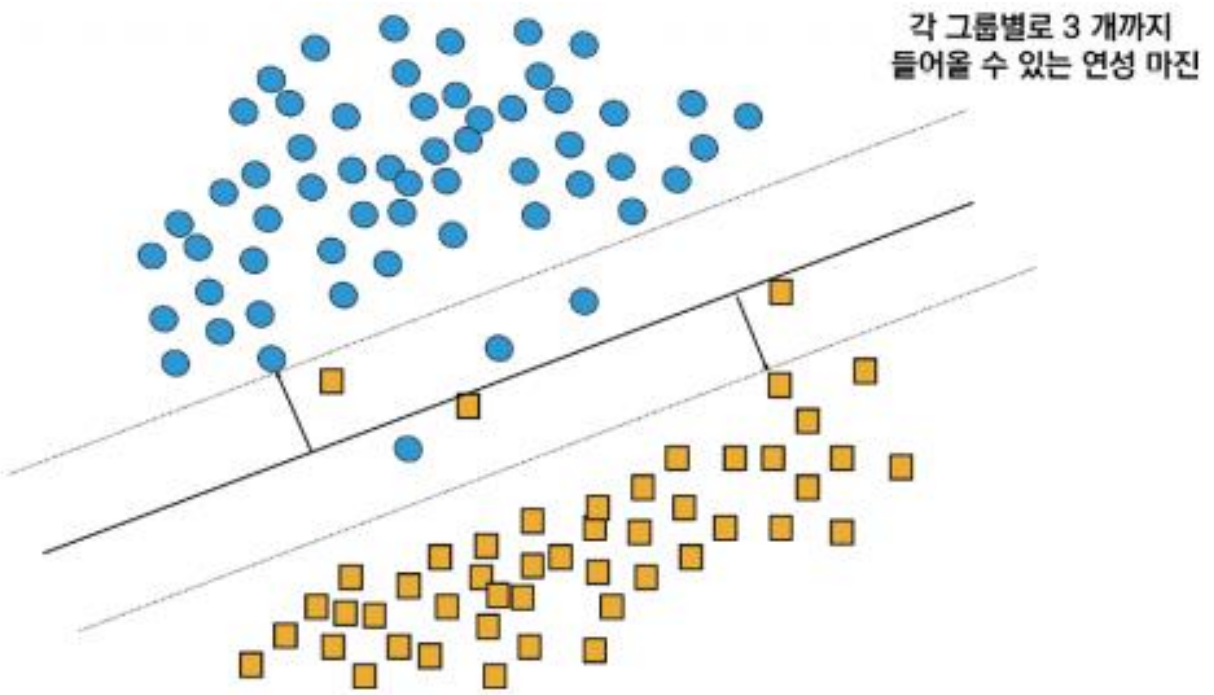
1. 서포트 벡터 머신의 소개

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



01 서포트 벡터 머신이란?

1. 서포트 벡터 머신의 소개



- 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 최대로 넓게 만들려고 하기 때문에 마진의 양쪽에는 서로 다른 그룹에 속하는 데이터들이 하나씩 닿아 있으며 이것을 **서포트 벡터(support vector)** 라고 함

01 서포트 벡터 머신이란?

1. 서포트 벡터 머신의 소개

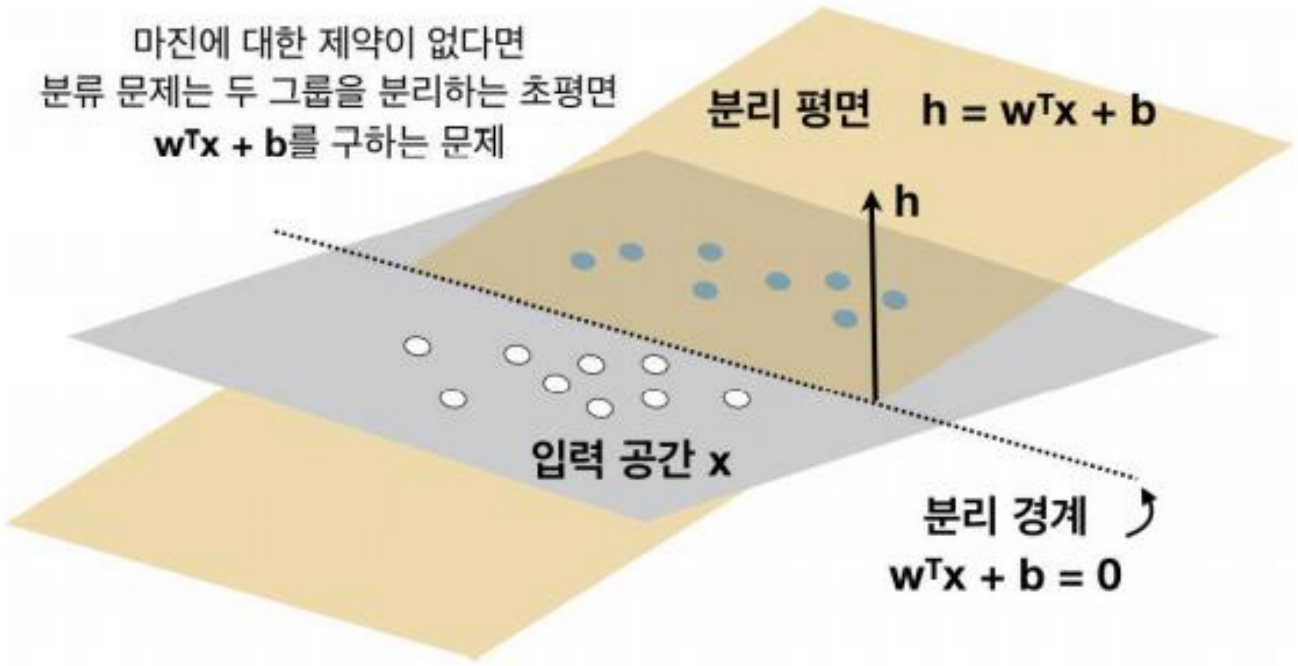
1.1 정의 및 특성

- 하드 마진을 사용할 경우에는 분류가 안될수도 있고, 잡음에 민감할 수밖에 없으므로 **소프트 마진을 사용하는 것이 바람직**
  - **잡음에 민감하다는 것은 데이터에 과적합된다는 의미**
  - 소프트 마진을 사용하는 것도 모델 정칙화의 일종
- 슬랙(slack) : 소프트 마진을 사용할 때는 마진 내에 들어갈 수 있는 데이터의 수를 제어하며 이 값을 제어하는 변수

01 서포트 벡터 머신이란?

2. 하드 마진 서포트 벡터 머신의 구현

- 서포트 벡터 머신의 기본적인 동작은 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 h를 $w^T\mathrm{x} + b$라고 정의하면, **$\boldsymbol{h = w^T\mathrm{x} + b}$ 은 노란색으로 비스듬하게 표시된 초평면**



01 서포트 벡터 머신이란?

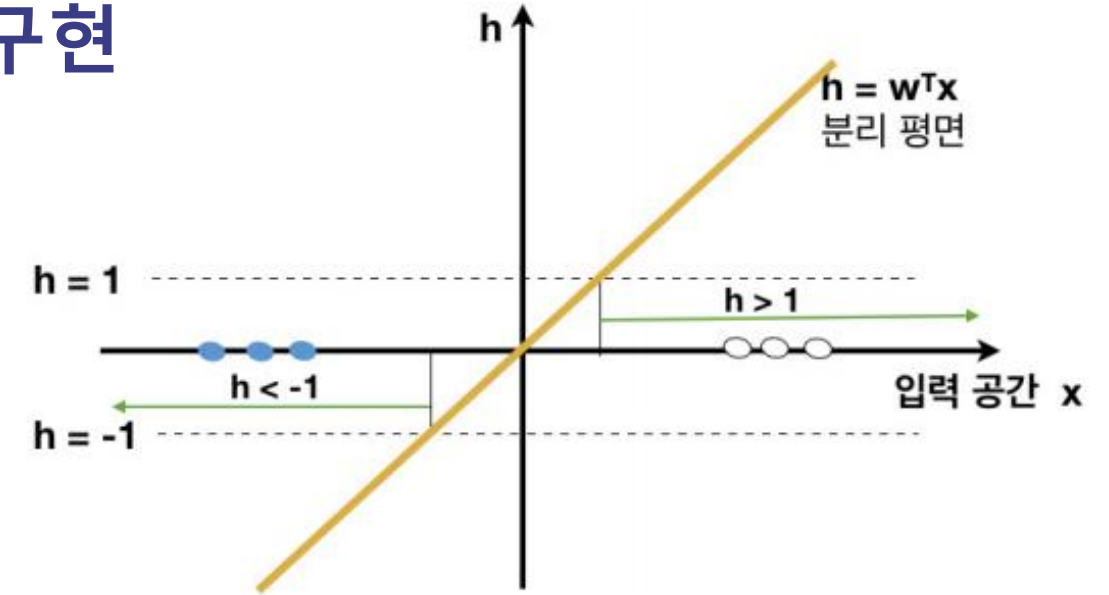
2. 하드 마진 서포트 벡터 머신의 구현

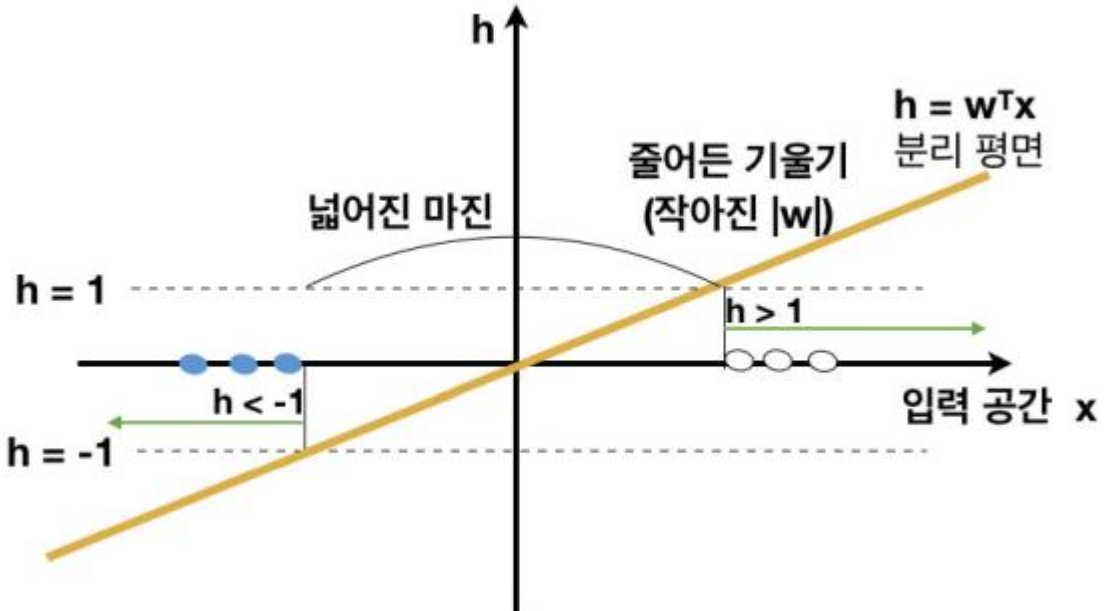
- 마진(margin)에 대한 제약 조건이 없다면, 서포트 벡터 머신은 흰색 레이블의 데이터들은 h 가 음수, 푸른색 레이블의 데이터들은 h 가 양수가 되게 하는 w 와 b 를 찾으면 됨
- 답이 되는 평면이 하나가 아니며 이런 경우에는 가장 "좋은" 평면을 찾아야함
  - 서포트 벡터 머신에서는 마진의 넓이가 큰 값이 될수록 좋은 답

01 서포트 벡터 머신이란?

2. 하드 마진 서포트 벡터 머신의 구현

- 평면을 수직선에 가깝게 눕혀 보자.
  - 직선으로 생각하면 기울기에 해당하는 w 벡터 가 0에 가까워지는 것.
  - 이러면 마진이 점점 넓어지는 것을 확인할 수 있음





2. 하드 마진 서포트 벡터 머신의 구현

- 다음과 같은 제약 조건을 가진 최적화 문제가 된다.

하드 마진 SVM의 최적화 문제

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

제약조건: $i = 1, 2, \cdots, m$ 인 m개 데이터 인스턴스 $\mathbf{x}^{(i)}$ 모두에 $\left|\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)} + b\right| \geq 1$

01 서포트 벡터 머신이란?

3. 소프트 마진 서포트 벡터 머신의 구현

- 소프트 마진에는 슬랙(slack) 변수가 사용됨
- 이 슬랙 변수는 각 데이터 인스턴스마다 정의되므로 m개의 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 $\zeta^{(i)} = 0$

3. 소프트 마진 서포트 벡터 머신의 구현

- 슬랙 변수에 의해 각각의 데이터 인스턴스에 대해 제약 조건이 다음과 같이 변경

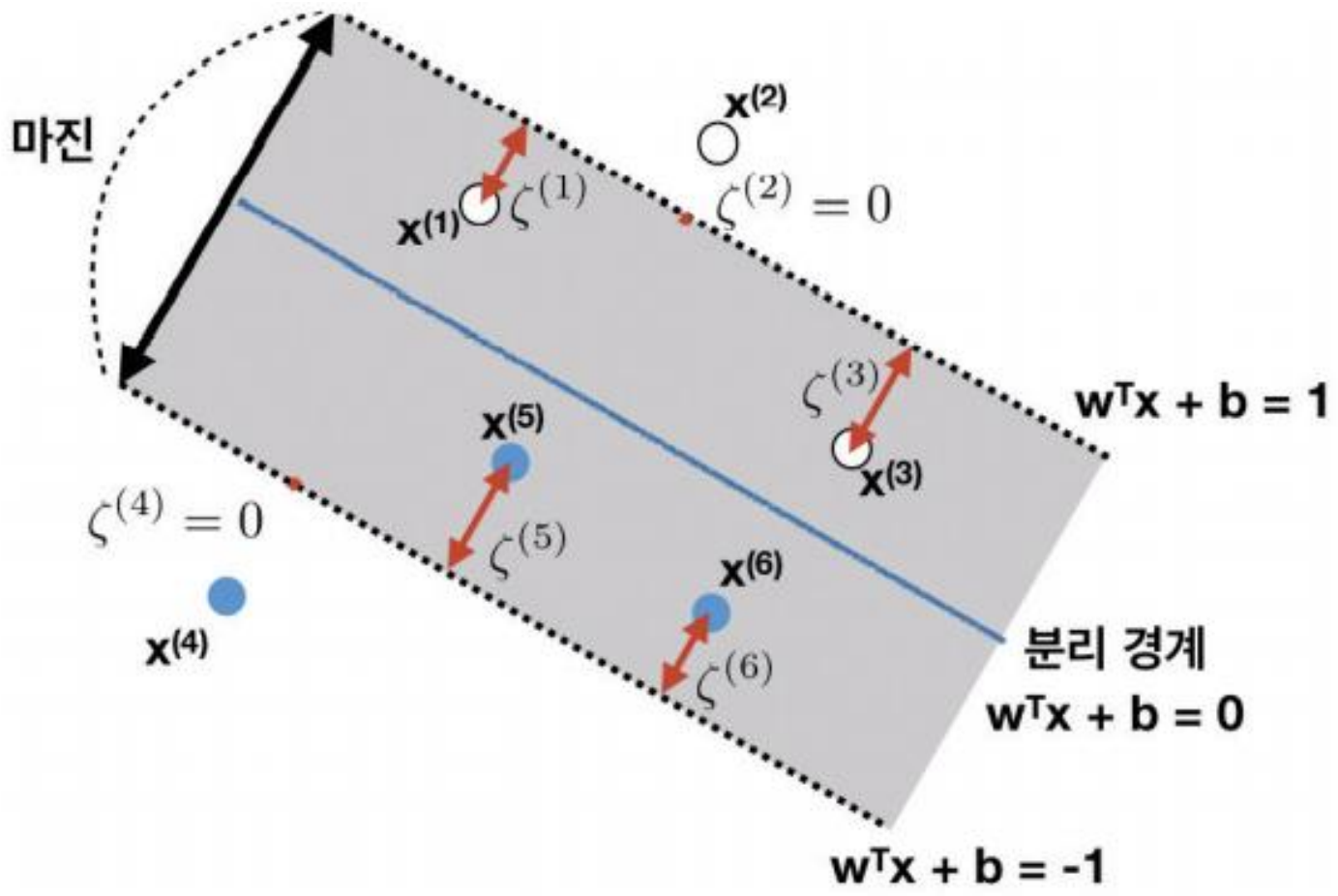
$i = 1, 2, \cdots, m$ 인 m 개 데이터 인스턴스 $\mathbf{x}^{(i)}$ 모두에 대해

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

01 서포트 벡터 머신이란?

3. 소프트 마진 서포트 벡터 머신의 구현

- 그림을 설명하면 다음과 같다.
- 각각의 데이터 인스턴스는 마진 밖이나 안에 존재할 수 있음
- 데이터 인스턴스가 마진 내에 많이 들어올수록 슬랙 변수의 값이 커지는 것
  - 좋은 분류를 위해서는 슬랙 변수를 최소로 만드는 최적화 문제를 품



01 서포트 벡터 머신이란?

3. 소프트 마진 서포트 벡터 머신의 구현

- 사이킷런을 이용하여 실제로 서포트 벡터 머신을 이용하여 데이터를 구분

In [1]:
```
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)
```

Out [1]:

	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

01 서포트 벡터 머신이란?

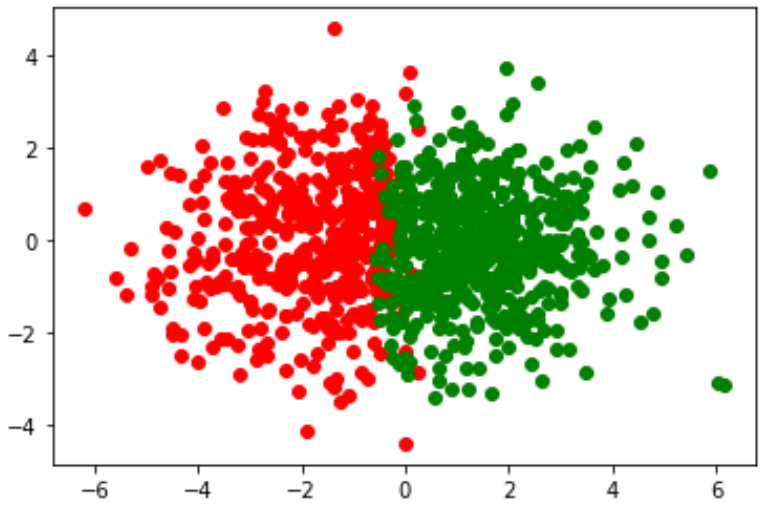
3. 소프트 마진 서포트 벡터 머신의 구현

- 사이킷런을 이용하여 실제로 서포트 벡터 머신을 이용하여 데이터를 구분

In [2]:
```
df_positive = df[df['y']>0]      # y가 1인 데이터만 출력
df_negative = df[df['y']==0]     # y가 0인 데이터만 출력
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')
```

Out [2]:

```
<matplotlib.collections.PathCollection at 0x1836c6961c0>
```



01 서포트 벡터 머신이란?

3. 소프트 마진 서포트 벡터 머신의 구현

- 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(hinge) 손실 함수
  - 평균 제곱 오차를 쓰기 위해 'mse' 등을 지정하면 오류 ('hinge' 혹은 'squared_hinge'만 가능)

In [3]:

```
from sklearn.svm import LinearSVC
X = df[['x1', 'x2']].to_numpy() # X1, X2를 입력 벡터로 함
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
```

Out [3]:

```
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)
```

01 서포트 벡터 머신이란?

3. 소프트 마진 서포트 벡터 머신의 구현

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

In [4]:	svm_simple.predict([[0.12, 0.56], [-4, 40], [0, 40], [5,20]])
Out [4]:	array([0, 1, 1, 0], dtype=int64)

Assignment

Assignment

- 본인이 머신러닝 기술을 사용하여 구현하고 싶은 시스템에 대하여 간략하게 서술하시오. (A4용지 1장 정도)

예제) 머신 러닝 기술을 이용한 인간-로봇 상호작용 시스템

Assignment

- 강의 PPT 35~49쪽 사이의 코드 In [1] ~ In [15]의 코드를 실행시킨 후 각 결과를 화면 캡쳐하여 제출하시오. (한글, 워드, PPT 등 이용 가능)

Thank You !