

07. Classification

- Classification이란?
- Logistic Regression(Classification)
- Multinomial Logistic Regression(Classification)

Classification이란?

Classification(분류)

■ 분류(Classification) 문제

- 독립변수값이 주어졌을 때 그 값과 가장 연관성이 큰 종속변수값 (클래스)을 예측하는 문제
- 즉, 어떤 표본에 대한 데이터가 주어졌을 때 그 표본이 어떤 카테고리 혹은 클래스에 속하는지를 알아내는 문제

■ 분류모형

■ 확률적 모형

- 각 카테고리 혹은 클래스가 정답일 조건부확률(conditional probability)을 계산하는 모형

■ 판별함수 모형

- 주어진 데이터를 카테고리에 따라 서로 다른 영역으로 나누는 경계면(decision boundary)을 찾아낸 다음 이 경계면으로부터 주어진 데이터가 어느 위치에 있는지를 계산하는 판별함수(discriminant function)를 이용하는 모형

(1) 확률적 모형

- 출력변수 y 가 K 개의 클래스 $1, 2, \dots, K$ 중 하나의 값을 가진다고 가정하면

1. 입력 x 가 주어졌을 때,
2. 출력변수 y 가 클래스 k 가 될 확률 $P(y = k|x)$ 을 계산

$$P_1 = P(y = 1|x)$$

$$P_2 = P(y = 2|x)$$

...

$$P_K = P(y = K|x)$$

3. 이 중에서 가장 확률이 큰 클래스를 선택

$$\hat{y} = \operatorname{argmax}_k P(y = k|x)$$

■ 조건부 확률 $P(y = k|x)$ 을 사용하는 분류모형

- 독립변수 x 가 주어지면 종속변수 y 의 모든 카테고리 값에 대해 조건부확률 또는 조건부확률의 로그값을 계산
- Scikit-learn 모듈은 다음과 같은 함수 지원
 - `predict_proba()`
 - `predict_log_proba()`

■ 조건부확률을 계산하는 방법

- 생성모형 (generative model)
- 판별모형 (discriminative model)

■ 확률적 생성모형

- 확률분포 $P(x|y = k)$ 을 추정한 다음 베이즈 정리 (Bayes' Theorem)를 사용하여 $P(y = k|x)$ 를 계산하는 방법
 - Naive Bayes Classifier 등

$$P(y = k|x) = \frac{P(x|y = k) P(y = k)}{P(x)}$$

- 클래스가 많은 경우 모든 클래스에 대해 $P(x|y = k)$ 를 추정하는 것은 많은 데이터를 필요로 할 뿐더러 최종적으로는 사용하지도 않을 확률분포를 계산하는데 계산량을 너무 많이 필요로 한다는 단점이 있음

■ 확률적 판별모형

- 확률적 생성모형은 조건부확률 $P(y|x)$ 를 구하기 위해 우선 likelihood $P(x|y)$ 를 구하고 베이즈 정리를 사용하여 조건부확률을 계산
- 반면, 확률적 판별모형은 조건부확률 $P(y = 1|x)$ 이 x 에 대한 함수 $f(x)$ 로 표시될 수 있다고 가정하고 그 함수를 직접 찾아내는 방법

$$P(y = k|x) = f(x), \quad 0 \leq f(x) < 1$$

- logistic regression 등

Naive Bayes Classifier

Weather	Play
Sunny	No
Sunny	No
Overcast	Yes
Rainy	Yes
Rainy	Yes
Rainy	No
Overcast	Yes
Sunny	No
Sunny	Yes
Rainy	Yes
Sunny	Yes
Overcast	Yes
Overcast	Yes
Rainy	No

■ 예제 데이터

■ 날씨에 따른 운동여부 데이터(총 14건)

- 맑은날씨(Sunny) : Yes(3일), No(2일)
- 흐린날씨(Overcast) : Yes(4일), No(0일)
- 비오는날(Rainy) : Yes(2일), No(3일)

■ 우도(likelihood)

- $P(Yes) = \frac{9}{14} = 0.64$, $P(No) = \frac{5}{14} = 0.36$
- $P(Sunny) = \frac{5}{14} = 0.36$, $P(Overcast) = \frac{4}{14} = 0.29$, $P(Rainy) = \frac{5}{14} = 0.36$
- $P(Sunny|Yes) = \frac{P(Sunny \cap Yes)}{P(Yes)} = \frac{2}{9} = 0.22$, $P(Sunny|No) = \frac{P(Sunny \cap No)}{P(No)} = \frac{3}{5} = 0.6$
- $P(Overcast|Yes) = \frac{P(Overcast \cap Yes)}{P(Yes)} = \frac{4}{9} = 0.44$
- $P(Rainy|Yes) = \frac{P(Rainy \cap Yes)}{P(Yes)} = \frac{3}{9} = 0.33$, $P(Rainy|No) = \frac{P(Rainy \cap No)}{P(No)} = \frac{2}{5} = 0.4$

■ 흐린날이면, 운동할 클래스에 속할까? 아닐까?

■ 사전에 계산된 확률

- $P(Yes) = 0.64$, $P(No) = 0.36$
- $P(Overcast) = 0.29$
- $P(Overcast|Yes) = 0.44$

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)}$$

우도(likelihood)

사전확률

사후확률

■ 따라서

■ 흐린날 운동할 확률

$$P(Yes|Overcast) = \frac{P(Overcast|Yes)P(Yes)}{P(Overcast)} = \frac{0.44 * 0.64}{0.29} = 0.97$$

■ 흐린날 운동하지 않을 확률

$$P(No|Overcast) = \frac{P(Overcast|No)P(No)}{P(Overcast)} = \frac{0.0 * 0.36}{0.29} = 0.0$$

■ 이 두가지 확률을 비교하여 확률이 높은 것으로 분류

■ Scikit-learn에서 제공하는 나이브베이지스 분류기

- GaussianNB: 정규분포 나이브베이지스
- BernoulliNB: 베르누이분포 나이브베이지스
- MultinomialNB: 다항분포 나이브베이지스

```
from sklearn.preprocessing import LabelEncoder
```

```
weather = ['Sunny', 'Sunny', 'Overcast', 'Rainy', 'Rainy',  
           'Rainy', 'Overcast', 'Sunny', 'Sunny', 'Rainy',  
           'Sunny', 'Overcast', 'Overcast', 'Rainy']
```

```
play = ['No', 'No', 'Yes', 'Yes', 'Yes',  
        'No', 'Yes', 'No', 'Yes', 'Yes',  
        'Yes', 'Yes', 'Yes', 'No']
```

```
le = LabelEncoder()
```

```
X = le.fit_transform(weather)
```

```
y = le.fit_transform(play)
```

```
print(X) # 0:Overcast, 1:Rainy, 2:Sunny
```

```
print(y) # 0:No, 1:Yes
```

```
[2 2 0 1 1 1 0 2 2 1 2 0 0 1]  
[0 0 1 1 1 0 1 0 1 1 1 1 1 0]
```

```
from sklearn.naive_bayes import GaussianNB
```

```
model = GaussianNB()
```


```
model.fit(X.reshape(-1,1), y.reshape(-1,1))
```

```
# Predict
```

```
predicted = model.predict([[0]]) # 0:Overcast
```

```
print("Predicted Value:", predicted)
```

```
Predicted Value: [1]
```



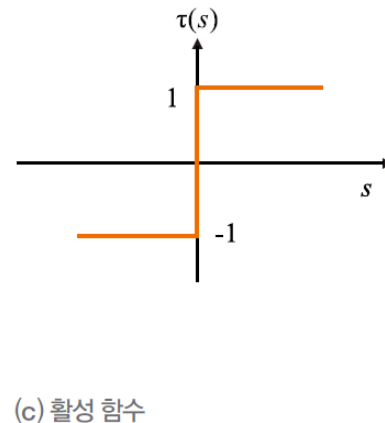
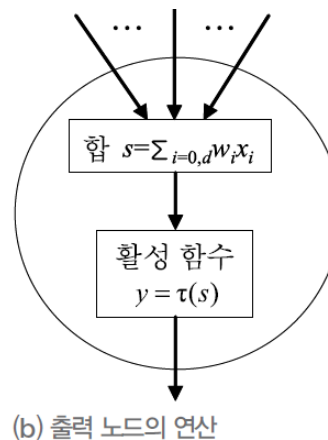
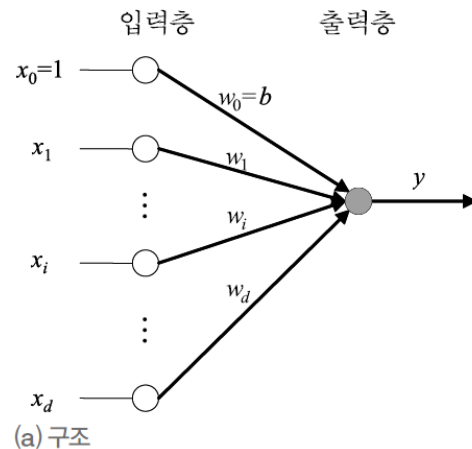
[[2]	[[0]
[2]	[0]
[0]	[1]
[1]	[1]
[1]	[1]
[1]	[0]
[0]	[1]
[2]	[0]
[2]	[1]
[1]	[1]
[2]	[1]
[0]	[1]
[0]	[1]
[1]]	[0]]

(2) 판별함수 모형

- 동일한 클래스가 모여 있는 영역과 그 영역을 나누는 경계면(boundary plane)을 정의하는 것
 - Perceptron
 - SVM(support vector machine)
 - Neural Network 등
- 경계면의 정의
 - 경계면으로부터의 거리를 계산하는 판별함수(discriminant function) $f(x)$ 로 정의
 - (예) 이진분류의 경우
 - 판별경계선 $f(x) = 0$
 - class 1 : $f(x) > 0$
 - class 0 : $f(x) \leq 0$

■ 퍼셉트론(Perceptron)

- 입력은 특징 벡터 $\mathbf{x}=(x_1, x_2, \dots, x_d)$
- \mathbf{x} 를 두 개의 부류 ω_1 과 ω_2 중의 하나로 분류하는 이진 분류기
- 뇌 구조를 모방한 계산 모형
 - 1950년대 Rosenblatt의 퍼셉트론
 - 1980년대 퍼셉트론을 확장한 다층퍼셉트론(MLP)
 - 일반화 능력이 뛰어남



Perceptron을 Python으로 직접 구현

```
import numpy as np

class Perceptron():
    def __init__(self, eta=0.01, n_iter=50, random_state=1):
        self.eta = eta
        self.n_iter = n_iter
        self.random_state = random_state

    def predict(self, X):
        hyperthesis = np.dot(X, self.w_[1:]) + self.w_[0]
        return np.where(hyperthesis>0.0, 1, -1)

    def fit(self, X, y):
        다음장에.....
```

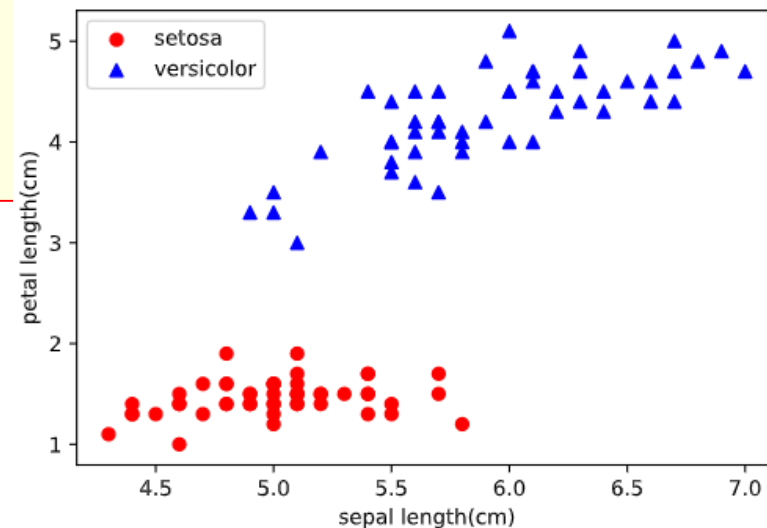
```
def fit(self, X, y):
    rgen = np.random.RandomState(self.random_state) # 결과의 재현
    self.w_ = rgen.normal(loc=0.0, scale=0.01, size=X.shape[1]+1)
    self.errors_ = []
    for _ in range(self.n_iter):
        errors = 0
        for features, target in zip(X, y):
            update = self.eta * (target - self.predict(features))
            self.w_[1:] += update * features
            self.w_[0] += update
            errors += int(update != 0.0)
        self.errors_.append(errors)
    return self
```


데이터 준비

```
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

iris = load_iris()
X = iris["data"][0:100, (0,2)] # 처음 100개의 데이터 중에서 꽃받침 길이, 꽃잎 길이
y = iris["target"][0:100] # 처음 50개는 Iris-setosa, 다음 50개는 Iris-versicolor
y = np.where(y==0, -1, 1) # 만일 Iris-setosa이면 -1, 아니면 1로 변경

plt.scatter(X[0:50, 0], X[0:50, 1], color="r", marker="o", label="setosa")
plt.scatter(X[50:100, 0], X[50:100, 1], color="b", marker="^", label="versicolor")
plt.xlabel('sepal length(cm)')
plt.ylabel('petal length(cm)')
plt.legend()
plt.show()
```



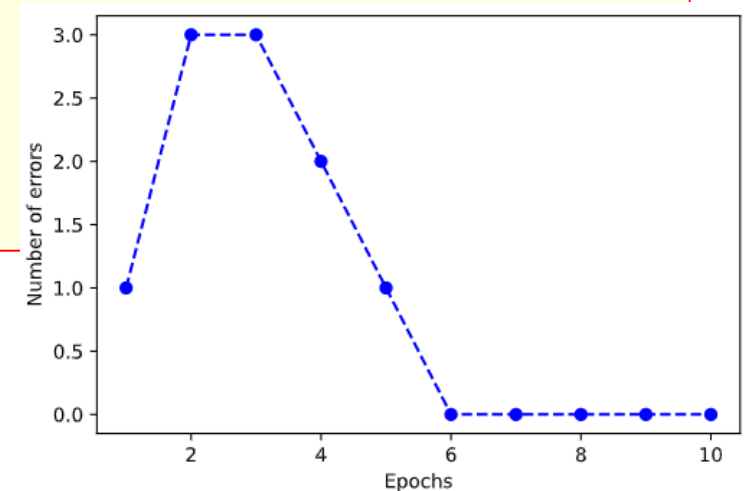
학습결과

```
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

iris = load_iris()
X = iris["data"][:,0:2] # 처음 100개의 데이터 중에서 꽃받침 길이, 꽃잎 길이
y = iris["target"][:,0] # 처음 50개는 Iris-setosa, 다음 50개는 Iris-versicolor
y = np.where(y==0, -1, 1) # 만일 Iris-setosa이면 -1, 아니면 1로 변경

p = Perceptron(eta=0.1, n_iter=10)
p.fit(X, y)

plt.plot(range(1, len(p.errors_)+1), p.errors_, "bo--")
plt.xlabel('Epochs')
plt.ylabel('Number of errors')
plt.show()
```



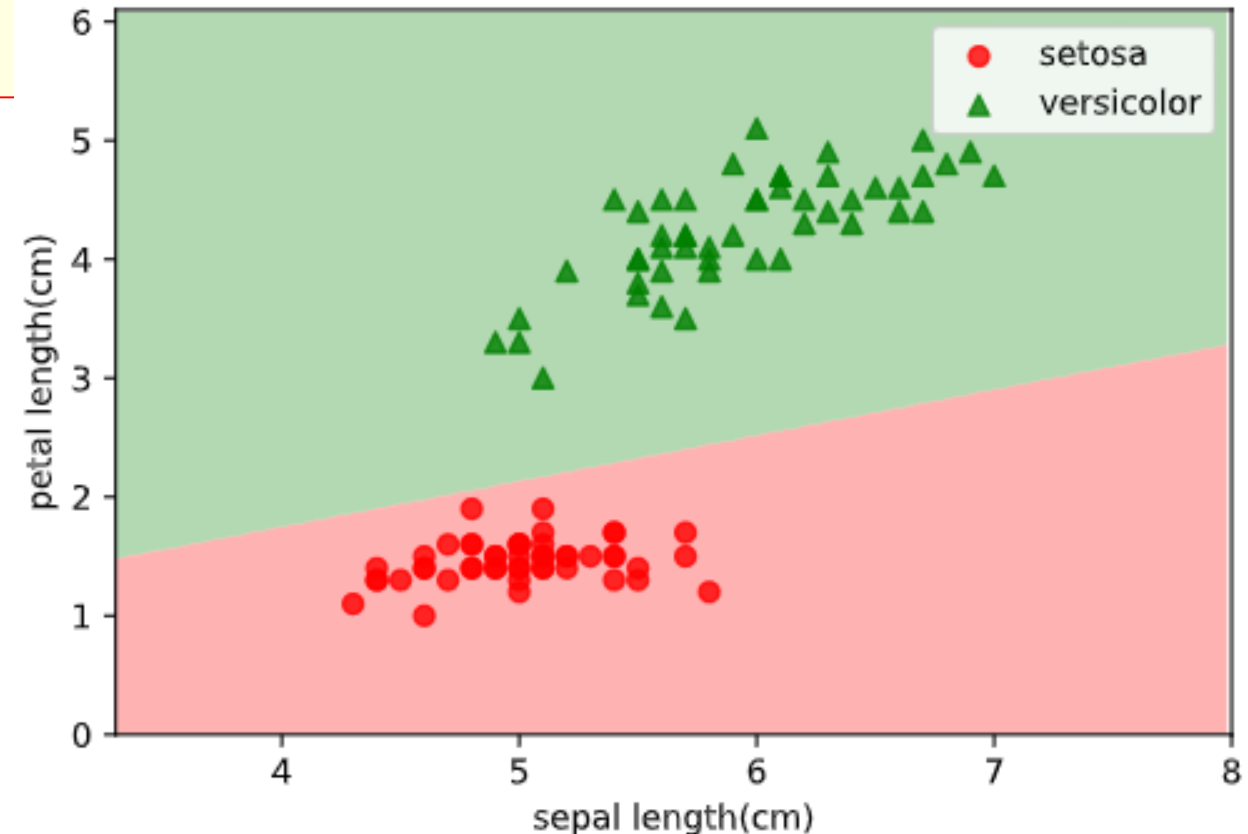
참고: 결정영역 그리기

```
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

def plot_decision_region(X, y, classifier):
    markers = ('o', '^', 'x')
    colors = ('red', 'green', 'blue')
    cmap = ListedColormap(colors[:len(np.unique(y))])
    x_min, x_max = X[:, 0].min()-1, X[:, 0].max()+1
    y_min, y_max = X[:, 1].min()-1, X[:, 1].max()+1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02), np.arange(y_min, y_max, 0.02))
    Z = classifier.predict(np.array([xx.ravel(), yy.ravel()]).T)
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, alpha=0.3, cmap=cmap)
    plt.axis([x_min, x_max, y_min, y_max])
    for idx, target in enumerate(np.unique(y)):
        plt.scatter(X[y==target, 0], X[y==target, 1],
                    alpha=0.8, c=colors[idx], marker=markers[idx],
                    label=np.where(target==1, 'setosa', 'versicolor'))
```

```
iris = load_iris()
X = iris["data"][0:100, (0,2)] # 처음 100개의 데이터 중에서 꽃받침 길이, 꽃잎 길이
y = iris["target"][0:100] # 처음 50개는 Iris-setosa, 다음 50개는 Iris-versicolor
y = np.where(y==0, -1, 1) # 만일 Iris-setosa이면 -1, 아니면 1로 변경
```

```
plot_decision_region(X, y, classifier=p)
plt.xlabel('sepal length(cm)')
plt.ylabel('petal length(cm)')
plt.legend()
plt.show()
```



참고: sklearn.linear_model.Perceptron 사용하기

```
from sklearn.datasets import load_iris
from sklearn.linear_model import Perceptron
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = load_iris()
X = iris["data"][:100, (0,2)] # 처음 100개의 데이터 중에서 꽃받침 길이, 꽃잎 길이
y = iris["target"][:100] # 처음 50개는 Iris-setosa, 다음 50개는 Iris-versicolor
y = np.where(y==0, -1, 1) # 만일 Iris-setosa이면 -1, 아니면 1로 변경

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

ppn = Perceptron(eta0=0.1, max_iter=30, random_state=1) # random_state는 재현위해
ppn.fit(X_train, y_train)
y_pred = ppn.predict(X_test)

print(f"accuracy: {accuracy_score(y_test, y_pred)}")
```

Logistic Regression(Classification)

Binary Logistic Regression(Classification)

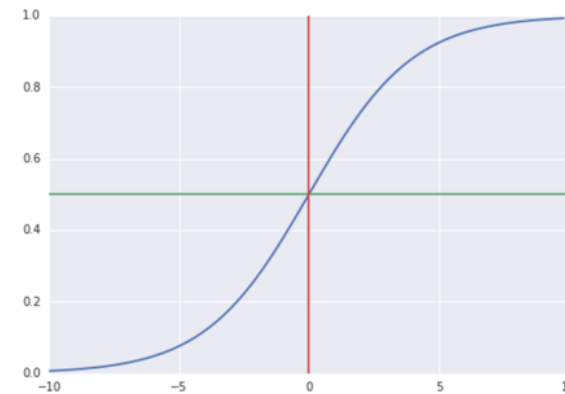
■ Class가 2개인 경우

- 주어진 데이터가 어느 클래스에 속하는가?
- Class(label)는 보통 0과 1로 encoding

■ Hypothesis

- $H(x) = Wx + b$
 - 이 가설은 0보다 작거나 1보다 큰 경우가 있음
 - 즉, 0 ~ 1사이의 값으로 만드는 함수가 필요
- Sigmoid Function
 - $g(z) = \frac{1}{1+e^{-z}}$
 - 0 ~ 1 사이의 값을 가짐

$$\blacksquare H(X) = \frac{1}{1+e^{-W^T X+b}}$$



■ 새로운 Cost Function에 대해...

- Sigmoid Function의 cost함수는 log함수로 정의될 수 있음

- $cost(W) = \frac{1}{m} \sum c(H(x), y)$

- $c(H(x), y) = \begin{cases} -\log(H(x)) & , y = 1 \\ -\log(1 - H(x)) & , y = 0 \end{cases}$

만일 0으로 분류하면, cost가 무한대.
그렇지 않으면, 0에 가까워짐

만일 1으로 분류하면, cost가 무한대.
그렇지 않으면, 0에 가까워짐

- 이 수식은 하나로 표현하면,

- $c(H(x), y) = -y \log(H(x)) - (1 - y) \log(1 - H(x))$

- $y=1$ 이면, $-\log(H(x))$

- $y=0$ 이면, $-\log(1 - H(x))$

- $cost(W) = -\frac{1}{m} \sum y \log(H(x)) + (1 - y) \log(1 - H(x))$

■ How to minimize cost?

■ cost function

- $cost(W) = -\frac{1}{m} \sum y \log(H(x)) + (1 - y) \log(1 - H(x))$

■ 경사하강알고리즘을 이용

- Gradient descent algorithm

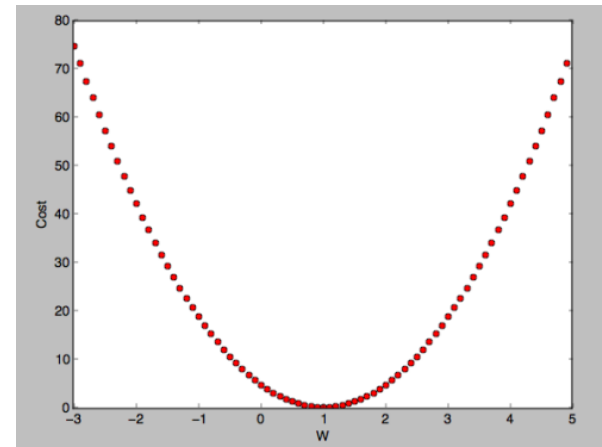
- 임의의 곳에서 시작하여 경사도(gradient)에 따라 W 를 변경시켜가면서 cost함수의 값이 최소화되는 W 를 구하는 알고리즘

- 경사도(gradient)는 미분값

- W 값의 변화

- $W = W - \alpha \frac{\partial}{\partial W} cost(W)$

- α 는 learning rate



sklearn.linear_model.LogisticRegression

```
LogisticRegression(C=1.0, solver='lbfgs',  
                    max_iter=100, multi_class='auto')
```

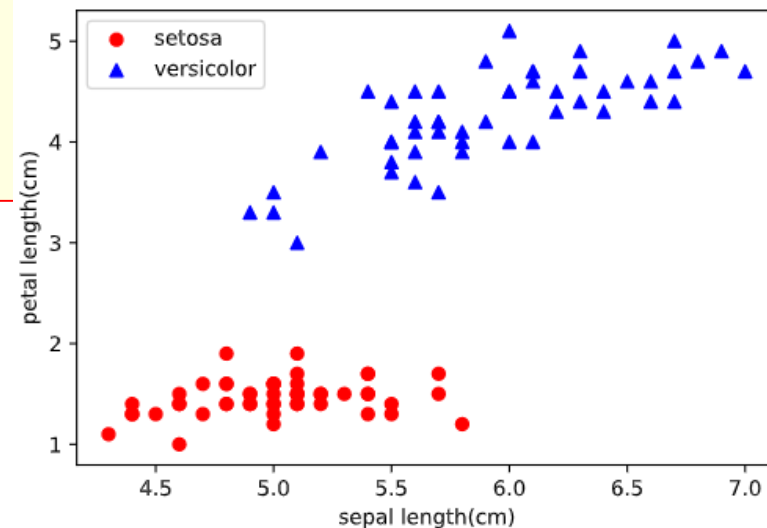
- C : 정규화강도의 역수(값이 작을수록 강한 정규화)
- solver : 최적화 문제해결 알고리즘
{‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’, ‘saga’}
- max_iter : 최대 반복수
- multi_class : 옵션 {‘auto’, ‘ovr’, ‘multinomial’}
 - OvR(One-versus-the-Rest) : 어떤 하나의 클래스에 속한 것과 다른 모든 것을 분류하는 분류기를 만들고, 각 분류기의 점수 중에서 가장 높은 것을 선택하는 전략. OvA(One-versus-One)라고도 함
 - OvO : 어떤 하나의 클래스와 다른 하나의 클래스에 속한 것을 분류하는 각각의 분류기를 만들고, 각 분류기의 점수들을 사용하는 전략

데이터 준비

```
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

iris = load_iris()
X = iris["data"][0:100, (0,2)] # 처음 100개의 데이터 중에서 꽃받침 길이, 꽃잎 길이
y = iris["target"][0:100] # 처음 50개는 Iris-setosa, 다음 50개는 Iris-versicolor
y = np.where(y==0, -1, 1) # 만일 Iris-setosa이면 -1, 아니면 1로 변경

plt.scatter(X[0:50, 0], X[0:50, 1], color="r", marker="o", label="setosa")
plt.scatter(X[50:100, 0], X[50:100, 1], color="b", marker="^", label="versicolor")
plt.xlabel('sepal length(cm)')
plt.ylabel('petal length(cm)')
plt.legend()
plt.show()
```



Binary Logistic Regression(Classification)

```
from sklearn.linear_model import LogisticRegression
from mlxtend.plotting import plot_decision_regions
```

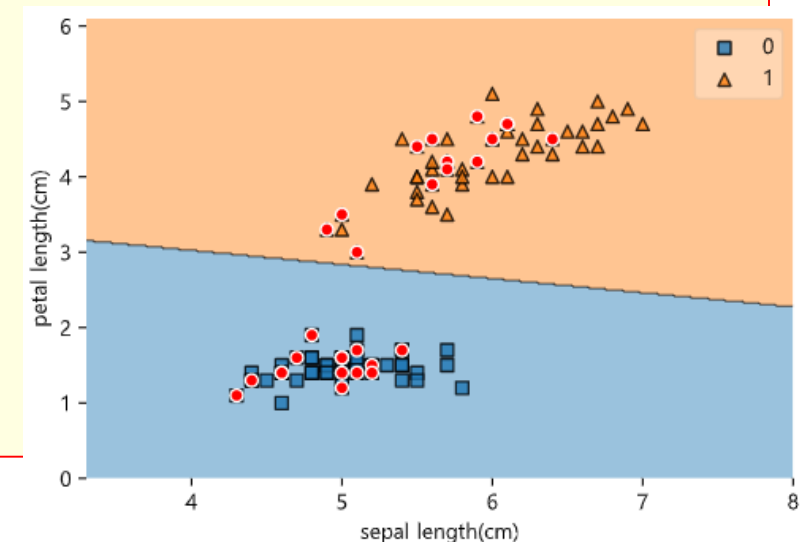
mlxtend 설치

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

```
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(f"accuracy: {accuracy_score(y_test, y_pred)}")
```

```
plot_decision_regions(X, y, clf=model)
plt.scatter(X_test[:,0], X_test[:,1],
            marker="o", color="r", edgecolor="w")
plt.xlabel('sepal length(cm)')
plt.ylabel('petal length(cm)')
plt.show()
```

accuracy: 1.0



Multinomial Logistic Regression(Classification)

Multinomial Logistic Regression(Classification)

■ Class가 N개인 경우

- 주어진 데이터가 A, B, C 중 어느 클래스에 속하는가?
- Softmax Regression이라고도 함

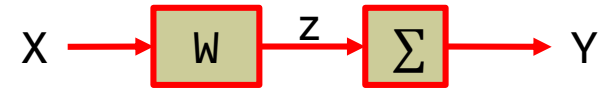
■ (예)

hours(x1)	attendance(x2)	grade(y)
10	5	A
9	5	A
3	2	B
2	4	B
11	1	C

■ N개의 binary classifier 결합

■ Class가 2개인 경우

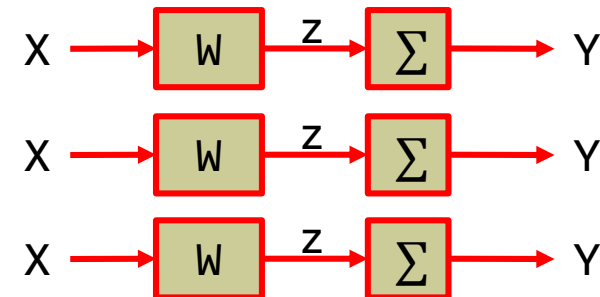
- $H(X) = \frac{1}{1+e^{-W^T X}}$



■ Class가 N개인 경우

- $$\begin{pmatrix} W_{A1} & W_{A2} & W_{A3} \\ W_{B1} & W_{B2} & W_{B3} \\ W_{C1} & W_{C2} & W_{C3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} W_{A1}x_1 + W_{A2}x_2 + W_{A3}x_3 \\ W_{B1}x_1 + W_{B2}x_2 + W_{B3}x_3 \\ W_{C1}x_1 + W_{C2}x_2 + W_{C3}x_3 \end{pmatrix} = \begin{pmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{pmatrix}$$

x1	x2	y
10	5	A
9	5	A
3	2	B
2	4	B
11	1	C

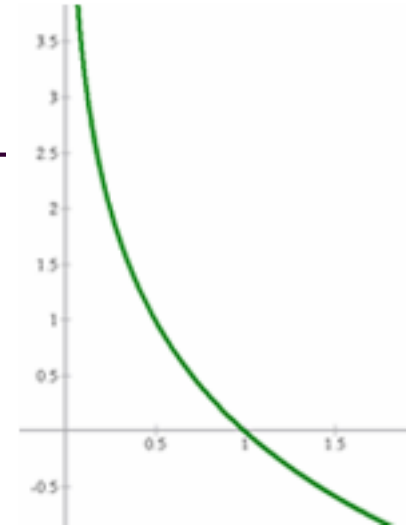


■ 자주 사용하는 Cost Function은?

■ Cross-Entropy

$$D(S, L) = - \sum_i L_i \log(S_i)$$

$$= \sum_i L_i * -\log(S_i) \quad \# \text{ elementary production}$$



■ 왜?

- 예측된 값 Li에 대해,
 - 옳은 결과 : cost를 0에 가깝게
 - 틀린 결과 : cost를 무한대에 가깝게

$$cost(W) = \frac{1}{m} \sum_i D(S(WX_i + b), L_i)$$

Softmax란?

$$\begin{bmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B1} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} w_{A1}x_1 + w_{A2}x_2 + w_{A3}x_3 \\ w_{B1}x_1 + w_{B2}x_2 + w_{B3}x_3 \\ w_{C1}x_1 + w_{C2}x_2 + w_{C3}x_3 \end{pmatrix} = \begin{pmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{pmatrix}$$

$$WX = Y$$

$$\begin{bmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{bmatrix} = \begin{pmatrix} 2.0 \\ 1.0 \\ 0.1 \end{pmatrix} = \begin{pmatrix} ? \\ ? \\ ? \end{pmatrix} \quad \begin{bmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{bmatrix} = \begin{pmatrix} 2.0 \\ 1.0 \\ 0.1 \end{pmatrix} = \begin{pmatrix} 0.7 \\ 0.2 \\ 0.1 \end{pmatrix}$$

- 이 수식의 값 2.0, 1.0, 0.1을 0~1사이의 값으로 변경할 수 있을까?
- 즉, 값을 확률(Probability)로 변경할 수 있을까?
 - 모두 더하면 1이 되도록...

$$\text{■ 확률 } S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

■ Softmax

$$\begin{matrix} \blacksquare \end{matrix} \begin{pmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{pmatrix} = \begin{pmatrix} 2.0 \\ 1.0 \\ 0.1 \end{pmatrix} \Rightarrow \underset{S(y)}{argmax} \begin{pmatrix} 0.7 \\ 0.2 \\ 0.1 \end{pmatrix} = \underset{L(Label)}{\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}}$$

- 계산결과를 확률로 표현하고,
- 이중 가장 max인 값을 1로 나머지를 0으로하는 것을 말함

Iris Dataset

- 데이터개수 : 150개

- 데이터 정보

- 데이터 세트에는 각각 50 개 인스턴스의 3 개 클래스가 포함되어 있으며, 각 클래스는 붓꽃의 유형을 나타냄

- 데이터 속성

1. sepal length in cm (꽃받침 길이)
2. sepal width in cm (꽃받침 너비)
3. petal length in cm (꽃잎 길이)
4. petal width in cm (꽃잎 너비)
5. class:
 - 0: Iris Setosa
 - 1: Iris Versicolour
 - 2: Iris Virginica



Iris dataset 전체

```
from sklearn.datasets import load_iris
import pandas as pd

iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df['target'] = iris.target
df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

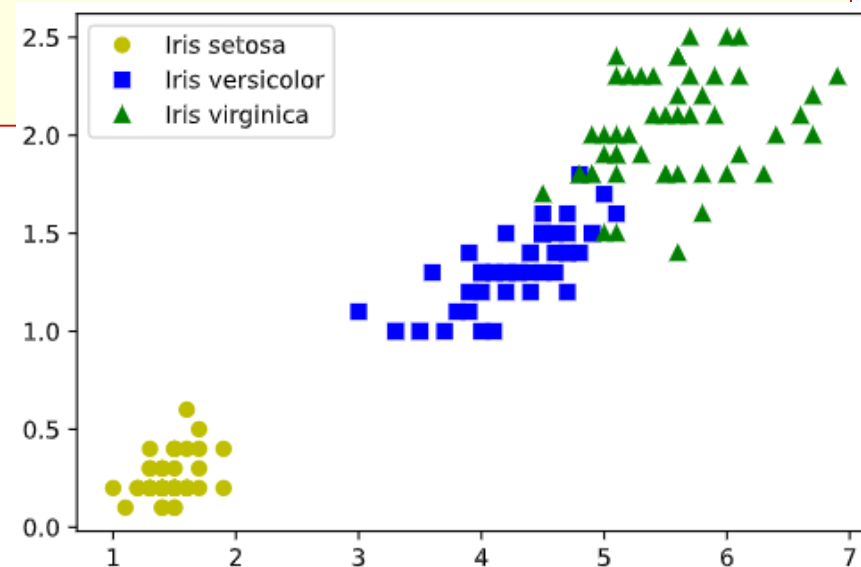
150 rows × 5 columns

Iris dataset 중에서 꽃잎의 길이와 너비만...

```
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

iris = load_iris()
X = iris["data"][:, (2,3)] # 꽃잎의 길이와 너비만 fancy indexing
y = iris["target"]

plt.plot(X[y==0, 0], X[y==0, 1], "yo", label="Iris setosa")
plt.plot(X[y==1, 0], X[y==1, 1], "bs", label="Iris versicolor")
plt.plot(X[y==2, 0], X[y==2, 1], "g^", label="Iris virginica")
plt.legend()
plt.show()
```



Multinomial Logistic Regression(scikit-learn)

- Class가 2개 이상인 경우

- 주어진 데이터가 어느 클래스에 속하는가?
- Softmax Regression이라고도 함

- **scikit-learn**

- LogisticRegrssion의 `multi_class`에 “multinomial”을 지정해야 함

Multinomial Logistic Regression(Iris dataset)

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn import model_selection
from sklearn import metrics

iris = load_iris()
X = iris["data"][:, (2,3)] # 꽃잎의 길이와 너비만 fancy indexing
y = iris["target"]
X_train, X_test, y_train, y_test
    = model_selection.train_test_split(X, y, test_size=0.3)

softmax_reg = LogisticRegression(multi_class="multinomial", C=1, penalty='l2')
softmax_reg.fit(X_train, y_train)

y_predict = softmax_reg.predict(X_test)
score = metrics.r2_score(y_test, y_predict)
print(f"accuracy = {score}")
```

정규화 강도의 역수

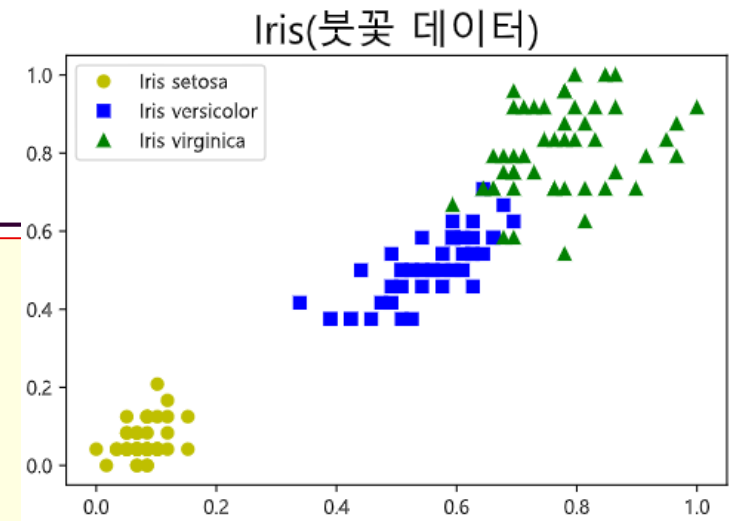
accuracy = 0.9662668665667167

참고: MinMaxScaler

```
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
```

```
iris = load_iris()
x_data = iris["data"][:, (2,3)] # 꽃잎의 길이와 너비만 fancy indexing
scaler = MinMaxScaler()
scaler.fit(x_data)
X = scaler.transform(x_data)
y = iris["target"]
```

```
plt.title("Iris(붓꽃 데이터)", fontsize=20)
plt.plot(X[y==0, 0], X[y==0, 1], "yo", label="Iris setosa")
plt.plot(X[y==1, 0], X[y==1, 1], "bs", label="Iris versicolor")
plt.plot(X[y==2, 0], X[y==2, 1], "g^", label="Iris virginica")
plt.legend()
plt.show()
```



SGDClassifier(sklearn)

■ SGD 알고리즘을 사용한 선형 분류기

■ SGD(Stochastic Gradient Descent) 알고리즘

- 임의의 곳에서 시작하여 경사도(gradient)에 따라 w 를 변경시켜가면서 cost함수의 값이 최소화되는 w 를 구하는 알고리즘

```
SGDClassifier(loss='hinge', *, penalty='l2', alpha=0.0001,
              l1_ratio=0.15, fit_intercept=True, max_iter=1000,
              tol=0.001, shuffle=True, verbose=0, epsilon=0.1,
              n_jobs=None, random_state=None, learning_rate='optimal',
              eta0=0.0, power_t=0.5, early_stopping=False,
              validation_fraction=0.1, n_iter_no_change=5,
              class_weight=None, warm_start=False, average=False)
```

■ 주요 cost function

- loss="hinge": (소프트 마진)선형 SVM,
- loss="modified_huber": 부드러운 힌지 손실,
- loss="log": 로지스틱 회귀

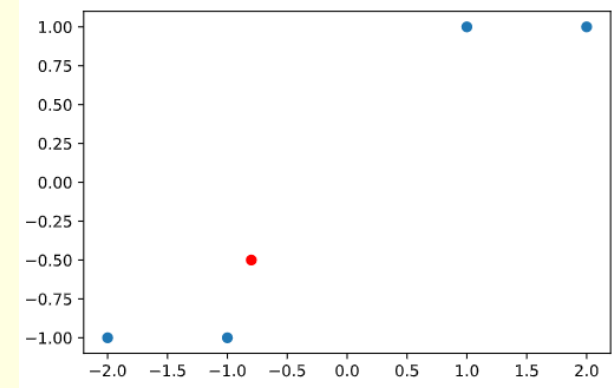
sklearn.linear_model.SGDClassifier

```
from sklearn.linear_model import SGDClassifier
import matplotlib.pyplot as plt
```

```
X = np.array([[ -1, -1], [-2, -1], [ 1,  1], [ 2,  1]])
y = np.array([ 1,  1,  2,  2])
```

```
clf = SGDClassifier(max_iter=1000, tol=1e-3)
clf.fit(X, y)
```

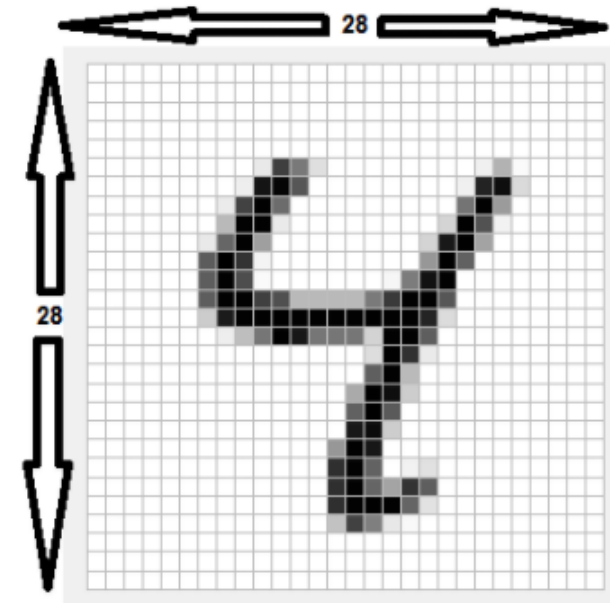
```
print(clf.predict([[ -0.8, -0.5]]))
plt.scatter(X[:,0], X[:,1])
plt.scatter(-0.8, -0.5, marker="o", color="r")
plt.show()
```



MNIST & Digits Dataset

■ MNIST: 우편번호 필기숫자 데이터베이스

- 학습데이터 60,000개, 테스트데이터 10,000개 (해상도: 28 x 28)



■ Digits: 필기숫자 데이터베이스

- 데이터 1797개 (해상도: 8 x 8)

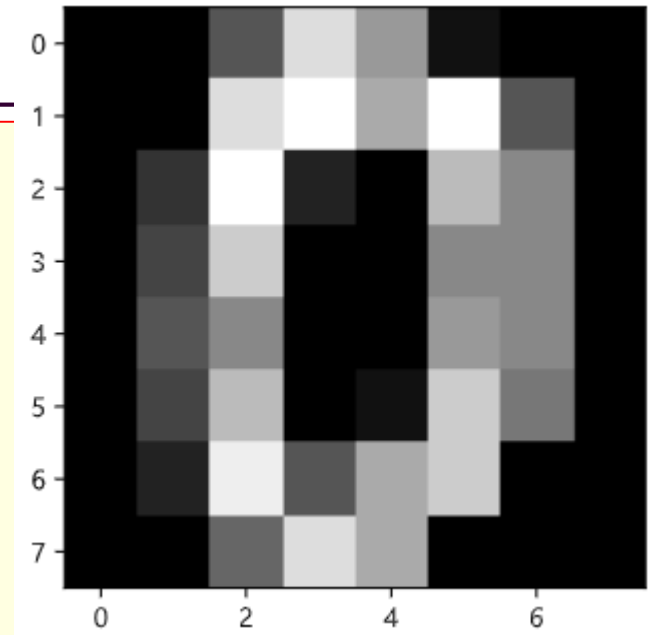
Digits Dataset(sklearn)

```
import numpy as np
from sklearn import datasets

digits = datasets.load_digits()
X, y = digits.data, digits.target
print(X.shape, y.shape)
```

```
first_digit = X[0]
first_digit_image = first_digit.reshape(8, 8)
```

```
plt.imshow(first_digit_image, cmap="gray")
plt.show()
```



```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score
import numpy as np
import matplotlib.pyplot as plt

digits = datasets.load_digits()
X, y = digits.data, digits.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

model = SGDClassifier()
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print(f"accuracy: {accuracy_score(y_test, y_pred)}")
```

```
accuracy: 0.9592592592592593
```

MNIST

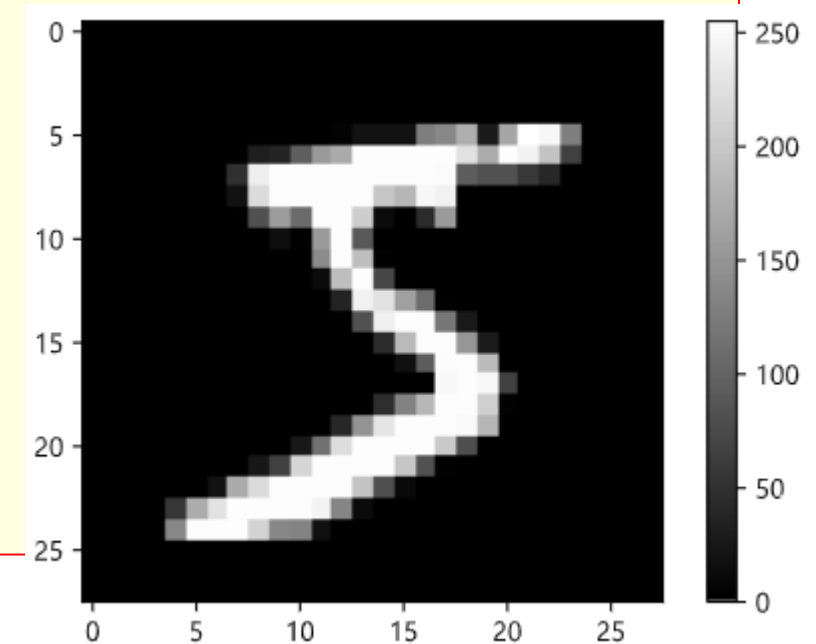
```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784')
mnist.keys()
```

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

```
X, y = mnist.data, mnist.target
print(X.shape, y.shape)
```

```
first_digit = np.array(X.iloc[0,:].tolist())
first_digit_image = first_digit.reshape(28, 28)
```

```
plt.imshow(first_digit_image, cmap="gray")
plt.show()
```



```
from sklearn.datasets import fetch_openml
```

```
mnist = fetch_openml('mnist_784')
```

```
X, y = mnist.data, mnist.target
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.linear_model import SGDClassifier
```

```
from sklearn.metrics import accuracy_score
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

```
model = SGDClassifier()
```

```
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
```

```
print(f"accuracy: {accuracy_score(y_test, y_pred)}")
```

```
accuracy: 0.8807619047619047
```