파이썬 머신러닝
- SVM
- MLP

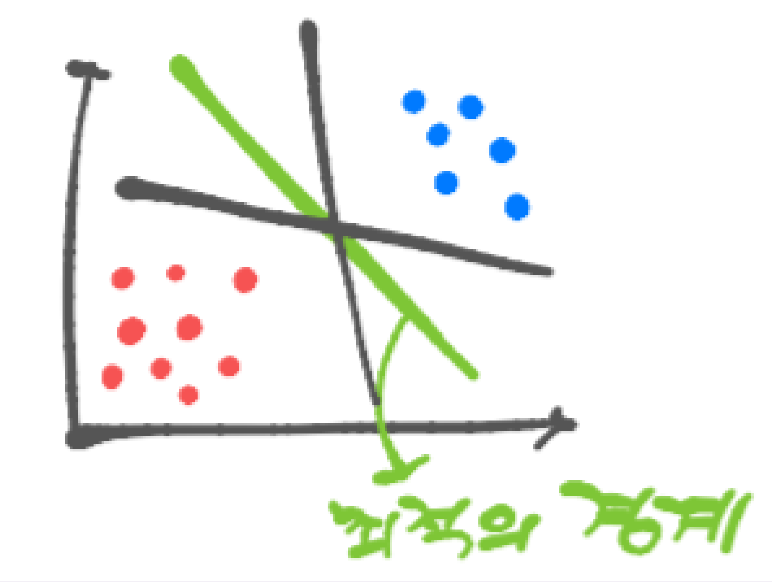
2017010698
수학과 오서영

목차

1 SVM

서포트 벡터 머신
Support Vector Machine

-분류 알고리즘
- 데이터를 나누는 최적의 경계를 만드는 방식



초평면(Hyperplane)

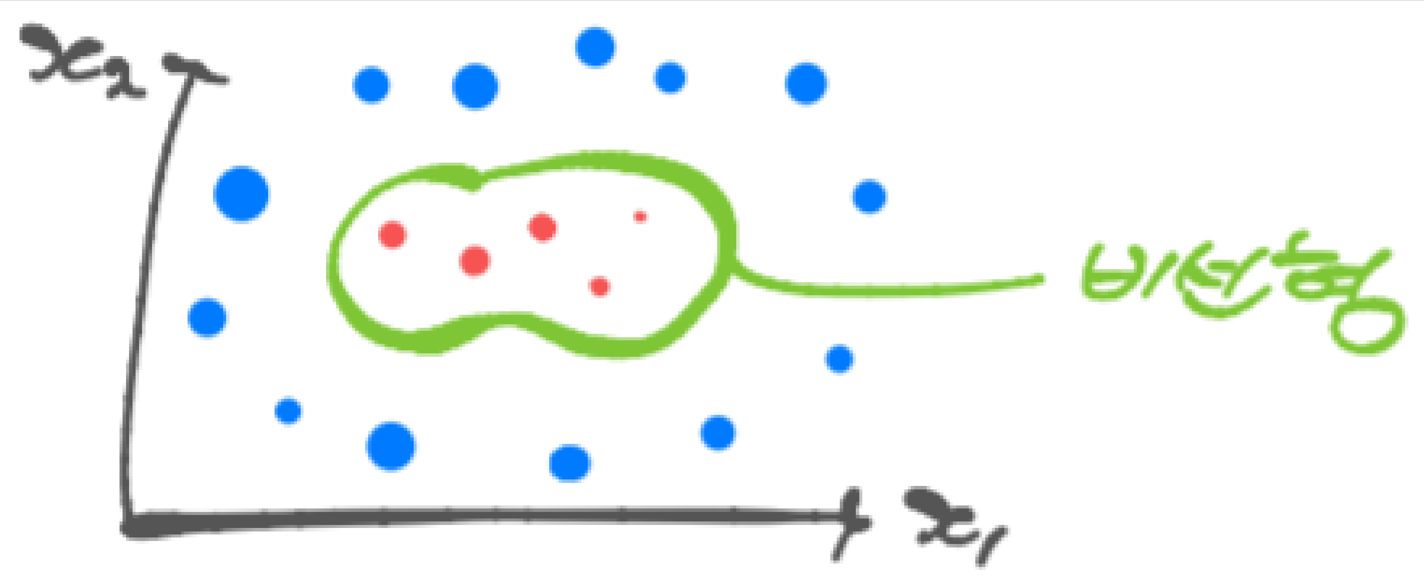
- **margin** : 데이터와 경계 사이의 거리
- **support vector** : margin에서 가장 가까운 데이터

- **초평면** : support vector와 margin을 이용하여 그린 선
= **최적의 경계**

- SVM에서는 데이터의 차원을 늘리는 **커널트릭**이라는 기법으로 초평면을 3차원으로 늘릴 수 있다.

커널(Kernel)

Non-linear Decision Boundary



$$\text{예측} = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \cdots \geq 0 \\ 0 & \text{그 외} \end{cases}$$

$$f_1 = x_1,\ f_2 = x_2,\ f_3 = x_1 x_2,\ f_4 = x_1^2$$

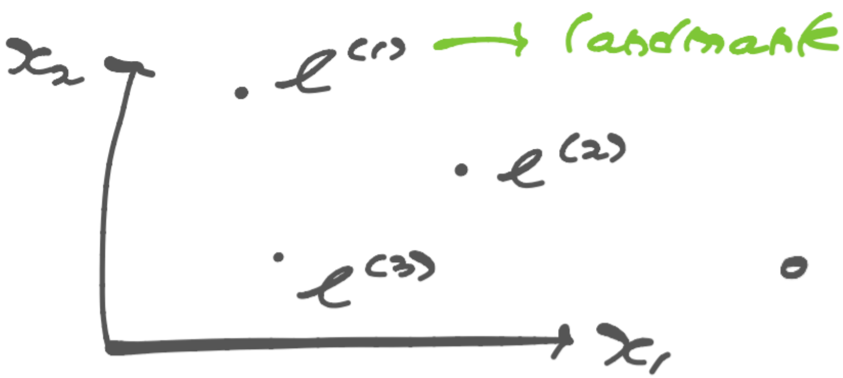
↳ feature

f(feature)을 고르는 더 나은 방법? -> 커널

- kernels → 유사도

" $$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x-l^{(1)}\|^2}{2\sigma^2}\right)$$

가우시안 커널

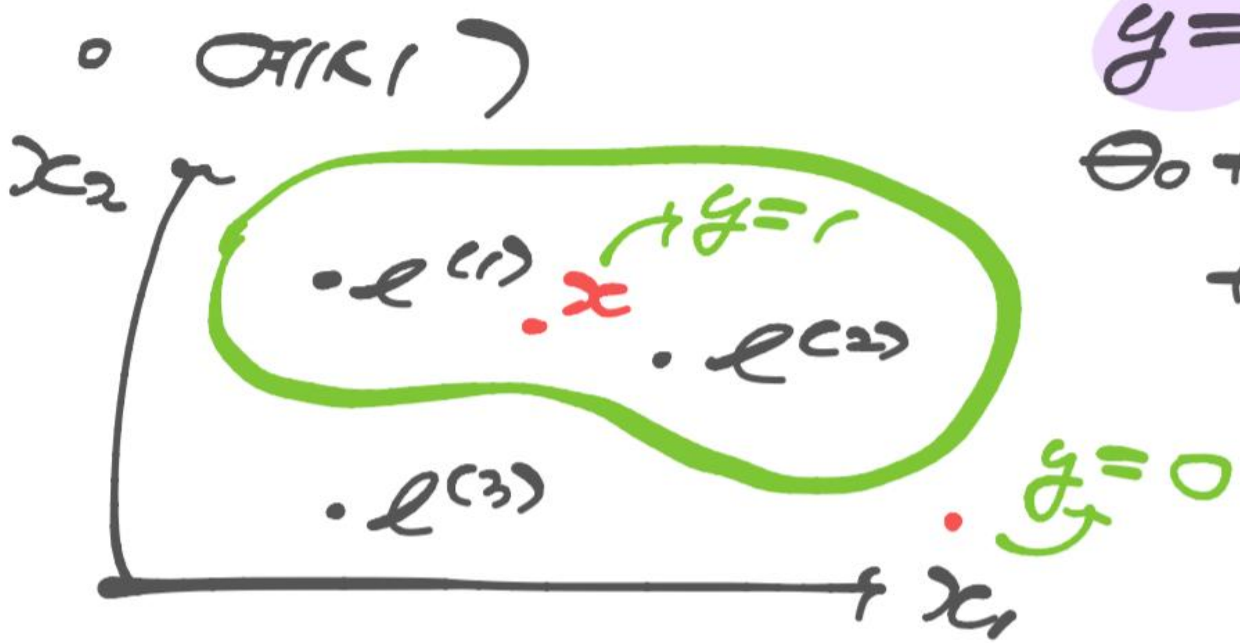


- If $x = l^{(1)}$:

  $f_1 \approx 1$

  If x : far from $l^{(1)}$:

  $f_1 \approx 0$



$y=1$ when

$\theta_0+\theta_1 f_1$
$+\theta_2 f_2$
$+\theta_3 f_3 \geq 0$

실습 1 : 간단한 SVM 구현하기

4. SVM

```
svc = SVC()
svc.fit(x_train, y_train)
```

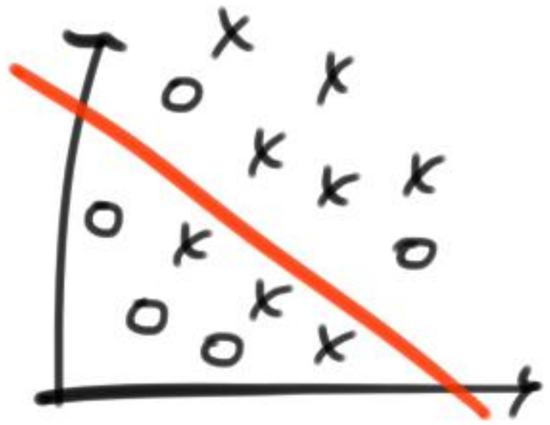
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

5. Accuracy Analysis

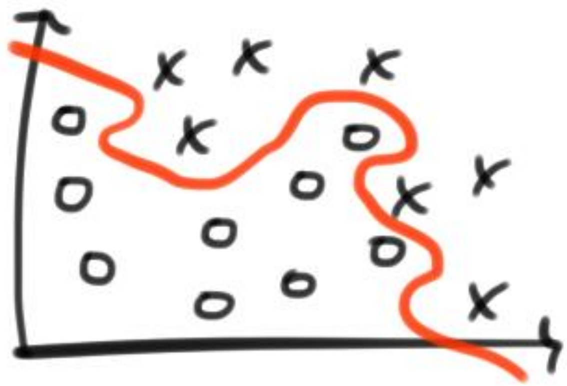
```
print("prediction :", svc.predict(x_test))
print("train accuracy :", svc.score(x_train, y_train))
print("test accuracy :", svc.score(x_test, y_test))
```

```
prediction : [1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 0.
 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 0.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1. 1. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0.
 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 0. 1. 0. 1. 0.]
train accuracy : 0.9225352112676056
test accuracy : 0.916083916083916
```

Overfitting vs Underfitting



Underfitting (과소적합)
= high bias
1) 새로운 모델
2) 더 오래 학습하기



Overfitting (과대적합)
= high variance
1) 더 많은 데이터
2) 규제(Regularization)

SVM 파라미터

- Cost 함수

$$\min_{\theta} C \sum_{i=1}^{m} \left(y^{(i)} \mathrm{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \mathrm{cost}_2(\theta^T x^{(i)}) \right) + \underbrace{\frac{1}{2} \sum \theta^2}_{\text{규제}}$$

① cost_1 :

② cost_2 :

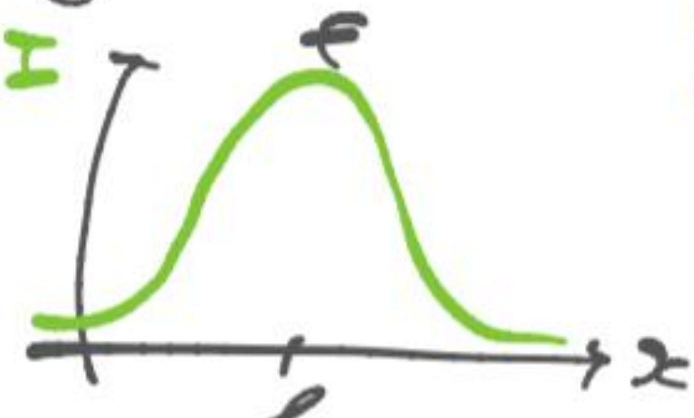
즉 CA + B 형태

◦ kernels → 유사도

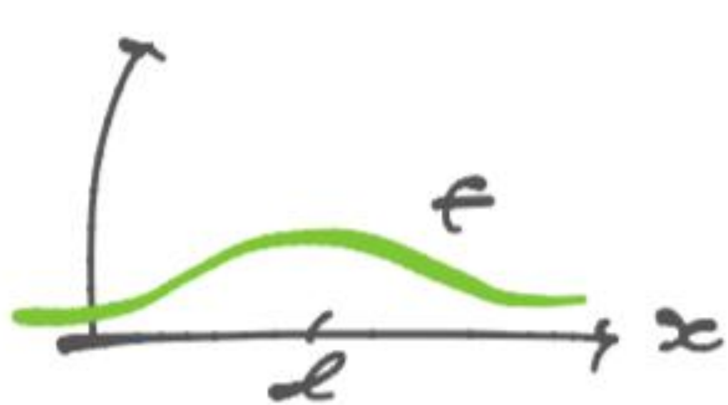
① C
- 큰 경우 : variance ↑
- 작은 경우 : bias ↑

② γ $\left(\approx \frac{1}{\sigma}\right)$
- I 큰 경우 : DB 복잡 , var ↑
- II 작은 경우 : DB 단순 , bias ↑

I



II



실습 2 : Cost 매개변수 조정

6. Underfit? (high bias)

```
svc1 = SVC(C = 1000)   # variance up!
svc1.fit(x_train, y_train)
```

```
SVC(C=1000, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
print("prediction :", svc1.predict(x_test))
print("train accuracy :", svc1.score(x_train, y_train))
print("test accuracy :", svc1.score(x_test, y_test))
```

```
prediction : [1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 0. 1. 0. 0. 1.
 1. 0. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 0.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 1. 1. 0. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0.
 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 0. 1. 1. 1. 0. 0. 1. 0. 1. 0. 1. 0.]
train accuracy : 0.9624413145539906
test accuracy : 0.9440559440559441
```

실습 3 : Gamma 매개변수 조정

```
svc2 = SVC(C = 1000, gamma = 0.00001)  # variance up!
svc2.fit(x_train, y_train)
```

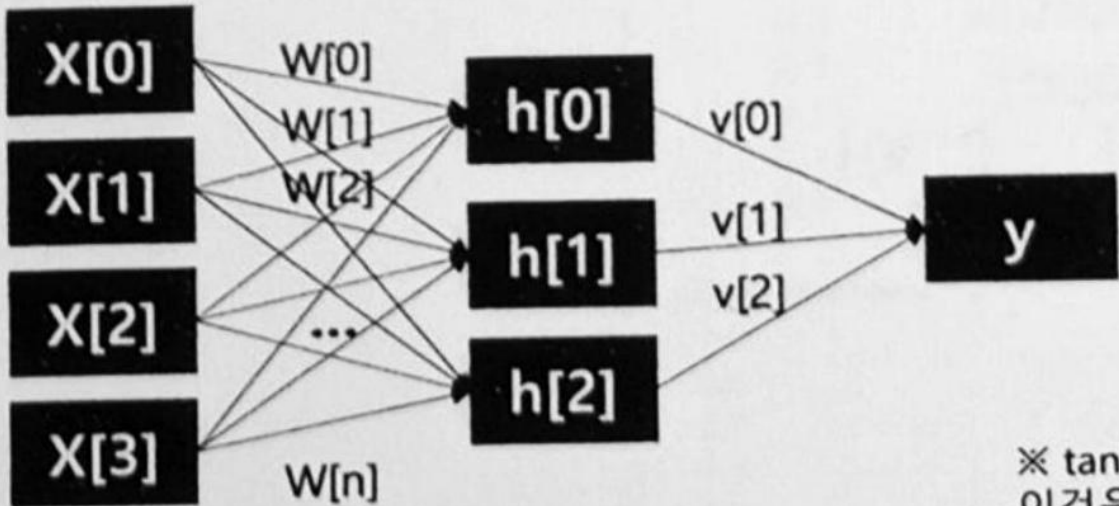
```
SVC(C=1000, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1e-05, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
print("prediction :", svc2.predict(x_test))
print("train accuracy :", svc2.score(x_train, y_train))
print("test accuracy :", svc2.score(x_test, y_test))
```

```
prediction : [1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0.
 1. 0. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 0.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 0. 0. 0. 1. 1. 0. 1. 1.
 1. 1. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0.
 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 0. 0. 1. 0. 1. 0. 1. 0.]
train accuracy : 0.9835680751173709
test accuracy : 0.958041958041958
```

2 MLP

- Multi-Layer Perceptron = 신경망 = 딥러닝



※ tanh 함수는 값이 -1~+1로 수렴하게 만든
이것은 복잡한 함수를 학습할 수 있게 한다

$$h_0 = \tanh(w_{[0,0]}x_0 + w_{[1,0]}x_1 + w_{[2,0]}x_2 + w_{[3,0]}x_3 + b_0)$$
$$h_1 = \tanh(w_{[0,1]}x_0 + w_{[1,1]}x_1 + w_{[2,1]}x_2 + w_{[3,1]}x_3 + b_1)$$
$$h_2 = \tanh(w_{[0,2]}x_0 + w_{[1,2]}x_1 + w_{[2,2]}x_2 + w_{[3,2]}x_3 + b_2)$$

※ 편향 b는 그래프에는 보통 표기하지 않는
그러나 은닉 유닛마다 각각의 편향이 있다

$$y = v_0h_0 + v_1h_1 + v_2h_2 + v_3h_3 + b$$

- 최적의 가중치를 찾을 수 있어 우수한 모델을 만들 수 있다.
- 학습 시간이 오래 걸린다.
- 매개변수 조정을 세심하게 해야한다
- 은닉층의 수, 노드의 수를 잘 조절해야 한다

1) 은닉층 : 1~2개 정도로 늘려보기
2) 노드 수 : 1000정도 까지 늘려보기

실습 1 : 간단한 MLP 구현하기

4. MLP

```python
mlp = MLPClassifier(random_state=42)
mlp.fit(x_train, y_train)
```

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_iter=200, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=42, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

5. Accuracy Analysis

```python
print("prediction :", mlp.predict(x_test))
print("train accuracy :", mlp.score(x_train, y_train))
print("test accuracy :", mlp.score(x_test, y_test))
```

```
prediction : [1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 0.
 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 0.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0.
 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 0. 1. 0. 1. 0.]
train accuracy : 0.9389671361502347
test accuracy : 0.9300699300699301
```

실습 2 : 더 오래 학습하기

6. Underfit? (high bias)

```
mlp1 = MLPClassifier(max_iter=1000, random_state=42)  # trains longer
mlp1.fit(x_train, y_train)
```

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_iter=1000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=42, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

```
print("prediction :", mlp1.predict(x_test))
print("train accuracy :", mlp1.score(x_train, y_train))
print("test accuracy :", mlp1.score(x_test, y_test))
```

```
prediction : [1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 0. 1. 0. 1. 0. 1. 1.
 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 0.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 1. 0. 1. 0. 0. 1. 1. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0.
 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 0. 1. 0. 1. 0.]
train accuracy : 0.9389671361502347
test accuracy : 0.9300699300699301
```

실습 3 : 은닉층

```python
# hidden layer modification
mlp2 = MLPClassifier(max_iter=200, hidden_layer_sizes=[100,100,100], random_state=42)
mlp2.fit(x_train, y_train)
```

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=[100, 100, 100], learning_rate='constant',
              learning_rate_init=0.001, max_iter=200, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=42, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

```python
print("prediction :", mlp2.predict(x_test))
print("train accuracy :", mlp2.score(x_train, y_train))
print("test accuracy :", mlp2.score(x_test, y_test))
```

```
prediction : [1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 1. 1. 1. 0. 0. 0. 1. 0. 1. 1.
 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 1. 1. 1. 1.
 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0. 1. 1. 0. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1. 1. 1. 1. 1. 1. 1. 0. 1. 0. 0.
 1. 1. 1. 0. 0. 1. 0. 1. 1. 1. 0. 0. 1. 1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 1. 1. 1. 0. 1. 0. 1. 0. 1. 0.]
train accuracy : 0.9154929577464789
test accuracy : 0.9370629370629371
```

끝