UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

www.ust.ac.kr

5장

서포트 벡터 머신

이 홍 석 (hsyi@kisti.re.kr)
한국과학기술정보연구원 슈퍼컴퓨팅응용센터

Contents

목차

UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

01.
선형 SVM 소개

이홍석 (hsyi@kisti.re.kr)

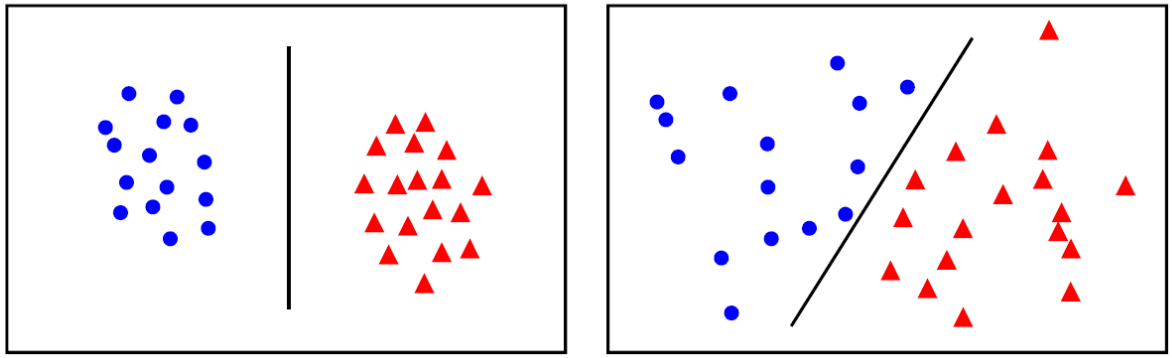
SVM의 역사

- SVM은 통계 학습 이론과 관련
  - SVM은 1992년에 처음 소개됨
  - SVM은 손글씨 인식 문제에서 성능을 보임
  - SVM의 테스트 오차율은 1.1%로, 신경망 LeNet 4과 비슷한 성능
- SVM의 'kernel trick'은 머신러닝의 주요 분야

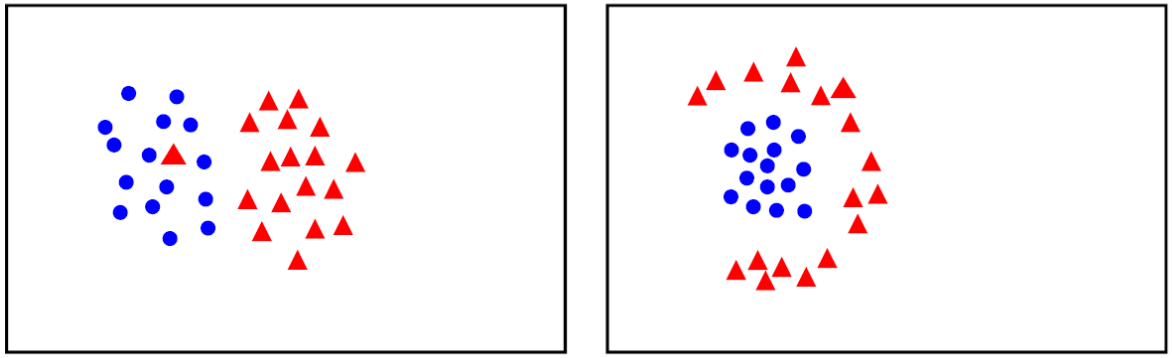
선형분리와 비선형 분리 예제

샘플의 선형적 분리 예제

linearly separable

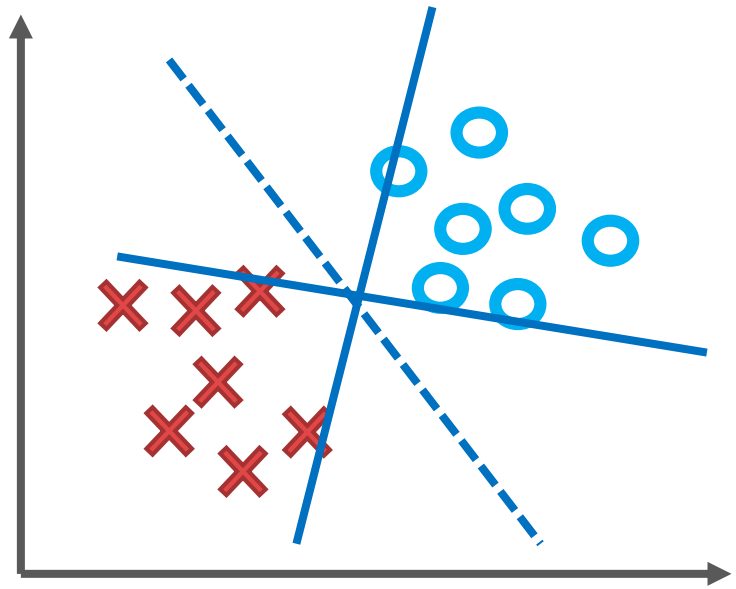


not linearly separable



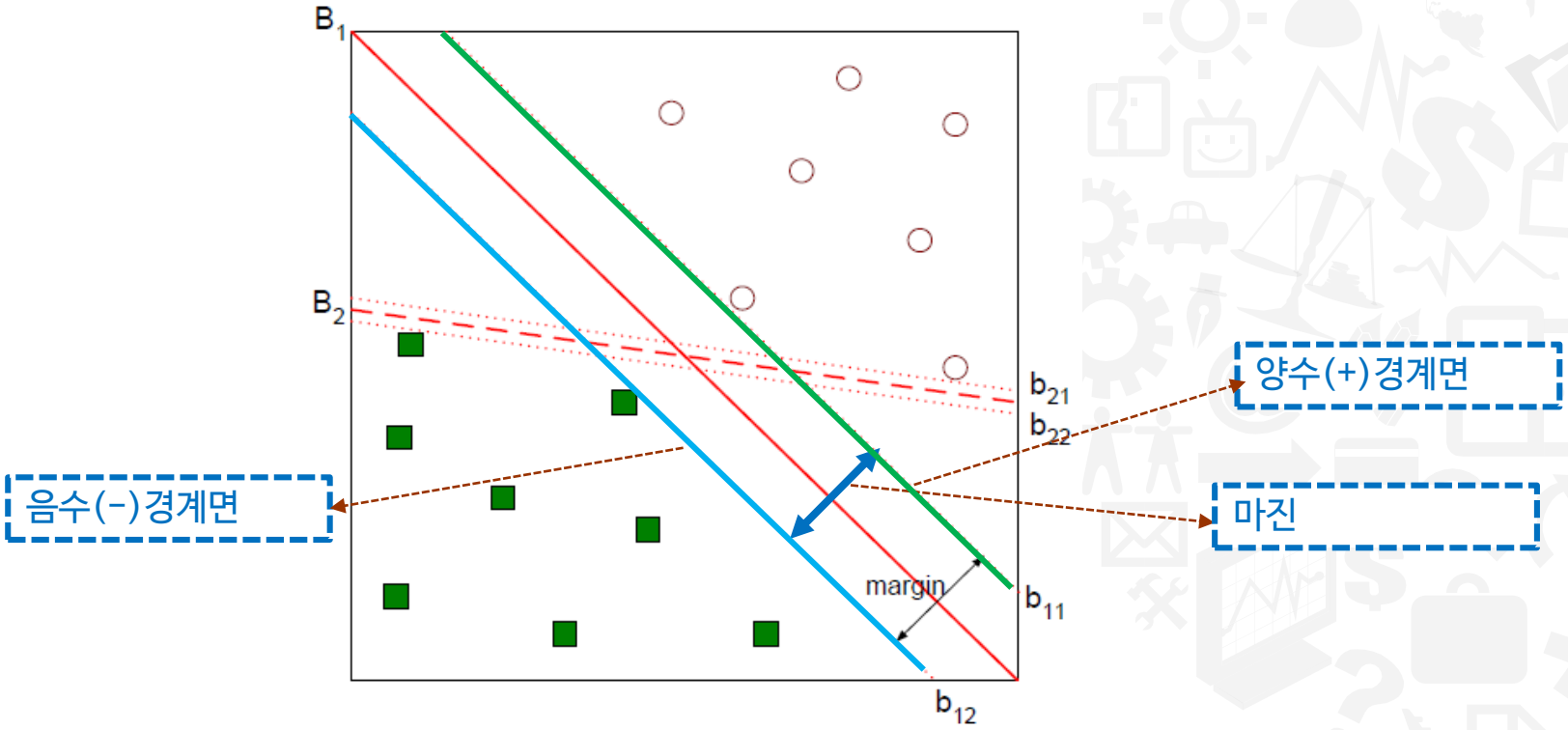
무엇인 결정 경계인가?

- 결정 경계 (decision boundary)
  - 아래 그림처럼 2개의 범주를 나누는 선형 분리가 가능할 때,
  - 많은 결정 경계가 가능하다.
  - 어떤 선이 가장 적절하게 두 데이터를 구분한 선일까요?



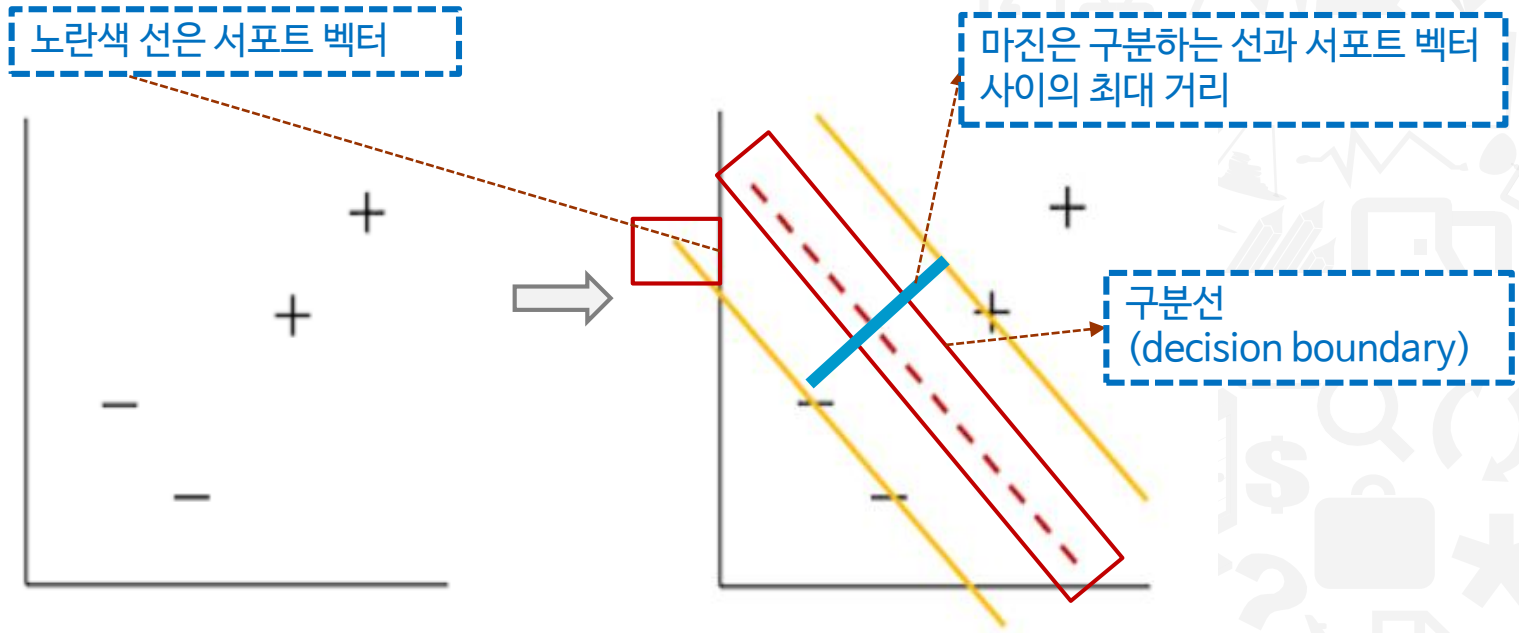
마진(margin)이란 무엇인가?

- 결정 경계를 정하기 위해 마진(margin)을 도입하자.
  - 마진은 두 경계면 사이의 거리이며, 이 거리가 최대일때 좋은 결정 경계를 얻음.



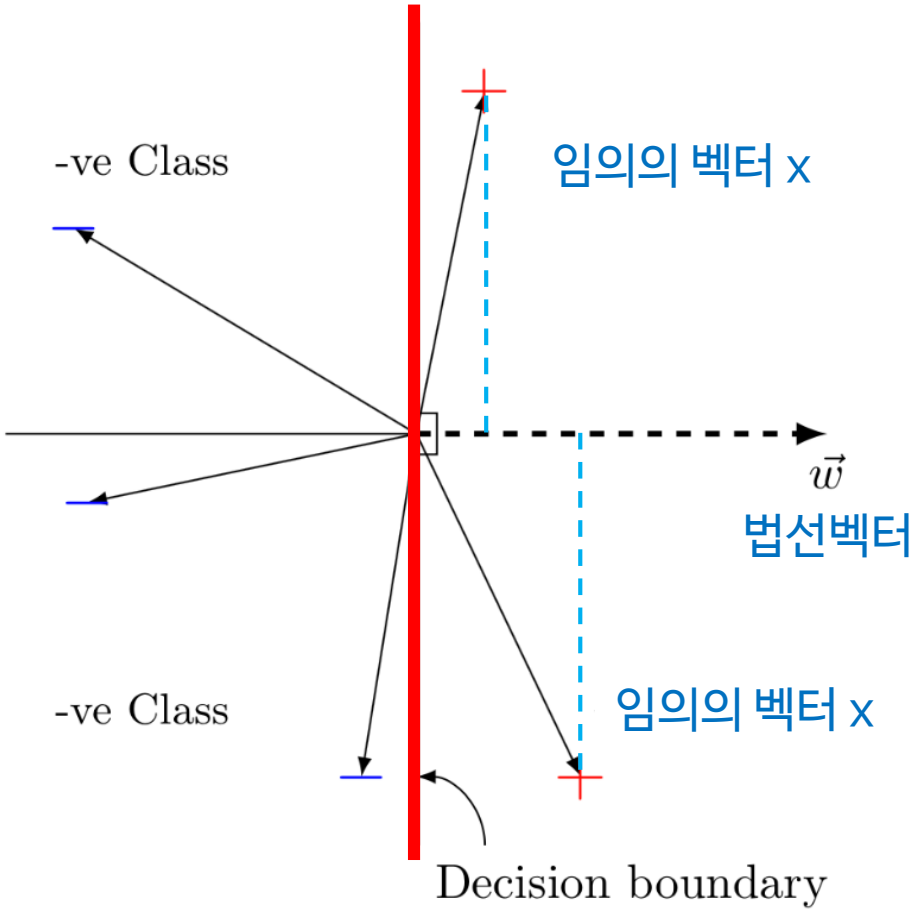
마진의 이해

- 질문?
  - 왼쪽 그림에서 +와 - 샘플을 구분하려면 어떻게 나누어야 할까?
  - 마진(Margin)이란
    - '+'와 '-' 샘플 사이의 거리를 가장 넓게 쓰는 어떤 라인(선)으로 점선



결정 경계(구분선) 결정 규칙(1)

결정경계(구분선)의 법선 벡터를 고려하자



$\vec{w} \cdot \vec{x} > c$ then '+'

$c = -b$

$\vec{w} \cdot \vec{x} + b > 0$ then '+'

내적은 |w||u|코사인이므로
결정경계 오른쪽이 +, 왼쪽이 - 이다.

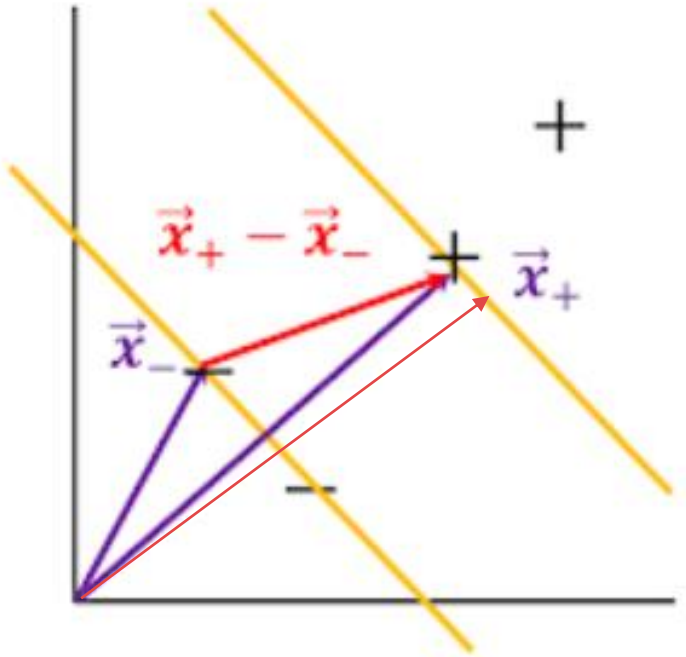
결정 경계(구분선) 결정 규칙(2)

어떤 선을 잡되 이로 인해 생기는 +와 - 샘플 사이의 거리를 가능한 최대로 넓게 하려면?

$$\vec{w} \cdot \vec{x^+} + b > 1$$
$$\vec{w} \cdot \vec{x^-} + b > -1$$

$$x^+ = x^- + \lambda w$$

구분선에 대한 법선 벡터(w)의 크기는 스칼라(w)이고, 람다는 임의의 계수이다. 두 개의 샘플 사이의 거리인 빨강색 선 크기이고, 이 값은 법선벡터 쪽으로 내적한 크기와 관련됨



$$w^T x^+ + b = 1$$
$$w^T (x^- + \lambda w) + b = 1$$
$$w^T x^- + b + \lambda w^T w = 1$$
$$-1 + \lambda w^T w = 1$$

$$\lambda = \frac{2}{w^T w}$$

결정 경계(구분선) 결정 규칙(3)

마진은 결정 경계 두 샘플 거리가 최대가 되도록 정한다.

$$
\begin{aligned}
Margin &= distance(x^+, x^-) \\
&= \left\| x^+ - x^- \right\|_2 \\
&= \left\| x^- + \lambda w - x^- \right\|_2 \\
&= \left\| \lambda w \right\|_2 \\
&= \lambda \sqrt{w^T w} \\
&= \frac{2}{w^T w} \sqrt{w^T w} \\
&= \frac{2}{\sqrt{w^T w}} \\
&= \frac{2}{\left\| w \right\|_2}
\end{aligned}
$$

$$
\lambda = \frac{2}{w^T w}
$$

목적식과 제약식 정의

- SVM의 목적은 마진을 최대화하는 경계면을 찾는 것
  - 계산상 편의를 위해 마진 절반을 제곱한 것에 역수를 취한 뒤 그 절반을 최소화하는 문제로 바꾸겠습니다. 이렇게 해도 문제의 본질은 바뀌지 않습니다.

$$max \frac{2}{\|w\|_2} \rightarrow \min \frac{1}{2}\|w\|_2^2$$

  - 여기엔 다음과 같은 제약조건이 관측치 개수만큼 붙습니다.

$$y_i(w^T x_i + b) \geq 1$$

라그랑지안 문제로 변환

- **라그랑지안 승수법(Lagrange multiplier method)**
  - 제약식에 형식적인 라그랑지안 승수를 곱한 항을 최적화하려는 목적식에 더하여, 제약된 문제를 제약이 없는 문제로 바꾸는 기법입니다
  - 정의한 목적식과 제약식을 라그랑지안 문제로 식을 다시 쓰면

$$\min L_p(w, b, \alpha_i) = \frac{1}{2}\|w\|_2^2 - \sum_{i=1}^{n} \alpha_i (y_i(w^T x_i + b) - 1)$$

$$\alpha_i \geq 0, \quad i = 1, \ldots, n$$

  - 라그랑지안 함수의 최소값을 주는 계수(alpha를 찾자)
    - 법선벡터(w)에 2차 함수 모양으로 1개의 전역(Global) 최소점이 있다.

라그랑지안 해를 구하기

최소값을 구하기 위해서 미분해주고,

$$\nabla_{\vec{w}}\mathcal{L} = \vec{w} - \sum_i \alpha_i y_i \vec{x}_i = 0 \quad \Longrightarrow \quad \vec{w} = \sum_i \alpha_i y_i \vec{x}_i$$

$$\nabla_b \mathcal{L} = -\sum_i \alpha_i y_i = 0$$

해석하면,
결정벡터가 '어떤' 샘플의 선형 합이다.

이제, 라그랑쥐 멀티플라이어만 알면 되는데,

$$\sum_{i=1}^{N} \alpha_i + \frac{1}{2}||\vec{w}||^2 - \sum_{i=1}^{N} \alpha_i y_i \vec{w}^T \vec{x}_i \iff \sum_{i=1}^{N} \alpha_i + \frac{1}{2}\vec{w}^T\vec{w} - \vec{w}^T\vec{w}$$

$$\iff \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\vec{w}^T\vec{w}$$

$$\iff \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \vec{x}_i^T \vec{x}_j$$

$$\iff \mathcal{L}(\alpha).$$

2차함수 성격으로 quadratic programming로 해를 구함

SVM의 해

- SVM의 해는 마진이 최대화 된 분류경계면의 w와 b를 찾자.

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i$$

x_i, y_i는 알고 있는 학습 데이터

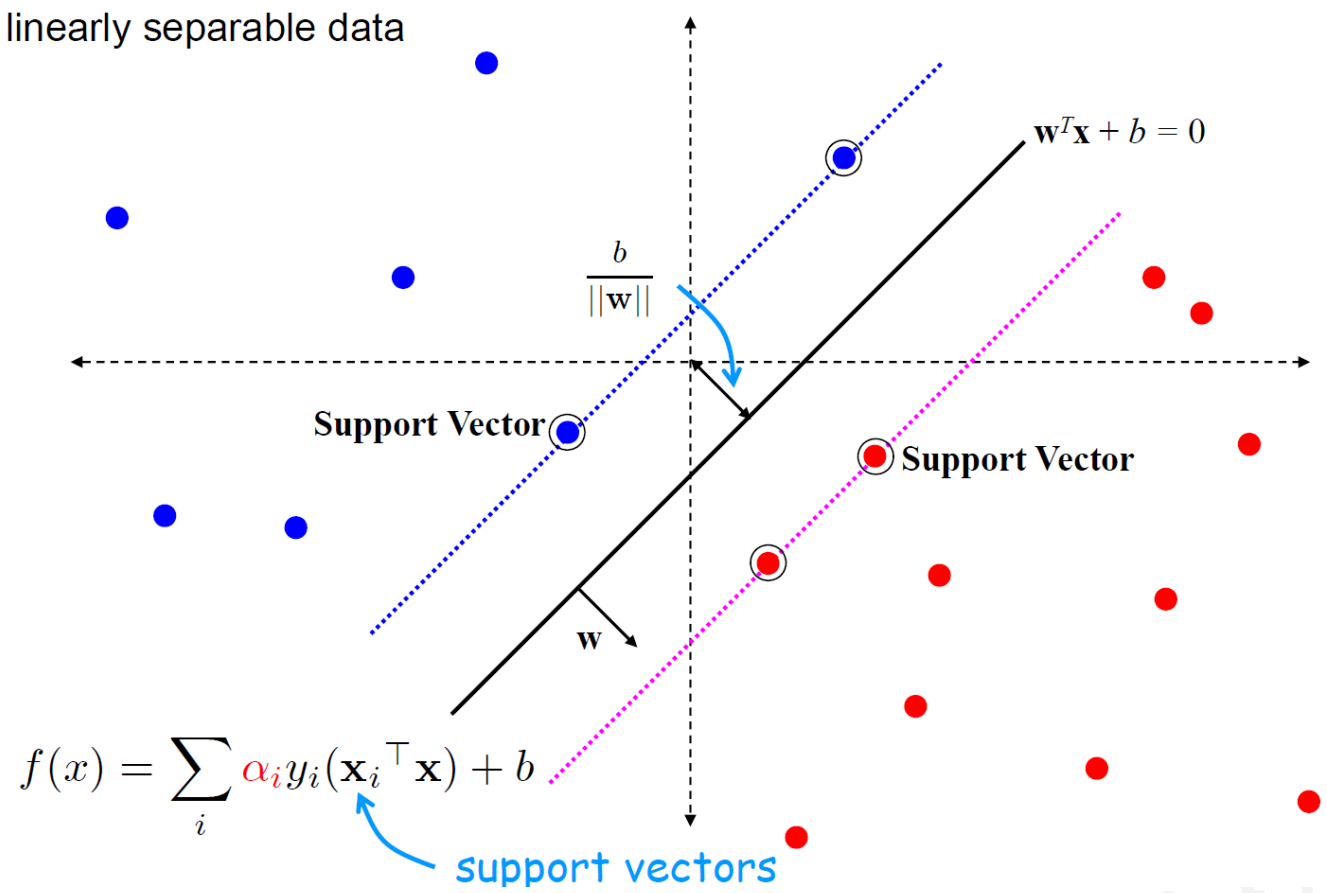
  - 함수가 최적값을 갖는다면 아래 두 개 가운데 하나는 반드시 0입니다

$$(1)\alpha_i$$
$$(2)y_i(w^T x_i + b) - 1$$
$$y_i(w^T x_i + b) - 1 = 0$$

  - 위의 마지막 식은 알파(alpha)가 0이 아닌 경우로, 서포트 벡터라고 한다.

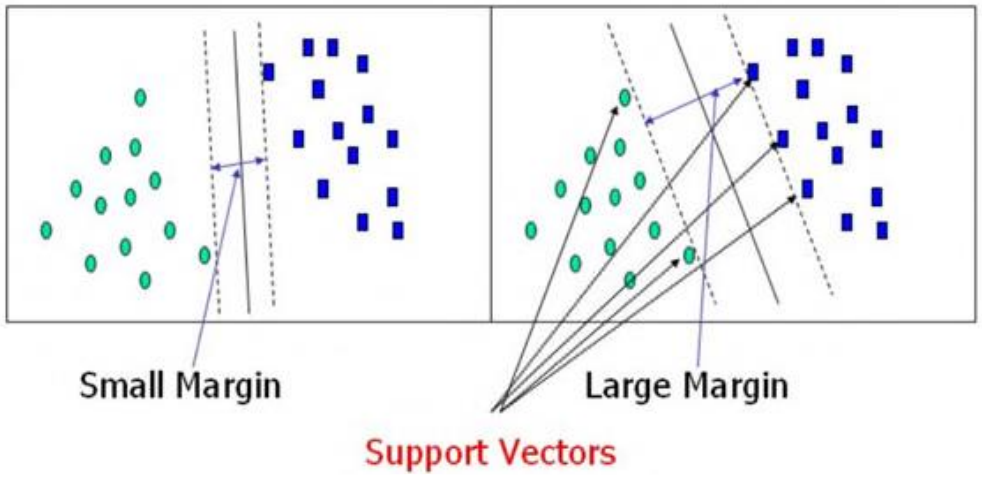
서포트 벡터 머신 (SVM)의 해

알파는 해당 벡터(x)가 경계선을 정하는 샘플이라는 뜻으로 “서포트 벡터 " 라고 부름



선형 분리 SVM 요약

- SVM 정리
  - 직관적으로 자료를 군집별로 가장 잘 분리하는 초평면은 가장 가까운 훈련용 자료까지의 거리가 가장 큰 경우
    - 최대 마진을 가지는 선형판별에 기초하며, 속성들 간의 의존성은 고려하지 않는 방법
    - 마진이 가장 큰 초평면을 분류기로 사용할 때, 새로운 자료에 대한 오분류가 가장 낮다



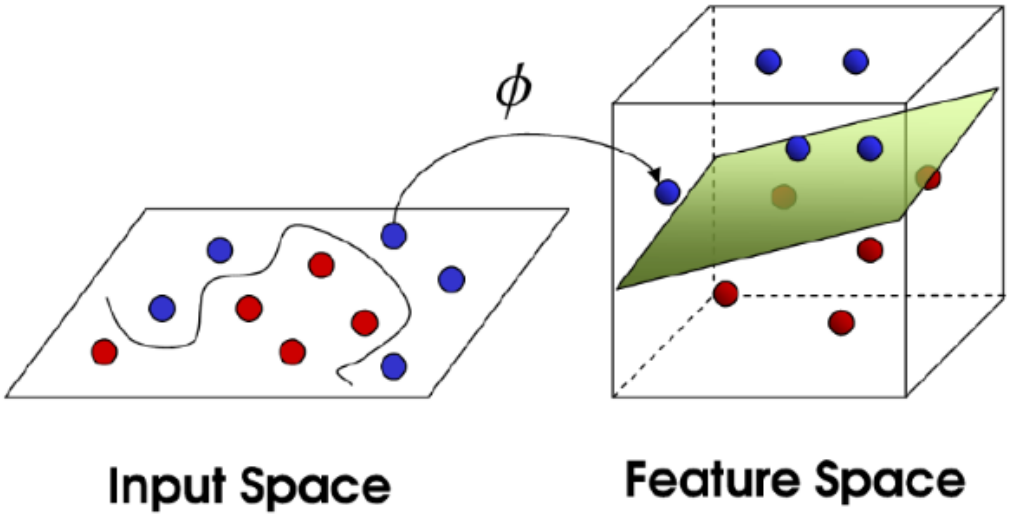
UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

01b.
SVM 커널트릭 소개

이홍석 (hsyi@kisti.re.kr)

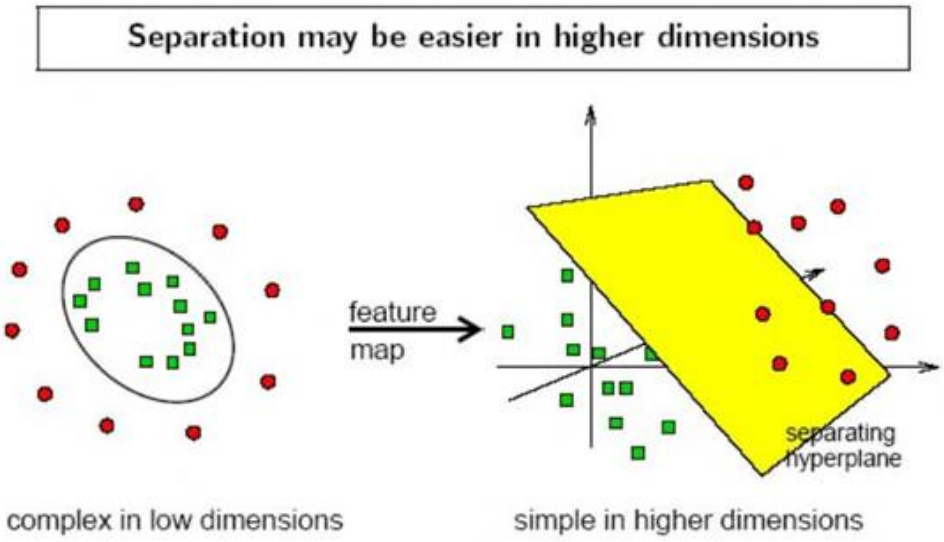
비선형 SVM

- 선형으로 분리 되지 않을 경우
  - 원공간(Input Space)의 데이터를 선형분류가 가능한 고차원 공간(Feature Space)으로 매핑한 뒤 두 범주를 분류하는 초평면을 찾는다.



SVM의 커널 트릭

- 커널 트릭(kernel trick)
  - 입력 자료의 다차원 공간상으로의 맵핑(mapping) 기법을 사용하여 비선형분류도 효율적으로 수행
  - 고차원 매핑과 내적을 한방에 할 수는 커널 트릭



$$K(\vec{x}, \vec{y}) = \phi(\vec{x}) \cdot \phi(\vec{y})$$

Kernel trick:
SVM에서 커널은 내적임으로,
샘플 공간에서 선형적으로
나눌 수 있는 공간을 샘플을
보내주고 SVM을 적용함.

UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

02. 선형 SVM
붓꽃 데이터에 적용

이홍석 (hsyi@kisti.re.kr)

붓꽃에 LinearSVC 적용 (1)

마진과 서프트 벡터 이해하기

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn import datasets
from sklearn import svm
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
```

붓꽃에 LinearSVC 적용 (2)

```
iris = datasets.load_iris()
X = iris['data'][:,(2,3)]
```

Petal (꽃잎) 길이와 넓이

```
scaler = StandardScaler()
Xstan = scaler.fit_transform(X)

data = pd.DataFrame(data=Xstan,
       columns=['petal length','petal width'])
data['target'] = iris['target']
data = data[data['target']!=2]
# 0, 1에 관심 Iris-setosa and Iris-Versicolor
data.head()
```

	petal length	petal width	target
0	-1.340227	-1.315444	0
1	-1.340227	-1.315444	0
2	-1.397064	-1.315444	0
3	-1.283389	-1.315444	0
4	-1.340227	-1.315444	0

입력값을 정규분포에 맞도록 변화함

Target은 0 (Setosa)와 1 (Versicolor)를 사용함

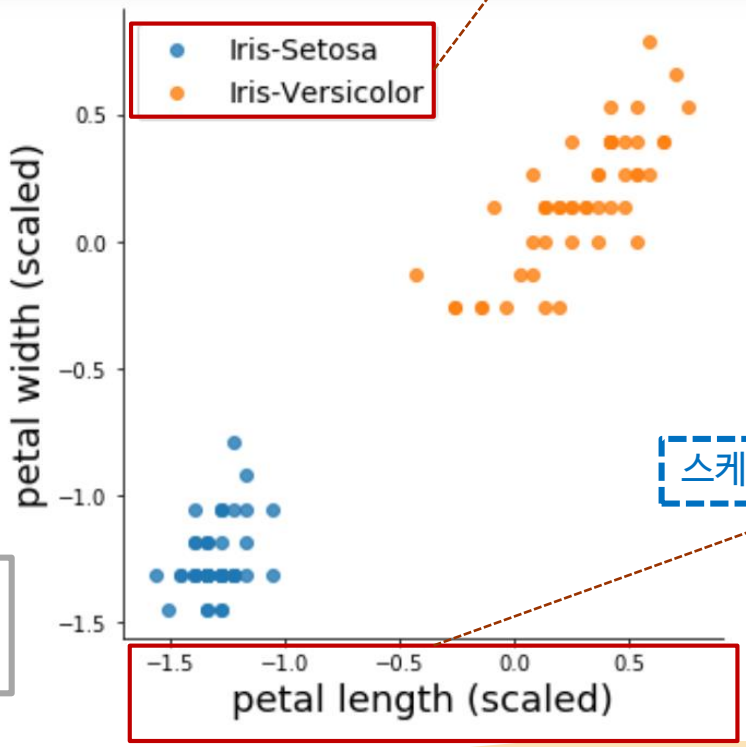
질문) 붓꽃의 꽃잎 3종류가 있는데, 2종류 보다 3종류를 선택하는 것이 올바른 것인가? 어떻게 처리할 것인가?

붓꽃에 LinearSVC 적용 (3)

```
sns.lmplot(x='petal length',y='petal width',
           hue='target',data=data, fit_reg=False, legend=False)

plt.legend(['Iris-Setosa','Iris-Versicolor'], fontsize = 14)
plt.xlabel('petal length (scaled)', fontsize = 18)
plt.ylabel('petal width (scaled)', fontsize = 18)
plt.show()
```

Seaborn 가시화

2종류 꽃잎이라서,
데이터가 잘 분리 되었음



스케일 된 꽃잎 길이

(질문) 3종류의 꽃잎일 경우?
또한, 3종류의 꽃받침(sepal)?

붓꽃에 LinearSVC 적용 (4)

SVC의 'linear' 커널과 비슷하다. 하지만, 손실함수와 페널티를 선택 폭이 넓고, 많은 샘플에 좋은 성능을 낸다.

훈련데이터를 이용하여 모델을 훈련한다.

SVC에서 대표적인 손실함수로 'hinge'를 사용한다. 디폴트는 제곱인 'squared_hinge'이다.

```
svc = LinearSVC(C=1, loss="hinge")

svc.fit(data[['petal length','petal width']].values,data['target'].values)
```

```
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)
```

디폴트 규제화는 'L2'이다.

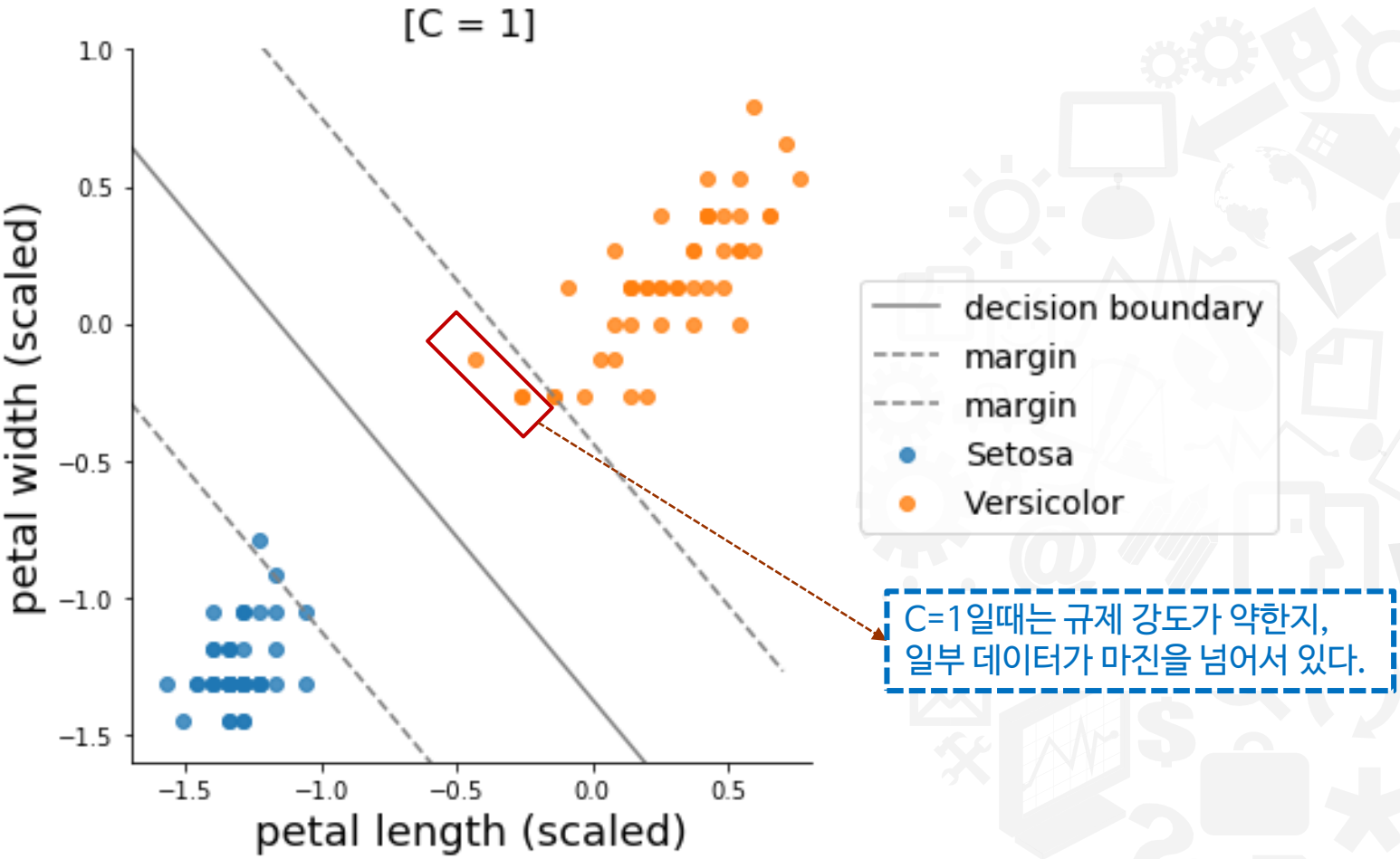
C는 규제화를 조절한다. 디폴트는 1이고, C 값의 역수가 규제화의 강도이다.

붓꽃에 LinearSVC 적용 (5)

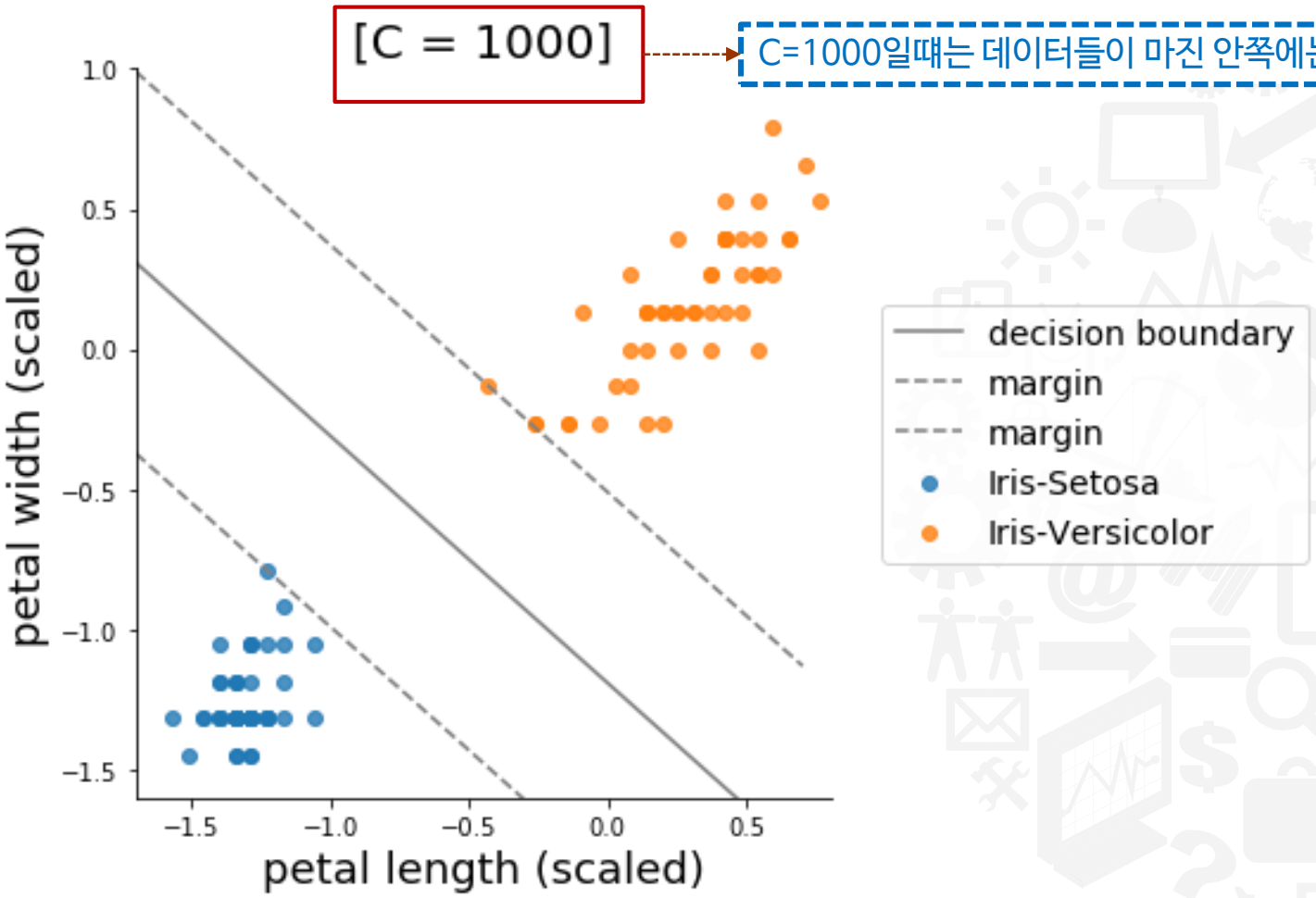
```
# 그림을 그리기 위한 파라미터 설정
w0,w1 = svc.coef_[0]
b = svc.intercept_[0]
x0 = np.linspace(-1.7, 0.7, num=100)
x1_decision = -b/w1 - w0/w1*x0 # 결정경계
x1_plus = x1_decision + 1/w1   # +1 마진
x1_minus = x1_decision - 1/w1  # -1 마진
```

```
sns.lmplot(x='petal length',y='petal width',
           hue='target',data=data, fit_reg=False, legend=False)
plt.plot(x0,x1_decision, color='grey')
plt.plot(x0,x1_plus,x0,x1_minus,color='grey', linestyle='--')
plt.legend(['decision boundary','margin','margin','Setosa','Versicolor'],
           fontsize = 14, loc='center left', bbox_to_anchor=(1.05,0.5))
plt.xlabel('petal length (scaled)', fontsize = 18)
plt.ylabel('petal width (scaled)', fontsize = 18)
plt.title('[C = 1]', fontsize = 16)
plt.ylim(-1.6,1)
plt.xlim(-1.7,0.8)
plt.show()
```

붓꽃에 LinearSVC 적용 (6)



붓꽃에 LinearSVC 적용 (7)



UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

03. 비선형 SVM
붓꽃 데이터에 적용

이홍석 (hsyi@kisti.re.kr)

붓꽃에 SVM.SVC 적용하기 (1)

SVM의 커널 사용 ¶

#Modeling Different Kernel Svm classifier using Iris Sepal features

```
iris = datasets.load_iris()
X = iris.data[:, :2]  # 꽃받침
y = iris.target
C = 1.0
```

Sepal 꽃받침 길이, 너비를 사용하며,
IRIS 꽃 종류에 대한 제한은 없다. 3종류 모두 사용

```
lin_svc = svm.LinearSVC(C=C)
lin_svc.fit(X, y)
```

```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

붓꽃에 SVM.SVC 적용하기 (2)

Libsvm 라이브러리를 사용하며, 계산 시간은 크기의 3승이다. 따라서 1000개 이상 데이터에는 매우 긴 계산 시간을 요구하므로, 이 경우는 linearSVC 사용을 권장

```
svc = svm.SVC(kernel='linear', C=C)
svc.fit(X, y)
```

```
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='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

SVC 커널은 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'에서 1개를 선택할 것

붓꽃에 SVM.SVC 적용하기 (3)

```
rbf_svc = svm.SVC(kernel='rbf', gamma=0.7, C=C)
rbf_svc.fit(X, y)
```

Rbf와 sigmoid에서 사용하며, gamma=scale(디폴트)

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.7, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
poly_svc = svm.SVC(kernel='poly', degree=3, C=C)
poly_svc.fit(X, y)
```

폴리노미얼 커널 함수의 계수이며, 디폴트는 degree=3이다.

```
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='poly',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

붓꽃에 SVM.SVC 적용하기 (4)

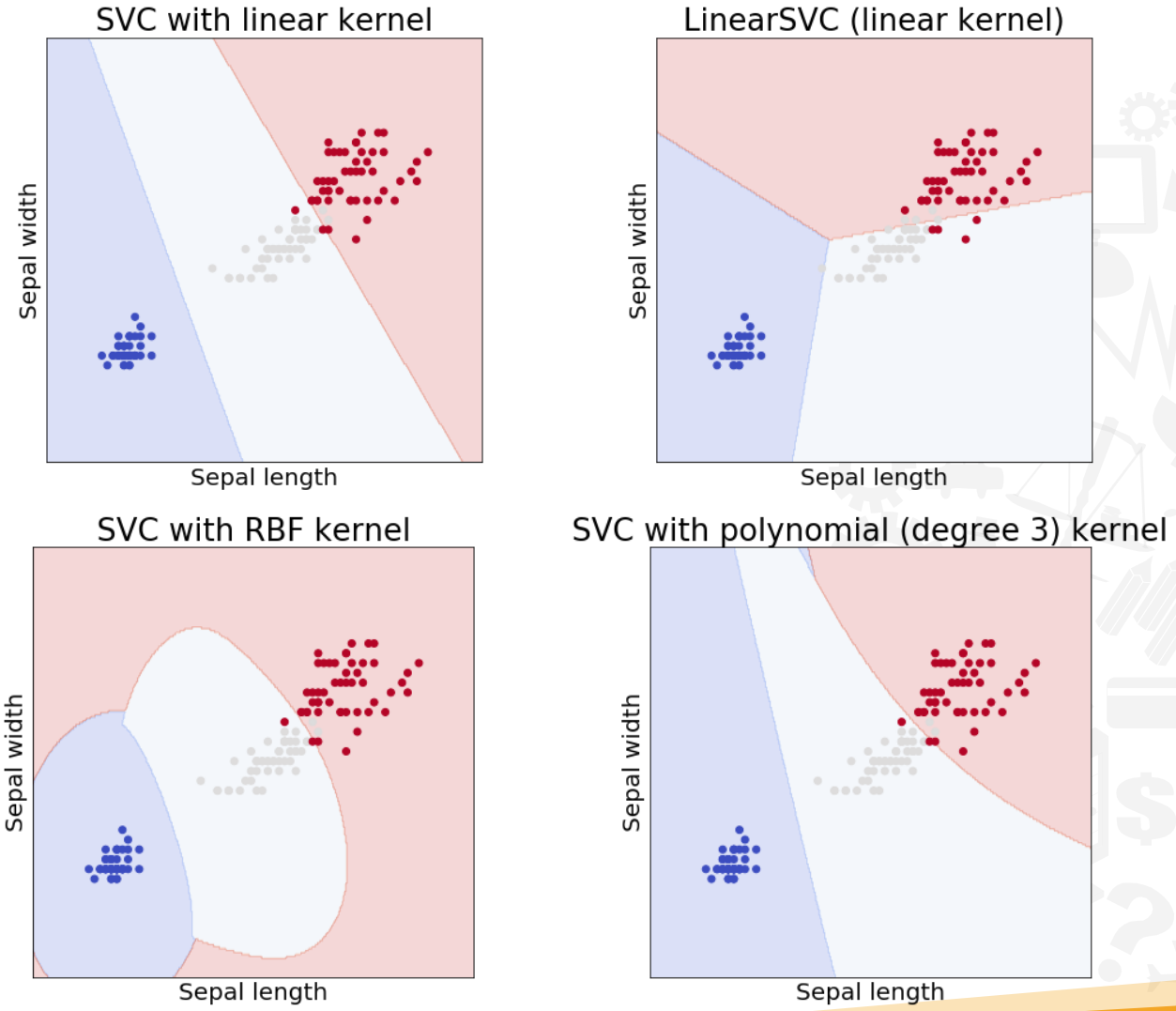
Visualizing the modeled svm classifiers with Iris Sepal features

```
h = .02  # step size in the mesh

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, h),np.arange(y_min, y_max, h))
titles = ['SVC with linear kernel','LinearSVC (linear kernel)',
          'SVC with RBF kernel','SVC with polynomial (degree 3) kernel']
plt.figure(figsize=(16,16))
```

붓꽃에 SVM.SVC 적용하기 (5)

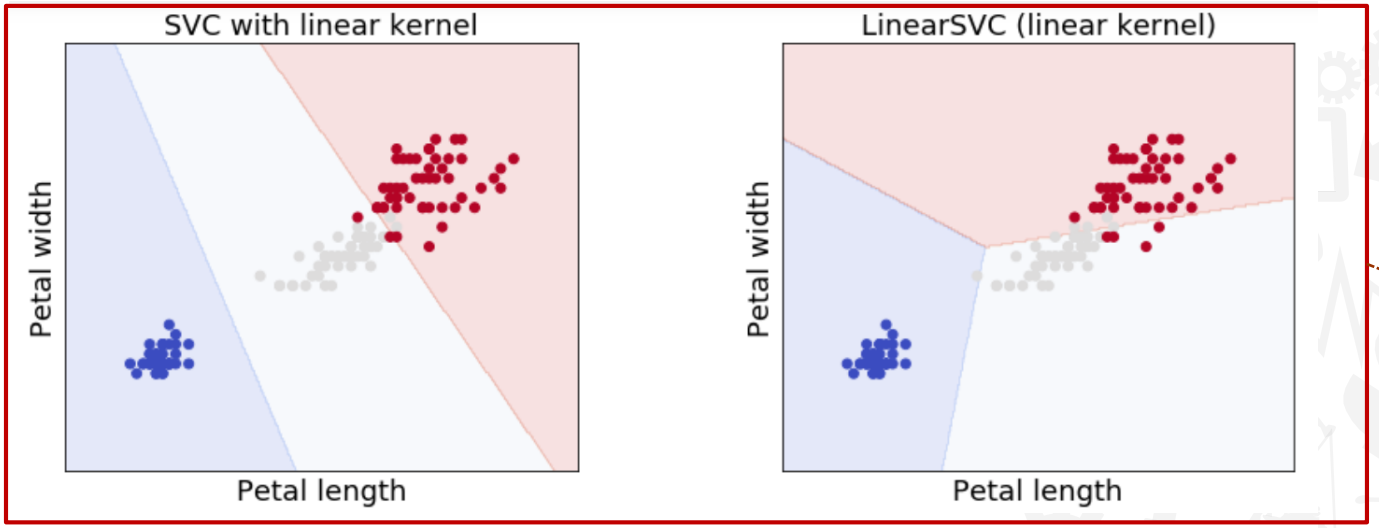
```
for i, clf in enumerate((svc, lin_svc, rbf_svc, poly_svc)):
    # Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    plt.subplot(2, 2, i + 1)
    plt.subplots_adjust(wspace=0.4, hspace=0.4)
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.2)
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
    plt.xlabel('Sepal length',fontsize = 18)
    plt.ylabel('Sepal width',fontsize = 18)
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks(())
    plt.yticks(())
    plt.title(titles[i],fontsize = 18)

plt.show()
```

붓꽃에 SVM.SVC 적용하기 (6)

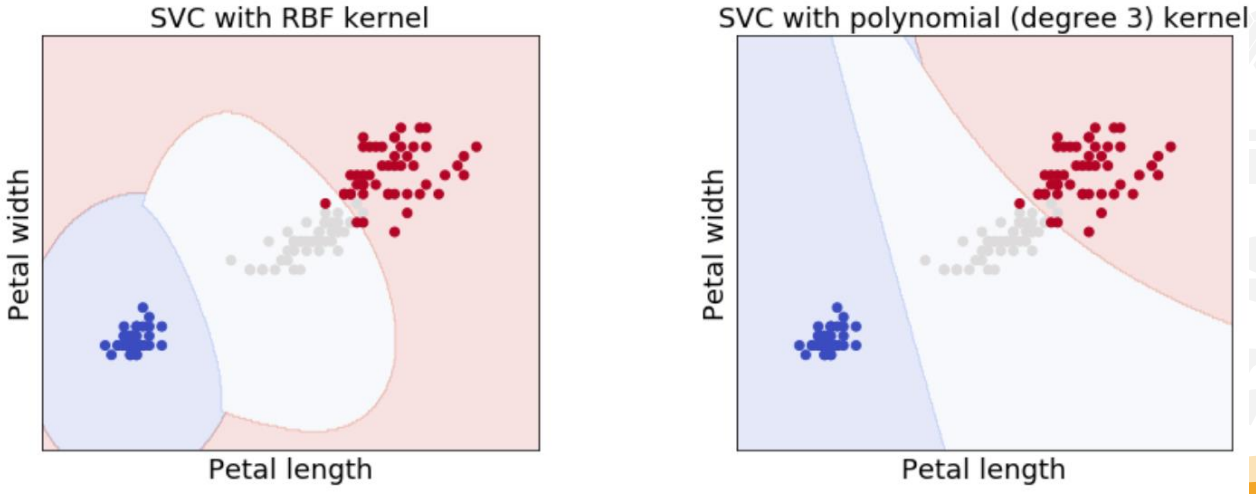


붓꽃에 SVM.SVC 적용하기 (7)

IRIS PETAL (꽃잎) 데이터를 SVM 4개의 다른 종류로 분류



Linear 분류 성능을 비교하면, 어느 것이 더 우수한가?



붓꽃에 SVM.SVC 적용하기 (8)

```
predictions = lin_svc.predict(iris.data[:, :2])
accuracy_score(predictions, iris.target)
```

Iris sepal 꽃받침

```
0.8
```

```
predictions = lin_svc.predict(iris.data[:, 2:])
```

Iris Petal 꽃잎의 경우

```
accuracy_score(predictions, iris.target)
```

```
0.94
```

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(lin_svc, iris.data, iris.target, cv=5)
print("Accuracy: %0.2f (+/- %0.2f)" % (scores.mean(), scores.std() * 2))
```

```
Accuracy: 0.97 (+/- 0.08)
```

Petal(꽃잎) 데이터의 교차검증 성능은 97%로 매우 높다

UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

04. 비선형 SVM
초승달 데이터에 적용

이홍석 (hsyi@kisti.re.kr)

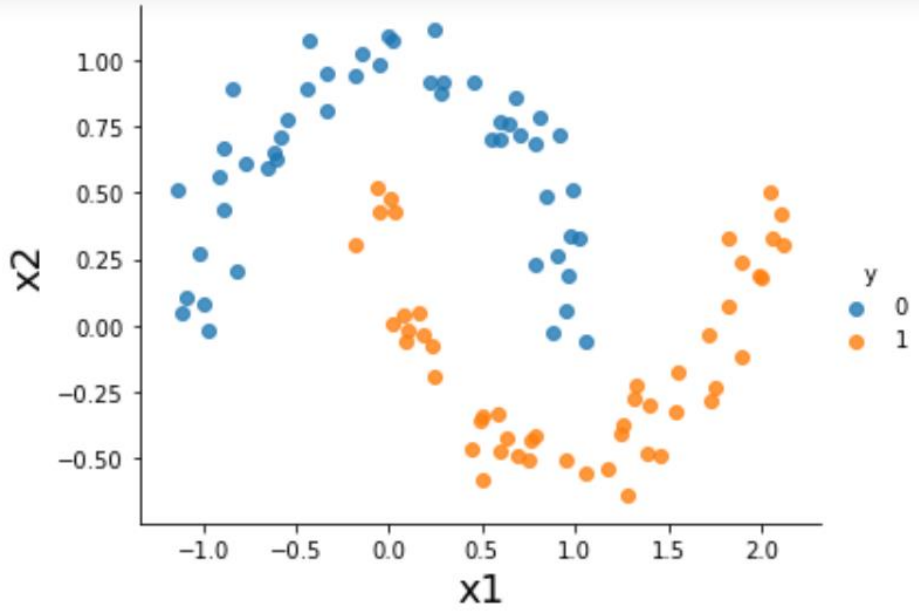
비선형 SVM 적용 (1)

```
from sklearn.datasets import make_moons

X,y=make_moons(noise=0.1, random_state=2)

data = pd.DataFrame(data = X, columns=['x1','x2'])
data['y']=y
data.head()
```

초승달 모양의 데이터 생성

9]:

	x1	x2	y
0	1.327241	-0.222425	1
1	-0.429116	1.071136	0
2	0.014901	0.003679	1
3	0.000352	1.087226	0
4	0.676553	0.857039	0



비선형 SVM 적용 (2)

```
# tranform the features, here we use a 3rd degree polynomials
print('Shape of X before tranformation:', X.shape)
poly = PolynomialFeatures(degree = 3, include_bias=False)
Xpoly = poly.fit_transform(X)
print('Shape of X aftere tranformation:', Xpoly.shape)
```

```
Shape of X before tranformation: (100, 2)
Shape of X aftere tranformation: (100, 9)
```

```
scaler = StandardScaler()
Xpolystan = scaler.fit_transform(Xpoly)

svm_clf = LinearSVC(C=10,loss='hinge',max_iter=10000)
svm_clf.fit(Xpolystan,y)
print(svm_clf.intercept_, svm_clf.coef_)
```

```
[0.14733125] [[-1.48192841 -0.38934979 -3.63169665 -0.24403263  0.84163192  6.20758509
  -0.98197827  0.70828887 -1.94865339]]
```

비선형 SVM 적용 (3)

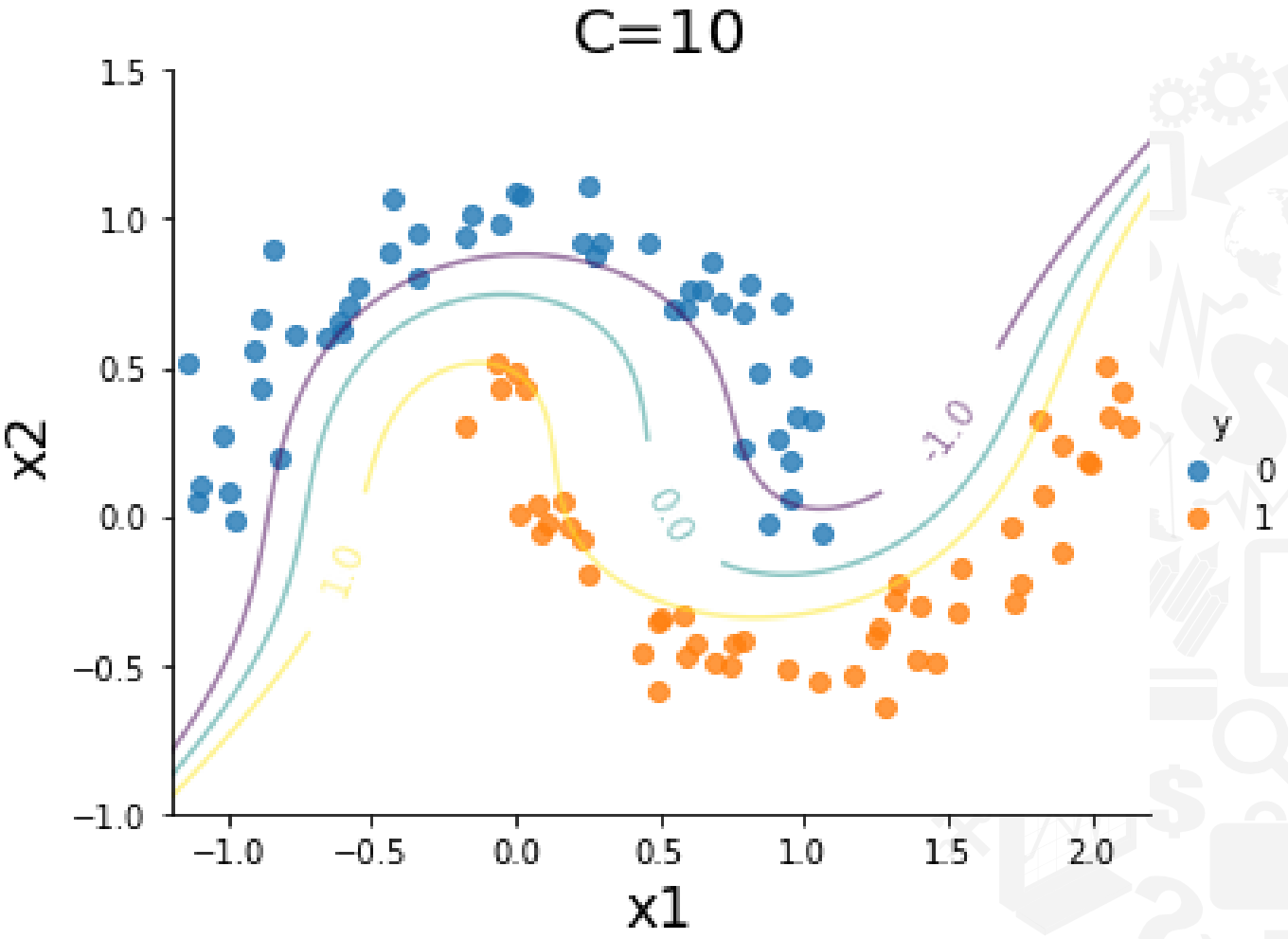
```
# preparing to plot decision boundary of the classifier
def make_meshgrid(x, y, h=.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy
```

```
X0, X1 = X[:, 0], X[:, 1]
xx0, xx1 = make_meshgrid(X0, X1)
# polynomial transformation and standardization on the grids
xgrid = np.c_[xx0.ravel(), xx1.ravel()]
xgridpoly = poly.transform(xgrid)
xgridpolystan = scaler.transform(xgridpoly)
# prediction
Z = xgridpolystan.dot(svm_clf.coef_[0].reshape(-1,1)) + svm_clf.intercept_[0] # wx + b
#Z = svm_clf.predict(xgridpolystan)
Z = Z.reshape(xx0.shape)
```

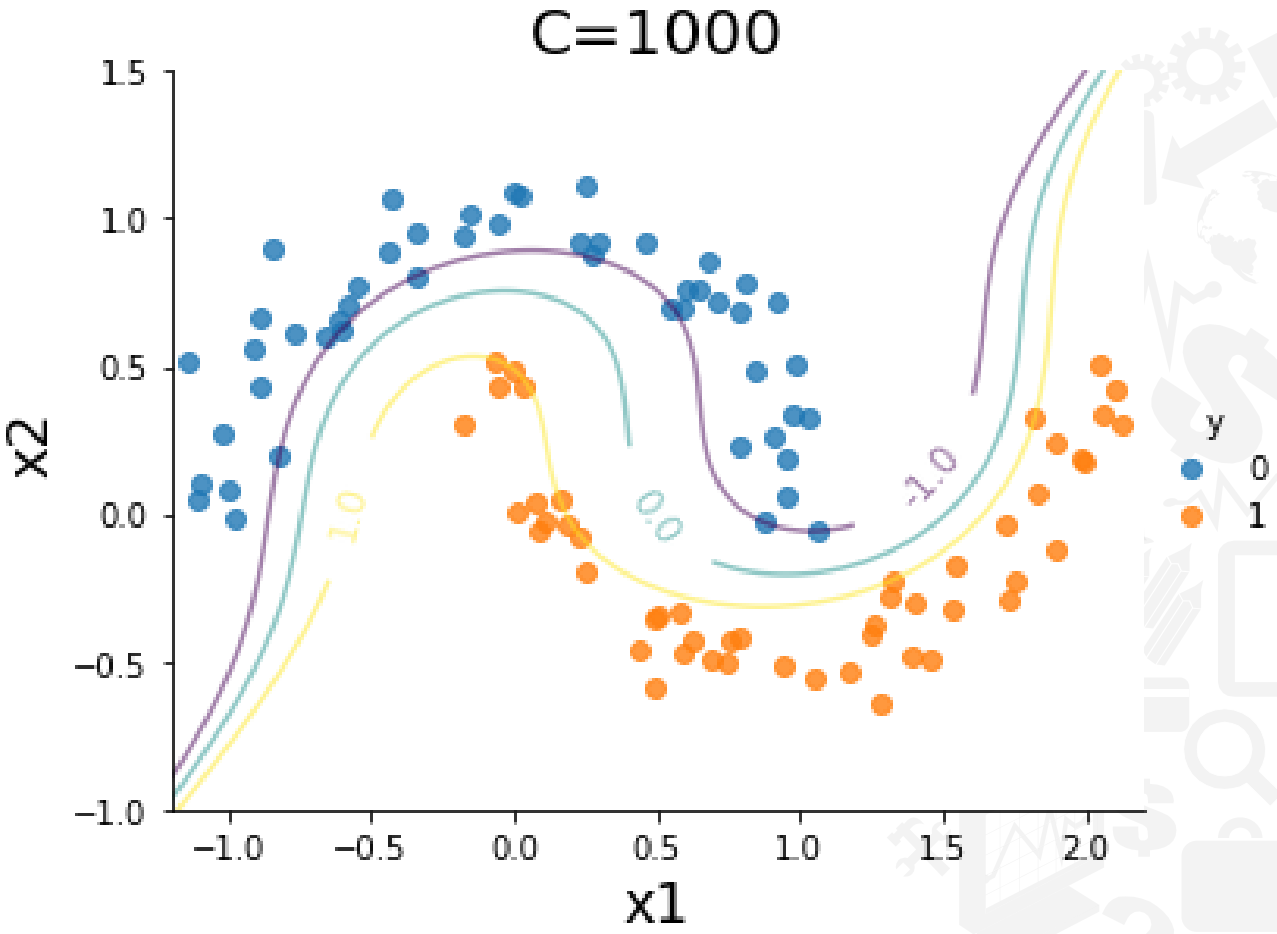
비선형 SVM 적용 (4)

```
# plotting prediction contours - decision boundary (Z=0), and two margins (Z = 1 or -1)
sns.lmplot(x='x1',y='x2',hue='y',data=data,
           fit_reg=False, legend=True, size=4, aspect=4/3)

CS=plt.contour(xx0, xx1, Z, alpha=0.5, levels=[-1,0,1])

#plt.clabel(CS, inline=1,levels=[-1.0,0,1.0], fmt='%1.1f',
plt.clabel(CS, inline=1,fmt='%1.1f',
           fontsize=12, manual=[(1.5,0.3),(0.5,0.0),(-0.5,-0.2)])

#
plt.xlim(-1.2,2.2)
plt.ylim(-1,1.5)
plt.title('C=10', fontsize = 20)
plt.xlabel('x1', fontsize = 18)
plt.ylabel('x2', fontsize = 18)
plt.show()
```

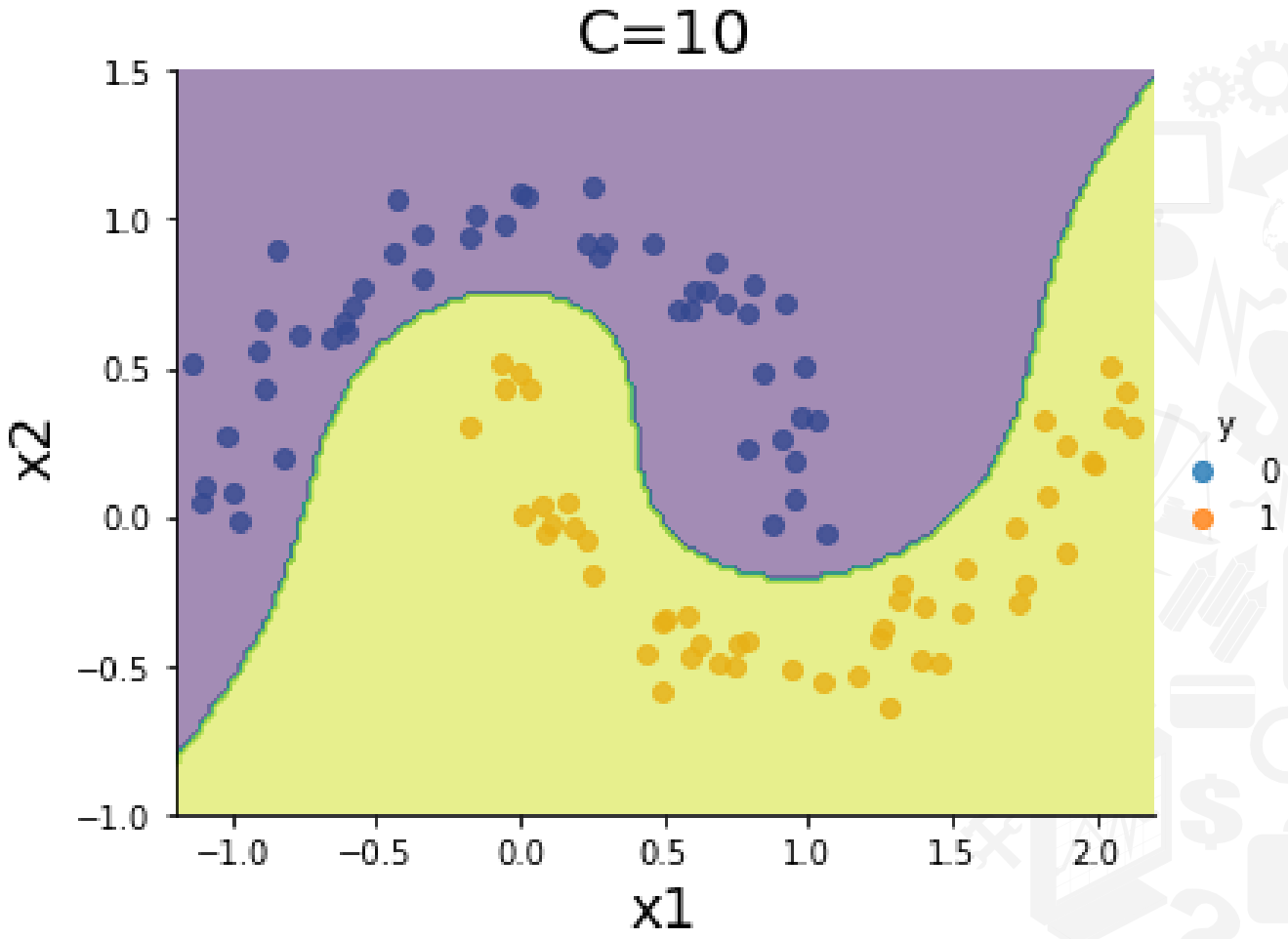
비선형 SVM 적용 (5)



비선형 SVM 적용 (6)



비선형 SVM 적용 (7)



05. SVM 실습
심장질병 데이터 적용

이홍석 (hsyi@kisti.re.kr)

SVM을 심장질병 데이터에 적용(1)

실습05. SVM 적용을 위해 심장병 데이터를 이용하기

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
sns.set_palette('Set1')
```

1) Reading Dataset

```
data = pd.read_csv('./input/heart.csv')
data.head()
```

SVM을 심장질병 데이터에 적용(2)

Reading Dataset

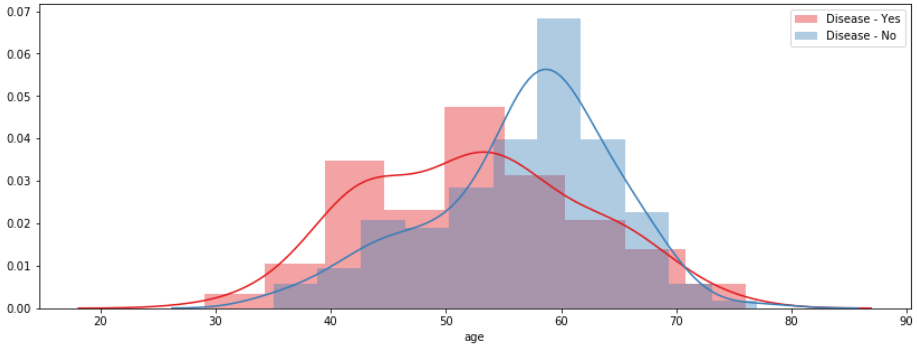
```
data = pd.read_csv('./input/heart.csv')
data.head()
```

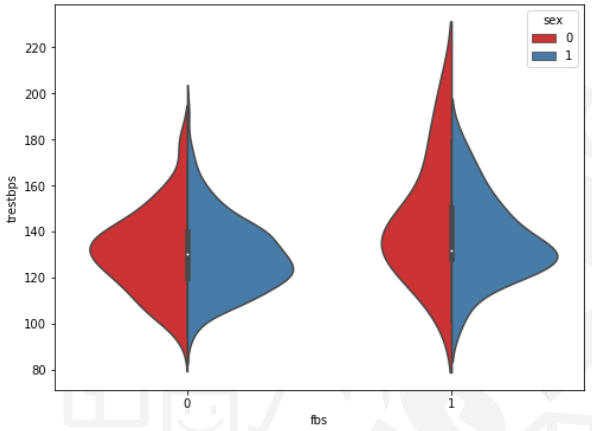
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

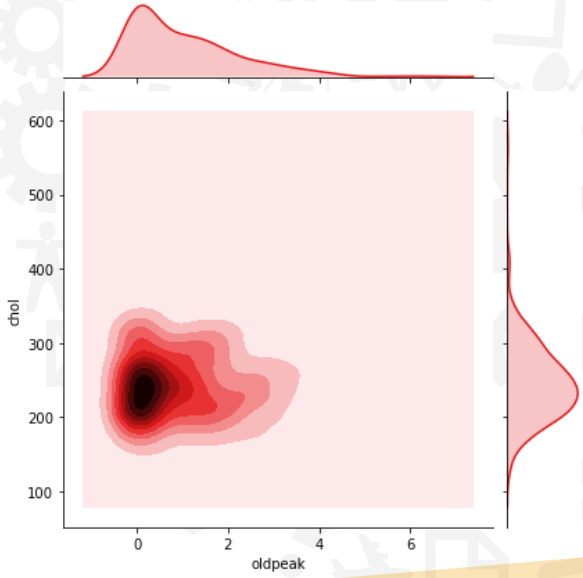
SVM을 심장질병 데이터에 적용(3)

1. age	나이 (int)
2. sex	성별 (1, 0 / int)
3. chest pain type (4 values)	가슴 통증 타입 (0 ~ 3 / int)
4. resting blood pressure	혈압
5. serum cholestoral in mg/dl	혈청 콜레스테롤
6. fasting blood sugar > 120 mg/dl	공복 혈당
7. resting electrocardiographic results	심전도
8. maximum heart rate achieved	최대 심장박동 수
9. exercise induced angina	운동 유도 협심증 (이게 뭐죠?)
10. oldpeak = ST depression induced by exercise relative to rest	노약 =운동에 의해 유발되는 St 우울증 (이건 또 뭐죠?)
11. the slope of the peak exercise ST segment	ST 세그먼트의 기울기
12. number of major vessels (0-3) colored by flourosopy	혈관의수
13. thal : 3 = normal; 6 = fixed defect; 7 = reversable defect	뭔지 모르겠네요

SVM을 심장질병 데이터에 적용(4)







SVM을 심장질병 데이터에 적용(5)

Data Preprocessing ¶

- 카테로리컬 데이터를 변환하지

```
sex = pd.get_dummies(data['sex'])
cp = pd.get_dummies(data['cp'])
fbs = pd.get_dummies(data['fbs'])
restecg = pd.get_dummies(data['restecg'])
exang = pd.get_dummies(data['exang'])
slope = pd.get_dummies(data['slope'])
ca = pd.get_dummies(data['ca'])
thal = pd.get_dummies(data['thal'])
```

```
data = pd.concat([data, sex, cp, fbs, restecg, exang, slope, ca, thal], axis = 1)
```

SVM을 심장질병 데이터에 적용(6)

```
data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	...	2	0	1	2	3	4	0	1	2	3
0	63	1	3	145	233	1	0	150	0	2.3	...	0	1	0	0	0	0	0	1	0	0
1	37	1	2	130	250	0	1	187	0	3.5	...	0	1	0	0	0	0	0	0	1	0
2	41	0	1	130	204	0	0	172	0	1.4	...	1	1	0	0	0	0	0	0	1	0
3	56	1	1	120	236	0	1	178	0	0.8	...	1	1	0	0	0	0	0	0	1	0
4	57	0	0	120	354	0	1	163	1	0.6	...	1	1	0	0	0	0	0	0	1	0

5 rows × 39 columns

```
data.drop(['sex', 'cp', 'fbs', 'restecg', 'exang', 'slope', 'ca', 'thal'], axis = 1, inplace= True)
```

SVM을 심장질병 데이터에 적용(7)

4) Support Vector Machine (스케일 없이 그냥 해볼까?)

```
from sklearn.svm import SVC
model = SVC(probability=True)
from sklearn.model_selection import train_test_split

X = data.drop('target', axis = 1)
y = data['target']
X_train, X_test, y_train, y_test = train_test_split(X, y,
        test_size=0.3, random_state=42)
```

SVM을 심장질병 데이터에 적용(8)

```
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

from sklearn.metrics import classification_report, accuracy_score, roc_curve, auc

print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.78	0.44	0.56	41
1	0.66	0.90	0.76	50
accuracy			0.69	91
macro avg	0.72	0.67	0.66	91
weighted avg	0.72	0.69	0.67	91

SVM을 심장질병 데이터에 적용(9)

5) Support Vector Machine (스케일을 적용하자)

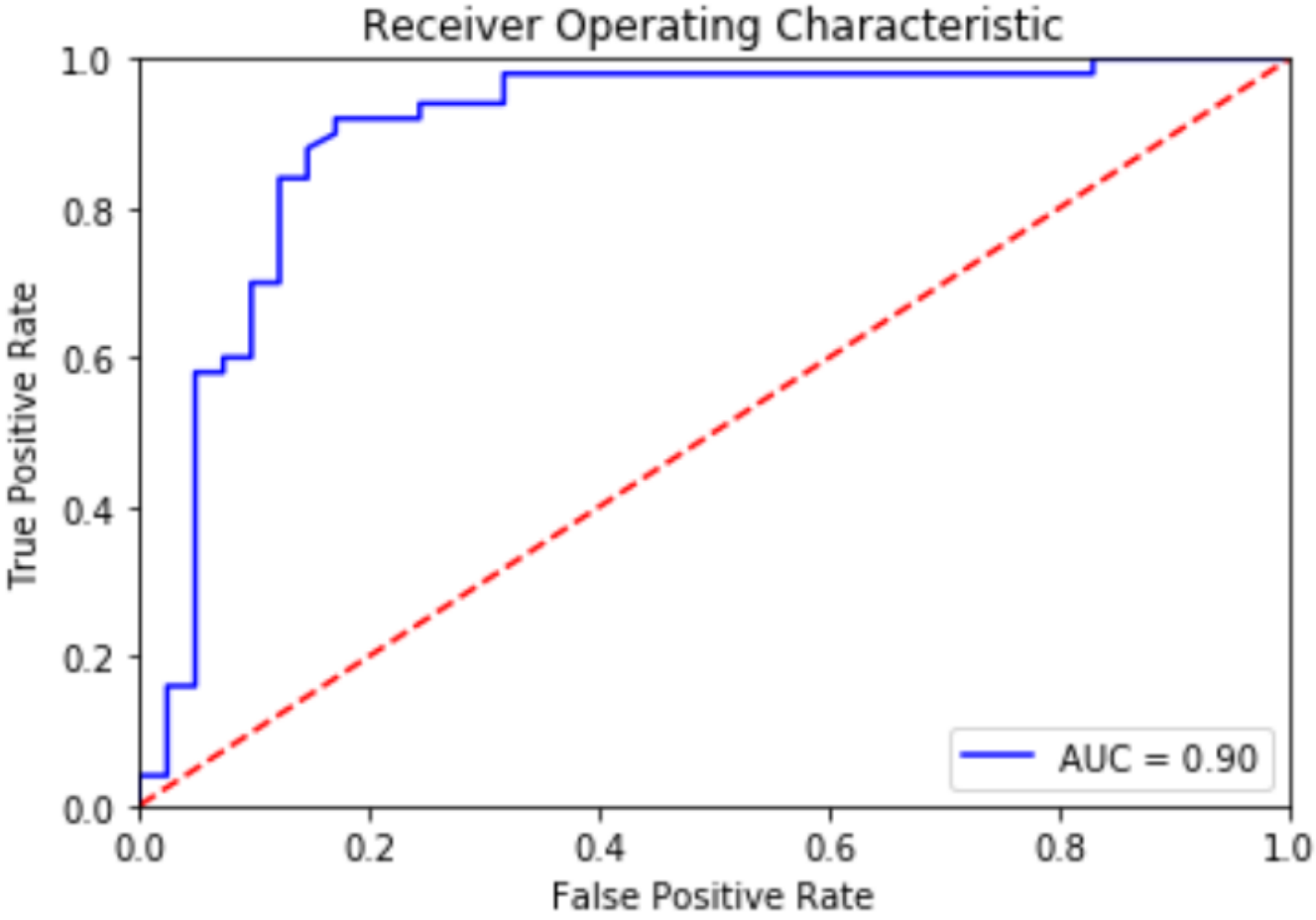
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.85      0.85        41
           1       0.88      0.88      0.88        50

    accuracy                           0.87        91
   macro avg       0.87      0.87      0.87        91
weighted avg       0.87      0.87      0.87        91
```

SVM을 심장질병 데이터에 적용(10)

```
y_prob = model.predict_proba(X_test)
fpr, tpr, thresholds = roc_curve(y_test, y_prob[:,1])
roc_auc = auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

SVM을 심장질병 데이터에 적용(11)

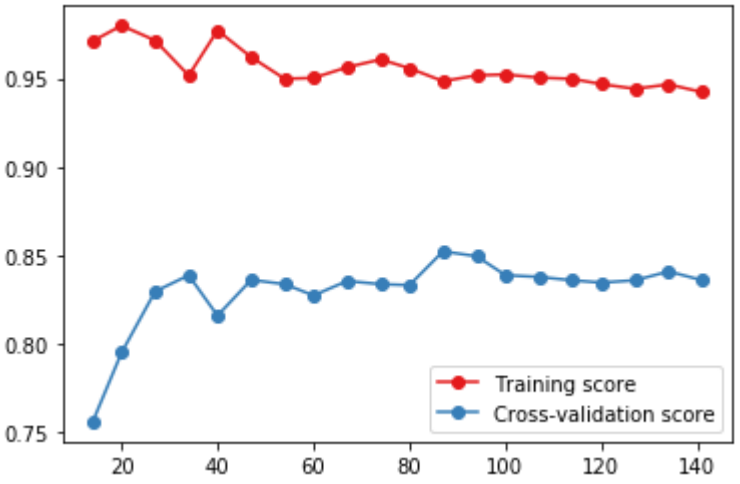


SVM을 심장질병 데이터에 적용(12)

디폴트 C=1

```
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(SVC(), X_train,
    y_train, scoring='f1', train_sizes=np.linspace(0.1, 1.0, 20), cv = 3)

train_scores = np.mean(train_scores, axis = 1)
test_scores = np.mean(test_scores, axis = 1)

plt.plot(train_sizes, train_scores, 'o-', label="Training score")
plt.plot(train_sizes, test_scores, 'o-', label="Cross-validation score")
plt.legend();
```

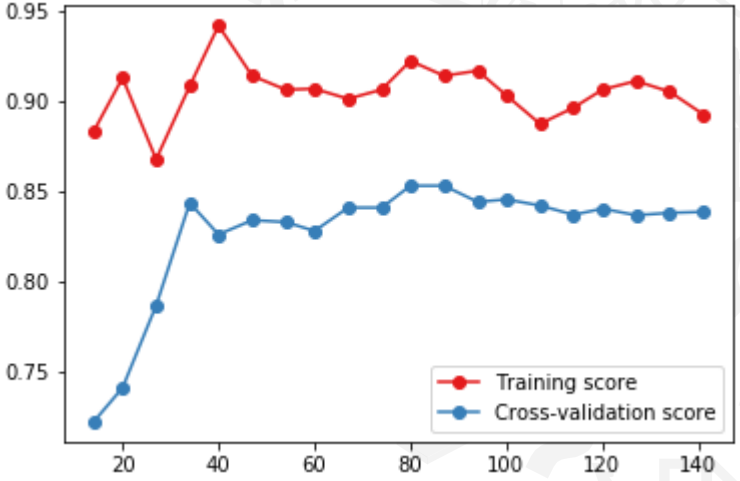
SVM을 심장질병 데이터에 적용(13)




```
learning_curve(SVC(),
```

디폴트 값 사용
C=1

```
learning_curve(SVC(C=3, gamma=0.01)
```



C=3,
Gamma=0.01

SVM을 심장질병 데이터에 적용(14)

- GridSearchCV

```
from sklearn.model_selection import GridSearchCV
param_grid = {'C':[1,2,3,4,5,6,7,8,14],
    'gamma':[0.1, 0.01, 0.001, 0.0001],
    'kernel':['linear', 'poly', 'rbf'],
    'degree': [1,2,3,4,5]}

grid = GridSearchCV(param_grid= param_grid, estimator= SVC(),
    scoring='f1', refit= True, verbose=1)
```

```
grid.fit(X_train, y_train)
grid.best_params_
```

```
Fitting 5 folds for each of 540 candidates, totalling 2700 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 2700 out of 2700 | elapsed:    8.0s finished
```

46]: {'C': 5, 'degree': 1, 'gamma': 0.1, 'kernel': 'poly'}

SVM을 심장질병 데이터에 적용(15)

GridSearchCV를 자세히 파인튜닝을 해보면

```
param_grid = {'C':[6,7,8],
              'gamma':np.linspace(0.01, 0.02, 10),
              'kernel':['rbf'], 'degree': [1,2,3,4,5]}
grid = GridSearchCV(param_grid= param_grid, estimator= SVC(probability= True),
                    scoring='f1', refit= True, verbose=1)
grid.fit(X_train, y_train)
grid.best_params_
```

```
Fitting 5 folds for each of 150 candidates, totalling 750 fits
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 750 out of 750 | elapsed:    4.7s finished
```

```
{'C': 6, 'degree': 1, 'gamma': 0.01, 'kernel': 'rbf'}
```

SVM을 심장질병 데이터에 적용(16)

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

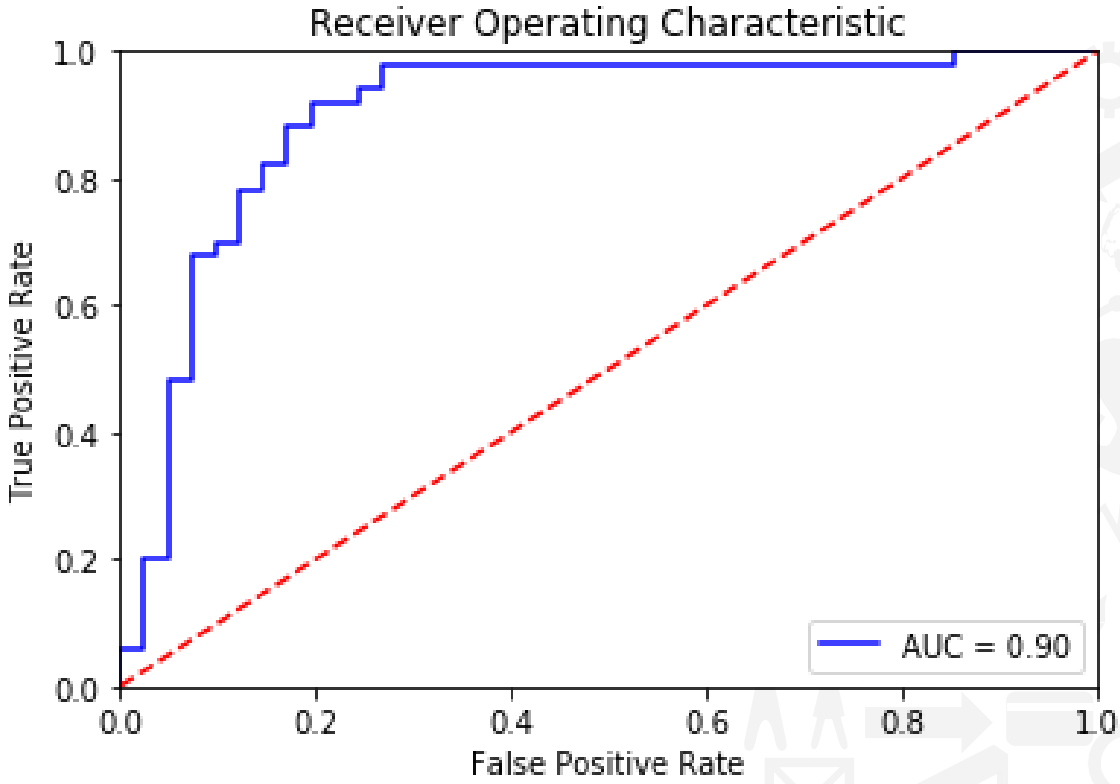
```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.80      0.83        41
           1       0.85      0.88      0.86        50

    accuracy                           0.85        91
   macro avg       0.85      0.84      0.84        91
weighted avg       0.85      0.85      0.85        91
```

SVM을 심장질병 데이터에 적용(17)

```
y_prob = grid.predict_proba(X_test)

fpr, tpr, thresholds = roc_curve(y_test, y_prob[:,1])
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

SVM을 심장질병 데이터에 적용(18)




```
result = pd.DataFrame({'Test':y_test, 'Prediction':y_pred, 'Probability': y_prob[:,1]})
result.to_csv('Result.csv')
```

결과를 파일로 저장함

06. 실습 (옵션)
(Tree, 램덤 포레스트 로지스틱 회귀) 타이타닉 데이터

이홍석 (hsyi@kisti.re.kr)

Kaggle ~ 머신러닝 분석 대회 주관

- 타이타닉 데이터~Kaggle에서 구함
  - 파이썬 강의 ch12장 결정트리의 예제로 사용된 데이터를 이용함
  - ./input 폴더를 만들고
    - 2개 파일 train.csv와 test.csv
- 사용 라이브러리
  - Matplotlib
  - Seaborn
  - Pandas 등

(실습) 타이타닉 생존자 예측 (1)

데이터 위치는 교재하고는 다름. 이름 변경.주의

```
titanic_df = pd.read_csv('./input/train.csv')
titanic_df.head(5)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

(실습) 타이타닉 생존자 예측 (2)

```
print('₩n ### train 데이터 정보 ###  ₩n')
print(titanic_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
None
```

(실습) 타이타닉 생존자 예측 (3)

```
titanic_df['Age'].fillna(titanic_df['Age'].mean(),inplace=True)
titanic_df['Cabin'].fillna('N',inplace=True)
titanic_df['Embarked'].fillna('N',inplace=True)
print('데이터 세트 Null 값 갯수 ',titanic_df.isnull().sum().sum())
```

```
데이터 세트 Null 값 갯수  0
```

```
print(' Sex 값 분포 :₩n',titanic_df['Sex'].value_counts())
print('₩n Cabin 값 분포 :₩n',titanic_df['Cabin'].value_counts())
print('₩n Embarked 값 분포 :₩n',titanic_df['Embarked'].value_counts())
```

```
 Sex 값 분포 :
 male      577
female     314
Name: Sex, dtype: int64

 Cabin 값 분포 :
 N                   687
G6                    4
C23 C25 C27           4
B96 B98               4
F33                   3
C22 C26               3
```

(실습) 타이타닉 생존자 예측 (4)

```
titanic_df.groupby(['Sex','Survived'])['Survived'].count()
```

```
Sex     Survived
female  0            81
        1           233
male    0           468
        1           109
Name: Survived, dtype: int64
```

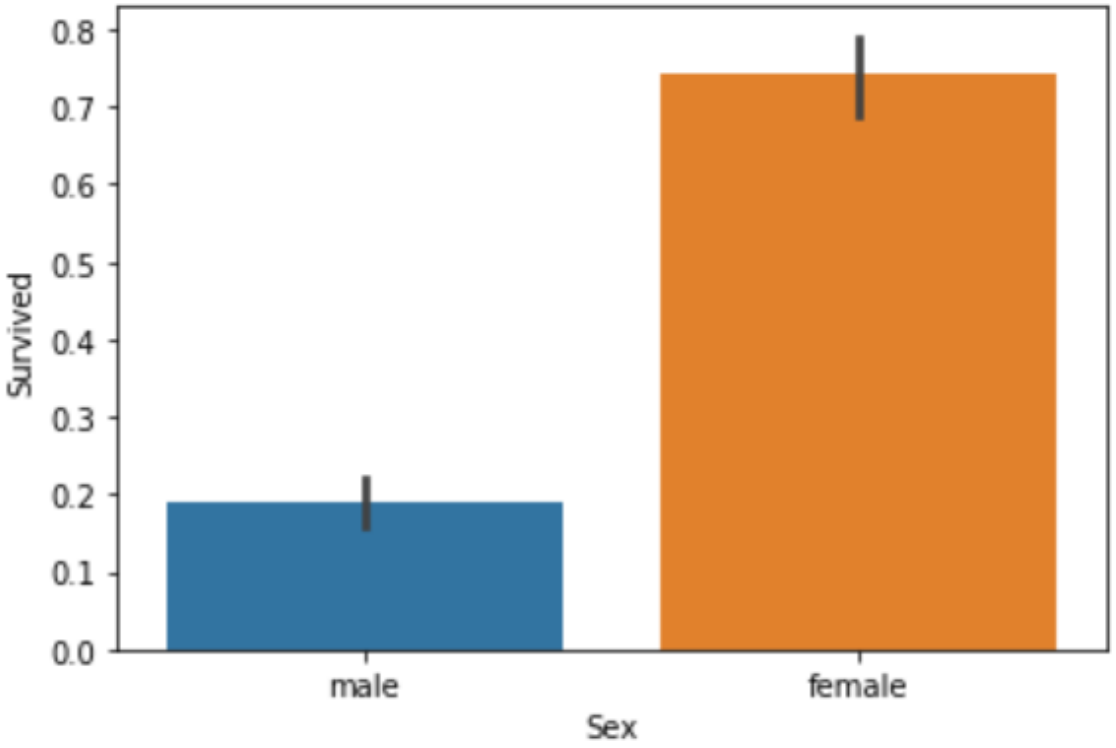
```
titanic_df['Cabin'] = titanic_df['Cabin'].str[:1]
print(titanic_df['Cabin'].head(12))
```

```
0     N
1     C
2     N
3     C
4     N
5     N
6     E
7     N
8     N
9     N
10    G
11    C
Name: Cabin, dtype: object
```

(실습) 타이타닉 생존자 예측 (5)

```
sns.barplot(x='Sex', y = 'Survived', data=titanic_df)
```

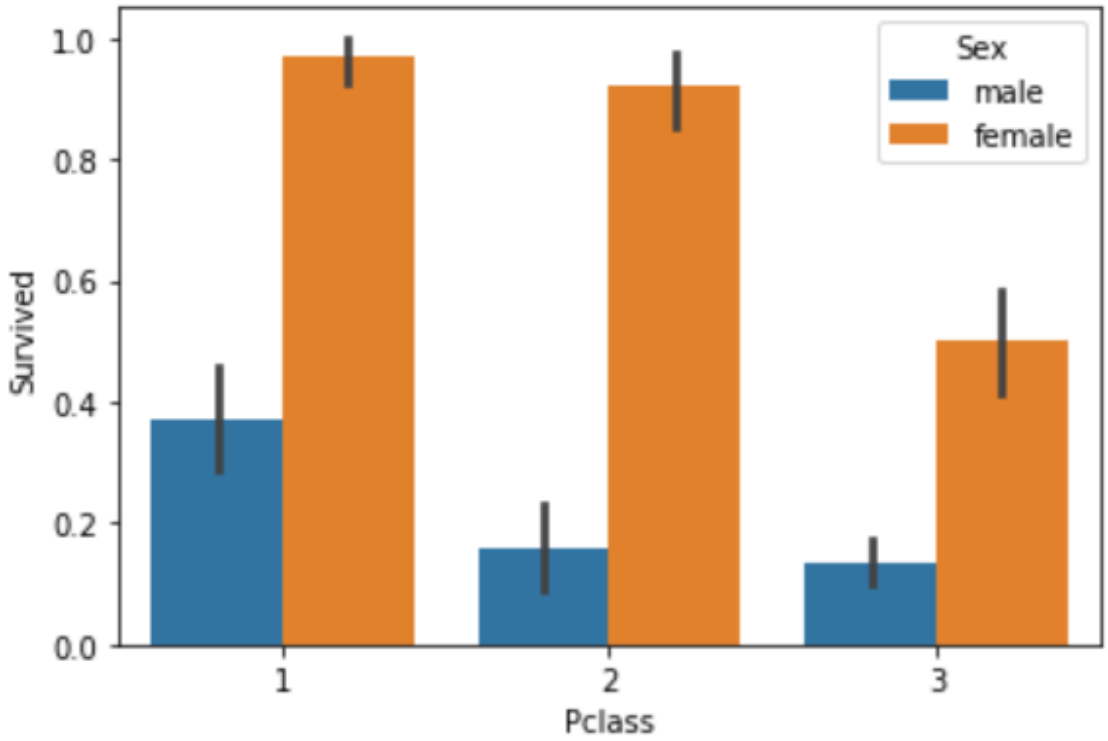
```
<matplotlib.axes._subplots.AxesSubplot at 0x2a8d613d080>
```



(실습) 타이타닉 생존자 예측 (5)

```
sns.barplot(x='Pclass', y='Survived', hue='Sex', data=titanic_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2a8d64d2048>
```



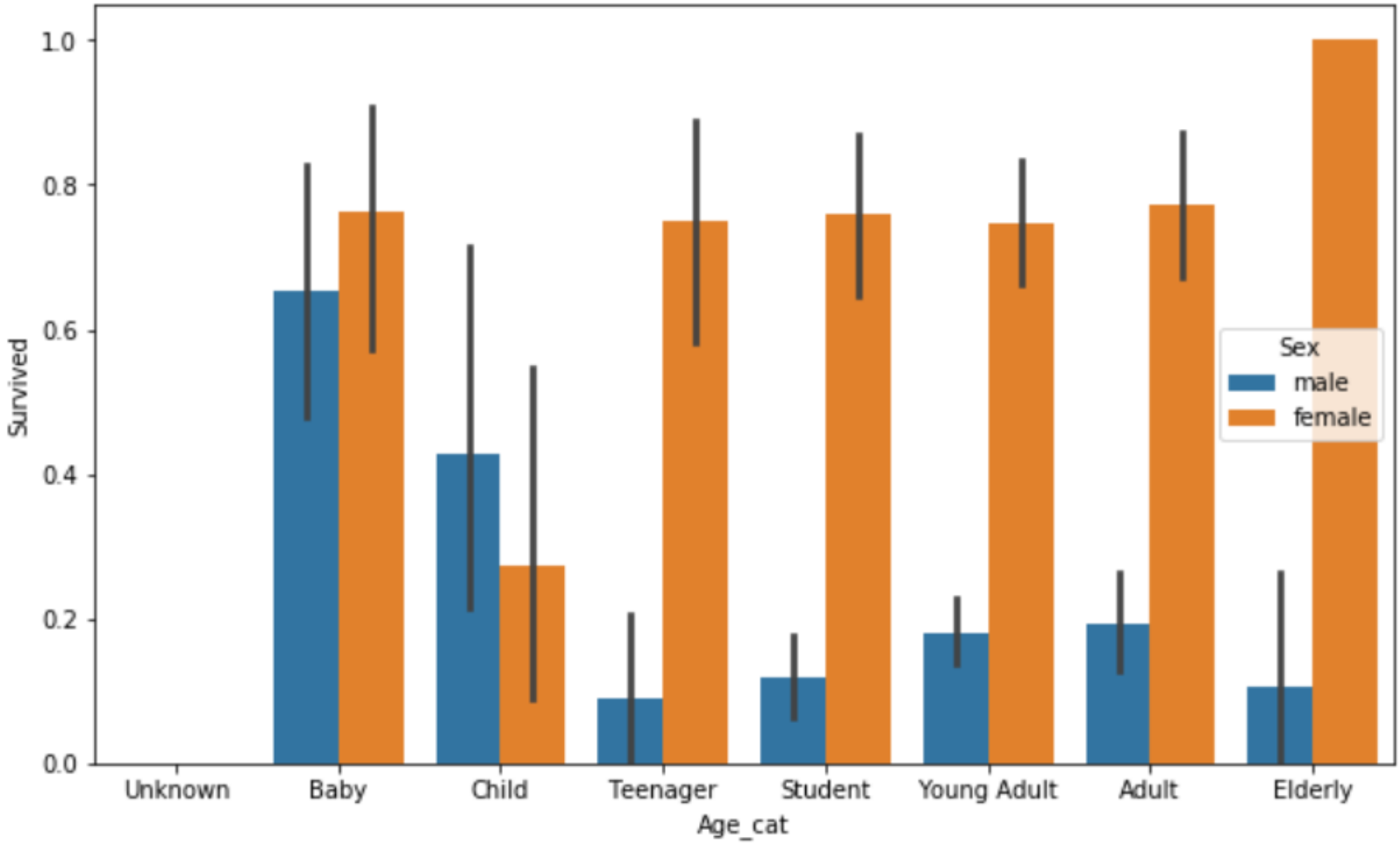
(실습) 타이타닉 생존자 예측 (6)

```
# 입력 age에 따라 구분값을 반환하는 함수 설정. DataFrame의
def get_category(age):
    cat = ''
    if age <= -1: cat = 'Unknown'
    elif age <= 5: cat = 'Baby'
    elif age <= 12: cat = 'Child'
    elif age <= 18: cat = 'Teenager'
    elif age <= 25: cat = 'Student'
    elif age <= 35: cat = 'Young Adult'
    elif age <= 60: cat = 'Adult'
    else : cat = 'Elderly'
```

(실습) 타이타닉 생존자 예측 (6)

```
# 막대그래프의 크기 figure를 더 크게 설정
plt.figure(figsize=(10,6))

#X축의 값을 순차적으로 표시하기 위한 설정
group_names = ['Unknown', 'Baby', 'Child', 'Teenager', 'Student', 'Young Adult', 'Adult',

# lambda 식에 위에서 생성한 get_category( ) 함수를 반환값으로 지정.
# get_category(X)는 입력값으로 'Age' 컬럼값을 받아서 해당하는 cat 반환
titanic_df['Age_cat'] = titanic_df['Age'].apply(lambda x : get_category(x))
sns.barplot(x='Age_cat', y = 'Survived', hue='Sex', data=titanic_df, order=group_names)
titanic_df.drop('Age_cat', axis=1, inplace=True)
```

(실습) 타이타닉 생존자 예측 (7)



(실습) 타이타닉 생존자 예측 (8)

```
from sklearn import preprocessing

def encode_features(dataDF):
    features = ['Cabin', 'Sex', 'Embarked']
    for feature in features:
        le = preprocessing.LabelEncoder()
        le = le.fit(dataDF[feature])
        dataDF[feature] = le.transform(dataDF[feature])

    return dataDF

titanic_df = encode_features(titanic_df)
titanic_df.head()
```

(실습) 타이타닉 생존자 예측 (8)

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	1	22.0	1	0	A/5 21171	7.2500	7	3
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	0	38.0	1	0	PC 17599	71.2833	2	0
2	3	1	3	Heikkinen, Miss. Laina	0	26.0	0	0	STON/O2. 3101282	7.9250	7	3
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	35.0	1	0	113803	53.1000	2	3
4	5	0	3	Allen, Mr. William Henry	1	35.0	0	0	373450	8.0500	7	3

(실습) 타이타닉 생존자 예측 (9)

```
from sklearn.preprocessing import LabelEncoder

# Null 처리 함수
def fillna(df):
    df['Age'].fillna(df['Age'].mean(),inplace=True)
    df['Cabin'].fillna('N',inplace=True)
    df['Embarked'].fillna('N',inplace=True)
    df['Fare'].fillna(0,inplace=True)
    return df

# 머신러닝 알고리즘에 불필요한 속성 제거
def drop_features(df):
    df.drop(['PassengerId','Name','Ticket'],axis=1,inplace=True)
    return df
```

(실습) 타이타닉 생존자 예측 (9)

```
# 레이블 인코딩 수행.
def format_features(df):
    df['Cabin'] = df['Cabin'].str[:1]
    features = ['Cabin','Sex','Embarked']
    for feature in features:
        le = LabelEncoder()
        le = le.fit(df[feature])
        df[feature] = le.transform(df[feature])
    return df

# 앞에서 설정한 Data Preprocessing 함수 호출
def transform_features(df):
    df = fillna(df)
    df = drop_features(df)
    df = format_features(df)
    return df
```

(실습) 타이타닉 생존자 예측 (10)

```
# 원본 데이터를 재로딩 하고, feature데이터 셋과 Label 데이터 셋 추출.
titanic_df = pd.read_csv('./input/train.csv')
y_titanic_df = titanic_df['Survived']
X_titanic_df= titanic_df.drop('Survived',axis=1)

X_titanic_df = transform_features(X_titanic_df)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(X_titanic_df, y_titanic_df, ₩
                                                  test_size=0.2, random_state=11)
```

(실습) 타이타닉 생존자 예측 (11)

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# 결정트리, Random Forest, 로지스틱 회귀를 위한 사이킷런 (
dt_clf = DecisionTreeClassifier(random_state=11)
rf_clf = RandomForestClassifier(random_state=11)
lr_clf = LogisticRegression()
```

```
# DecisionTreeClassifier 학습/예측/평가
dt_clf.fit(X_train , y_train)
dt_pred = dt_clf.predict(X_test)
print('DecisionTreeClassifier 정확도: {0:.4f}'.format(accuracy_score(y_test, dt_pred)))
```

(실습) 타이타닉 생존자 예측 (11)

```
# RandomForestClassifier 학습/예측/평가
rf_clf.fit(X_train , y_train)
rf_pred = rf_clf.predict(X_test)
print('RandomForestClassifier 정확도:{0:.4f}'.format(accuracy_score(y_test, rf_pred)))

# LogisticRegression 학습/예측/평가
lr_clf.fit(X_train , y_train)
lr_pred = lr_clf.predict(X_test)
print('LogisticRegression 정확도: {0:.4f}'.format(accuracy_score(y_test, lr_pred)))
```

```
DecisionTreeClassifier 정확도: 0.7877
RandomForestClassifier 정확도:0.8547
LogisticRegression 정확도: 0.8492
```

(실습) 타이타닉 생존자 예측 (12)

```
from sklearn.model_selection import KFold

def exec_kfold(clf, folds=5):
    # 폴드 세트를 5개인 KFold객체를 생성, 폴드 수만큼 예측결과 저장을 위한  리스트 객체 생성.
    kfold = KFold(n_splits=folds)
    scores = []

    # KFold 교차 검증 수행.
    for iter_count , (train_index, test_index) in enumerate(kfold.split(X_titanic_df)):
        # X_titanic_df 데이터에서 교차 검증별로 학습과 검증 데이터를 가리키는 index 생성
        X_train, X_test = X_titanic_df.values[train_index], X_titanic_df.values[test_index]
        y_train, y_test = y_titanic_df.values[train_index], y_titanic_df.values[test_index]

        # Classifier 학습, 예측, 정확도 계산
        clf.fit(X_train, y_train)
        predictions = clf.predict(X_test)
        accuracy = accuracy_score(y_test, predictions)
        scores.append(accuracy)
        print("교차 검증 {0} 정확도: {1:.4f}".format(iter_count, accuracy))

    # 5개 fold에서의 평균 정확도 계산.
    mean_score = np.mean(scores)
    print("평균 정확도: {0:.4f}".format(mean_score))
# exec_kfold 호출
```

(실습) 타이타닉 생존자 예측 (12)

```
from sklearn.model_selection import KFold
```

```
def exec_kfold(clf, folds=5):
    # 폴드 세트를 5개인 KFold객체를 생성, 폴드 수만큼 예측결과 저장을 위한  리스트 객체 생성
    kfold = KFold(n_splits=folds)
    scores = []

    # KFold 교차 검증 수행.
    for iter_count , (train_index, test_index) in enumerate(kfold.split(X_titanic_df)):
        # X_titanic_df 데이터에서 교차 검증별로 학습과 검증 데이터를 가리키는 index 생성
        X_train, X_test = X_titanic_df.values[train_index], X_titanic_df.values[test_index]
        y_train, y_test = y_titanic_df.values[train_index], y_titanic_df.values[test_index]

        # Classifier 학습, 예측, 정확도 계산
        clf.fit(X_train, y_train)
        predictions = clf.predict(X_test)
        accuracy = accuracy_score(y_test, predictions)
        scores.append(accuracy)
        print("교차 검증 {0} 정확도: {1:.4f}".format(iter_count, accuracy))

    # 5개 fold에서의 평균 정확도 계산.
    mean_score = np.mean(scores)
    print("평균 정확도: {0:.4f}".format(mean_score))
```

(실습) 타이타닉 생존자 예측 (12)

결정 트리 K-폴드 5개 교차 검증한 성능 78%

```
exec_kfold(dt_clf , folds=5)
```

```
교차 검증 0 정확도: 0.7542
교차 검증 1 정확도: 0.7809
교차 검증 2 정확도: 0.7865
교차 검증 3 정확도: 0.7697
교차 검증 4 정확도: 0.8202
평균 정확도: 0.7823
```

크로스 교차 검증을 한 성능 78.8%

(실습) 타이타닉 생존자 예측 (13)

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(dt_clf, X_titanic_df , y_titanic_df , cv=5)
for iter_count,accuracy in enumerate(scores):
    print("교차 검증 {0} 정확도: {1:.4f}".format(iter_count, accuracy))

print("평균 정확도: {0:.4f}".format(np.mean(scores)))
```

```
교차 검증 0 정확도: 0.7430
교차 검증 1 정확도: 0.7753
교차 검증 2 정확도: 0.7921
교차 검증 3 정확도: 0.7865
교차 검증 4 정확도: 0.8427
평균 정확도: 0.7879
```

- 결정트리를 이용할 경우 최적의 하이퍼-파라미터를 찾기 위한 Grid-Search를 수행
- 테스트 셋트에서 정확도 는 87.15%

(실습) 타이타닉 생존자 예측 (14)

```
from sklearn.model_selection import GridSearchCV

parameters = {'max_depth':[2,3,5,10],
              'min_samples_split':[2,3,5], 'min_samples_leaf':[1,5,8]}

grid_dclf = GridSearchCV(dt_clf , param_grid=parameters , scoring='accuracy' , cv=5)
grid_dclf.fit(X_train , y_train)

print('GridSearchCV 최적 하이퍼 파라미터 :',grid_dclf.best_params_)
print('GridSearchCV 최고 정확도: {0:.4f}'.format(grid_dclf.best_score_))
best_dclf = grid_dclf.best_estimator_

# GridSearchCV의 최적 하이퍼 파라미터로 학습된 Estimator로 예측 및 평가 수행.
dpredictions = best_dclf.predict(X_test)
accuracy = accuracy_score(y_test , dpredictions)
print('테스트 세트에서의 DecisionTreeClassifier 정확도 : {0:.4f}'.format(accuracy))
```

```
GridSearchCV 최적 하이퍼 파라미터 : {'max_depth': 3, 'min_samples_leaf': 5, 'min_samples_split': 2}
GridSearchCV 최고 정확도: 0.7992
테스트 세트에서의 DecisionTreeClassifier 정확도 : 0.8715
```

UST 과학기술연합대학원대학교
UNIVERSITY OF SCIENCE & TECHNOLOGY

Thank You!

www.ust.ac.kr