Machine Learning

Support Vector Machine

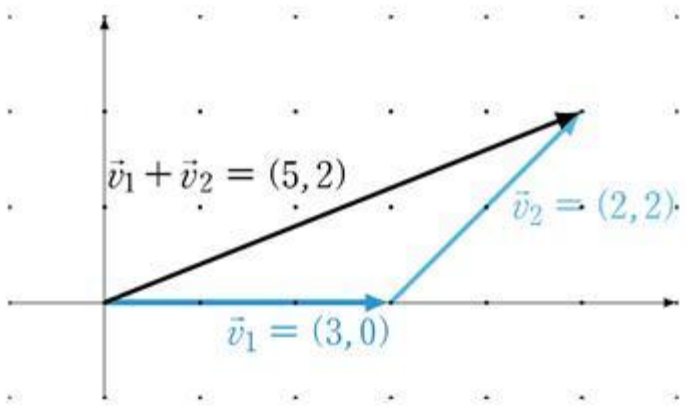
김선녕(sykim.lecture@gmail.com)

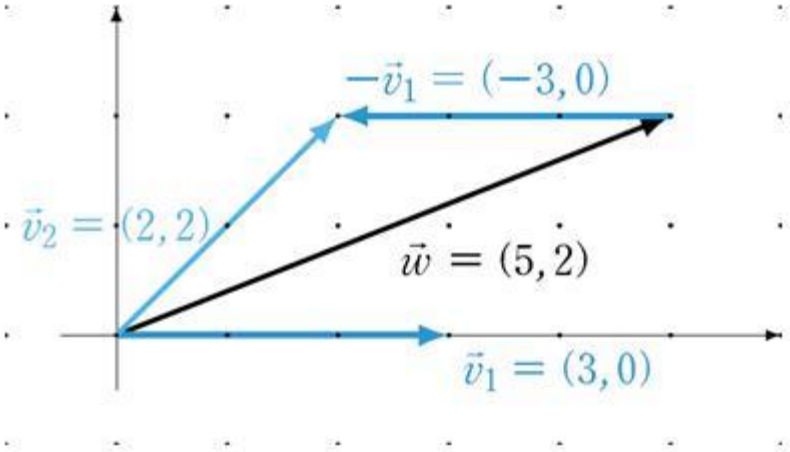
SVM분류

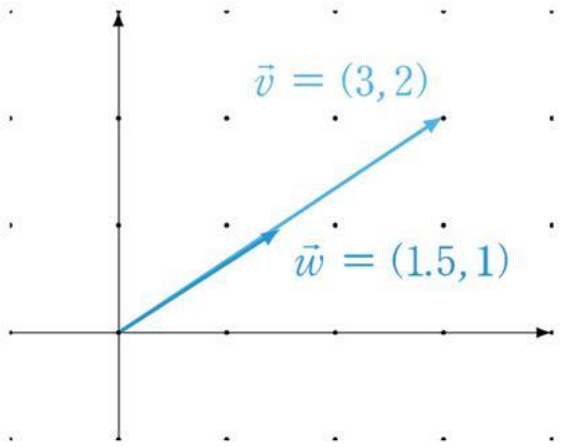
커널트릭

쌍대문제 - 라그랑주 승수

벡터의 덧셈과 뺄셈


$\vec{v}_1 + \vec{v}_2 = (5, 2)$
$\vec{v}_2 = (2, 2)$
$\vec{v}_1 = (3, 0)$


$-\vec{v}_1 = (-3, 0)$
$\vec{v}_2 = (2, 2)$
$\vec{w} = (5, 2)$
$\vec{v}_1 = (3, 0)$


$\vec{v} = (3, 2)$
$\vec{w} = (1.5, 1)$

노름(norm) : 거리측정, 벡터의 크기를 계산하는 함수

- 유클리디안 노름$(Euclidean\ norm)$: $\boldsymbol{l_2}$노름. $\|\cdot\|_2$ 혹은 $\|\cdot\|$로 표시

$$\|\vec{v}\|_2 = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} = \sqrt{\sum_{i=1}^{n} v_i^2}$$

- 맨해튼 노름$(Manhattan\ Norm)$: $\boldsymbol{l_1}$노름. $\|\cdot\|_1$ 로 표시

$$\|\vec{v}\|_1 = |v_1| + |v_2| + \cdots + |v_n| = \sum_{i=1}^{n} |v_i|$$

- 원소 n개인 벡터 v의 l_p노름

$$\|\vec{v}\|_p = \sqrt[p]{|v_0|^p + |v_1|^p + \cdots + |v_n|^p} = (|v_0|^p + |v_1|^p + \cdots + |v_n|^p)^{\frac{1}{p}}$$

$$= \left(\sum_{i=1}^{n} |v_i|^p\right)^{\frac{1}{p}}$$

  - $\boldsymbol{l_0}$: 단순히 벡터에 있는 0이 아닌 원소의 수
  - $\boldsymbol{p = 1}$ 인 경우 $\boldsymbol{l_1}$노름, $\boldsymbol{p = 2}$ 인 경우 $\boldsymbol{l_2}$노름
- 최대값 노름$(Maximum\ Norm)$: $\boldsymbol{l_\infty}$ 노름. 벡터에서 가장 큰 절댓값

$$\|\vec{v}\|_\infty = \mathbf{max}(|v_1|, |v_2|, \cdots, |v_n|)$$

노름 예제

$$v = \begin{bmatrix} 1 \\ -2 \\ 3 \end{bmatrix}$$

$$\|\vec{v}\|_2 = \sqrt{v_1^2 + v_2^2 + v_3^2} = \sqrt{1^2 + (-2)^2 + 3^2} = \sqrt{14}$$

$$\|\vec{v}\|_1 = |v_1| + |v_2| + |v_3| = |1| + |-2| + |3| = 6$$

$$\|\vec{v}\|_2 = (|1|^2 + |-2|^2 + |3|^2)^{\frac{1}{2}} = \sqrt{14} \qquad p = 2 \text{일때}$$

$$\|\vec{v}\|_\infty = \max(|1|, |-2|, |3|) = 3$$

단위벡터(unit vector)

단위벡터 : 노름이 1인 벡터

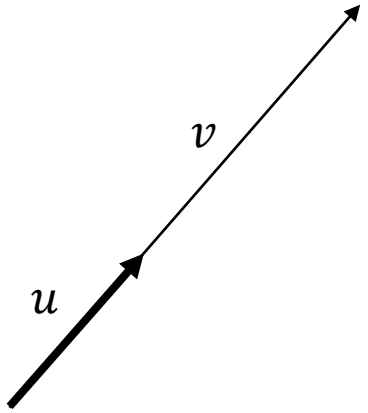
벡터 v의 단위 벡터 $\boldsymbol{u} = \dfrac{\boldsymbol{v}}{\|\boldsymbol{v}\|}$

(ex)

$$v = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

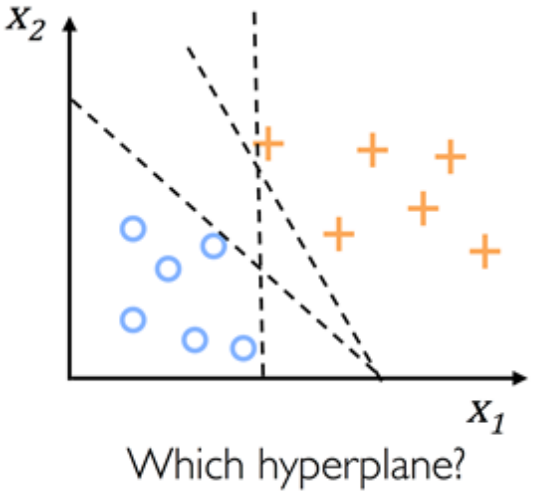
$$\|v\| = \sqrt{1^2 + (2)^2 + 3^2} = \sqrt{14}$$

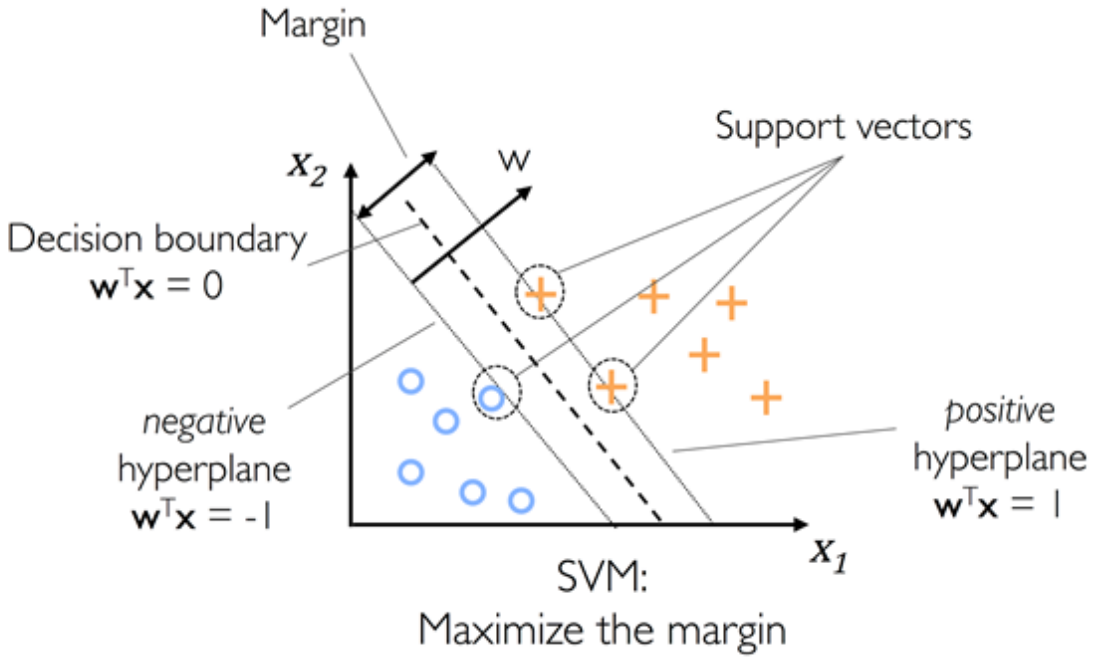
$$u = \frac{v}{\|v\|} = \begin{bmatrix} \dfrac{1}{\sqrt{14}} \\ \dfrac{2}{\sqrt{14}} \\ \dfrac{3}{\sqrt{14}} \end{bmatrix}$$



개요

- 결정함수 : n차원의 초평면($hyperplane$)
- 결정경계 : 결정 함수의 값이 0인 점. $(n-1)$차원의 초평면($hyperplane$)
- SVM의 최적화는 마진(Margin)을 최대화 하는 것(large margin classification)
  - 마진은 클래스를 구분하는 초평면(결정경계)과 이 초평면에 가장 가까운 훈련 샘플 사이의 거리. 이런 샘플을 서포트 벡터(support vector)라고 한다.
  - 클래스 사이에 가능한 한 가장 넓은 '도로'를 내는 것
  - 클래스를 구분하는 결정경계와 샘플 사이의 마진을 가능한 한 가장 크게 하는 것





https://github.com/rasbt/python-machine-learning-book-2nd-edition/commit/ba17d077c78dafb362971ca8b8f8cd92ee81b40f?short_path=34ed1bf#diff-34ed1bfbf22327fba9b6df3c88b2c539

SVM 이론

- n차원 공간에 있는 입력 데이터 $x = [x_1 x_2 \cdots x_n]^T$
- $h(x) = w_1 x_1 + w_2 x_2 \cdots w_n x_n + b = w^T x + b = 0$
- 경계와 가장 가까운 support vector

$$w^T x_{pos} + b = 1 \quad \cdots \quad (1)$$
$$w^T x_{neg} + b = -1 \quad \cdots \quad (2)$$
$$w^T (x_{pos} - x_{neg}) = 2 \quad \cdots \quad (1) - (2)$$

$$\frac{w^T (x_{pos} - x_{neg})}{\|w\|} = \frac{2}{\|w\|}$$

$$\|w\| = \sqrt{\sum_{j=1}^{m} w_j^2}$$

$$r_{pos} = \frac{w^T x_{pos} + b}{\|w\|} = \frac{1}{\|w\|}$$

$$r_{neg} = \frac{w^T x_{neg} + b}{\|w\|} = -\frac{1}{\|w\|}$$

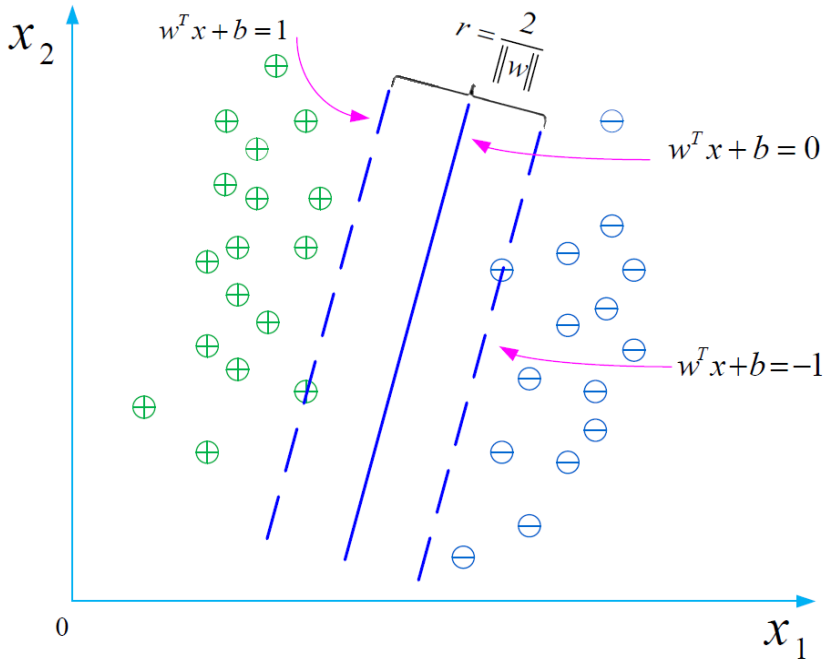
- Goal : maximize the margin

$$\therefore max \frac{2}{\|w\|}$$

⇒ 콰드라틱$(quadratic)$ 프로그래밍 방법

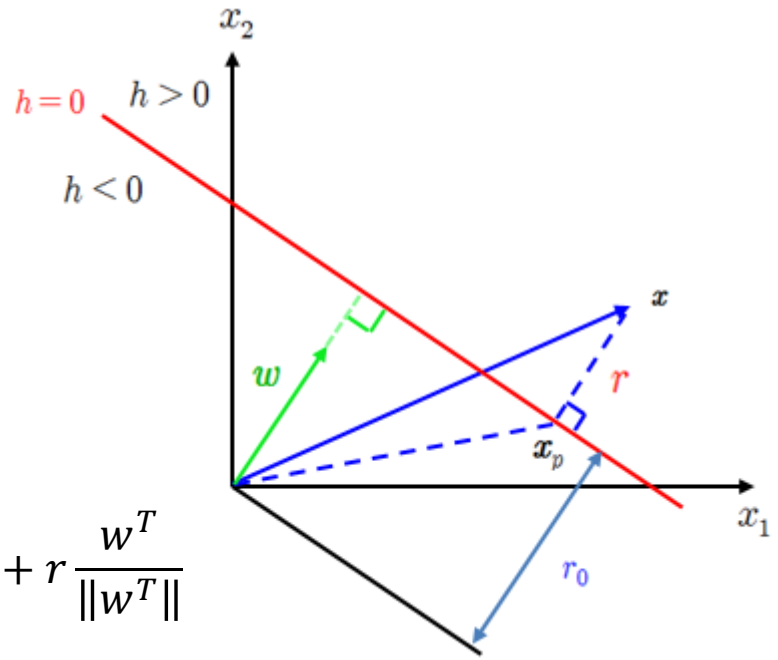
$min \frac{1}{2} \|w\|^2$

를 최소화(미분가능)



2차원 평면상에서 입력 벡터 x의 초평면 h와의 거리

- 벡터 $w = [w_1 w_2 \cdots w_n]^T$가 평면의 법선 벡터
- x_p벡터 : 벡터 x를 초평면 h에 투영 $(projection)$한 벡터
- r : 초평면에서 x까지의 거리



x를 단위 법선 벡터 $\frac{w}{\|w\|}$를 이용하여 r을 구하면

$$x = x_p + r\frac{w}{\|w\|}, \qquad column\ vector : x^T = x_p^T + r\frac{w^T}{\|w^T\|}$$

$$w^T x = w^T x_p + r\frac{w^T w}{\|w\|}, \qquad (w^T w = \|w\|^2)$$

$$w^T x = w^T x_p + r\|w\|$$

$$w^T x + b = w^T x_p + b + r\|w\|, \qquad (x_p\text{는 } h(x) = 0 \text{ 상의 벡터}, h(x) = w^T x_p + b = 0)$$

$$h(x) = r\,\|w\|$$

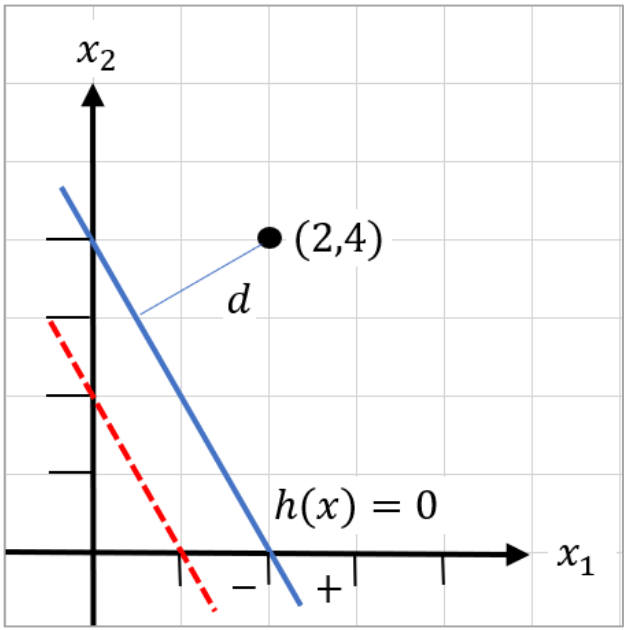
$$\boldsymbol{r = \frac{h(x)}{\|w\|}}$$

$$r_0 = \frac{h(0)}{\|w\|} = \frac{b}{\|w\|} \qquad (x = 0\ then\ b)$$

ref : 인공지능 - 튜링 테스트에서 딥러닝까지 (생능출판사)

결정초평면의 수학적 계산

$w = (2,1), \quad b = 4, \quad h(x) = 2x_1 + x_2 - 4 = 0$



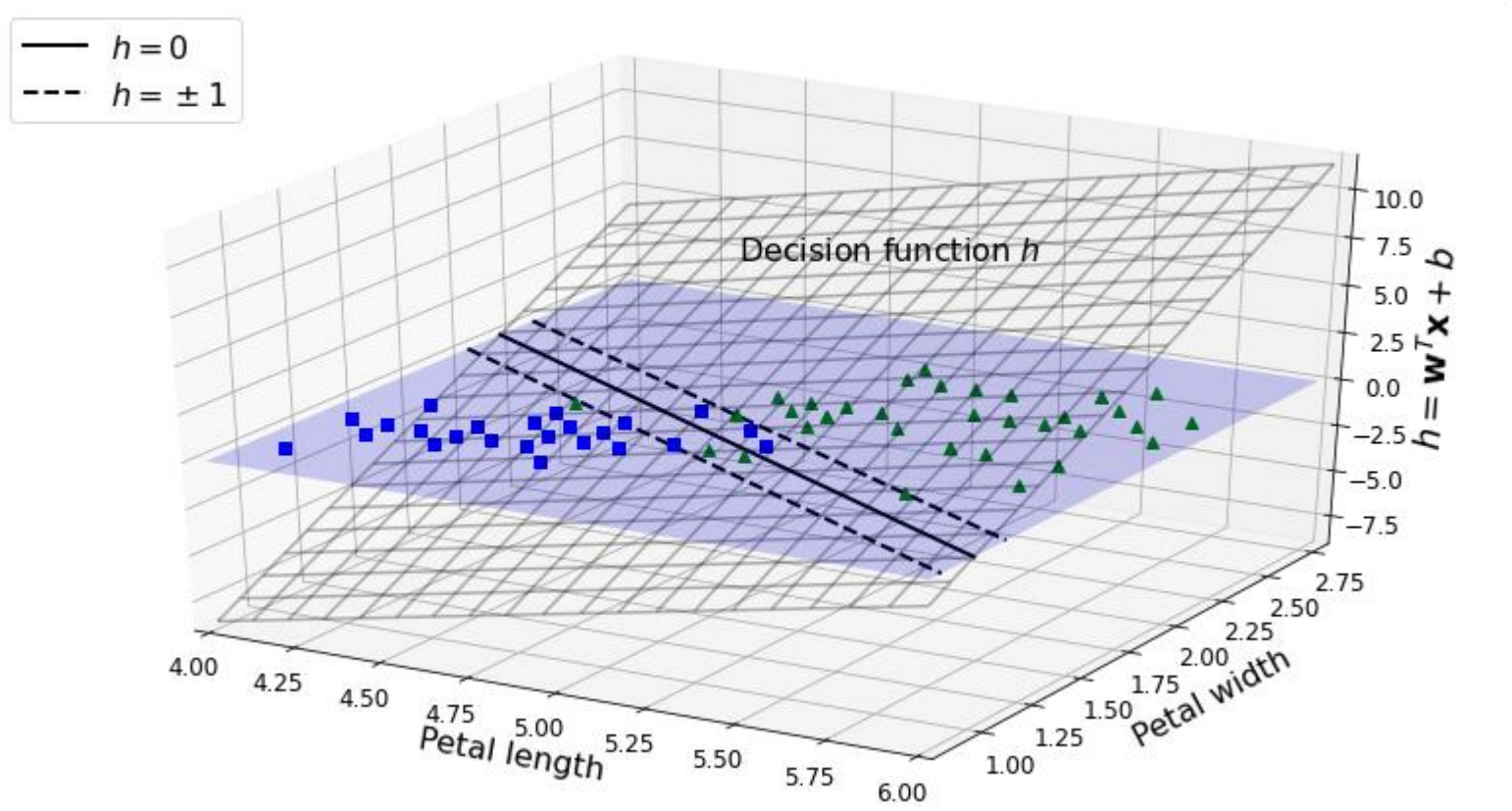
$(ex)h\big((0,0)\big) = -4 < 0$

- $h(x)$에 2를 곱하면 $h(x) = 4x_1 + 2x_2 - 8 = 0$ (파란실선)
- b를 -4에서 -2로 바꾸면 $h(x) = 2x_1 + x_2 - 2$ (빨간점선)
- 즉, w를 그대로 둔 채 b를 바꾸면 결정 직선은 방향을 유지한 채 위치만 바뀐다
- 점$(2,4)$에서 결정경계까지 거리

$$d = \frac{h(x)}{\|w\|} = \frac{2 * 2 + 4 * 1 - 4}{\sqrt{2^2 + 1}} = \frac{4}{\sqrt{5}}$$

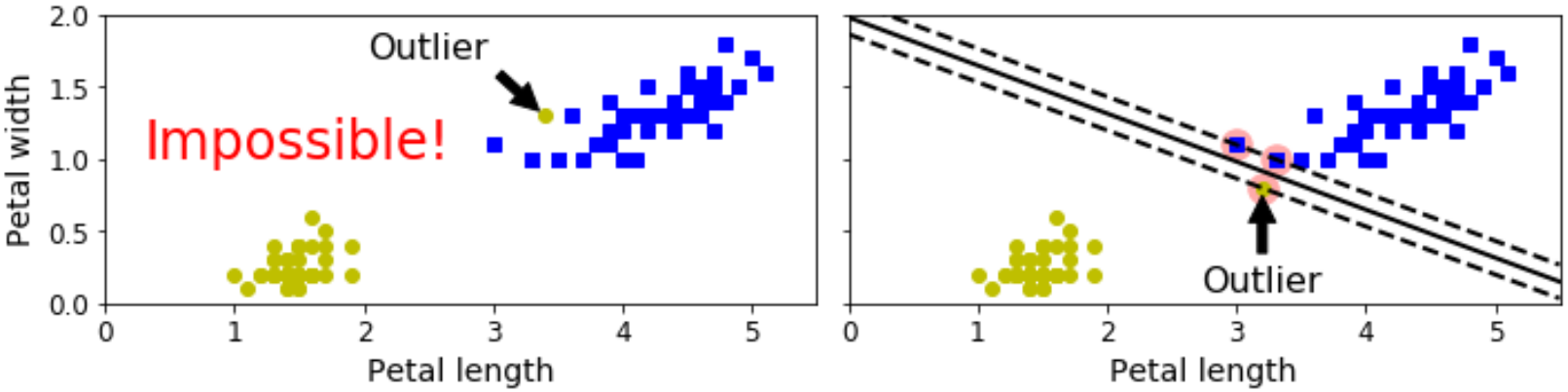
iris 데이터셋의 결정함수

- SVM의 최적화는 가능한 한 마진을 크게 하는 $w^T x + b$의 w와 b를 찾는 것
- 점선 : 결정함수의 값이 1 또는 -1인 점

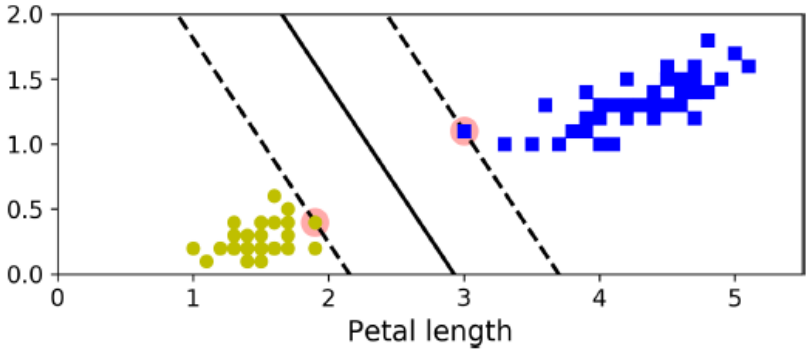


하드 마진 분류(hard margin classification)

- 모든 샘플이 경계선 바깥쪽에 올바르게 분류되어 있는 상태
- 데이터가 선형적으로 구분될 수 있어야 제대로 작동하며 이상치에 민감

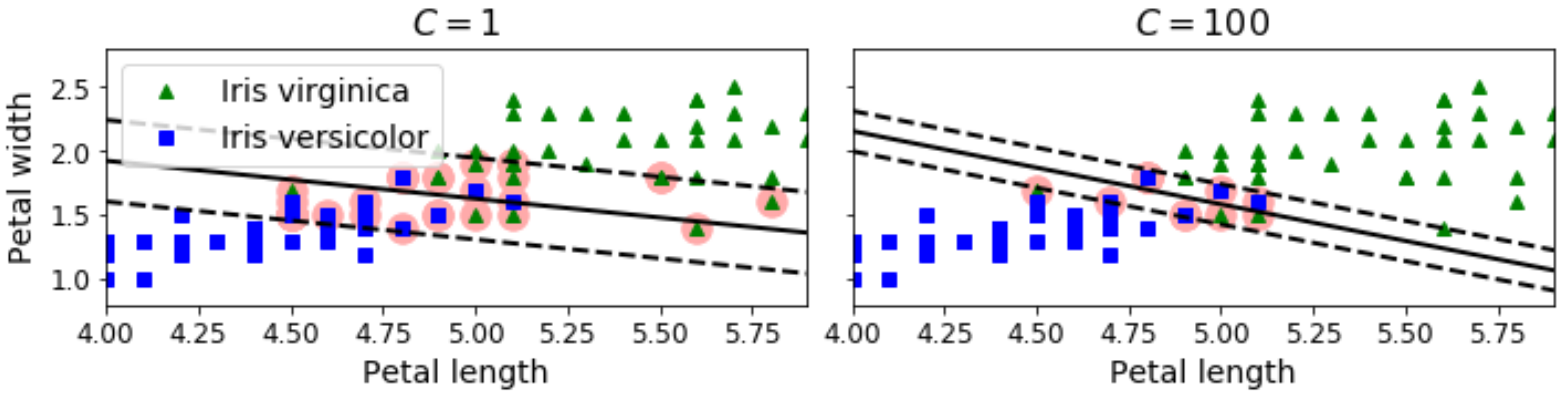


- 왼쪽그래프 : 하드마진을 찾을 수 없다.
- 오른쪽 그래프 : 결정경계는 이상치가 없던 경우(하단 그림)와 매우 다르게 바뀐다. 또한 일반화가 잘 될 것 같지 않다



소프트 마진 분류(soft margin classification)

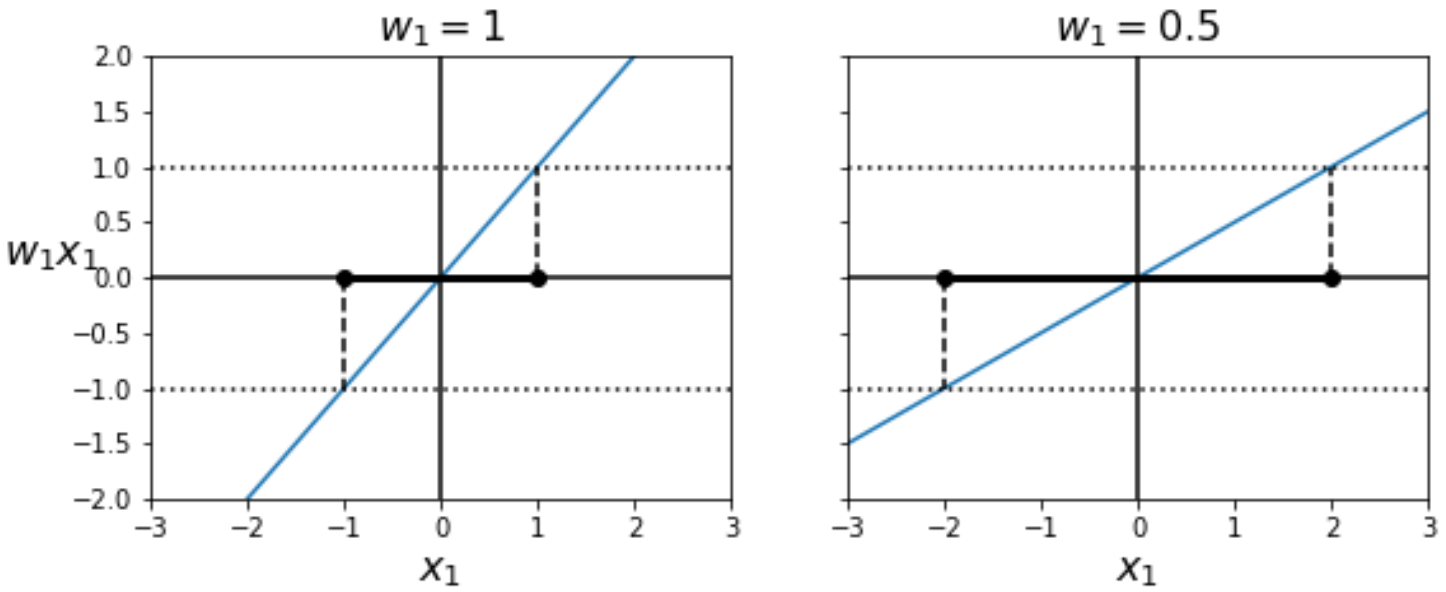
- 마진오류(margin violation): 샘플이 경계선 중간이나 반대쪽에 있는 경우
- 경계의 폭을 가능한 넓게 유지하는 것
- 마진 오류 사이에 적절한 균형을 잡는 것



- 사이킷런(sklearn)의 SVM모델에서 여러 하이퍼 파라미터 지정(C는 여러 하이퍼 파라미터 중 하나)
  - 왼쪽 그래프 : 낮게 설정
  - 오른쪽 그래프 : 높게 설정
- 마진 오류는 일반적으로 적은 것이 좋다.
  - 하지만 위의 이 경우에는 왼쪽 모델에 마진 오류가 많지만 일반화가 더 잘 될 것 같다

목적함수

- 결정함수의 기울기 : 가중치 벡터 노름 $\|w\|$
- 기울기를 2로 나누는 것은 마진에 2를 곱하는 것과 같다
- 가중치 벡터 w가 작을 수록 마진은 커진다.



목적함수

- 마진을 크게 하기 위해 $\|w\|$를 최소화 : $w^T w = \|w\|^2$
- 음성 샘플일 때 $t^{(i)} = -1$, 양성 샘플일 때 $t^{(i)} = 1$로 정의하면 $\boldsymbol{t^{(i)}\left(w^T x^{(i)} + b\right) \geq 1}$ 로 표현할 수 있다.

- 하드마진 선형 SVM 분류기의 목적함수

$$\underset{W,b}{\boldsymbol{minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2$$

[조건] $i = 1,2,\cdots,m$일 때 $t^{(i)}\left(w^T x^{(i)} + b\right) \geq 1$

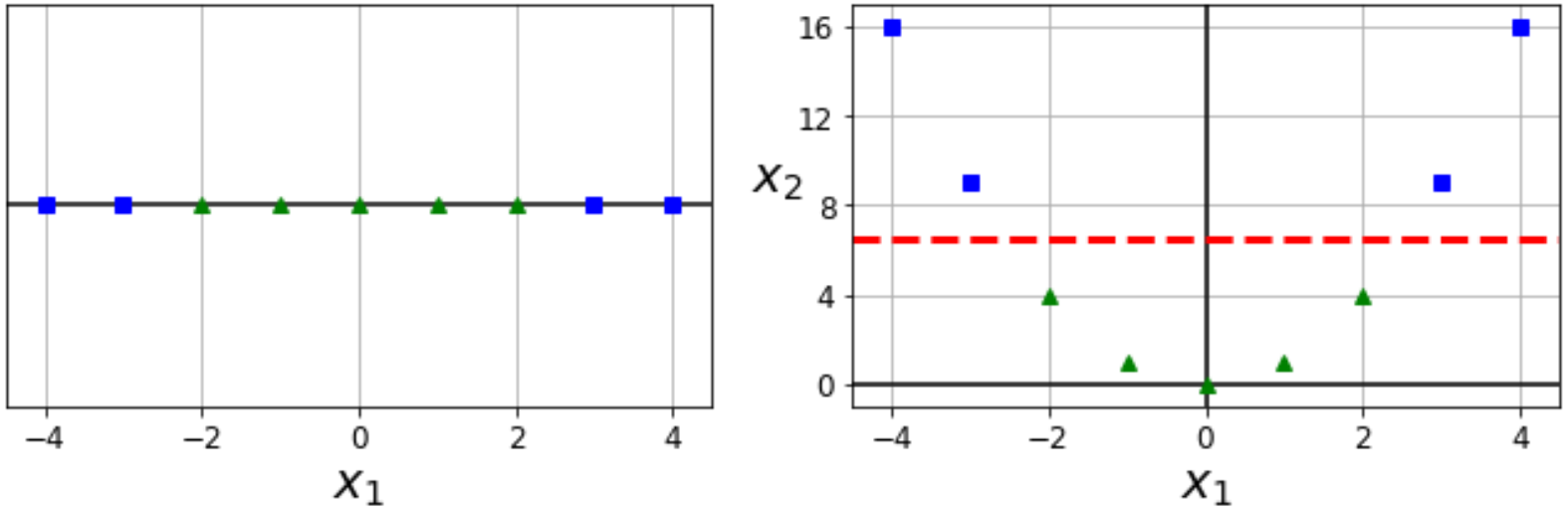
- 소프트마진 선형 SVM 분류기의 목적함수
  - 슬랙변수(slack variable) $\zeta^{(i)} \geq 0$을 도입. $\zeta^{(i)}$는 i번째 샘플이 얼마나 마진을 위반할지 결정
  - 하이퍼파라미터 C는 아래 두 목표 사이의 트레이드 오프를 정의
    - 슬랙변수의 값이 작으면 마진의 오류를 최소화
    - $\frac{1}{2}\|\boldsymbol{w}\|^2$가 작으면 마진을 크게 한다

$$\underset{W,b,\zeta}{\boldsymbol{minimize}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + \boldsymbol{C}\sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \boldsymbol{\zeta^{(i)}}$$

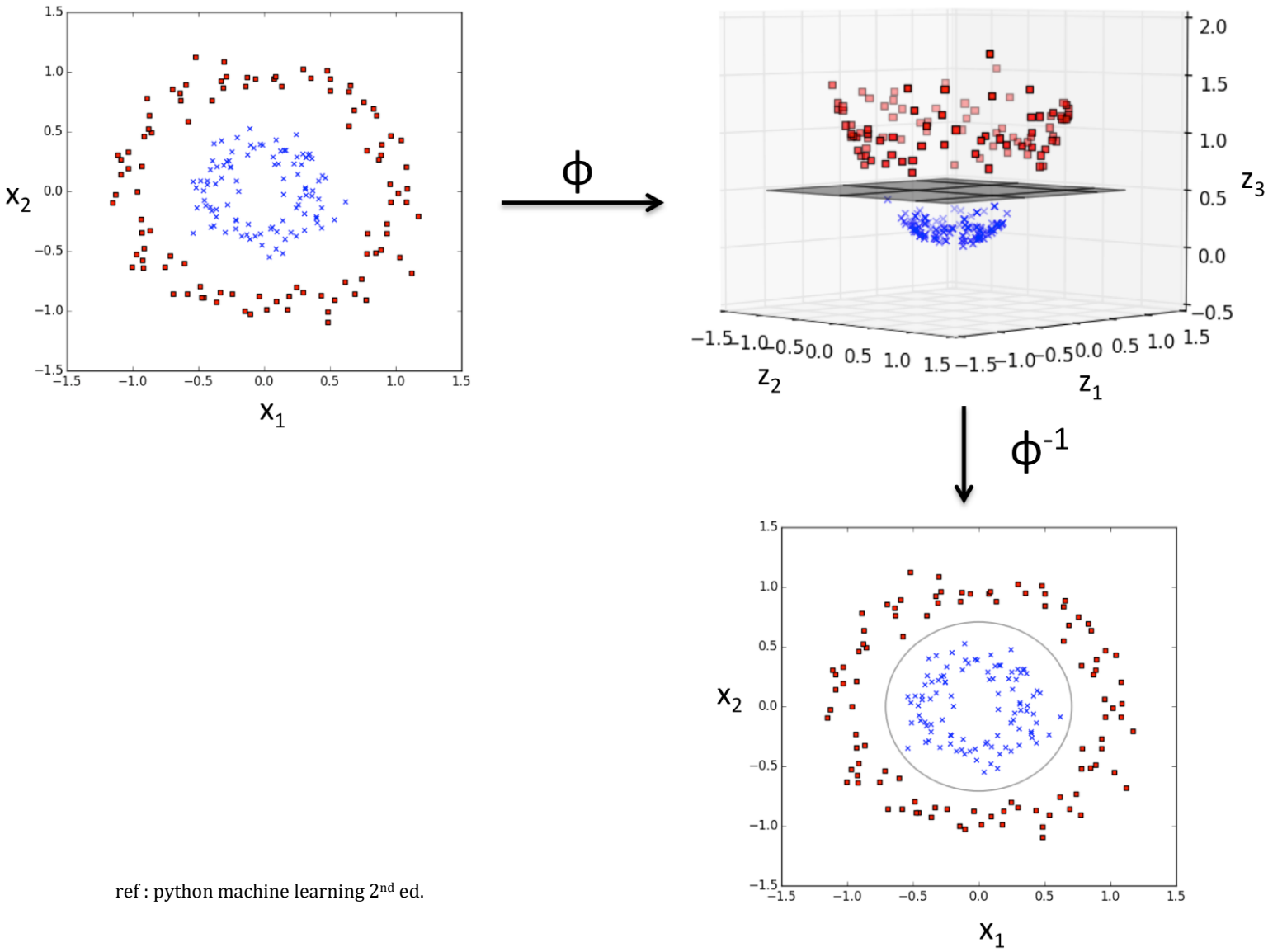
[조건] $i = 1,2,\cdots,m$일 때 $t^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 - \zeta^{(i)}$이고 $\zeta^{(i)} \geq 0$

비선형 SVM분류

- $x_2 = (x_1)^2$로 하여 만들어진 2차원 데이터셋은 선형적으로 구분 가능(오른쪽 그래프)



고차원 공간에서 찾은 결정 경계의 예



ref : python machine learning 2nd ed.

PolynomialFeatures(degree=d)

- 다항식 특성
  - 낮은 차수의 다항식 : 매우 복잡한 데이터셋을 잘 표현하지 못함
  - 높은 차수의 다항식 : 매우 많은 특성을 추가하므로 모델이 느림
- 특성이 n개인 배열을 특성이 $\frac{(n+d)!}{d!n!}$ 개인 배열로 변환
  - $n!$ 로서 특성수가 교차항을 포함해 엄청나게 늘어날 수 있다.
- (예) 두 개의 특성 a, b가 있을 때 degree=3으로 적용하면
  - $a^2, a^3, b^2b^3, ab, a^2b, ab^2$

```
from sklearn.datasets import make_moons
from sklearn.preprocessing import PolynomialFeatures
X, y = make_moons(n_samples=100, noise=0.15, random_state=42)

poly =PolynomialFeatures(degree=3)
poly.fit(X, y)
# 만들어진 특성의 차수 확인
poly.get_feature_names()
```

['1', 'x0', 'x1', 'x0^2', 'x0 x1', 'x1^2', 'x0^3', 'x0^2 x1', 'x0 x1^2', 'x1^3']

```
# interaction_only=True지정 : 거듭제곱이 포함된 항은 제외
poly =PolynomialFeatures(degree=3, interaction_only=True)
poly.fit(X, y)
poly.get_feature_names()
```

['1', 'x0', 'x1', 'x0 x1'

다항 특성을 사용한 선형 SVM분류(1)

- moons 데이터 셋(두 개의 반달 모양 데이터 셋) 적용

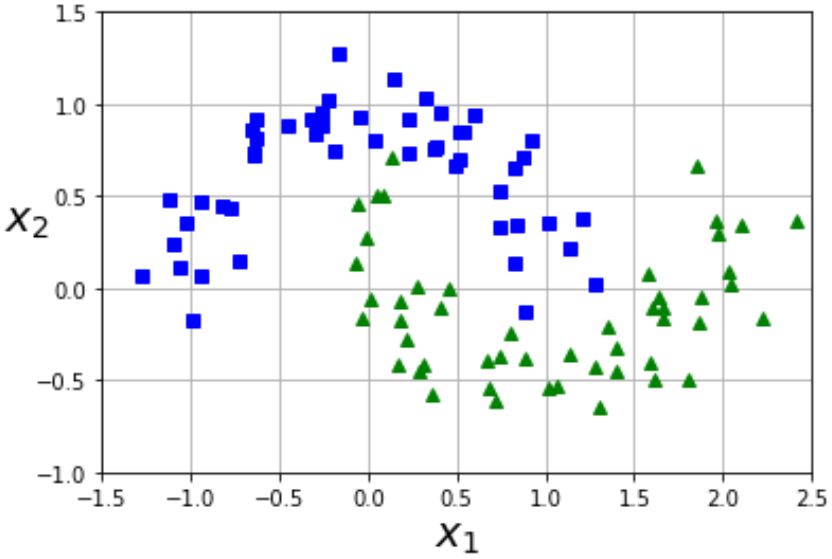
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

X, y = make_moons(n_samples=100, noise=0.15, random_state=42)

def plot_dataset(X, y, axes):
    plt.plot(X[:, 0][y==0], X[:, 1][y==0], "bs")
    plt.plot(X[:, 0][y==1], X[:, 1][y==1], "g^")
    plt.axis(axes)
    plt.grid(True, which='both')
    plt.xlabel(r"$x_1$", fontsize=20)
    plt.ylabel(r"$x_2$", fontsize=20, rotation=0)

plot_dataset(X, y, [-1.5, 2.5, -1, 1.5])
plt.show()
```

다항 특성을 사용한 선형 SVM분류(2)

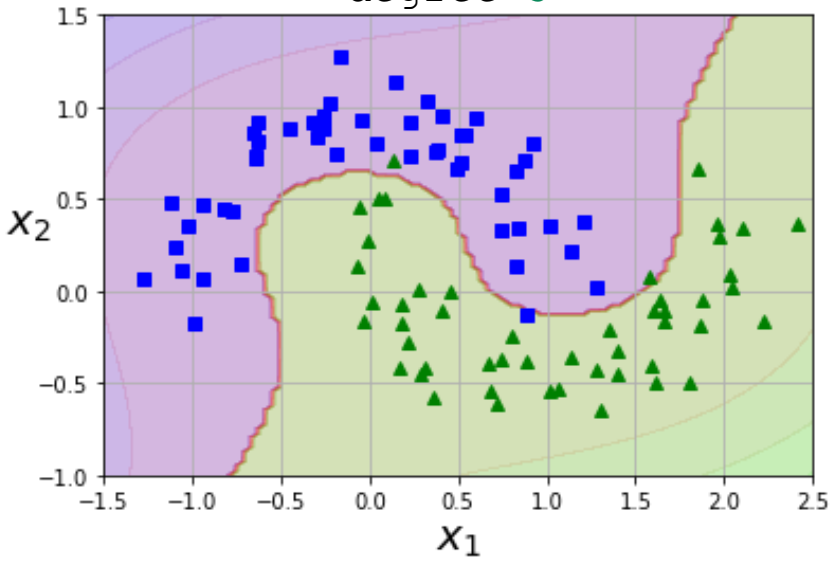

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

polynomial_svm_clf = Pipeline([
        ("poly_features", PolynomialFeatures(degree=3)),
        ("scaler", StandardScaler()),
        ("svm_clf", LinearSVC(C=10, loss="hinge", random_state=42))
    ])

polynomial_svm_clf.fit(X, y)
```

```
def plot_predictions(clf, axes):
    x0s = np.linspace(axes[0], axes[1], 100)
    x1s = np.linspace(axes[2], axes[3], 100)
    x0, x1 = np.meshgrid(x0s, x1s)
    X = np.c_[x0.ravel(), x1.ravel()]
    y_pred = clf.predict(X).reshape(x0.shape)
    y_decision = clf.decision_function(X).reshape(x0.shape)
    plt.contourf(x0, x1, y_pred, cmap=plt.cm.brg, alpha=0.2)
    plt.contourf(x0, x1, y_decision, cmap=plt.cm.brg, alpha=0.1)
plot_predictions(polynomial_svm_clf, [-1.5, 2.5, -1, 1.5])
plot_dataset(X, y, [-1.5, 2.5, -1, 1.5])

plt.show()
```

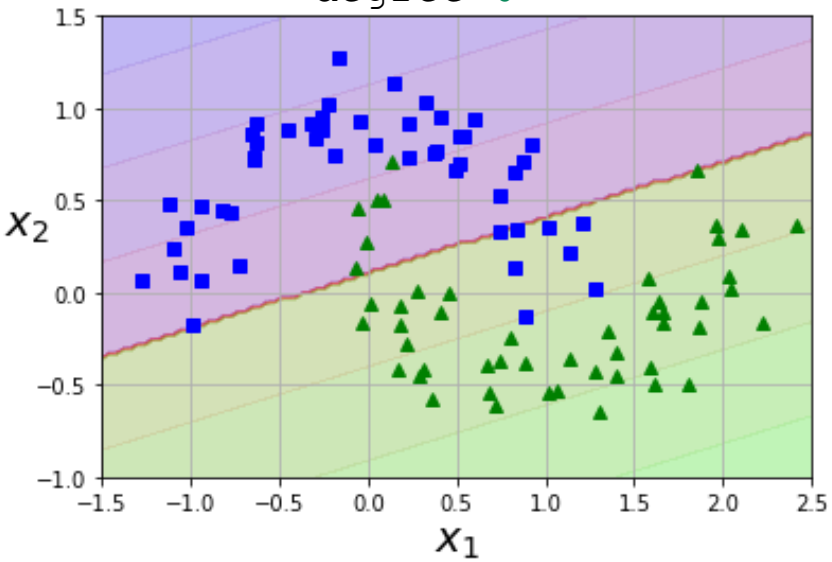
다항 특성을 사용한 선형 SVM분류(3)



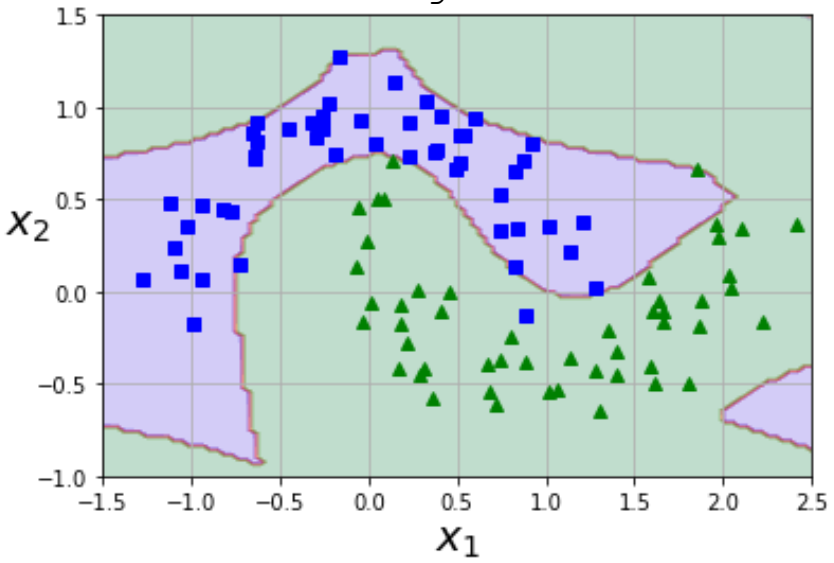
degree=3



degree=0



degree=300



SVM분류

커널트릭

쌍대문제 - 라그랑주 승수

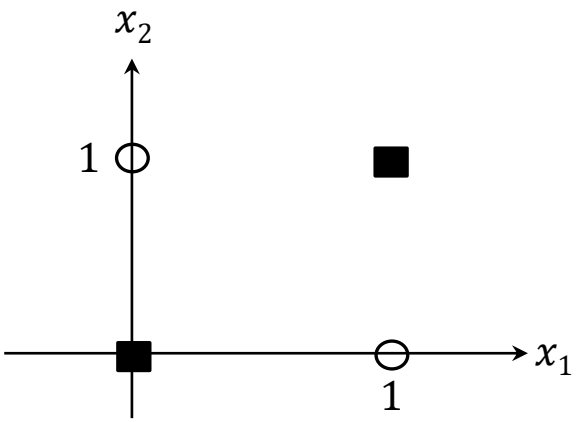
커널(kernel trick)

- 고차원 공간으로 매핑
  - SVM으로 비선형을 해결하기 위하여 **매핑함수 ϕ**를 사용하여 훈련 데이터를 고차원 특성 공간으로 변환
  - 새로운 특성을 만드는 계산 비용이 매우 비싸다(특히 고차원 데이터일 때)
  - ϕ를 고차원으로 변환하여 계산하지 않고 원래 데이터에서 계산. 따라서 높은 비용을 절감하기 위해 커널함수를 정의
- 커널(kernel)
  - 샘플간의 유사도 함수(similarity function)
  - 0과 1사이의 범위
- 커널 트릭(kernel trick)
  - 원래 공간에서 더 높은 차원의 새로운 공간으로 매핑
  - 실제로 특성을 추가하지 않으면서 다항식 특성을 많이 추가한 것과 같은 결과를 얻을 수 있다

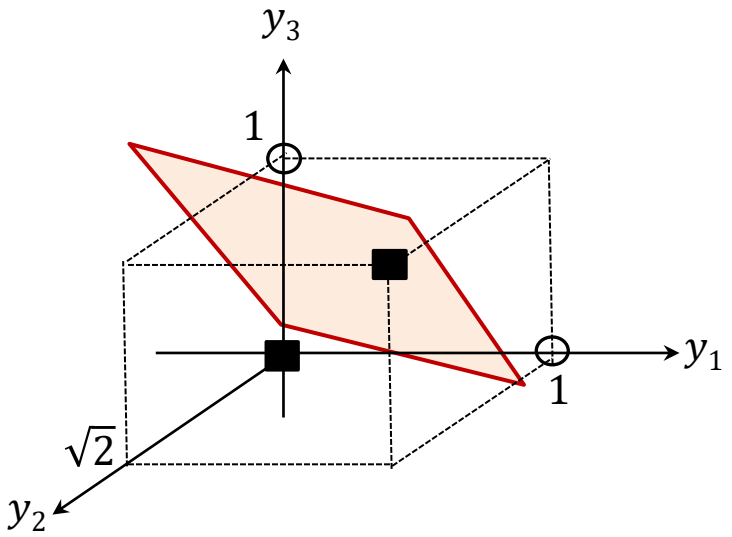
커널기법(kernel trick)

(예) XOR의 2차원 입력(x_1, x_2)를 **매핑함수 $\phi(x_1, x_2)$** 를 이용하여 3차원으로 변환

$$\phi(x_1, x_2) = \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right) = (y_1, y_2, y_3)$$



(a)원래공간 x



(b)변환 공간 $\phi(x)$

커널함수(kernel function)

- 어떤 특징공간에 정의된 두 특징벡터$x^{(i)}, x^{(i)}$에 대해$K\left(x^{(i)}, x^{(j)}\right) = \phi\left(x^{(i)}\right) \cdot \phi\left(x^{(j)}\right)$인 변환함수 ϕ 가 존재하면 $K\left(x^{(i)}, x^{(j)}\right)$를 커널함수라 부른다
- **선형 커널**

  $K\left(x^{(i)}, x^{(j)}\right) = (x^{(i)})^T \cdot x^{(j)}$
- **다항식 커널(polynomial kernel)**

  $K\left(x^{(i)}, x^{(j)}\right) = \left(\Upsilon (x^{(i)})^T \cdot x^{(j)} + r\right)^d, \qquad d: \text{integer}(+)$
- **방사기저 함수(radial basis function, RBF), 가우시안 커널(Gaussian kernel)**

  $$K\left(x^{(i)}, x^{(j)}\right) = exp\left(-\frac{\left\|x^{(i)} - x^{(j)}\right\|^2}{2\sigma^2}\right)$$

  $$K\left(x^{(i)}, x^{(j)}\right) = exp\left(-\Upsilon\left\|x^{(i)} - x^{(j)}\right\|^2\right), \qquad \gamma = \frac{1}{2\sigma^2}$$
- **쌍곡 탄젠트 커널(hyperbolic tangent), 시그모이드(sigmoid)**

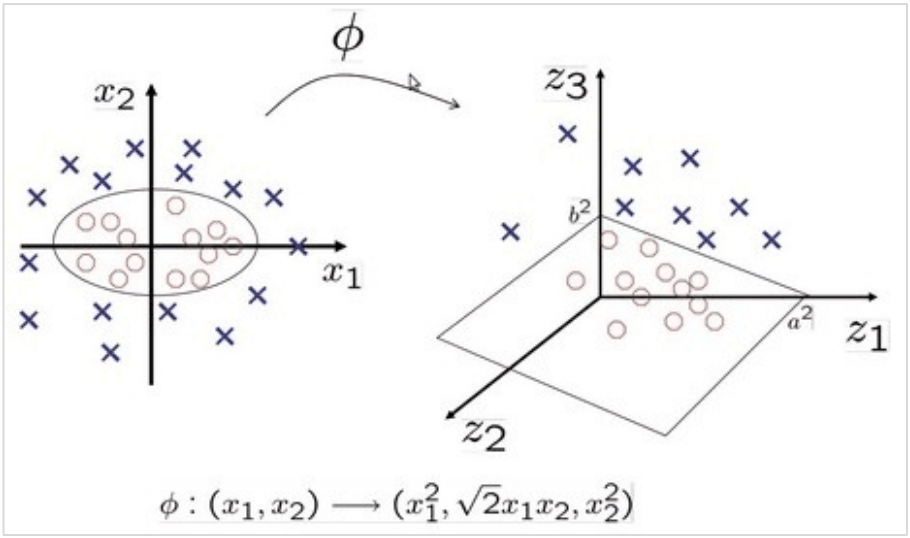
  $K\left(x^{(i)}, x^{(j)}\right) = tanh\left(\Upsilon (x^{(i)})^T \cdot x^{(j)} + r\right)$

- 사이킷런의 SVC, SVR에서 매개변수 kernel에 지정할 수 있는 함수
  - "linear", "poly", "rbf", "sigmoid"

2차원 다항식 커널

- 2차 다항식 **매핑 함수 ϕ**

$$\phi(x) = \phi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix}$$



- 2차 다항식 매핑을 위한 커널 트릭

$$\phi(a)^T\phi(b) = \begin{pmatrix} a_1^2 \\ \sqrt{2}a_1a_2 \\ a_2^2 \end{pmatrix}^T \begin{pmatrix} b_1^2 \\ \sqrt{2}b_1b_2 \\ b_2^2 \end{pmatrix} = a_1^2b_1^2 + 2a_1b_1a_2b_2 + a_2^2b_2^2$$

$$= (a_1b_1 + a_2b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^T \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 = (a^Tb)^2$$

$$\therefore\ \boldsymbol{\phi(a)^T\phi(b) = \left(a^Tb\right)^2}$$

비선형 SVM분류 - 다항식 커널(1)


```
# 3차 다항식 커널
from sklearn.svm import SVC
poly_kernel_svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("svm_clf", SVC(kernel="poly", degree=3, coef0=1, C=5))
    ])
poly_kernel_svm_clf.fit(X, y)
```

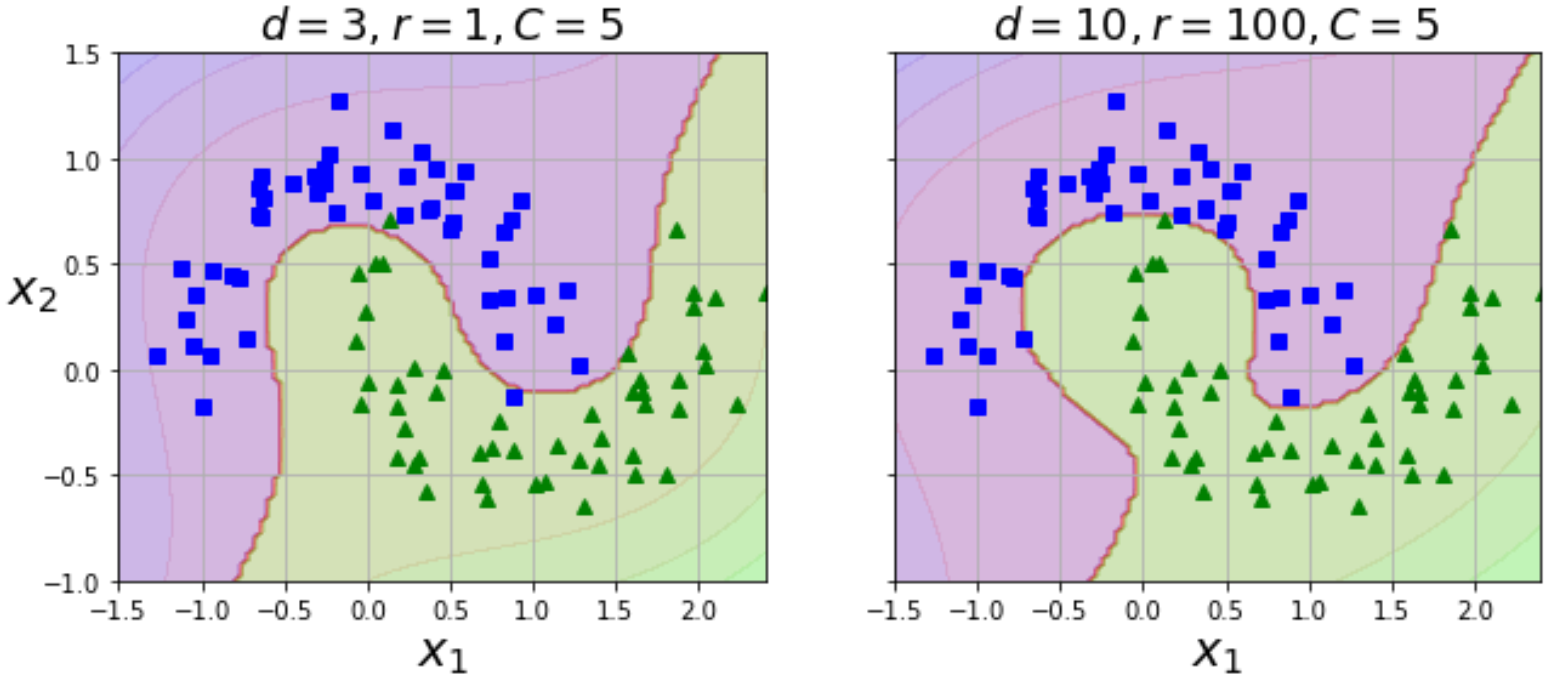
```
# 10차 다항식 커널
poly100_kernel_svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("svm_clf", SVC(kernel="poly", degree=10, coef0=100, C=5))
    ])
poly100_kernel_svm_clf.fit(X, y)
```

비선형 SVM분류 - 다항식 커널 (2)


```
fig, axes = plt.subplots(ncols=2, figsize=(10.5, 4), sharey=True)

plt.sca(axes[0])
plot_predictions(poly_kernel_svm_clf, [-1.5, 2.45, -1, 1.5])
plot_dataset(X, y, [-1.5, 2.4, -1, 1.5])
plt.title(r"$d=3, r=1, C=5$", fontsize=18)

plt.sca(axes[1])
plot_predictions(poly100_kernel_svm_clf, [-1.5, 2.45, -1, 1.5])
plot_dataset(X, y, [-1.5, 2.4, -1, 1.5])
plt.title(r"$d=10, r=100, C=5$", fontsize=18)
plt.ylabel("")

plt.show()
```

비선형 SVM분류 - 다항식 커널 (3)

- 과대적합(overfitting) : 다항식의 차수를 줄인다
- 과소적합(underfitting) : 다항식의 차수를 늘린다
- coef0 : 모델이 높은 차수와 낮은 차수에 얼마나 영향을 받을 지 조절
  - 차수가 높아질수록 1보다 작은 값과 1보다 큰 값의 차이가 크게 벌어지므로 coef0을 적절한 값으로 지정하면 고차항의 영향을 줄일 수 있다.
  - 다항식 커널에 있는 상수항 r (기본값은 0)



비선형 SVM분류 - RBF(1)

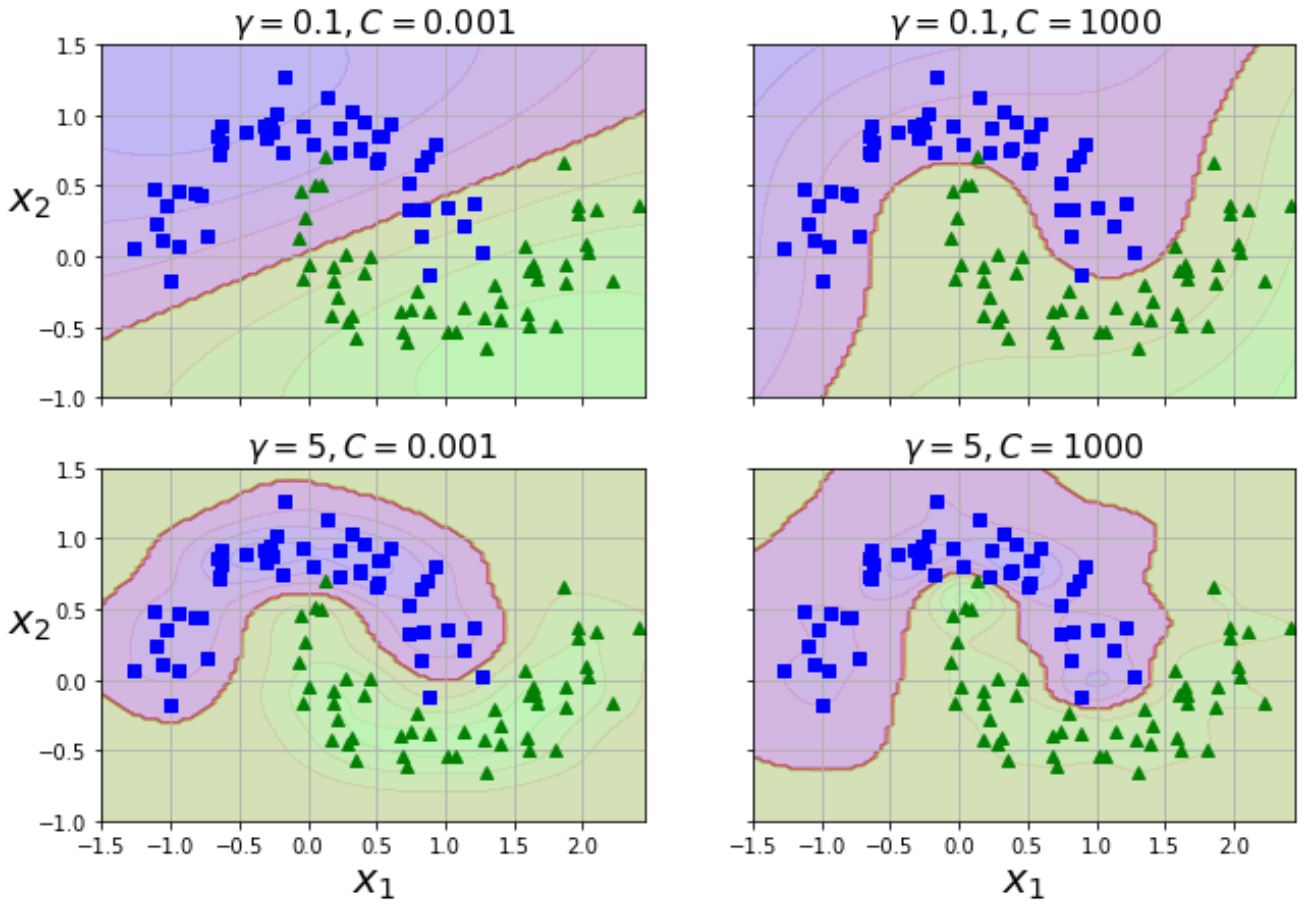

```
rbf_kernel_svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("svm_clf", SVC(kernel="rbf", gamma=5, C=0.001))
    ])
rbf_kernel_svm_clf.fit(X, y)
```

```
from sklearn.svm import SVC

gamma1, gamma2 = 0.1, 5
C1, C2 = 0.001, 1000
hyperparams = (gamma1, C1), (gamma1, C2), (gamma2, C1), (gamma2, C2)

svm_clfs = []
for gamma, C in hyperparams:
    rbf_kernel_svm_clf = Pipeline([
            ("scaler", StandardScaler()),
            ("svm_clf", SVC(kernel="rbf", gamma=gamma, C=C))
        ])
    rbf_kernel_svm_clf.fit(X, y)
    svm_clfs.append(rbf_kernel_svm_clf)

fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(10.5, 7), sharex=T
rue, sharey=True)
```

비선형 SVM분류 - RBF (2)


```
for i, svm_clf in enumerate(svm_clfs):
    plt.sca(axes[i // 2, i % 2])
    plot_predictions(svm_clf, [-1.5, 2.45, -1, 1.5])
    plot_dataset(X, y, [-1.5, 2.45, -1, 1.5])
    gamma, C = hyperparams[i]
    plt.title(r"$\gamma = {}, C = {}$".format(gamma, C), fontsize=16)
    if i in (0, 1):
        plt.xlabel("")
    if i in (1, 3):
        plt.ylabel("")
plt.show()
```

비선형 SVM분류 - RBF(3)



- gammer가 규제역할 : 과대적합일 경우 감소, 과소적합일 경우 증가
- γ 를 증가시키면 결정경계가 불규칙해지고 각 샘플을 따라 구불구불하게 휘어진다
- 작은 γ 값은 샘플이 넓은 범위에 걸쳐 영향을 주므로 결정경계가 더 부드러워진다
- 모델의 복잡도를 조절하려면 γ 와 C 를 함께 조정하는 것이 좋다

SVM분류
커널트릭
쌍대문제 - 라그랑주 승수

라그랑주 승수법(Lagrange multiplier method)

- 제한조건이 있는 함수의 최적화$(constrained\ optimization)$ 문제
- 제한조건은 연립방정식 또는 연립부등식
  - 등식 제약조건$(equality\ constraint)$
  - 부등식 제약조건$(inequality\ constraint)$:$KKT(Karush-Kuhn-Tucker)$조건 추가
- 목적함수$\mathcal{L}(x,\alpha)$ 를 원래의 목적함수$f(x)$를 사용하지 않고 제한조건 등식 $g(x)$에 새로운 $\boldsymbol{\alpha}$ **라그랑주 승수**를 곱하여 계산한 함수

$$\mathcal{L}(x,\alpha) = h(x_1, x_2, \cdots, x_n, \alpha_1\alpha_2, \cdots, \alpha_m) = f(x) - \sum_{j=1}^{m} \alpha_j g_j(x)$$

- 목적함수$\mathcal{L}(x,\alpha)$ 최적화 : 모든 편도함수가 0값을 찾는다.

$$\frac{\delta\mathcal{L}}{\delta x_1} = \frac{\delta f}{\delta x_1} - \sum_{j=1}^{m} \alpha_j \frac{\delta g_j}{\delta x_1} = 0, \quad \cdots \quad , \frac{\delta\mathcal{L}}{\delta x_n} = \frac{\delta f}{\delta x_n} - \sum_{j=1}^{m} \alpha_j \frac{\delta g_j}{\delta x_n} = 0$$

$$\frac{\delta\mathcal{L}}{\delta \alpha_1} = g_1 = 0, \quad \cdots \quad , \frac{\delta\mathcal{L}}{\delta \alpha_m} = g_m = 0$$

$\therefore\ n+m$개의 연립방정식을 풀면 $\boldsymbol{x_1, x_2, \cdots, x_n, \alpha_1\alpha_2, \cdots, \alpha_m}$을 구할 수 있다.

- $f(x,y)$가 제약이 있는 최적화 문제라면$\mathcal{L}(x,y,\alpha)$의 라그랑주 승수 α(0이 아닌값)가 존재해야 한다.
  - $\alpha = 0$이면 원래의 문제와 제한조건이 있는 문제의 최적화 조건이 같다.

등식제약 예제

- 제약이 있는 최적화 문제에서 목적함수로 제약을 옮김으로써 제약이 없는 문제로 변환하는 것
- 등식제약$(equality\ constraint) 3x + 2y + 1 = 0$ 조건하에서
  $f(x, y) = x^2 + 2y$를 최소화하는 x, y값은?

- 라그랑주 함수로 정의
  $\mathcal{L}(x, y, \alpha) = f(x, y) - \alpha(3x + 2y + 1)$
  $= \boldsymbol{x^2 + 2y - \alpha(3x + 2y + 1)}$
- 편도함수
  $\frac{\delta}{\delta x}\mathcal{L}(x, y, \alpha) = 2x - 3\alpha$
  $\frac{\delta}{\delta y}\mathcal{L}(x, y, \alpha) = 2 - 2\alpha$
  $\frac{\delta}{\delta \alpha}\mathcal{L}(x, y, \alpha) = -3x - 2y - 1$

- 해$(solution)$
  $2x - 3\alpha = 2 - 2\alpha = -3x - 2y - 1 = 0$
  $x = \frac{3}{2}, \quad y = -\frac{11}{4}, \quad \alpha = 1$

부등식 제약

- 부등식 제약(inequality constraint)

  $g(x) = 3x + 2y + 1 \geq 0 \ (j = 1,2,\cdots,m)$

  목적함수 $\mathcal{L}(x,\alpha) = f(x) - \sum_{j=1}^{m} \alpha_j g_i(x)$

- $KKT(Karush - Kuhn - Tucker)\ Condition$
  - 모든 독립 변수 $x_1, x_2, \cdots, x_n$ 에 대한 미분은 0이다

    $$\frac{\delta \mathcal{L}(x,\alpha)}{\delta x_i} = 0$$

  - 모든 라그랑주 승수 $\alpha_1, \alpha_2, \cdots, \alpha_m$ 과 제한조건 부등식(α 에 대한 미분)의 곱은 0이다

    $$\alpha_j \cdot \frac{\delta \mathcal{L}(x,\alpha)}{\delta \alpha_i} = \alpha_j \cdot g_i = 0$$

  - 라그랑주 승수는 음수가 아니어야 한다

    $$\alpha_j \geq 0$$

하드 마진 문제를 위한 일반화된 라그랑주 함수

- 제약함수 : $t^{(i)}\left(w^T x^{(i)} + b\right) \geq 1$ (음성 샘플 : $t^{(i)} = -1$, 양성 샘플 : $t^{(i)} = 1$로 정의)
- 목적함수 : $\underset{W,b}{minimize}\ \frac{1}{2}\|w\|^2$, $(i = 1,2,\cdots,m$일때 $t^{(i)}\left(w^T x^{(i)} + b\right) \geq 1)$
- 라그랑주 함수 정의 : **[식 1]**

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha^i \left(t^i \left(w^T x^i + b\right) - 1\right)$$

- 편도함수

$$\frac{\delta}{\delta \boldsymbol{w}} \mathcal{L}(w, b, \alpha) = w - \sum_{i=1}^{m} \alpha^i t^i x^i$$

$$\frac{\delta}{\delta \boldsymbol{b}} \mathcal{L}(w, b, \alpha) = -\sum_{i=1}^{m} \alpha^i t^i$$

- 해(solution) : 임계점 (편도함수를 0으로 놓으면)

$$w = \sum_{i=1}^{m} \alpha^i t^i x^i$$

$$\sum_{i=1}^{m} \alpha^i t^i = 0$$

새로운 목적함수

$$w = \sum_{i=1}^{m} \alpha^{(i)} t^{(i)} x^{(i)}, \qquad \sum_{i=1}^{m} \alpha^{(i)} t^{(i)} = 0$$

를 식[1]에 대입

$$\mathcal{L}(\hat{w}, \hat{b}, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha^{(i)} \left(t^{(i)} \left(w^T x^{(i)} + b \right) - 1 \right)$$

$$= \frac{1}{2} w^t w - \sum_{i=1}^{m} \alpha^{(i)} \left(t^{(i)} \left(w^T x^{(i)} + b \right) - 1 \right)$$

$$= \frac{1}{2} w^t w - \sum_{i=1}^{m} \left(\alpha^{(i)} t^{(i)} w^T x^{(i)} + \alpha^{(i)} t^{(i)} b - \alpha^{(i)} \right)$$

$$= \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} t^{(i)} t^{(j)} x^{(i)^T} x^{(j)} - \sum_{i=1}^{m} \alpha^{(i)}$$

$\hat{\alpha}$를 찾는다. 제약이 있는 최적화 문제로 쌍대문제를 찾았다

쌍대문제(dual problem)

- 제약이 있는 최적화 문제인 원 문제(primal problem)는 쌍대문제(dual problem)로 표현할 수 있다.
- 쌍대문제 해는 원 문제해의 하한 값이지만, SVM에서 원 문제와 똑같은 해를 제공
- 선형 SVM 목적 함수의 쌍대 형식
  - 미분결과를 라그랑주 함수정의에 대입

$$\underset{a}{minimize}\,\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha^{(i)}\alpha^{(j)}t^{(i)}t^{(j)}x^{(i)^T}x^{(j)}-\sum_{i=1}^{m}\alpha^{(i)}$$

[조건] $i=1,2,\cdots,m$일때 $\alpha^{(i)}\geq 0$

  - 쌍대문제에서 구한 해로 원 문제의 해 계산 : 찾은 $\hat{\alpha}$로 $\hat{w},\hat{b}$ 를 계산할 수 있다.

$$\hat{w}=\sum_{i=1}^{m}\hat{\alpha}^{(i)}t^{(i)}x^{(i)}$$

$$\hat{b}=\frac{1}{n_s}\sum_{\substack{i=1\\ \hat{\alpha}^{(i)}>0}}^{m}\left(t^{(i)}-\hat{w}^T x^{(i)}\right)$$

쌍대문제에 커널트릭 적용

- 커널트릭을 사용한다면 예측식에 $\phi(x^{(i)})$를 포함해야 하는데 $\hat{w}$의 차원이 매우 크거나 무한한 $\phi(x^{(i)})$의 차원과 같아져야 하므로 이를 계산할 수 없다
- 커널함수 사용 : 새로운 샘플 $x^{(n)}$의 결정함수에 $\hat{w}$에 대한식을 적용
- 커널 SVM으로 예측하기

$$h_{\hat{w}\hat{b}}\left(\phi(x^{(n)})\right) = \hat{w}^T\phi(x^{(n)}) + \hat{b} = \left(\sum_{i=1}^{m}\hat{\alpha}^{(i)}t^{(i)}\phi(x^{(i)})\right)^T\phi(x^{(n)}) + \hat{b}$$

$$= \sum_{i=1}^{m}\hat{\alpha}^{(i)}t^{(i)}\left(\phi(x^{(i)})^T\phi(x^{(n)})\right) + \hat{b}$$

$$= \sum_{i=1}^{m}\hat{\alpha}^{(i)}t^{(i)}\boldsymbol{K}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(n)}) + \hat{b}$$

- 커널 트릭을 사용한 편향 계산

$$\hat{b} = \frac{1}{n_s}\sum_{\substack{i=1\\ \hat{\alpha}^{(i)}>0}}^{m}\left(t^{(i)} - \hat{w}^T\phi(x^{(i)})\right) = \frac{1}{n_s}\sum_{\substack{i=1\\ \hat{\alpha}^{(i)}>0}}^{m}\left(t^{(i)} - \left(\sum_{i=1}^{m}\hat{\alpha}^{(i)}t^{(i)}\phi(x^{(i)})\right)^T\phi(x^{(n)})\right)$$

$$= \frac{1}{n_s}\sum_{\substack{i=1\\ \hat{\alpha}^{(i)}>0}}^{m}\left(t^{(i)} - \sum_{\substack{i=j\\ \hat{\alpha}^{(j)}>0}}^{m}\hat{\alpha}^{(j)}t^{(j)}\boldsymbol{K}(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})\right)$$

온라인 SVM

- 온라인 학습은 새로운 샘플이 생겼을 때 점진적으로 학습하는 것
- 원 문제로부터 유도된 비용함수(선형 SVM 분류기의 비용함수, $J(w,b)$)를 최소화하기 위한 경사 하강법을 사용

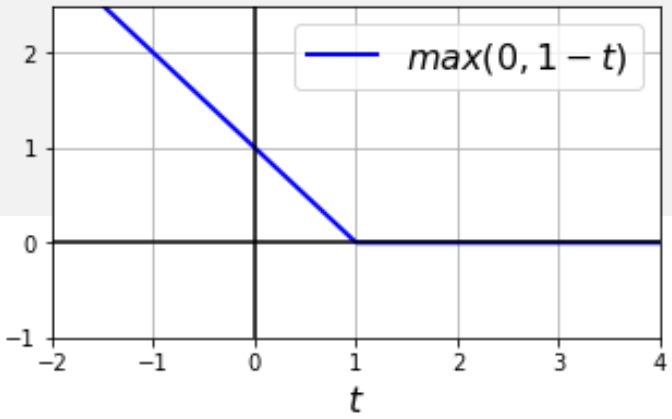
$$J(w,b) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{m}\max\left(0, 1 - t^{(i)}\left(w^T x^{(i)} + b\right)\right)$$

  - 첫 번째 항 : 작은 가중치 벡터 w를 가지도록 제약을 가해 마진을 크게 만든다
  - 두 번째 항 : 모든 마진 오류를 계산.
    샘플이 경계에서 올바른 방향으로 벗어나 있다면 마진오류는 0

- 대규모의 비선형 문제라면 신경망 알고리즘을 고려해 보는 것이 좋다.

힌지손실(hinge loss)함수


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

t = np.linspace(-2, 4, 200)
h = np.where(1 - t < 0, 0, 1 - t)   # max(0, 1-t)

plt.figure(figsize=(5,2.8))
plt.plot(t, h, "b-", linewidth=2, label="$max(0, 1 - t)$")
plt.grid(True, which='both')
plt.axhline(y=0, color='k')
plt.axvline(x=0, color='k')
plt.yticks(np.arange(-1, 2.5, 1))
plt.xlabel("$t$", fontsize=16)
plt.axis([-2, 4, -1, 2.5])
plt.legend(loc="upper right", fontsize=16)
plt.show()

# 참조 : scikit-learn의 SGDClassifier 에서
# loss 매개변수를 "hinge"로 지정하면 선형 SVM문제가 된다.
```



힌지함수 : $\max(0, 1-t)$

$t \geq 1$일때 0

기울기 : $t < 1$이면 -1이고 $t > 1$이면 0