In [8]:

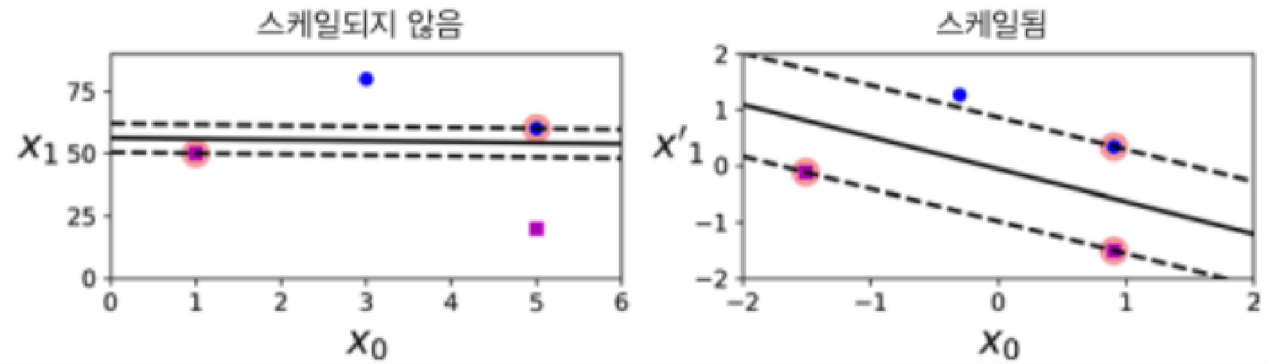
```
from matplotlib import font_manager, rc
import matplotlib as mpl
font_path = "C:\\Users\\이혜림\\Desktop\\Bita5/malgun.ttf" #폰트 파일의 위치
font_name = font_manager.FontProperties(fname=font_path).get_name()
rc("font",family=font_name)
mpl.rcParams["axes.unicode_minus"]=False

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

서포트 벡터 머신(SVM)

- gradient descent를 사용하든 말든 feature의 크기에 민감하기 때문에 회귀, 분류 모델 둘 다에서 스케일링 필수

1) 선형 SVM 분류

- 두 클래스 사이에 가장 폭이 넓은 분류기를 찾음 -> 일반화에 좋을 것
- large margin classification
- **서포트 벡터** : 도로 경계에 위치한 샘플들, 분류기를 찾는데에 영향을 주는 샘플들
- feature의 스케일에 민감
    - 왼쪽그림에서 x1축의 스케일이 x0축 보다 훨씬 커서 경계선이 수평에 가깝게 됨



1-1) 하드 마진 분류

- 모든 샘플이 도로 바깥쪽에 올바르게 분류되어 있는 것
- 문제점
    - 모든 데이터가 선형으로 구분 가능해야 함
    - 이상치에 상관없이 모든 샘플을 올바르게 분류해야 하기 때문에 이상치에 민감

1-2) 소프트 마진 분류

- 도로의 폭을 가능한 넓게 유지하는 것과 마진오류(샘플이 도로 중간이나 심지어 반대쪽에 있는 경우) 사이에 적절한 균형을 잡는 것

- **C** : 마진 오류에 대한 민감도
  - 낮게 설정 : 마진 오류에 대한 민감도가 낮기 때문에 보다 일반화에 용이
  - 높게 설정 : 마진 오류에 대한 민감도가 높기 때문에 보다 오류가 낮은 모델이 나올 수 있음. 하지만 트레인셋에 대한 과대적합의 위험성

In [3]:

```
import numpy as np
from sklearn import datasets
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC # 선형 SVM

iris = datasets.load_iris()
X = iris["data"][:,(2,3)] # 꽃잎 길이, 꽃잎 너비
y = (iris["target"]==2).astype(np.float64) # Iris-Virginica 인지 아닌지에 대한 target 값

svm_clf = Pipeline([("scaler", StandardScaler()),
                    ("linear_svc", LinearSVC(C=1, loss="hinge"))])
# LinearSVC == SGDCLassifier(loss="hinge", alpha=1(m+C)) == SVC(kernel="linear",C=1)

svm_clf.fit(X,y)
```

Out[3]:

```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('linear_svc',
                 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))],
         verbose=False)
```

In [4]:

```
svm_clf.predict([[5.5,1.7]]) # iris-Virginica 이다.

"""
linearSVC : 계산을 통해 구함(빠름, 메모리 많이듦)
SGDClassifier : 경사하강법을 통해 구함(느림, 메모리 좀 적게듦)
(장단점은 linear에서의 장단점과 같음)"""
```

Out[4]:

```
array([1.])
```

2) 비선형 SVM 분류

- 선형적으로 분류할 수 없는 데이터 셋이 많기 때문에 비선형 분류기 필요

2-1) 다항 특성 추가

- 특성들을 거듭제곱을 통해 새로운 특성을 만들어내어 이를 선형적으로 분류

- 모든 머신러닝 알고리즘에서 잘 작동 but
    - 낮은 차수의 다항식 -> 매우 복잡한 데이터셋을 잘 표현하지 못함
    - 높은 차수의 다항식 -> 굉장히 많은 특성을 추가하므로 모델이 느려짐, 오버피팅

In [6]:

```
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)
# 반드시 새로운 거듭제곱 feature을 만든 후에 scaler을 적용해 주어야 함!
polynomial_svm_clf = Pipeline([
    ("poly_features", PolynomialFeatures(degree=3)),
    ("scaler", StandardScaler()),
    ("svm_clf", LinearSVC(C=10, loss="hinge"))
])

polynomial_svm_clf.fit(X,y)
```

```
C:₩ProgramData₩Anaconda3₩lib₩site-packages₩sklearn₩svm₩_base.py:947: ConvergenceWarn
ing: Liblinear failed to converge, increase the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[6]:

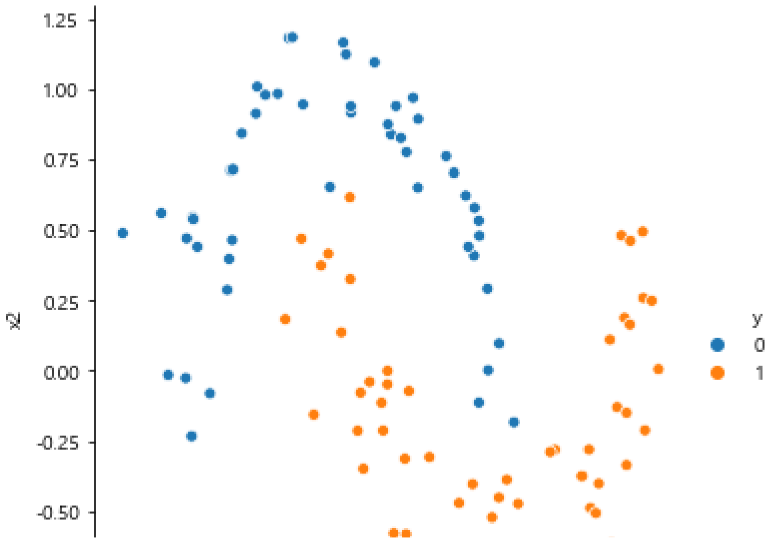
```
Pipeline(memory=None,
         steps=[('poly_features',
                 PolynomialFeatures(degree=3, include_bias=True,
                                    interaction_only=False, order='C')),
                ('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('svm_clf',
                 LinearSVC(C=10, 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))],
         verbose=False)
```

In [16]:

```
# moon 데이터 시각화 -> 비선형적 데이터
moon = pd.DataFrame(X, columns = ["x1","x2"])
moon["y"]=y
sns.relplot(x="x1", y="x2", hue="y", data=moon)
```

Out[16]:

<seaborn.axisgrid.FacetGrid at 0x26a1d990608>



2-2) 다항식 커널

- **커널트릭** -> 수학적 기교를 적용하여 실제로는 특성을 추가하지 않으면서 다항식 특성을 많이 추가한 것과 같은 결과를 얻음
  - 실제로는 어떠한 특성도 추가하지 않기 때문에 모델을 느리게 만들지 않음
- r(coef0) : 높은 차수와 낮은 차수에 얼마나 영향을 받을지를 조절

In [17]:

```
from sklearn.svm import SVC
poly_kernel_svm_clf = Pipeline([
    ("scaler", StandardScaler()),
    ("svm_clf", SVC(kernel="poly", degree = 3, coef0 = 1, C=5))
])
# 3차 다항식 커널을 사용
# coef0 -> 높은 차수와 낮은 차수에 얼마나 영향을 받을지를 조절, 상수항 r에 해당
# 적절한 값으로 지정하면 고차항의 영향을 줄일 수 있음
```

2-3) 유사도 특성

- 유사도 함수로 계산한 특성을 추가하여 선형적으로 구분이 가능하도록 하는 것.
- **유사도 함수** : 각 샘플(벡터 형식)이 특정 랜드마크와 얼마나 닮았는지 측정
- 가우시안 방사 기저 함수(RBF) 를 유사도 함수로 정의

식 5-1 가우시안 RBF

$$\phi_{\gamma}(\mathbf{x}, \ell) = \exp\left(-\gamma \left\| \mathbf{x} - \ell \right\|^2\right)$$

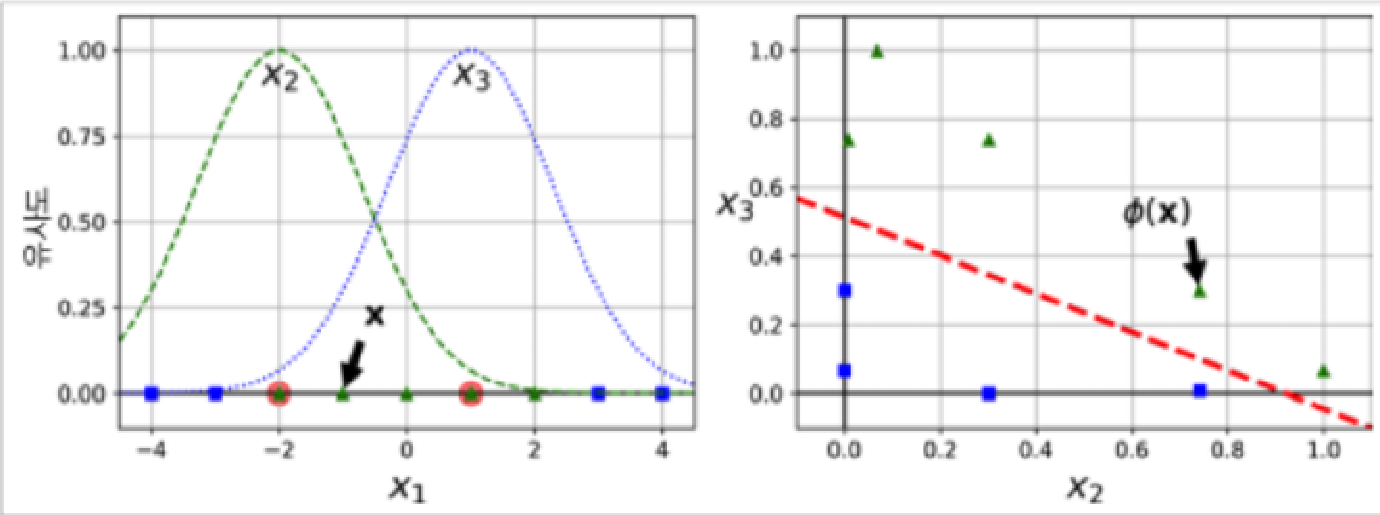
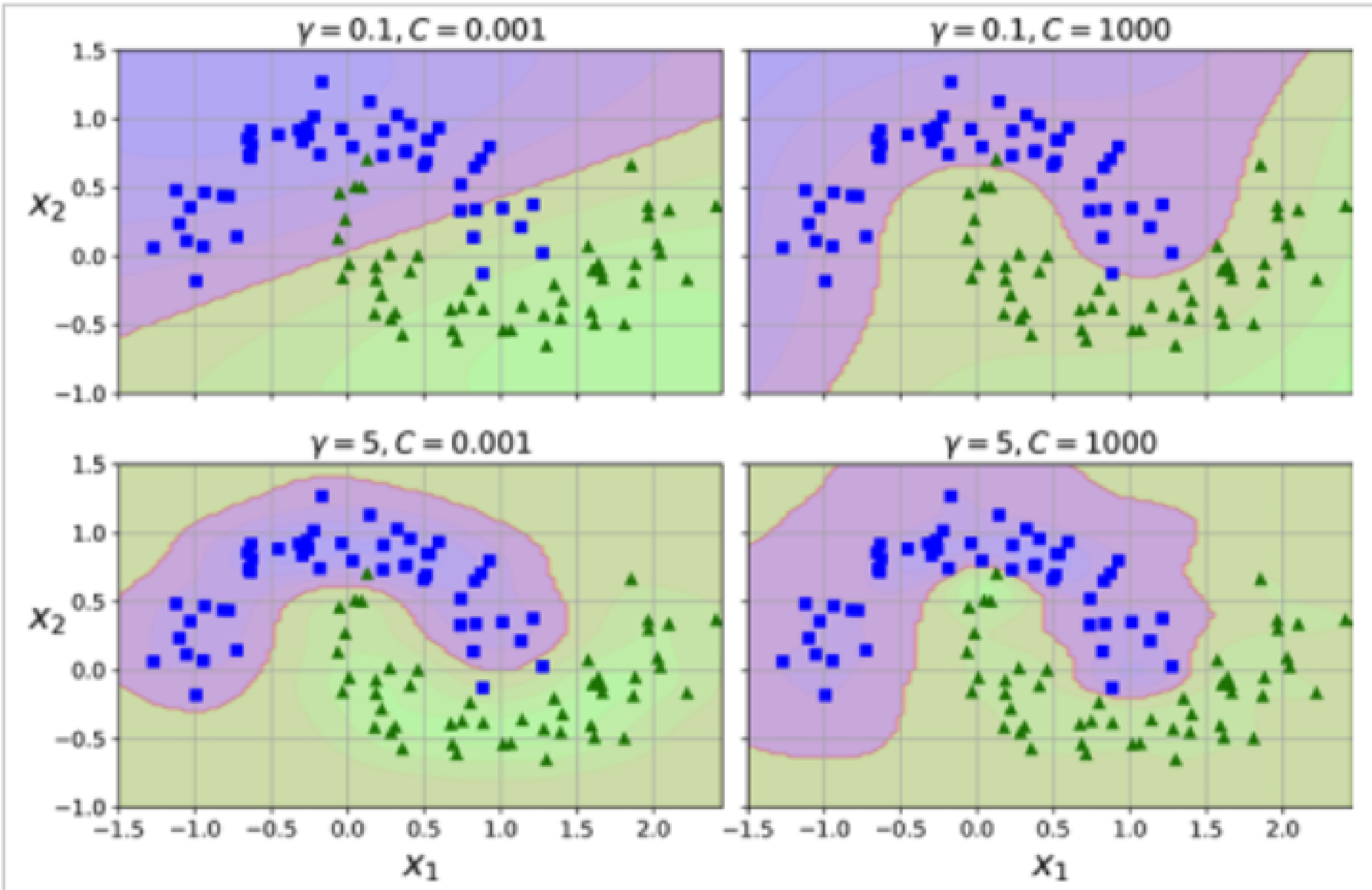


그림 5-8 가우시안 RBF를 사용한 유사도 특성

- 랜드마크 선택 방법 : 데이터셋에 있는 모든 샘플 위치에 랜드마크를 설정
  - 차원이 매우 커짐에 따라 변환된 훈련 세트가 선형적으로 구분될 가능성이 높음
  - n개의 특성, m개의 데이터 -> m개의 특성, m개의 데이터(원본 특성은 제외한다고 가정)
  - 차원이 매우 커짐에 따라 많은 연산 비용, 시간 소요(데이터 셋이 많을 경우 더욱 더)

2-4) 가우시안 RBF 커널

- **커널트릭** -> 수학적 기교를 이용하여 실제로는 유사도 특성을 추가하지 않으면서 유사도 특성을 많이 추가한 것과 비슷한 결과를 얻음
- gamma : 각 샘플의 영향 범위를 조절
  - 증가 : 종모양 그래프가 좁아지면서 각 샘플의 영향 범위 줄어듦
    - 각 샘플간 편차가 커지기 때문에 결정 경계가 조금 더 불규칙, 구불구불하게 휘어짐
    - overfitting의 위험
  - 감소 : 종모양 그래프가 넓어지면서 각 샘플의 영향 범위 늘어남
    - 각 샘플간 편차가 줄얻르면서 결정 경계가 조금 더 부드러워짐.
    - underfitting의 위험



In [21]:

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

Out[21]:

```
Pipeline(memory=None,
         steps=[('scaler',
                 StandardScaler(copy=True, with_mean=True, with_std=True)),
                ('svm_clf',
                 SVC(C=0.001, break_ties=False, cache_size=200,
                     class_weight=None, coef0=0.0,
                     decision_function_shape='ovr', degree=3, gamma=5,
                     kernel='rbf', max_iter=-1, probability=False,
                     random_state=None, shrinking=True, tol=0.001,
                     verbose=False))],
         verbose=False)
```

2-5) 계산복잡도

- LinearSVC : 최적화된 알고리즘을 구현한 liblinear 라이브러리 기반
  - 훈련샘플이 많고 간단할 때 좋음
- SGDClassifier : gradient descent를 이용하여 최적화
  - LinearSVC보다 다소 느리지만 메모리 덜 필요
- SVC : 커널 트릭 알고리즘을 구현한 libsvm 라이브러리 기반
  - 훈련샘플이 적당하고 복잡할 때 좋음
  - feature에는 덜 민감
  - 희소 특성인 경우에 잘 확장

표 5-1 SVM 분류를 위한 사이킷런 파이썬 클래스 비교

파이썬 클래스	시간 복잡도	외부 메모리 학습 지원	스케일 조정의 필요성	커널 트릭
LinearSVC	$O(m \times n)$	아니오	예	아니오
SGDClassifier	$O(m \times n)$	예	예	아니오
SVC	$O(m^2 \times n) \sim O(m^3 \times n)$	아니오	예	예

3) SVM 회귀

- 선형, 비선형 분류뿐만 아니라 선형, 비선형 회귀에도 사용 가능
- 일정한 마진 오류 안에 도로 안에 가능한 한 많은 샘플이 들어가도록 학습
- tol : 허용 오차
- epsilon : 도로의 폭(margin)을 정하는 하이퍼파라미터
  - epsilon 이 크면 많은 데이터들이 도로 안으로 들어오게 됨
    - 과대적합의 우려
  - epsilon 이 작으면 적은 데이터들이 도로 안으로 들어옴
    - 과소적합의 우려
  - 마진 안에서는 훈련 샘플이 추가되어도 모델의 예측에는 영향이 없음

In [18]:

```
from sklearn.svm import LinearSVR # 훈련 세트의 크기에 비례해서 선형적으로 시간 증가

svm_reg = LinearSVR(epsilon=1.5)
svm_reg.fit(X,y)
```

Out[18]:

```
LinearSVR(C=1.0, dual=True, epsilon=1.5, fit_intercept=True,
          intercept_scaling=1.0, loss='epsilon_insensitive', max_iter=1000,
          random_state=None, tol=0.0001, verbose=0)
```

In [19]:

```
from sklearn.svm import SVR # 훈련 세트의 크기가 커지면 폭발적으로 시간 증가

svm_poly_reg = SVR(kernel="poly", degree=2, C=100, epsilon=0.1)
svm_poly_reg.fit(X,y)
```

Out[19]:

```
SVR(C=100, cache_size=200, coef0=0.0, degree=2, epsilon=0.1, gamma='scale',
    kernel='poly', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

4) SVM 이론

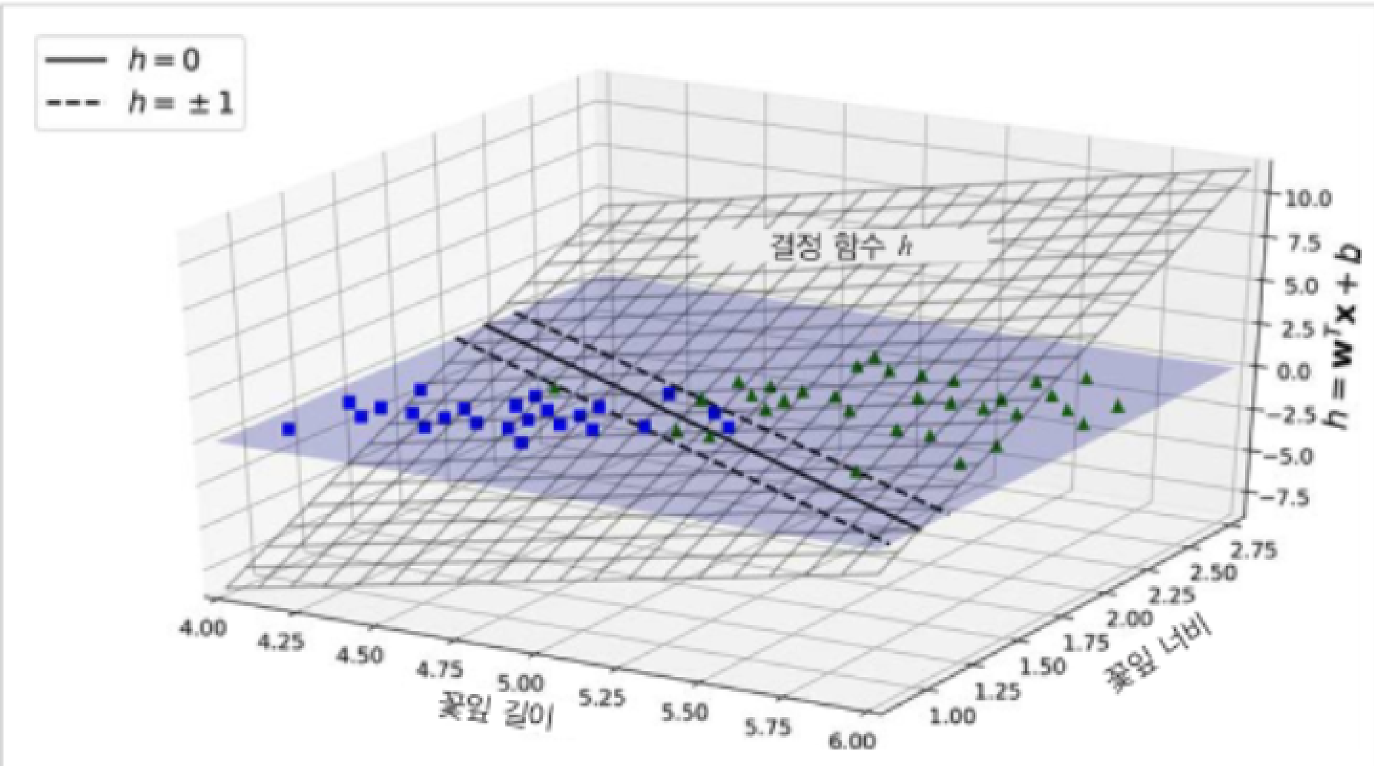
4-1) 선형 SVM 모델기

- **WT.dot(x) + b 의 W,b의 파라미터를 찾음**

식 5-2 선형 SVM 분류기의 예측

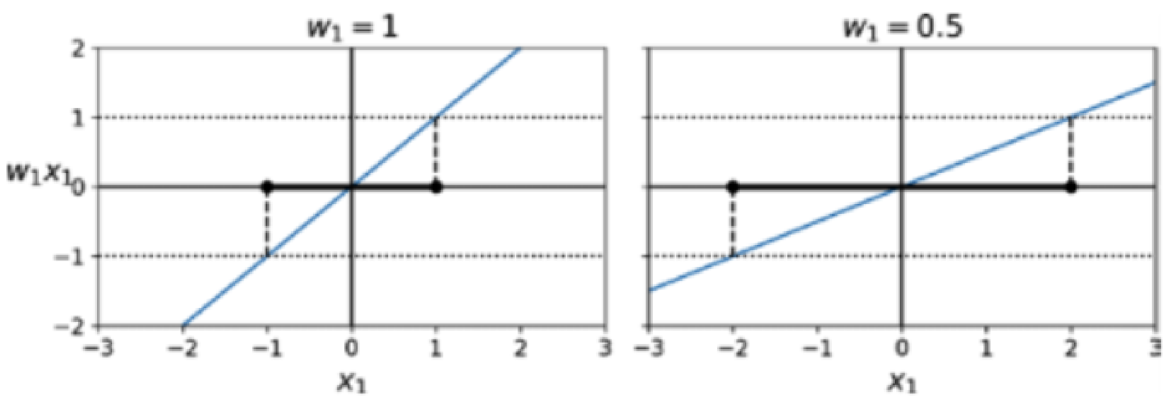
$$\hat{y} = \begin{cases} 0 & \mathbf{w}^T \mathbf{x} + b < 0 \text{일 때} \\ 1 & \mathbf{w}^T \mathbf{x} + b \geq 0 \text{일 때} \end{cases}$$

- n개의 특성이 있을 때
  - **결정함수** : n차원의 초평면
  - **결정경계** : (n-1)차원의 초평면
  - (ex 2개의 특성이 있을 때 결정함수는 2차원의 초평면(2x1+4x2=y) 이지만 결정경계는 1차원의 초평면(2x1+4x2<0.5))
- 결정경계에 나란하고 일정한 거리만큼 떨어진, 오류를 하나도 발생시키지 않거나(하드 마진) 제한적인 마진 오류를 가지면서 (소프트 마진) **가능한 한 마진을 크게하는 w와 b를 찾음**



4-1-1) 목적함수

- 결정 함수의 기울기(가파른 정도) == 가중치 벡터의 노름(||W||)
- 기울기가 작아질 수록 마진은 커짐 -> **마진을 크게 하기 위해 ||W||을 최소화**
- 결정 경계 : h=0
- 마진 : h = +=1 이기 때문에 기울기가 작아질 수록 마진이 커짐



4-1-1-1) 하드 마진의 목적함수

- 결정 함수가 모든 양의 훈련 샘플에서 1보다 커야 하고, 모든 음의 훈련 샘플에서는 -1보다 작아야 함.
- t(z) 를 이용하여 목적 함수 식을 하나로 표현
  - 음성 샘플일 때 : t(x) = -1
  - 양성 샘플일 때 : t(x) = 1

식 5-3 하드 마진 선형 SVM 분류기의 목적 함수

$$\underset{\mathbf{w},b}{\text{minimize}} \ \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

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

4-1-1-2) 소프트 마진의 목적함수

- 각 샘플에 대해 슬랙변수를 도입
  - 슬랙변수 : i번째 샘플이 얼마나 마진을 위반할지를 정하는 변수
- 마진 오류를 최소화(마진이 줄어듦)하기 위해 슬랙 변수의 값을 작게 만드는 것과 마진을 크게 하기 위해 기울기를 작게 만드는 것이 상충
  - C : 두 목표 사이의 트레이드 오프를 설정
    - 커지면 커질 수록 마진 오류를 최소화하는 목표에 가까워짐
    - 작아지면 작아질 수록 마진을 크게 하는 목표에 가까워짐
    - 0이되면 마진을 크게 하는 목표만 설정됨

식 5-4 소프트 마진 선형 SVM 분류기의 목적 함수[20]

$$\underset{w,b,\zeta}{\text{minimize}} \ \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m}\zeta^{(i)}$$

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

4-1-2) 콰트라틱 프로그래밍

- QP(콰트라틱 프로그래밍) : 선형적인 제약 조건이 있는 볼록 함수의 이차 최적화 문제

식 5-5 QP 문제

$$\underset{\mathbf{p}}{\text{minimize}} \ \frac{1}{2}\mathbf{p}^T \mathbf{H} \mathbf{p} + \mathbf{f}^T \mathbf{p}$$

$$[\text{조건}] \ \mathbf{A}\mathbf{p} \le \mathbf{b}$$

$$\text{여기서} \begin{cases} \mathbf{p}\text{는 } n_p \text{차원의 벡터} (n_p = \text{모델 파라미터 수}) \\ \mathbf{H}\text{는 } n_p \times n_p \text{ 크기 행렬} \\ \mathbf{f}\text{는 } n_p \text{차원의 벡터} \\ \mathbf{A}\text{는 } n_c \times n_p \text{ 크기 행렬} (n_c = \text{제약 수}) \\ \mathbf{b}\text{는 } n_c \text{차원의 벡터} \end{cases}$$

- 하드 마진과 소프트 마진의 목적 함수는 모두 콰트라틱 프로그래밍에 해당
  - $n_p = n + 1$, 여기서 n은 특성 수입니다(편향 때문에 +1이 추가되었습니다).
  - $n_c = m$, 여기서 m은 훈련 샘플 수입니다.
  - $\mathbf{H}$는 $n_p \times n_p$ 크기이고 왼쪽 맨 위의 원소가 0(편향을 제외하기 위해)인 것을 제외하고는 단위행렬입니다.
  - $\mathbf{f} = \mathbf{0}$, 모두 0으로 채워진 n_p차원의 벡터입니다.
  - $\mathbf{b} = \mathbf{1}$, 모두 1로 채워진 n_c차원의 벡터입니다.
  - $a^{(i)} = -t^{(i)}\dot{\mathbf{x}}^{(i)}$, 여기서 $\dot{\mathbf{x}}^{(i)}$는 편향을 위해 특성 $\dot{\mathbf{x}}_0 = 1$을 추가한 $\mathbf{x}^{(i)}$와 같습니다.

4-1-3) 쌍대문제

- SVM의 최적화 문제인 원문제는 특정 조건을 만족시키기 때문에 쌍대문제로 바꾸어서 푼 후에 다시 원 문제의 해로 변환했을 때 원문제와 똑같은 해를 제공
  - 특정 조건 : 목적 함수가 볼록 함수 이고, 부등식 제약 조건이 연속 미분 가능하면서 볼록 함수.
  - 특정조건을 만족시키지 못하면 쌍대 문제의 해는 원 문제 해의 하한 값
- 원 문제 또는 쌍대 문제 중 하나를 선택하여 풀 수 있음
  - 쌍대 문제를 이용하면
    - feature의 개수가 많을 때 더 빠른 연산 가능
    - 커널트릭을 사용하여 다양한 트릭을 사용할 수 있음

4-2) 커널 SVM

- 다양한 커널트릭방식을 이용하여 실제 feature을 추가하지 않고도 실제 feature을 추가한 것처럼 계산함으로써 속도를 현저히 줄일 수 있음.

- 훈련 시와 예측 시 모두 커널트릭을 사용하여 연산량과 연산 시간을 줄일 수 있음

4-3) 온라인 SVM

- QP로 SVM 문제를 푸는 경우 모든 데이터셋을 넣어서 한번에 종합적으로 최적의 W,b를 찾음
- 따라서 점진적으로 학습을 하는 온라인 학습의 경우 cost 함수를 이용하여 gradient descent 알고리즘을 사용해 주어야 함.
- 비용 함수 : hinge cost => max(0, 1-t)
    - t>=1 : 기울기 : 0
    - t<1 : 기울기 : 1
    - t=1 : 기울기 : 1(서브 그라이언트)
        - 서브 그라디언트 : 미분 불가능한 점에서 근처 값들의 미분값의 중앙값을 미분값으로 사용하는 것.

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

- 앞의 항 : 작은 가중치 벡터를 만들어서 마진을 최대화 하기 위한 제약식
- 뒤의 항 : 마진 오류 term, 마진 오류가 하나도 없으면 0이 됨. t는 target값에 따른 값
- C : 마진 오류를 반영할 가중치, 크면 클수록 마진 오류에 대한 가중치가 올라감으로 overfitting됨