[4주차]
서포트 백터 머신

1기 김희숙
1기 안하린
1기 최지우

EURON

목차

5.0 서포트 벡터 머신

SVM(support vector machine)란?

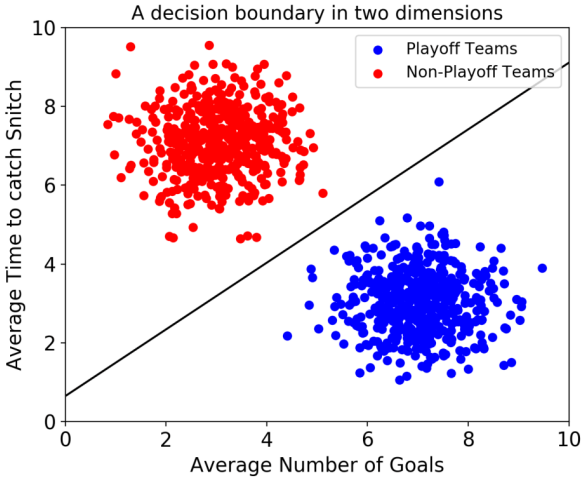
- 정의: 결정 경계 (Decision Boundary) , 즉 분류를 위한 기준 선을 정의하여 주어진 데이터가 어느 카테고리에 속할지 판단하는 이진 선형 분류 모델

- 복잡한 분류 문제에 잘 맞음

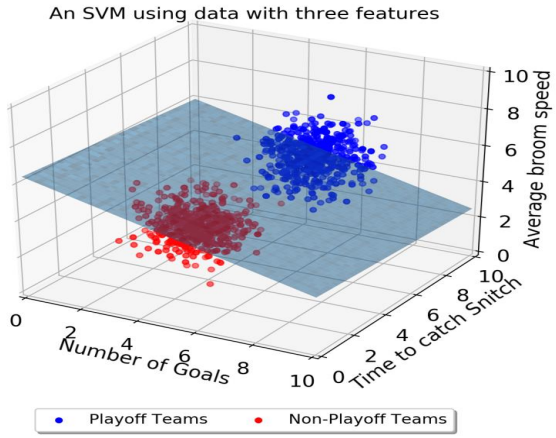
- 작거나 중간 크기의 데이터셋에 적합

결정 경계 (Decision Boundary)

속성이 두개일 경우

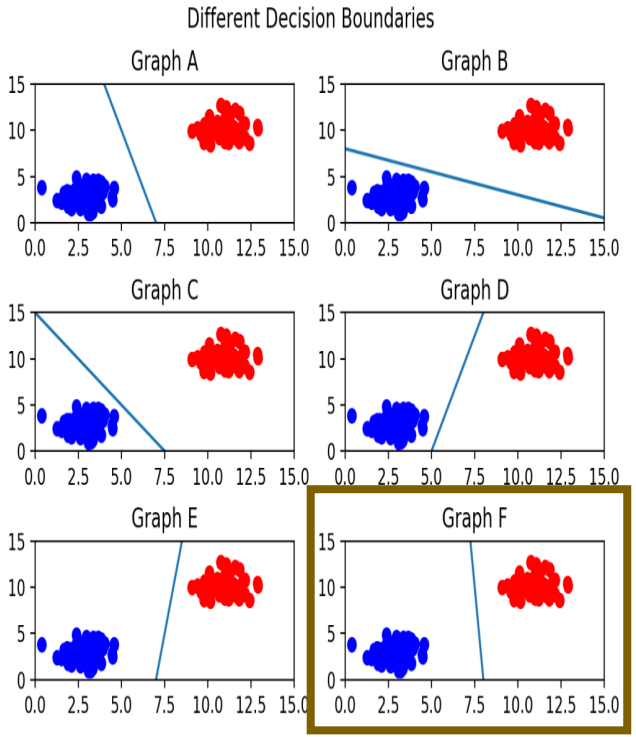


속성이 세개일 경우



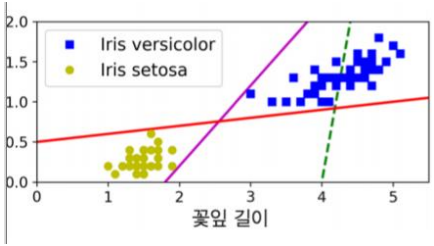
5.1 선형 SVM 분류

최적의 결정 경계는?



선형 SVM

```
# 나쁜 모델
x0 = np.linspace(0, 5.5, 200)
pred_1 = 5*x0 - 20
pred_2 = x0 - 1.8
pred_3 = 0.1 * x0 + 0.5
```

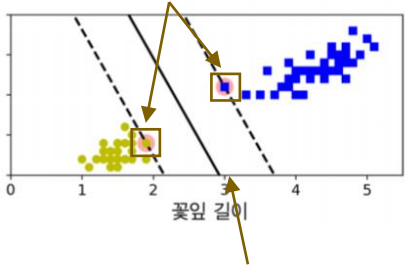



```
# SVM 분류 모델
svm_clf = SVC(kernel="linear", C=float("inf"))
svm_clf.fit(X, y)

def plot_svc_decision_boundary(svm_clf, xmin, xmax):
    w = svm_clf.coef_[0]
    b = svm_clf.intercept_[0]

    # 결정 경계에서 w0*x0 + w1*x1 + b = 0 이므로
    # => x1 = -w0/w1 * x0 - b/w1
    x0 = np.linspace(xmin, xmax, 200)
    decision_boundary = -w[0]/w[1] * x0 - b/w[1]

    margin = 1/w[1]
    gutter_up = decision_boundary + margin
    gutter_down = decision_boundary - margin

    svs = svm_clf.support_vectors_
    plt.scatter(svs[:, 0], svs[:, 1], s=180, facecolors='#FFAAAA
    plt.plot(x0, decision_boundary, "k-", linewidth=2)
    plt.plot(x0, gutter_up, "k--", linewidth=2)
    plt.plot(x0, gutter_down, "k--", linewidth=2)
```

마진의 폭

두 클래스 경계 사이가 가장 넓은 폭을 찾는 알고리즘 => 서포트 벡터

서포트 벡터



svm 분류기의 결정경계

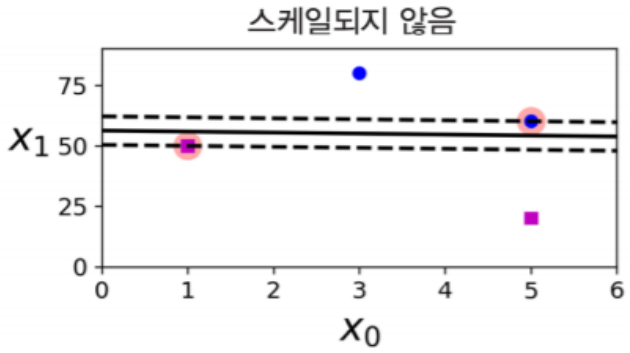
SVM 분류기 = 라지 마진 분류

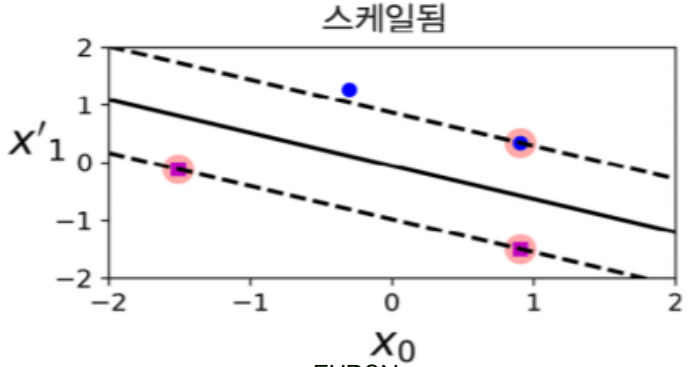
5.1 선형 SVM 분류

스케일 *SVM은 특성의 스케일에 민감

```
Xs = np.array([[1, 50], [5, 20], [3, 80], [5, 60]]).astype(np.float64)
ys = np.array([0, 0, 1, 1])
svm_clf = SVC(kernel="linear", C=100)
svm_clf.fit(Xs, ys)

plt.figure(figsize=(9,2.7))
plt.subplot(121)
plt.plot(Xs[:, 0][ys==1], Xs[:, 1][ys==1], "bo")
plt.plot(Xs[:, 0][ys==0], Xs[:, 1][ys==0], "ms")
plot_svc_decision_boundary(svm_clf, 0, 6)
plt.xlabel("$x_0$", fontsize=20)
plt.ylabel("$x_1$    ", fontsize=20, rotation=0)
plt.title("Unscaled", fontsize=16)
plt.axis([0, 6, 0, 90])
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(Xs)
svm_clf.fit(X_scaled, ys)

plt.subplot(122)
plt.plot(X_scaled[:, 0][ys==1], X_scaled[:, 1][ys==1], "bo")
plt.plot(X_scaled[:, 0][ys==0], X_scaled[:, 1][ys==0], "ms")
plot_svc_decision_boundary(svm_clf, -2, 2)
plt.xlabel("$x'_0$", fontsize=20)
plt.ylabel("$x'_1$  ", fontsize=20, rotation=0)
plt.title("Scaled", fontsize=16)
plt.axis([-2, 2, -2, 2])
```





5.1.1 소프트 마진 분류

마진분류

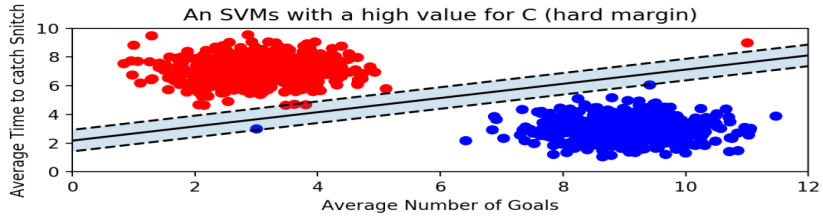
- 마진(Margin)이란?

→결정 경계와 서포트 벡터 사이의 거리

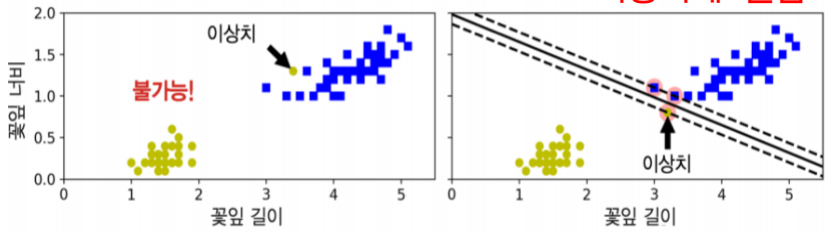
→최적의 결정경계 = 마진의 최대화

→이상치를 다루는 것이 필요

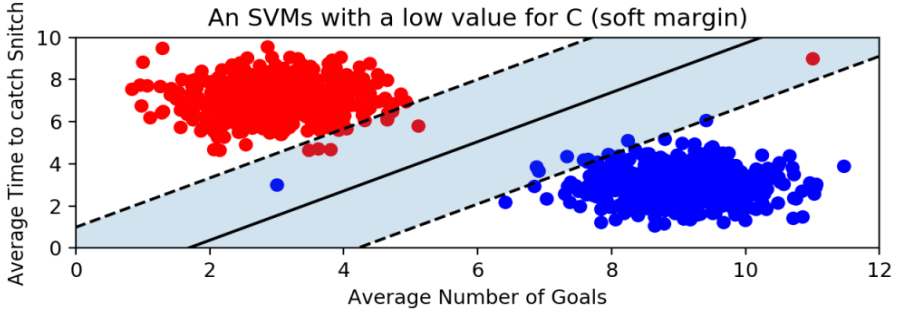
하드마진



이상치에 민감



소프트마진



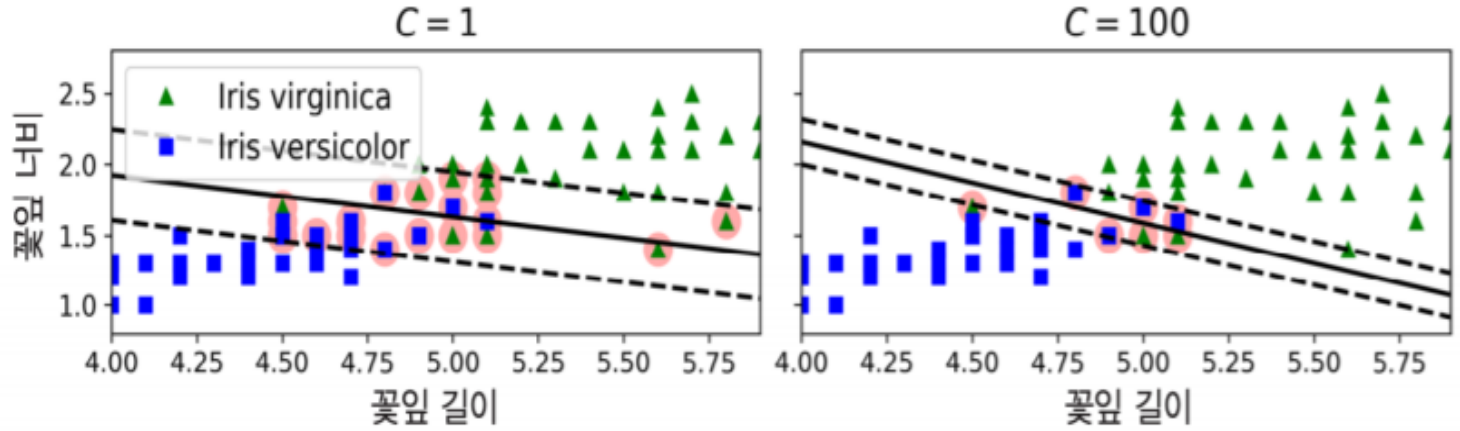
5.1.1 소프트 마진 분류

SVM 파라미터: C

• C: 마진의 넓이와 마진오류 사이의 tradeoff를 결정하는 파라미터

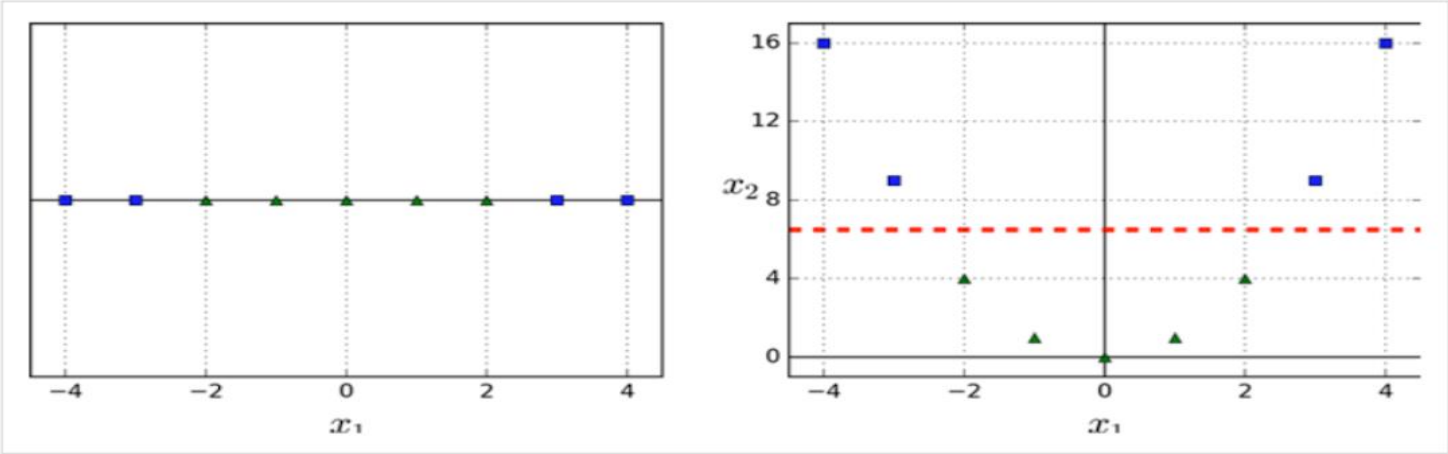
- 적을 경우 : 마진이 넓지만, train error 증가

- 높을 경우: 마진이 좁지만, train error 감소



비선형 SVM

- 선형적으로 분류할 수 없는 비선형 데이터셋을 다룰 때 활용

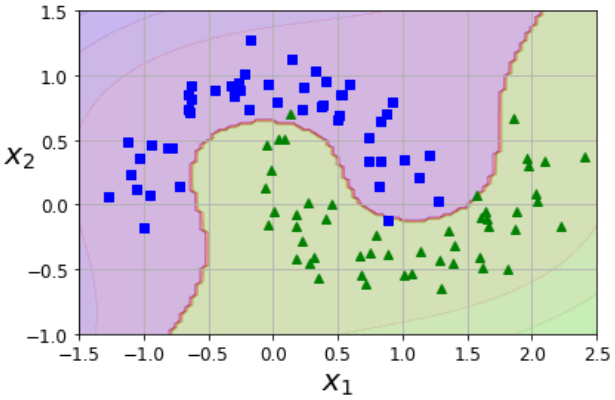


하나의 특성 X1만을 가진 비선형 데이터셋 → X2=(X1)^2을 추가하여 2차원 데이터셋으로 변환

5.2 비선형 SVM 분류

- Moons 데이터셋 활용

```
[ ]  from sklearn.datasets import make_moons
     from sklearn.pipeline import Pipeline
     from sklearn.preprocessing import PolynomialFeatures

     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)
```



다항식 특성을 추가하는 것은 간단하고 잘 작동하지만, 다항식의 차수가 높고 특성의 개수가 많아지면 모델 훈련 시간이 많이 소요되어 비효율적.

그래서 이때 커널트릭을 사용.

<u>커널 트릭</u>
실제로 특성을 추가하지 않았는데 다항식의 특성을 추가한 듯한 효과

다항식 커널 (Polynomial Kernel)

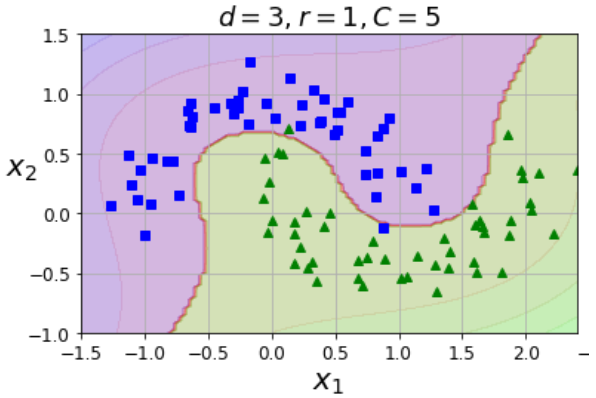
```
from sklearn.svm import SVC

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)
```

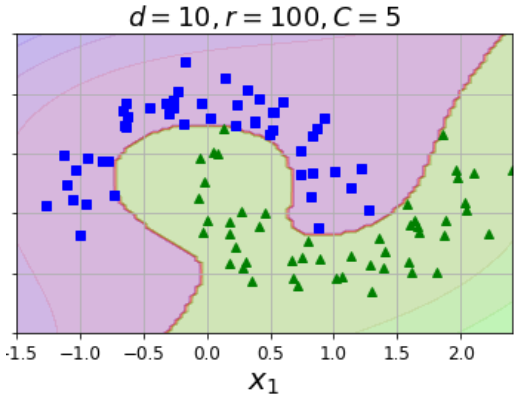
3차 다항식 커널

매개변수coef0는 모델이 높은 차수와 낮은 차수에 얼마나 영향 받을지 조절 (상수항 r) : 고차항의 영향 조절 가능

```
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)
```

10차 다항식 커널





5.2.2 유사도 특성

유사도 특성

- 비선형 특성을 다루는 또 다른 기법
- 유사도 함수로 계산한 특성을 추가

*유사도함수 : 샘플과 특정 랜드마크의 닮은 정도를 측정

유사도함수: 가우시안 rbf로 정의

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

< Exp. 1. 가우시안 RBF >

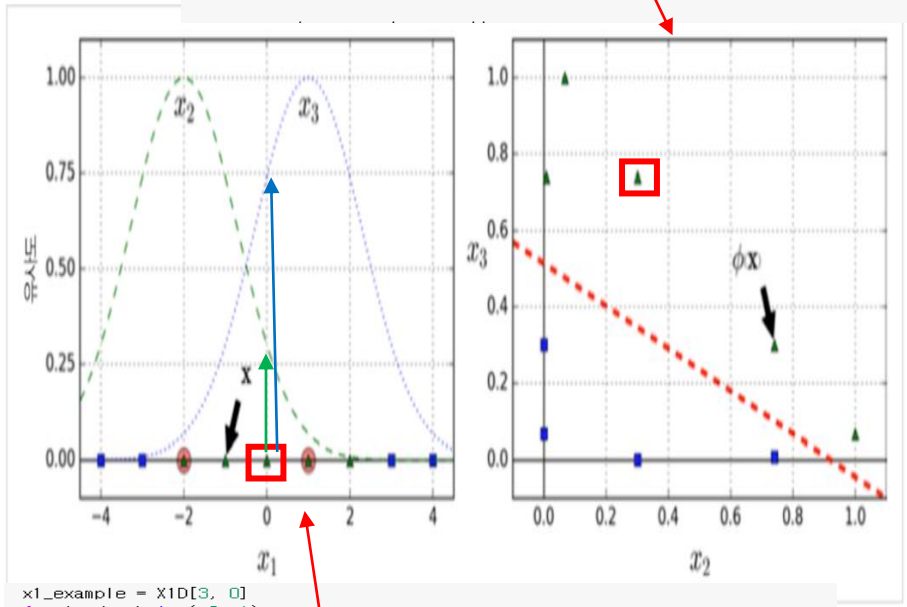
랜드마크 지점

하이퍼파라미터. 작을수록 폭이 넓은 종모양이 됨

```
def gaussian_rbf(x, landmark, gamma):
    return np.exp(-gamma * np.linalg.norm(x - landmark, axis=1)**2)

gamma = 0.3

x1s = np.linspace(-4.5, 4.5, 200).reshape(-1, 1)
x2s = gaussian_rbf(x1s, -2, gamma)
x3s = gaussian_rbf(x1s, 1, gamma)
```

랜드마크 지점

```
XK = np.c_[gaussian_rbf(X1D, -2, gamma), gaussian_rbf(X1D, 1, gamma)]
yk = np.array([0, 0, 1, 1, 1, 1, 1, 0, 0])
```



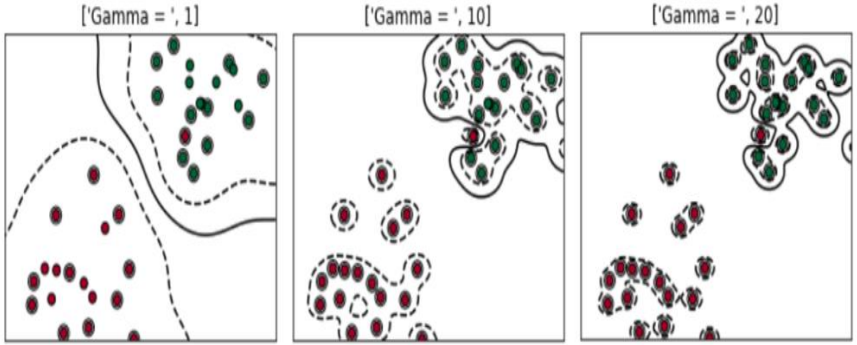

```
x1_example = X1D[3, 0]
for landmark in (-2, 1):
    k = gaussian_rbf(np.array([[x1_example]]), np.array([[landmark]]), gamma)
    print("Phi({}, {}) = {}".format(x1_example, landmark, k))
```

```
Phi(-1.0, -2) = [0.74081822]
Phi(-1.0, 1) = [0.30119421]
```

5.2.3 가우시안 RBF 커널

파라미터: gamma & c

- Gamma: 하나의 데이터 샘플이 영향력을 행사하는 거리를 결정
- Gamma가 클수록 데이터 포인터들이 영향력을 행사하는 길이가 짧아짐
- Gamma가 낮을수록 영향력이 커짐

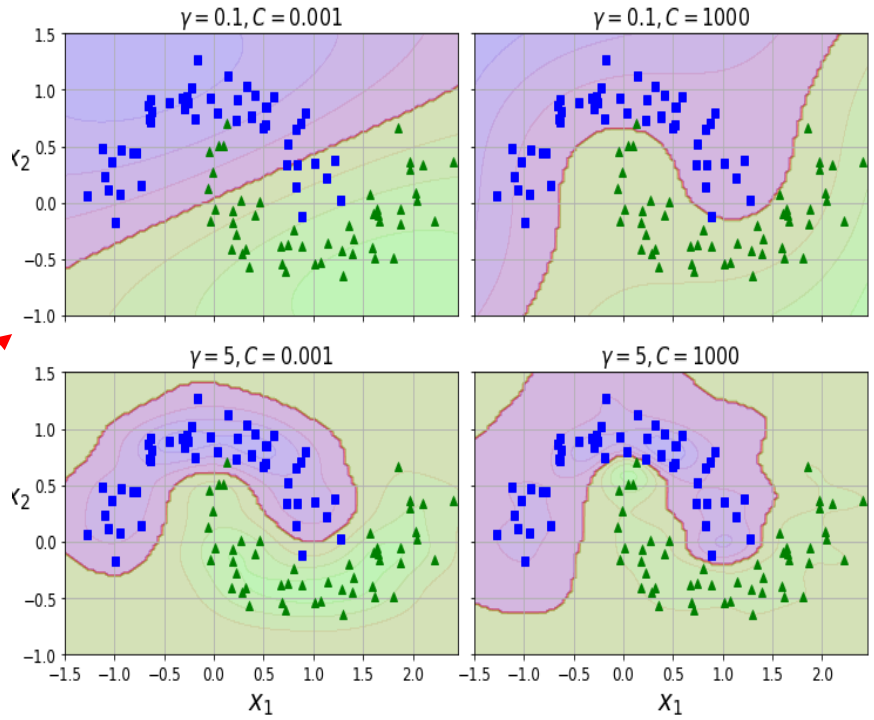


5.2.3 가우시안 RBF 커널

방사 기저 함수 커널 (RBF 커널)

```
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)
```



5.2.4 계산 복잡도

LinearSVC

- O(mn)
- liblinear 라이브러리 기반
- 훈련 샘플(m)과 특성 수(n)에 거의 선형적으로 늘어남 -> O(mn)
- 정밀도를 높이면 알고리즘 수행 시간이 증가함
- 허용 오차의 하이퍼파라미터(ε: tol 매개변수)로 조절

Table 5-1. Comparison of Scikit-Learn classes for SVM classification

Class	Time complexity	Out-of-core support	Scaling required	Kernel trick
`LinearSVC`	$O(m \times n)$	No	Yes	No
`SGDClassifier`	$O(m \times n)$	Yes	Yes	No
`SVC`	$O(m^2 \times n)$ to $O(m^3 \times n)$	No	Yes	Yes

m: 샘플의 수 n: 특성의 수

5.2.4 계산 복잡도

SGDClassifier

- O(mn)
- 확률적 경사하강법을 이용하여 선형 모델 구현

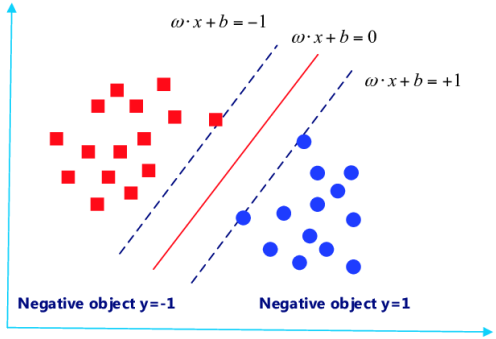
SVC

- O(m^2 n) ~ O(m^3 n)
- libsvm 라이브러리 기반
- 훈련 샘플의 수가 커지면 시간이 매우 오래 걸림
- 복잡하지만 작거나 중간 규모의 훈련 세트에 대해 적합함
- 희소 특성인 경우에 잘 확장됨
- 알고리즘의 성능은 샘플이 가진 0이 아닌 특성의 평균 수에 비례함
- 허용 오차 하이퍼파라미터(ε: tol 매개변수)로 조절

SVM 회귀와 분류의 목표

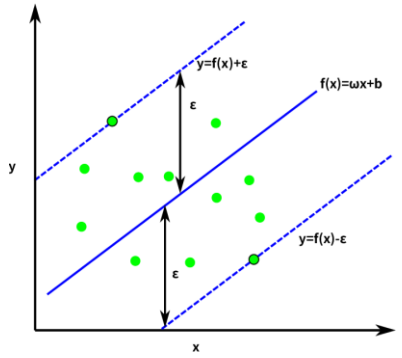
- SVM 분류

일정한 마진 오류 안에서 두 클래스 사이의 도로 폭을 최대한 넓힘



- SVM 회귀

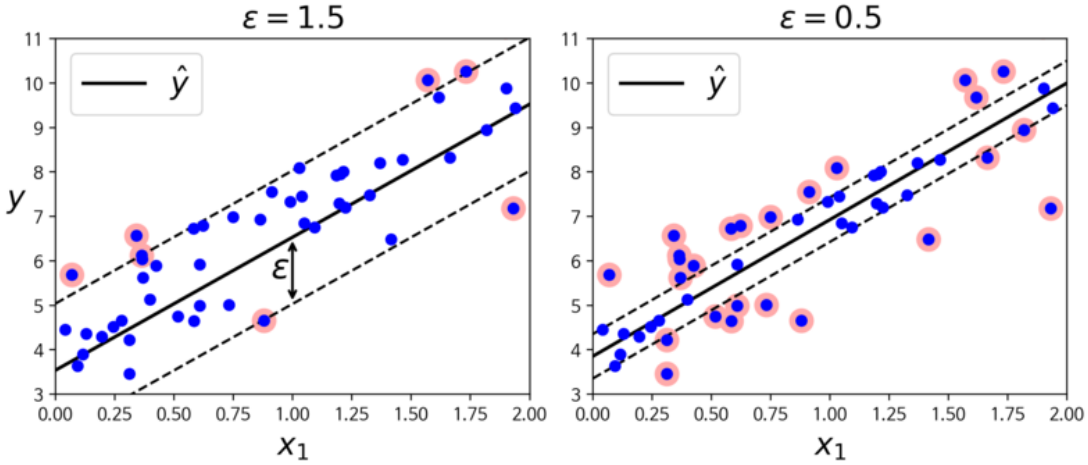
제한된 마진 오류 안에서 도로 폭을 최대한 넓혀서 도로 위에 가능한 많은 샘플을 포함하도록 학습



**회귀 모델의 마진 위반 사례: 도로 밖에 위치한 샘플

SVM 선형 회귀

- 선형 회귀 모델을 SVM을 이용하여 구현
- 마진 안에서는 훈련 샘플이 추가되어도 모델의 예측에는 변함이 없음
- 결과적으로 ε(epsilon) 에 민감하지 않다



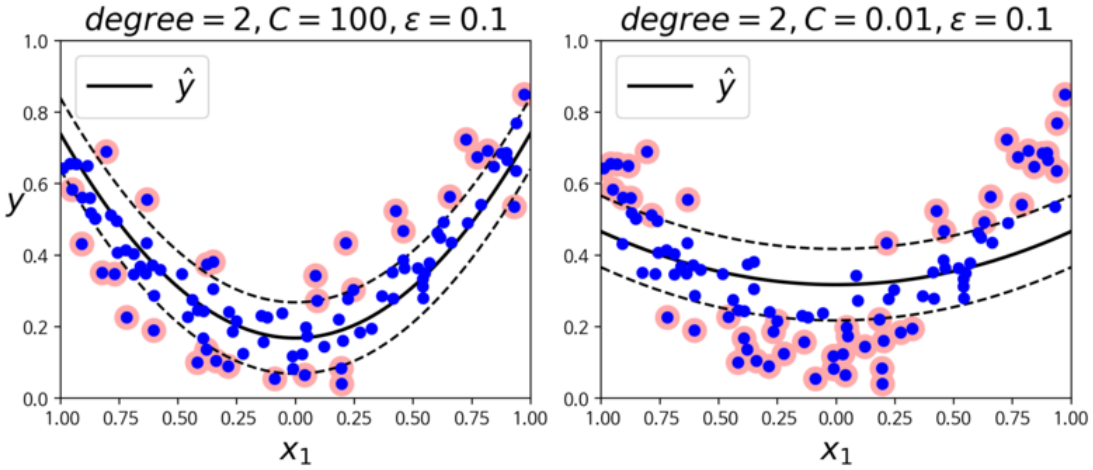

```
# LinearSVR 클래스 지정
from sklearn.svm import LinearSVR
svm_reg = LinearSVR(epsilon = e)
```

	왼편 그래프	오른편 그래프
epsilon=e:	e=1.5: 마진 크게	e=0.5 마진 작게

5.3 SVM 회귀

SVM 비선형 회귀

- 커널 트릭을 활용하여 비선형 회귀 모델 구현
- 마진 안에서는 훈련 샘플이 추가되어도 모델의 예측에는 변함이 없음
- 결과적으로 ε(epsilon) 에 민감하지 않다
- 하이퍼파라미터 C(정규화 매개 변수)가 커질수록 마진 오차에 민감 -> 과대 적합 가능성 ↑




```
# SVR + 다항식 커널
from sklearn.svm import SVR
svm_poly_reg = SVR(kernel = "poly", degree = d,
                   C = C, epsilon = e, gamma = "scale" )
```

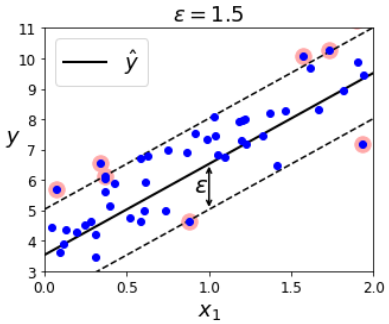
	왼편 그래프	오른편 그래프
degree=d:	d=2: 2차 다항 커널	d=2: 2차 다항 커널
epsilon=e:	e=0.1: 마진 작게	e=0.1 마진 작게
C=C:	C=100: 가중치 규제 거의 없음	C=0.01: 가중치 규제 많음
	샘플에 더 민감	샘플에 덜 민감
	도록폭을 보다 넓게	도로폭을 보다 좁게

5.3 SVM 회귀

모델 생성 학습, 예측

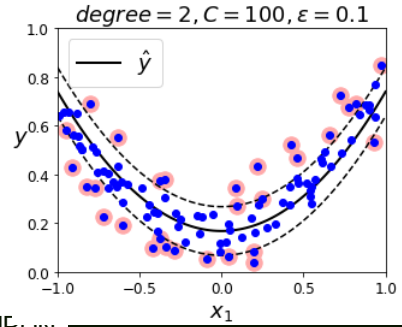
- 사이킷런 LinearSVR(LinearSVC의 회귀 버전)을 사용하여 선형 SVM 회귀 적용
  -> 필요한 시간이 훈련 세트의 크기에 비례하여 선형적으로 늘어남

```
#모델을 만들기 전 훈련 데이터의 스케일을 맞추고 평균을 0으로 맞춰야 함
from sklearn.svm import LinearSVR

svm_reg = LinearSVR(epsilon=1.5, random_state=42)
svm_reg.fit(X, y)

y_predict = svm_reg.predict(X)
```



- 사이킷런 SVR(SVC의 회귀 버전)을 사용하여 선형 SVM 회귀 적용
  -> 훈련 세트의 크기가 커지면 훨씬 느려짐

```
from sklearn.svm import SVR

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

y_predict = svm_reg.predict(X)
```



5.3 SVM 회귀

SVM의 장단점

장점

- 비선형 분리 데이터를 커널 트릭을 사용하여 분류 모델링 가능
- 고차원 공간에서 원활하게 작동함
- 텍스트 분류 및 이미지 분류에 효과적임

단점

- 데이터가 너무 많으면 속도가 느리고 메모리적으로 힘듦
- 확률 추정치를 제공하지 않음
- 선형 커널은 선형의 분리 가능한 데이터인 경우 로지스틱 회귀분석과 거의 유사함

5.4 SVM 이론

선형 SVM 작동 원리

- 결정 함수와 예측
- 목적 함수
- 2차 계획법(QP, quadratic programming)
- 쌍대 문제

커널 SVM 작동원리

- 쌍대 문제를 해결할 때 커널 기법 활용 가능

온라인 SVM

- 온라인 선형 SVM
- 온라인 커널 SVM

5.4.1 결정함수와 예측

선형 SVM 작동 원리: 결정 함수와 예측

- 선형 SVM 분류기 모델의 결정 함수

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$
$$= w_1 x_1 + \cdots + w_n x_n + b$$

- 선형 SVM 분류기 예측

$$\hat{y} = \begin{cases} 0 & \text{if } h(\mathbf{x}) < 0 \\ 1 & \text{if } h(\mathbf{x}) \geq 0 \end{cases}$$

단순히 결정 함수를 계산하여 새로운 샘플 X의 클래스를 예측
결과값이 0보다 크면 예측된 클래스 y.hat 은 양성 클래스(1), 0과 같거나 크면 음성 클래스(0)

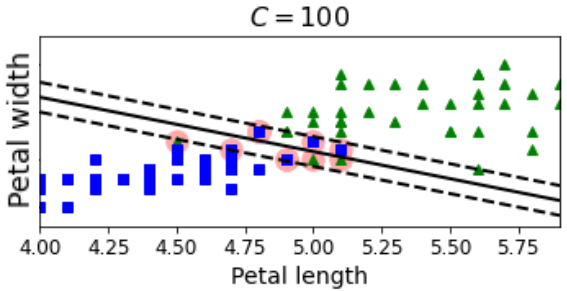
b: 편향 w: 가중치 벡터

5.4.1 결정함수와 예측

결정 경계

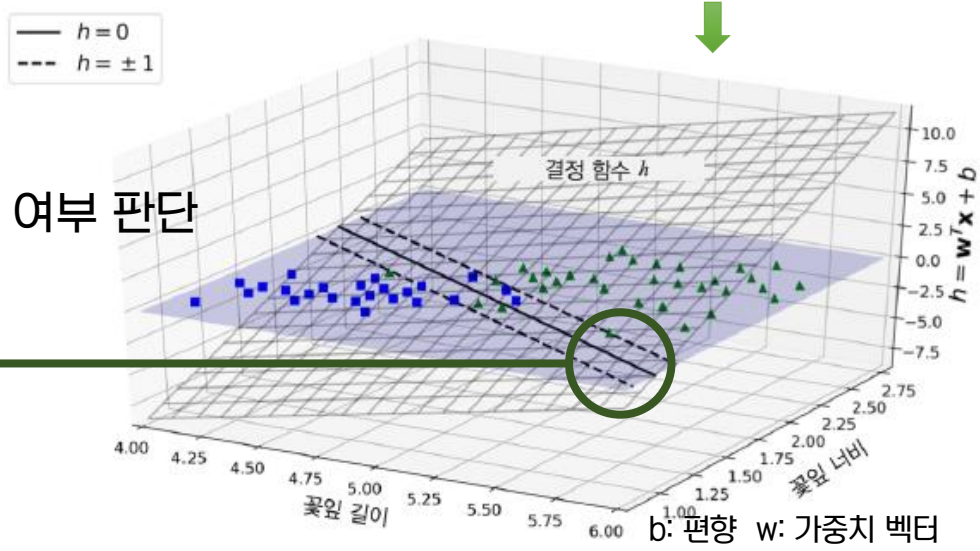
- 결정 함수의 값이 0인 점들의 집합

$$\{\mathbf{x} \mid h(\mathbf{x}) = 0\}$$



- 예제: 붓꽃 분류
- 꽃잎 길이와 너비를 기준으로

  Iris-Virginica(초록색 삼각형) 품종 여부 판단



두 점선에 유의
h(X)가 1 또는 -1인 샘플들의 집합
margin과 밀접하게 관련됨

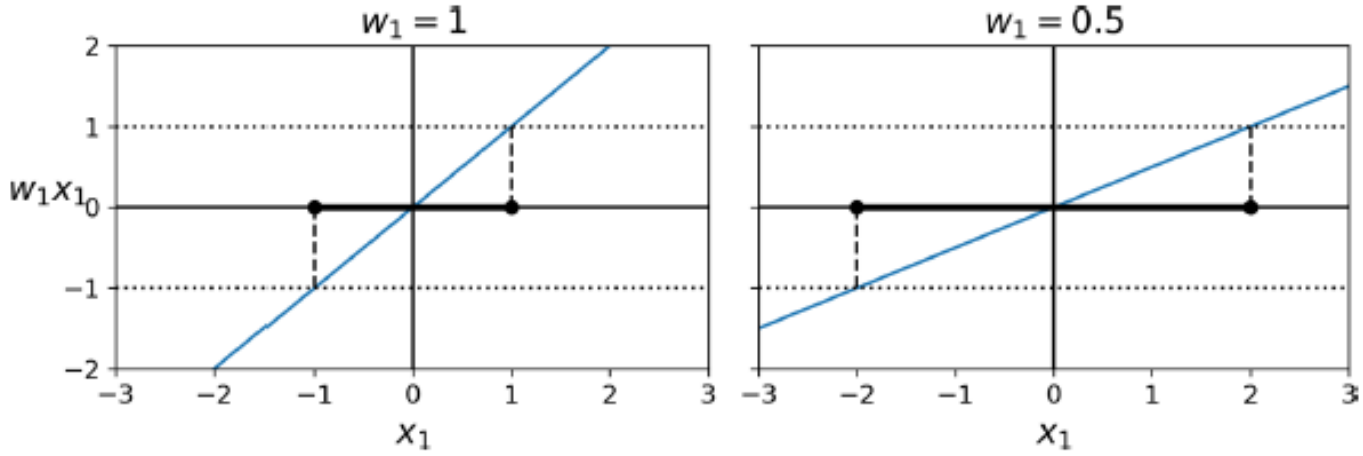
b: 편향 w: 가중치 벡터

5.4.2 목적 함수

선형 SVM 작동 원리: 목적 함수

결정 함수의 기울기와 마진 폭

- 결정 함수의 기울기가 작아질 수록 마진 폭이 커짐
- 결정 함수의 기울기가 $\|\mathbf{w}\|$에 비례



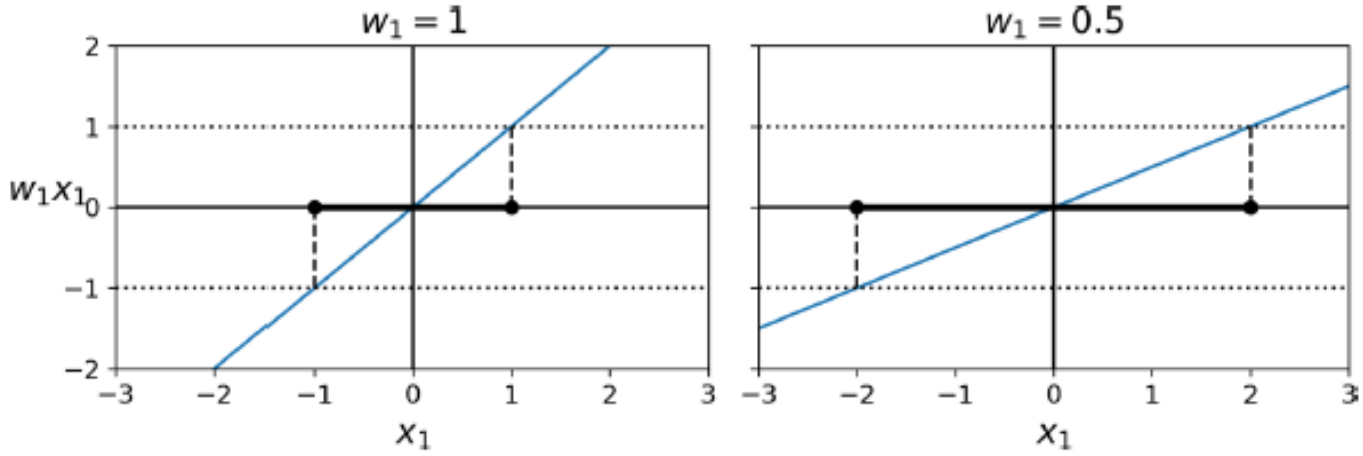
b: 편향 w: 가중치 벡터

5.4.2 목적 함수

선형 SVM 작동 원리: 목적 함수

마진을 크게 하기 위해 $\|\mathbf{w}\|$ 를 최소화

- 하드 마진: 모든 양성(음성) 샘플에 대한 결정 함수의 값이 1(-1)보다 큼(작음)
- 소프트 마진: 모든 샘플에 대한 결정 함수의 값이 지정된 값 이상 또는 이하



b: 편향 w: 가중치 벡터

5.4.2 목적 함수

하드 마진 선형 SVM 분류기의 목적 함수

- 목적 함수

$$\frac{1}{2}\mathbf{w}^T\mathbf{w}$$

- 조건

$$t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1$$

- 단, 다음이 성립:

$x^{(i)}$: i 번째 샘플
$t^{(i)}$: 양성 샘플($y^{(i)}$ =1)일 때 1, 음성 샘플($y^{(i)}$ =0)일 때 −1

b: 편향 w: 가중치 벡터

5.4.2 목적 함수

B: 편향 w: 가중치 벡터

소프트 마진 선형 SVM 분류기의 목적 함수

- 목적 함수

$$\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=0}^{m-1}\zeta^{(i)}$$

C: 아래 두 목표 사이의 트레이드오프를 조절하는 하이퍼파라미터
- 목표1: 슬랙 변수의 값을 작게 만들기
- 목표2: 마진을 크게 하기 위해 $\frac{1}{2}\mathbf{w}^T\mathbf{w}$ 값을 가능하면 작게 만들기

- 조건

$$t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \zeta^{(i)}$$

- 단, 다음이 성립:

$x^{(i)}$: i 번째 샘플
$t^{(i)}$: 양성 샘플($y^{(i)}$ =1)일 때 1, 음성 샘플($y^{(i)}$ =0)일 때 −1
$t^{(i)}(w^T+X^{(i)}+b) >= 1-\zeta^{(i)}$와 $\zeta^{(i)}>=0$: $\zeta^{(i)}$은 슬랙 변수로 i번째 샘플이 얼마나 마진을 위반할 지 정함

b: 편향 w: 가중치 벡터

5.4.3 콰드라틱 프로그래밍

선형 SVM 작동 원리: 2차 계획법(QP)

- 하드/소프트 마진 문제: 선형 제약조건이 있는 블록 2차 최적화 문제
- 2차 계획법(QP, quadratic programming) 문제
- 일반적인 문제 공식

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

$$if \quad A \cdot p \leq b$$

1) p는 n_p 차원의 벡터, (n_p =모델 파라미터 수)
2) H는 $n_p \times n_p$ 크기 행렬
3) f는 n_p 차원의 벡터
4) A는 $n_c \times n_p$ 크기 행렬(n_c = 제약 수)
5) b는 n_c 차원의 벡터

b: 편향 w: 가중치 벡터

5.4.4 쌍대문제

최적화 문제(optimization)

- 어떤 목적함수(objective function)의 함수 값을 최적화(최대화 또는 최소화)시키는 파라미터 조합을 찾는 것.

  최대화 문제: 목적함수가 이윤, 점수(score) 등인 경우

  최소화 문제: 목적함수가 비용(cost), 손실(loss), 에러(error) 등인 경우

쌍대 문제(dual problem)

- 최적화 문제를 푸는 과정에서, 원 문제를 깊게 관련된 다른 문제로 표현하는 것.
- 일반적으로 쌍대 문제의 해는 원 문제 해의 하한 값이지만, 특정 조건을 만족하면 원 문제와 똑같은 해를 제공함.

  특히, 원 문제에서 적용할 수 없는 '커널 트릭'을 가능하게 하기 때문에 중요함.

(핸즈온 머신러닝 교재 부록 C 챕터 참고)

5.4.4 쌍대문제

선형 SVM 분류의 최적화 문제

식 5-3: 하드 마진 선형 SVM 분류기 목적 함수

$$\underset{\mathbf{w},b}{\text{minimize}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$
$$\text{subject to} \quad t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 \quad \text{for } i = 1, 2, \ldots, m$$

마진을 크게 하기 위해
1/2w^Tw를 가능한 작게 만들어야 함 → 목적함수 최소화

위의 목적함수를 최소화하는 최적 파라미터(**w**, b)를 찾아야 함.

이 때, 특정 조건 (목적함수가 볼록 함수이고, 제약 조건이 연속 미분 가능하면서 볼록 함수이다.)
을 만족하기 때문에 **쌍대 문제**를 풀어서 목적함수의 해를 얻을 수 있다.

식 5-6 선형 SVM 목적 함수의 쌍대 형식

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

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

이 식을 최소화하는 α를 찾아
원 문제의 식을 최소화하는 **w**와 b를 찾을 수 있음.

식 5-7 쌍대 문제에서 구한 해로 원 문제의 해 계산하기

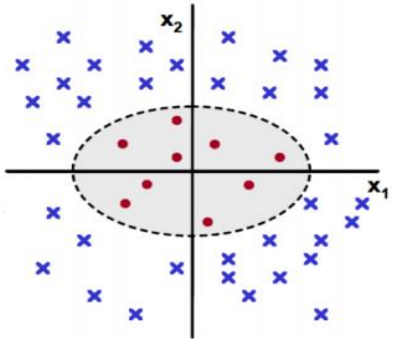
$$\hat{\mathbf{w}} = \sum_{i=1}^{m}\hat{\alpha}^{(i)}t^{(i)}\mathbf{x}^{(i)}$$

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

훈련 샘플 수가 특성 개수보다 작을 때, (m<n)
원 문제보다 쌍대 문제를 푸는 것이 더 빠름.

5.4.5 커널 SVM

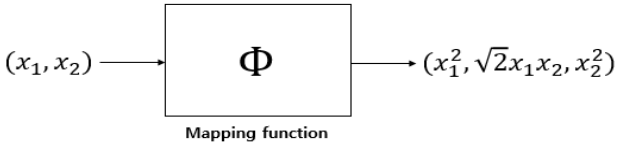
SVM을 사용할 때, 바로 선형적인 decision boundary로 분류하기 어려운 데이터를 높은 차원으로 매핑하여 선형으로 분리해줄 수 있음.

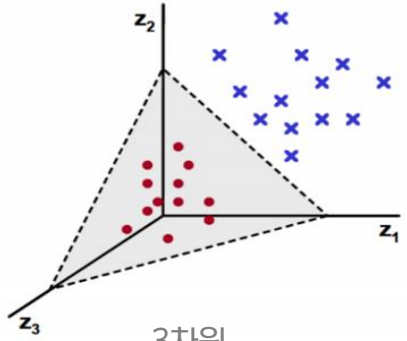


2차원

$$x = \{x_1, x_2\} \rightarrow z = \{x_1^2, \sqrt{2}x_1x_2, x_2^2\}$$

Mapping function





3차원

선형 SVM 쌍대 문제에 이를 적용하면,

$$\max L_D(\alpha_i) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j \longrightarrow \max L_D(\alpha_i) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \Phi(x_i)^T \Phi(x_j)$$

변환된 3차원 벡터의 내적 형태

이 때, 고차원 매핑과 내적까지의 결과를 연산량을 최소화해서 얻게 도와주는 것이 커널 트릭!

5.4.5 커널 SVM

224쪽 예시: 2차 다항식 매핑

식 5-8: 2차 다항식 매핑

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

식 5-9: 2차 다항식 매핑을 위한 커널 트릭

$$\begin{aligned} \phi(\mathbf{a})^T \phi(\mathbf{b}) &= \begin{pmatrix} {a_1}^2 \\ \sqrt{2}\, a_1 a_2 \\ {a_2}^2 \end{pmatrix}^T \begin{pmatrix} {b_1}^2 \\ \sqrt{2}\, b_1 b_2 \\ {b_2}^2 \end{pmatrix} = {a_1}^2 {b_1}^2 + 2 a_1 b_1 a_2 b_2 + {a_2}^2 {b_2}^2 \\ &= (a_1 b_1 + a_2 b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^T \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 = (\mathbf{a}^T \mathbf{b})^2 \end{aligned}$$

원래 벡터 a, b를 매핑(변환)한 후 내적(점곱)한 것과 원래 벡터를 내적하여 제곱한 것이 같음
즉, 훈련 샘플을 변환할 필요 없음!

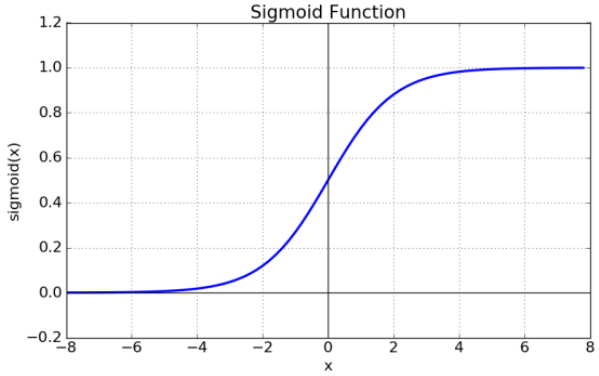
2차 다항식 커널: K(a,b)=(a^Tb)^2

5.4.5 커널 SVM

자주 쓰이는 커널의 종류

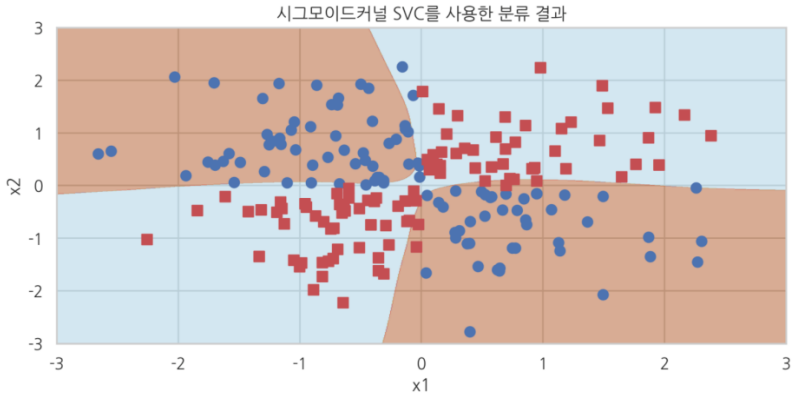
식 5-10: 일반적인 커널

$$\begin{aligned}
\text{선형:} \quad & K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T\mathbf{b} \\
\text{다항식:} \quad & K(\mathbf{a}, \mathbf{b}) = \left(\gamma\mathbf{a}^T\mathbf{b} + r\right)^d \\
\text{가우시안 RBF:} \quad & K(\mathbf{a}, \mathbf{b}) = \exp(-\gamma\|\mathbf{a} - \mathbf{b}\|^2) \\
\text{시그모이드:} \quad & K(\mathbf{a}, \mathbf{b}) = \tanh\left(\gamma\mathbf{a}^T\mathbf{b} + r\right)
\end{aligned}$$



추가 내용) 시그모이드 커널

```
svclassifier = SVC(kernel='sigmoid', degree=8)
svclassifier.fit(X_train, y_train)

y_pred = svclassifier.predict(X_test)
print('score : ',svclassifier.score(X_test,y_test))
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```



5.4.5 커널 SVM

226쪽: 선형 SVM 분류기의 쌍대 문제에 커널 트릭 적용하기

식 5-2: 선형 SVM 분류기의 예측

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

식 5-7 쌍대 문제에서 구한 해로 원 문제의 해 계산하기

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

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

식 5-11 커널 SVM으로 예측하기

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

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

$$= \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)}>0}}^{m} \hat{\alpha}^{(i)} t^{(i)} K\left(\mathbf{x}^{(i)}, \mathbf{x}^{(n)}\right) + \hat{b}$$

식 5-12 커널 트릭을 사용한 편향 계산

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

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

w를 구하지 않아도
결정 함수에 w식을 대입함으로써
예측을 만들 수 있음.

5.4.6 온라인 SVM

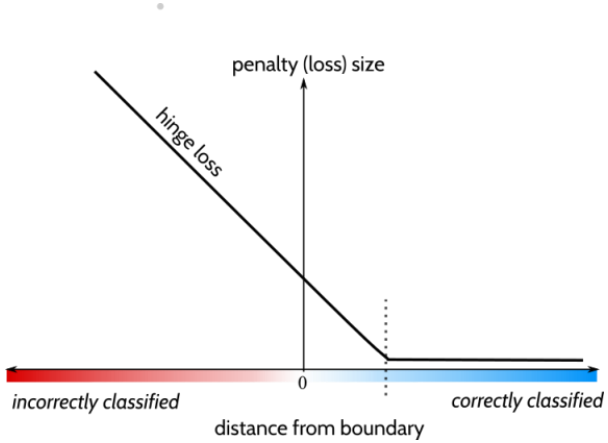
온라인 학습: 새로운 샘플이 생겼을 때마다 점진적으로 학습하는 것.

온라인 SVM 분류기
방법 1) 비용함수 최소화 시키기 위한 경사 하강법 사용하기 (ex. SGDClassifier)

식 5-13: 선형 SVM 분류기의 비용 함수

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

힌지 손실(hinge loss): max(0,1-t)



방법 2) 온라인 커널 SVM 구현하기
매트랩 또는 C++로 구현 가능
대규모의 비선형 문제라면, 신경망 알고리즘 쓰는 것이 추천됨

감사합니다

Q&A