서포트 벡터 머신(SVM)

19기 분석 신은빈

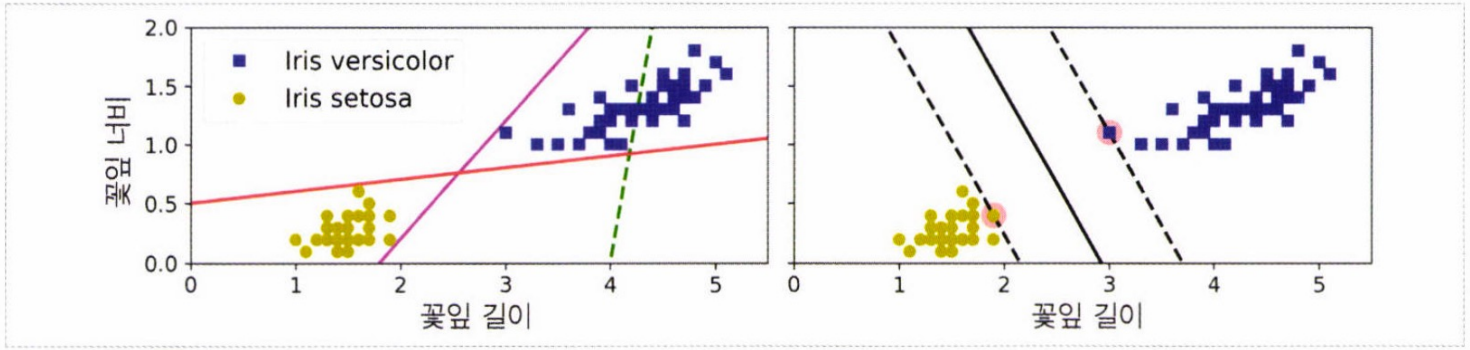
목차

- 선형 SVM 분류
- 비선형 SVM 분류
- SVM 회귀
- SVM 이론

선형 SVM 분류

- SVM이란?
- 선형, 비선형 분류, 회귀, 이상치 탐색에도 사용 가능한 모델
- 라지 마진 분류 라고 부름
- 마진(Margin)이란? 결정 경계와 서포트 벡터 사이의 거리, 즉 실선으로부터 점선까지의 거리, 엡실론
- 우리의 목표 : 마진을 최대화할 수 있는 결정 경계 찾기

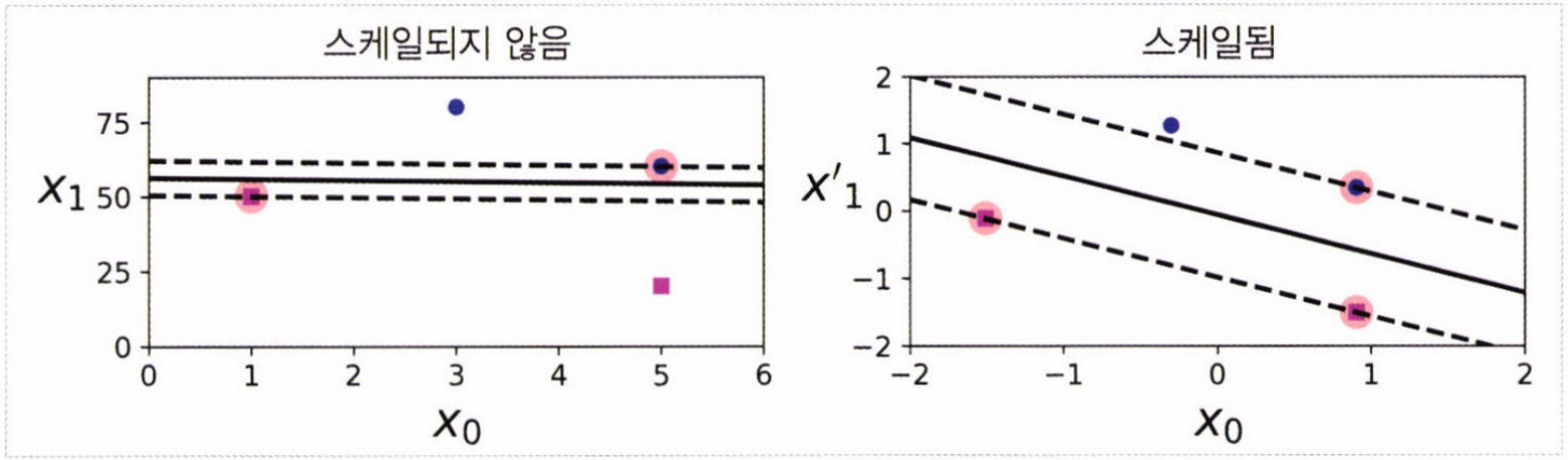
- 서포트 벡터(Support vector)란?
- 결정 경계와 가장 가까운 데이터
- 분류를 할 때 이 서포트 벡터만 가지고 사용, 데이터 전체를 사용하지 않음 -> 시간이 단축되는 장점



선형 SVM 분류

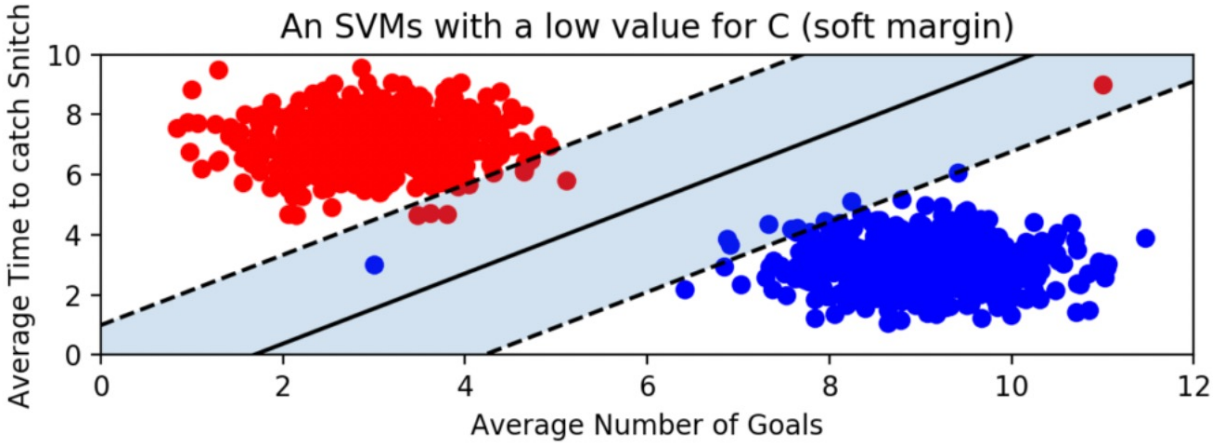
- 스케일링(Scaling)

- SVM은 특성의 스케일에 민감하기 때문에 StandardScaler, MinMaxScaler 등을 이용해서 조정해줘야한다.

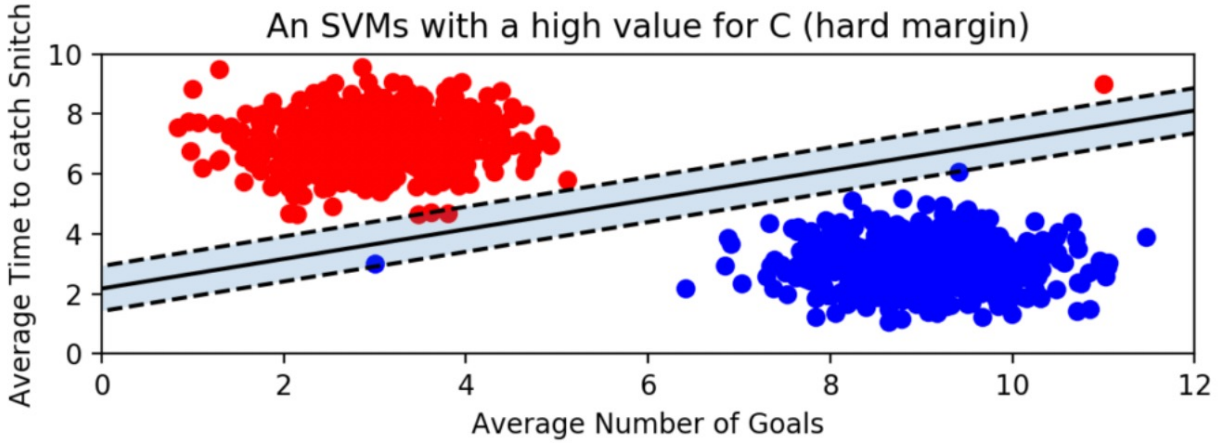


선형 SVM 분류

- 소프트 마진 분류
  - 이상치에 덜 민감
  - 언더피팅 문제 발생 가능



- 하드 마진 분류
  - 이상치에 민감
  - 데이터가 선형적으로 구분 될 수 있어야 작동 가능
  - 오버피팅 문제 발생 가능

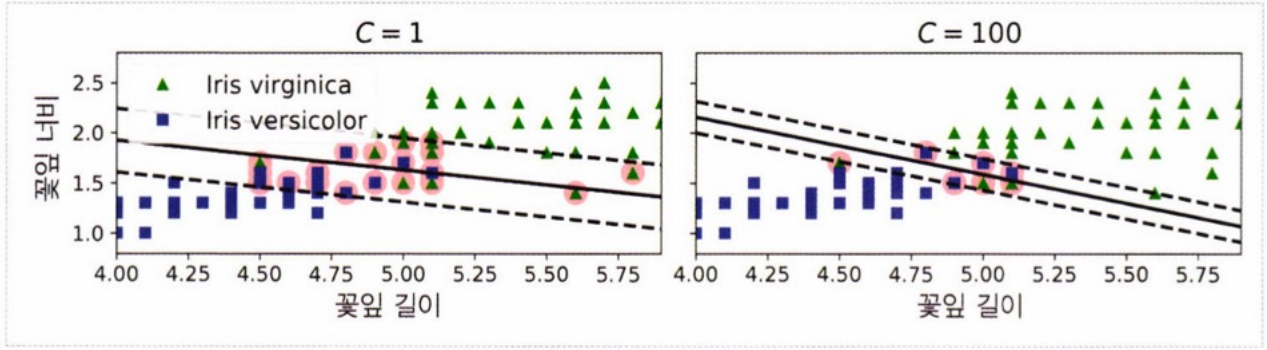


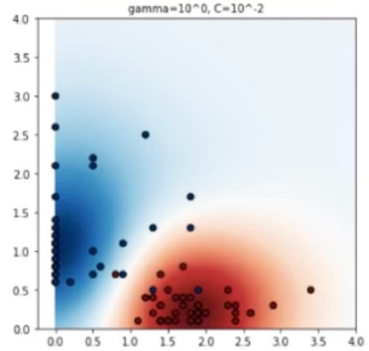
선형 SVM 분류

SVM의 하이퍼파라미터
① C(cost)
② kernel
③ gamma

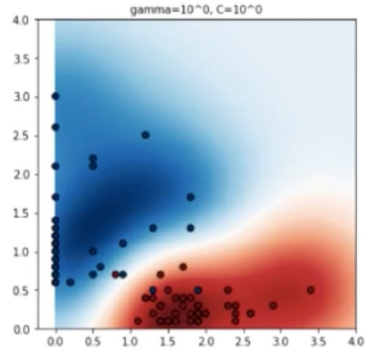
- SVM의 하이퍼파라미터 ①: Cost

- C가 작으면 직선모양의 결정 경계가 나타나고, 에러를 많이 허용합니다.(소프트 마진)
- C가 크면 구불구불한 모양의 결정 경계가 나타나고, 에러를 허용하지 않는다.(하드 마진)
- SVM 모델이 과적합이 되었다면 C를 감소시켜 모델을 규제할 수 있다.





cost= 0.01



cost= 1

선형 SVM 분류

- SVM 예시 코드

```
[2]  import numpy as np
     from sklearn import datasets
     from sklearn.pipeline import Pipeline
     from sklearn.preprocessing import StandardScaler
     from sklearn.svm import LinearSVC
```

```
[4]  iris = datasets.load_iris()
     X = iris['data'][:, (2,3)]
     y = (iris['target'] == 2).astype(np.float64)
```

```
[7]  svm_clf = Pipeline([
                        ("scaler", StandardScaler()),
                        ("liner_svc", LinearSVC(C = 1, loss= "hinge")),
     ])
     svm_clf.fit(X,y)
```

```
     Pipeline(steps=[('scaler', StandardScaler()),
                     ('liner_svc', LinearSVC(C=1, loss='hinge'))])
```

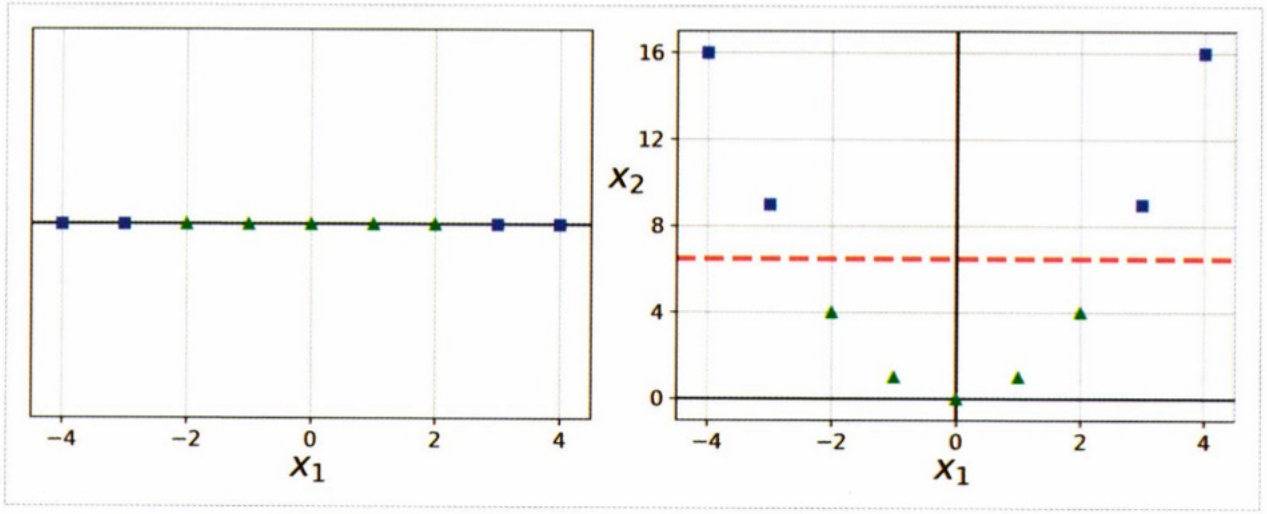
```
[8]  svm_clf.predict([[5.5, 1.7]]) #예측
```

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

- iris dataset을 이용하여 SVM 모델 적용해보기
- iris-virginia 품종이 맞으면 1, 아니면 0

비선형 SVM 분류

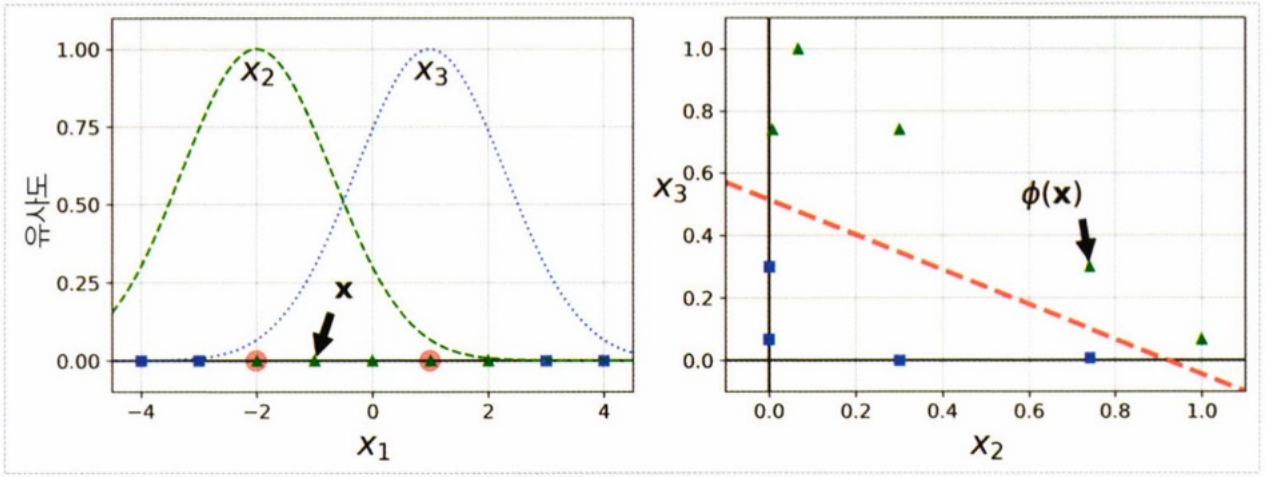
- 비선형 데이터셋을 다루는 방법 ① 다항 특성 추가
- 아래 그림은 $X_2 = (X_1)^2$을 추가하여 만든 2차원 데이터셋



비선형 SVM 분류

- 비선형 데이터셋을 다루는 방법 ② 유사도함수
- 각 샘플이 특정 랜드마크와 얼마나 닮았는지 측정하는 함수
- 유사도 함수로 계산한 특성을 추가한다.
- 유사도 함수는 '방사 기저 함수(RBF)'를 이용

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

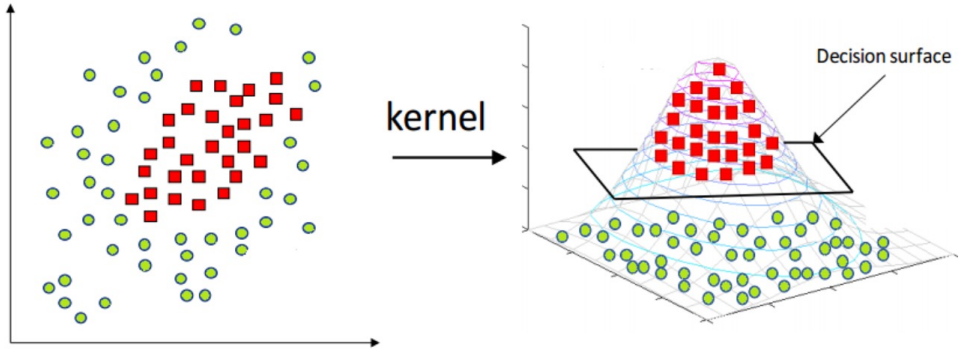


비선형 SVM 분류

• 커널 트릭

-실제로 데이터를 고차원으로 보내진 않지만 보낸 것과 동일한 효과를 줘서 매우 빠른 속도로 결정 경계선을 찾는 방법

- 커널은 원래 가지고 있는 데이터를 더 높은 차원의 데이터로 변환한다.



-> x,y 2차원 평면에서 분류를 하지 못해 z축을 추가하면 분류가 가능하다.

SVM
① 선형 분류 – LinearSVC / SVC(kernel = “linear”, C = 1) / SGDClassifier(loss="hinge", alpha=1/(m*C))
② 비선형 분류 – SVC(kernel = “rbf”) / SVC(kernel = “poly”)
③ 회귀 – LinearSVR

비선형 SVM 분류

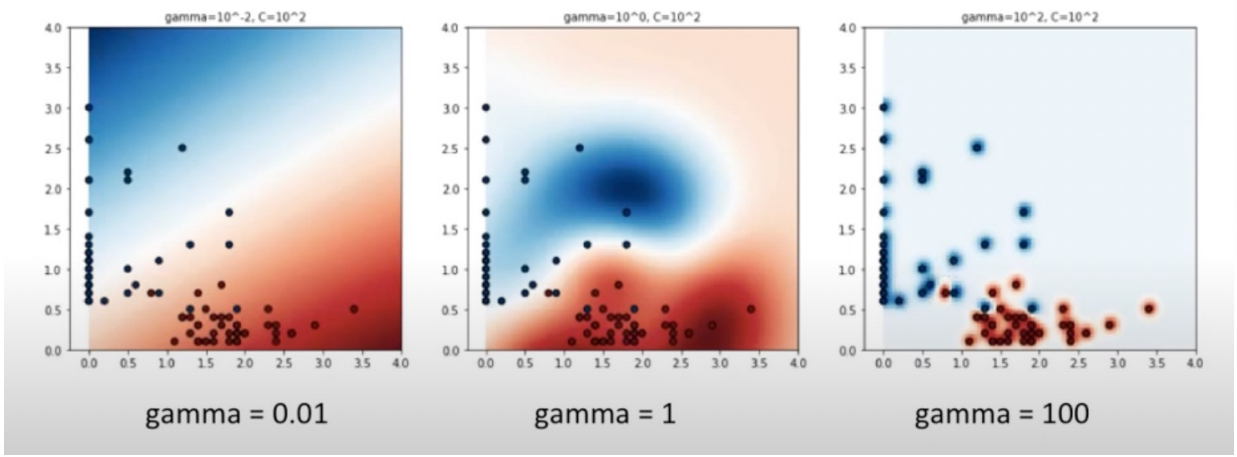
SVM의 하이퍼파라미터
① C(cost)
② kernel
③ gamma

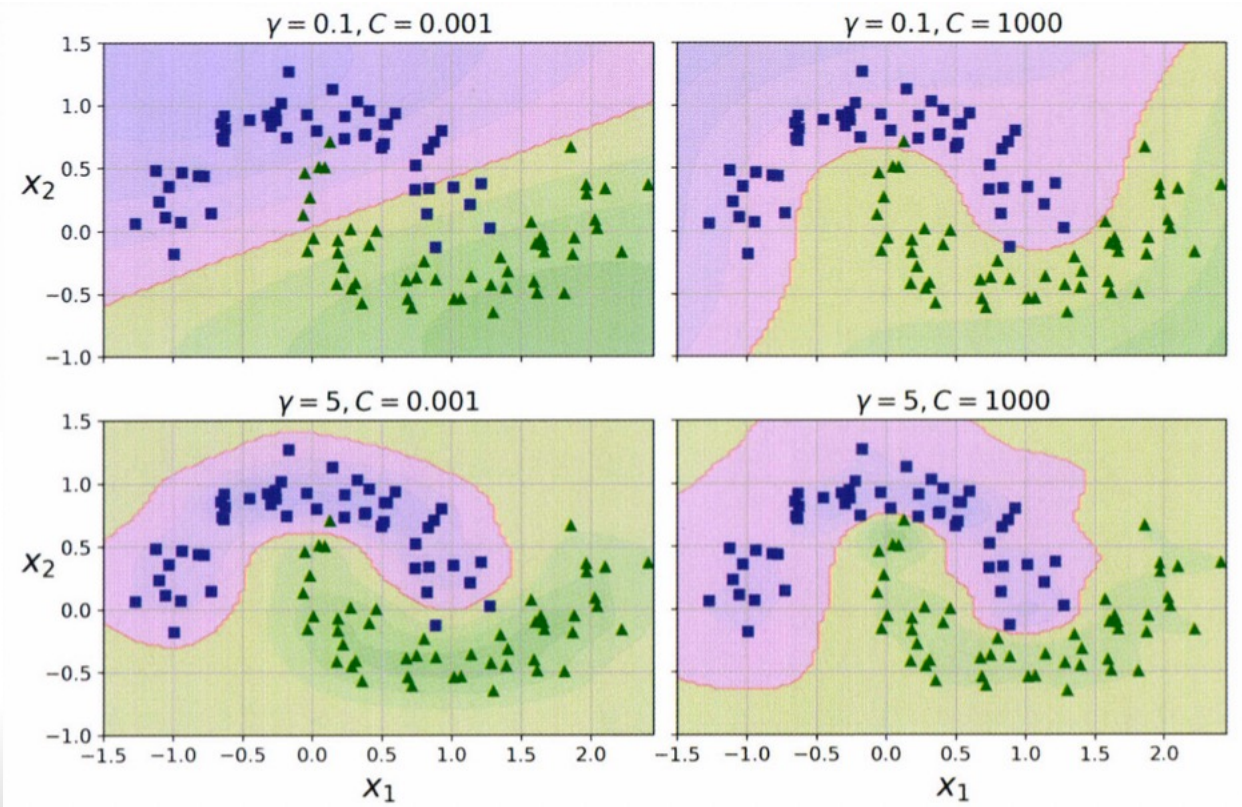
- SVM 하이퍼파라미터 ② kernel
- Kernel = 'rbf'를 추가하여 비선형 데이터셋을 사용가능하다.
- SVM 하이퍼파라미터 ③ gamma
- 감마 값이 증가할수록 과대적합

```
from sklearn.svm import SVC

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

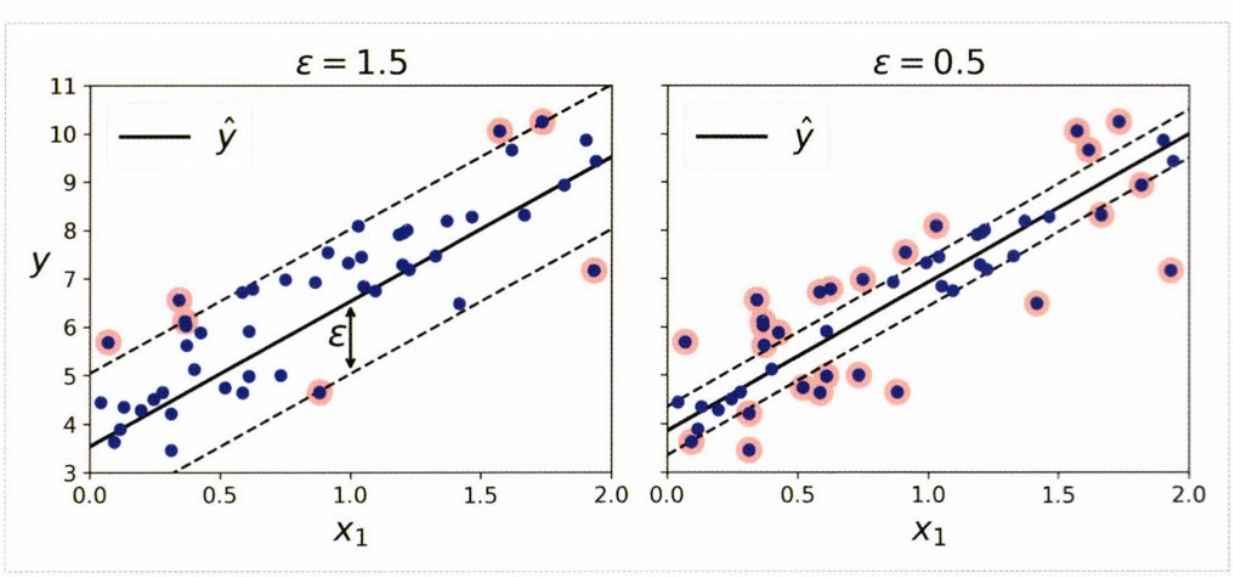
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('svm_clf', SVC(C=0.001, gamma=5))])
```





SVM 회귀

- SVM 회귀의 목표?
- 도로(마진) 안에 가능한 많은 샘플이 들어가도록 학습하는것, 분류와 반대
- 마진, 엡실론값이 작으면 도로 밖에 더 많은 데이터가 들어감



SVM 회귀

- 실습

```
import numpy as np
from sklearn import datasets
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, explained_variance_score
from sklearn.utils import shuffle

data = datasets.load_boston()

X, y = shuffle(data.data, data.target, random_state = 7)

num_training = int(0.8 * len(X))
X_train, y_train = X[:num_training], y[:num_training]
X_test, y_test = X[num_training:], y[num_training:]

sv_regressor = SVR(kernel='linear', C=1.0, epsilon=0.1)

sv_regressor.fit(X_train, y_train)

y_pred = sv_regressor.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
print(mse)
```

- 회귀에서는 SVC 대신 Regressor 를 의미하는 SVR을 사용하면 됩니다.

SVM 이론

- 결정함수

- 선형SVM에서는 결정함수 $w^T * \mathrm{x} + \mathrm{b}$를 이용함

- SVM에서 풀고자 하는 문제는 아래의 그림에서 마진(margin)을 가장 크게하는 w를 찾는 것

$$\max\ Margin = \max \frac{2}{\|w\|_2} \Leftrightarrow \min \frac{1}{2}\|w\|_2$$

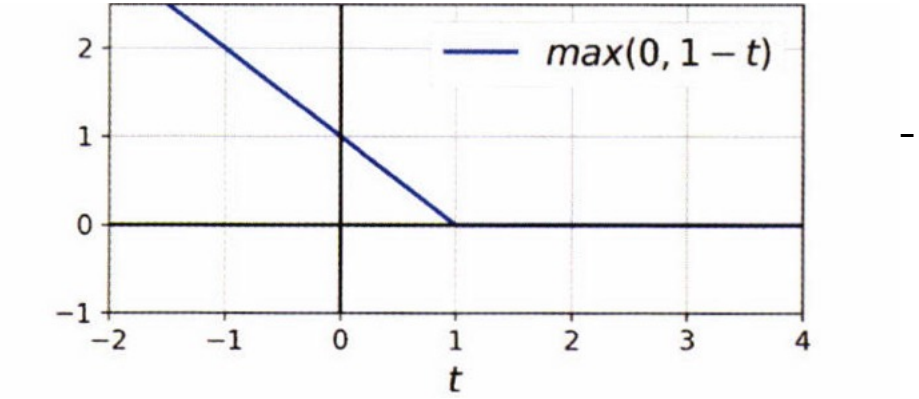
SVM 이론

- 목적함수(손실함수)
- C와 관련한 식은 조금의 오차를 허용한다는 slack variables. 여유변수

$$\min \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\zeta_i$$

$$\zeta_i = \max\left(0, 1 - t_i\left(\mathbf{w}^T\mathbf{x}_i + b\right)\right)$$

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



->max(0, 1-t)의 형태를 hinge loss라고 한다.

감사합니다 ☺