Support Vector Machines -
Linear & Nonlinear SVM Classification, SVM Regression

Samkeun Kim <skim@hknu.ac.kr>

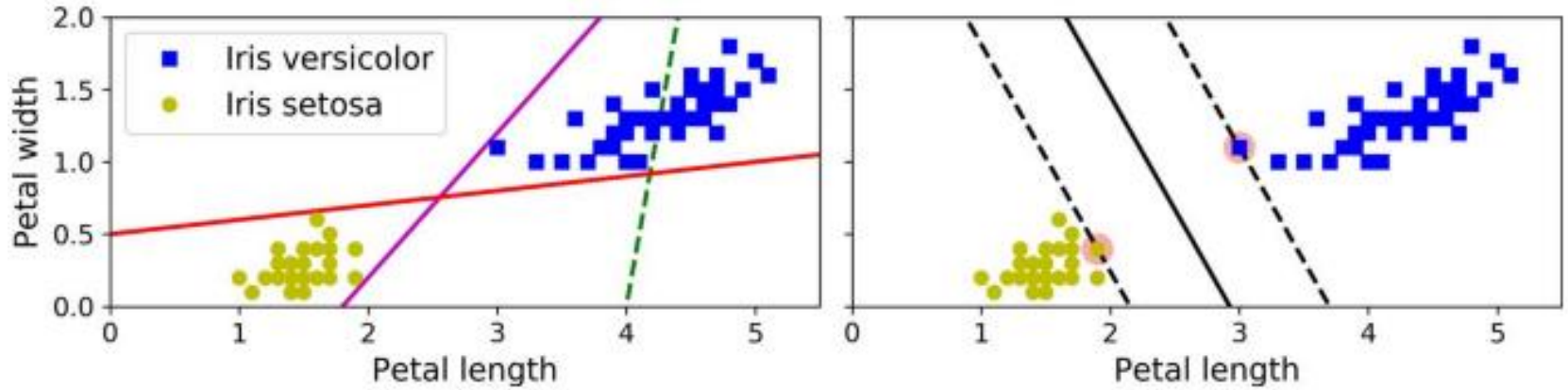
http://cyber.hankyong.ac.kr

국립한경대학교
HANKYONG NATIONAL UNIV.

Support Vector Machine (SVM)

- Linear or nonlinear classification, regression, and outlier detection을 수행할 수 있는 강력한 ML 모델
- 가장 인기있는 모델들 중의 하나
- 특히 복잡하지만 소중규모 사이즈의 데이터셋에 적합

Linear SVM Classification



왼쪽 그림에서 3개의 decision boundary:

- 녹색 점선 => 전혀 클래스를 구분 못함
- 2개의 실선 => 완벽하게 구분 함
- 문제점: 인스턴스와 너무 가깝다

  ⇒ 새 인스턴스에 대해 잘 구분 못할 것으로 예상

오른쪽 그림에서:

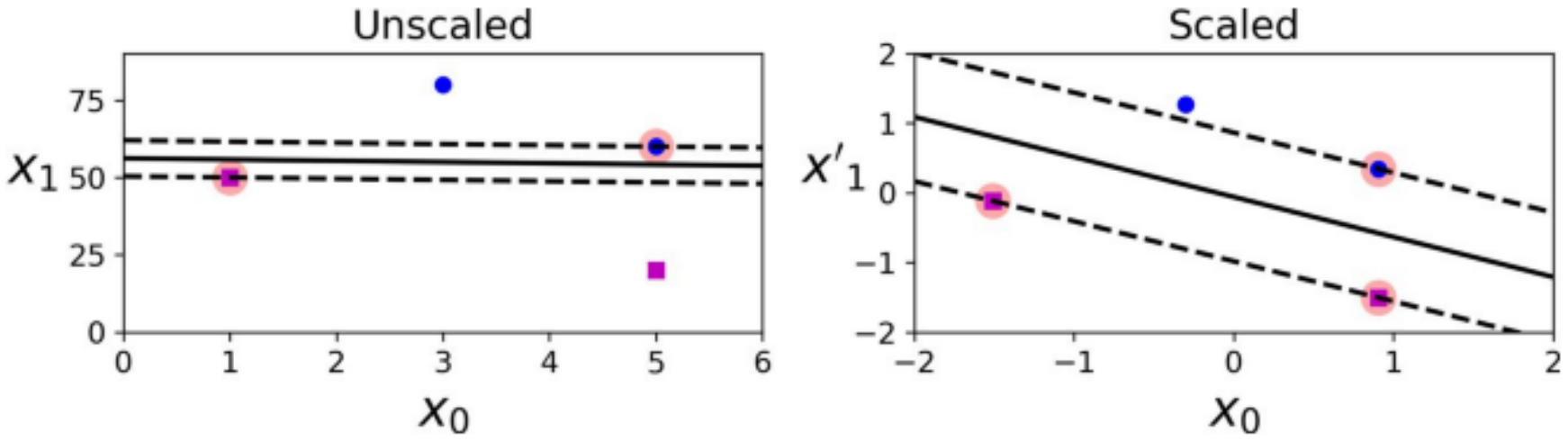
- 실선 => SVM classifier의 decision boundary
- 두 클래스를 정확하게 구분
- + 가장 가까운 인스턴스로부터 가급적 멀리 떨어져 있음

  ⇒ SVM classifier를 가장 넓은 도로(평행한 두 점선)가 되도록 학습시키는 것으로 간주

  ⇒ "*large margin classification*"이라 함

"off the street"에 더 많은 인스턴스 추가 => decision boundary에 전혀 영향을 미치지 않는다.

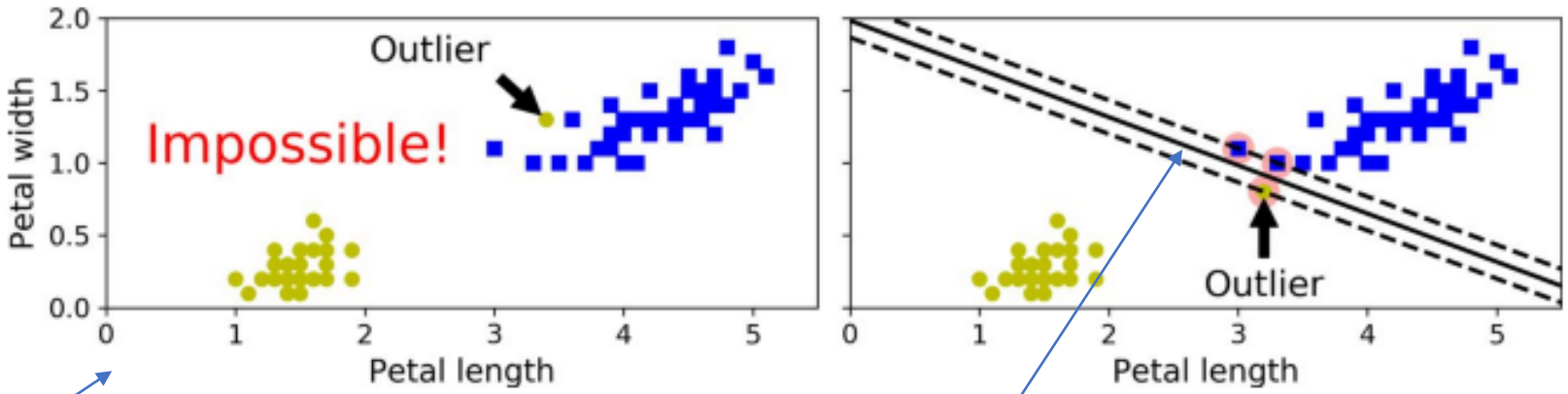
- 오히려 도로의 가장자리에 위치한 인스턴스들에 의해 결정된다.
- 이런 인스턴스들을 "**support vectors**"라 한다. (아래 그림에서 원으로 표시된 것)
- SVM은 입력 특성 스케일에 민감 (오른쪽 그림 => Scikit-Learn의 `StandardScaler`를 사용한 Scaling)



Soft Margin Classification

Hard margin classification

- 모든 인스턴스가 올바른 쪽의 "off the street"에 있어야 한다고 강요하는 분류법
- 두 가지 주요 이슈:
  1. 오로지 선형적으로 분리할 수 있을 때만 동작
  2. Outlier에 민감 => 제대로 일반화 할 수 없음



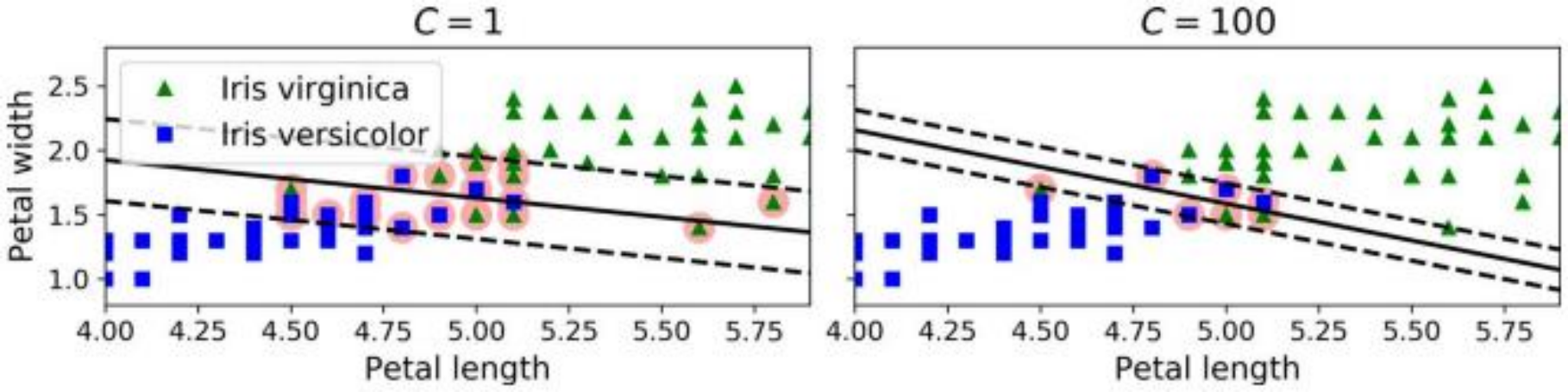
Hard margin을 찾을 수 없다.

Decision boundary가 크게 달라졌다.

Soft Margin Classification

보다 유연한 모델 필요: "**soft margin classification**"

- 도로는 가급적 넓게 + "margin violations"(도로에 있거나 반대쪽 측에 있는 인스턴스)를 제한시킴
- Scikit-Learn 이용 SVM 모델 생성 시: 수많은 hyperparameters 지정 필요
  - ✓ Hyperparameter *c* 를 이용해서 "margin violations" 조정 가능
  - ✓ 낮은 *c* 값 (왼쪽 그림) vs. 높은 *c* 값 (오른쪽 그림)
  - ✓ Margin violations are bad!! => 가급적 적게 가지는 것이 좋음. 그러나 왼쪽 그림처럼 많은 "margin violations"를 갖지만 오히려 일반화 성능은 더 좋을 수 있다.



Tip: Overfitting? => c 를 줄여보라!

Iris virginica 탐색을 위해 Iris dataset 로딩, 입력 특성 스케일링, 선형 SVM 모델 학습:

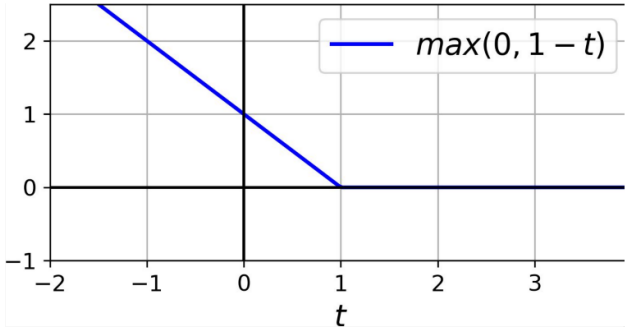
- LinearSVC with c=1, hinge 함수 사용 (앞 페이지 왼쪽 그림)

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

iris = datasets.load_iris()
X = iris["data"][:, (2, 3)]  # petal length, petal width
y = (iris["target"] == 2).astype(np.float64)  # Iris virginica

svm_clf = Pipeline([
        ("scaler", StandardScaler()),
        ("linear_svc", LinearSVC(C=1, loss="hinge")),
    ])

svm_clf.fit(X, y)
```



<Hinge loss function>

- 모델을 이용하여 예측 수행

```
>>> svm_clf.predict([[5.5, 1.7]])
array([1.])
```

NOTE

SVM classifier는 Logistic Regression classifier 와 달리 각 클래스에 대한 확률을 출력하지 않는다!

LinearSVC 클래스 대신 linear kernel을 갖는 SVC 클래스 사용 가능:

- SVC 모델 생성: **`SVC(kernel="linear", C=1)`**

또 다른 방법으로 SGDClassifier 사용: **`SGDClassifier(loss="hinge", alpha=1/(m*C))`**

- Linear SVM classifier를 학습시키기 위해 Stochastic Gradient Descent 적용

  ⇒ LinearSVC 클래스 만큼 빠르게 수렴하진 않지만 온라인 분류 작업이나 메모리에 로딩하지 못하는 대규모 데이터셋에 유용

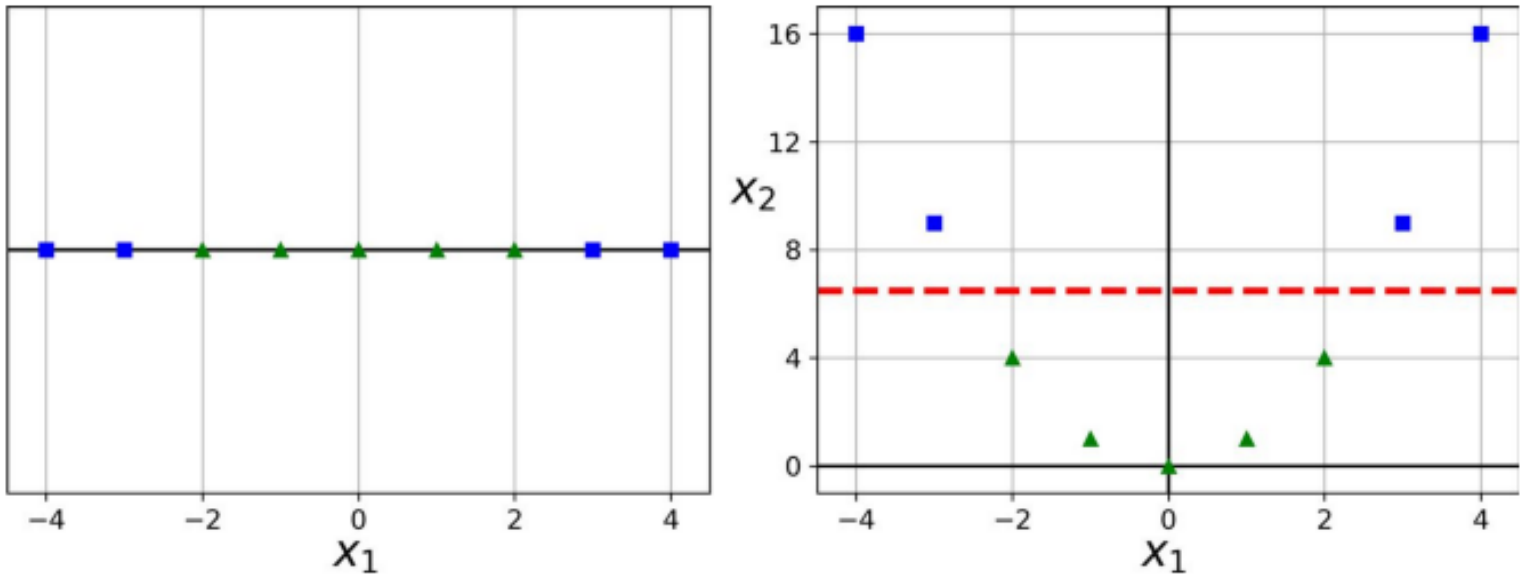
Nonlinear SVM Classification

현실 세계 => 많은 비선형 데이터셋들이 존재

비선형 데이터셋을 다루기 위한 방법 필요 => Polynomial 특성 등을 추가 => 선형으로 구분 가능하게 될 수도!!

왼쪽 그림: 단 하나의 입력 특성 x_1만을 갖는 단순한 데이터셋

⇒ 두 번째 특성 $x_2 = (x_1)^2$을 추가하면 결과 데이터셋은 완벽하게 선형적으로 분리 가능한 상태가 된다!!



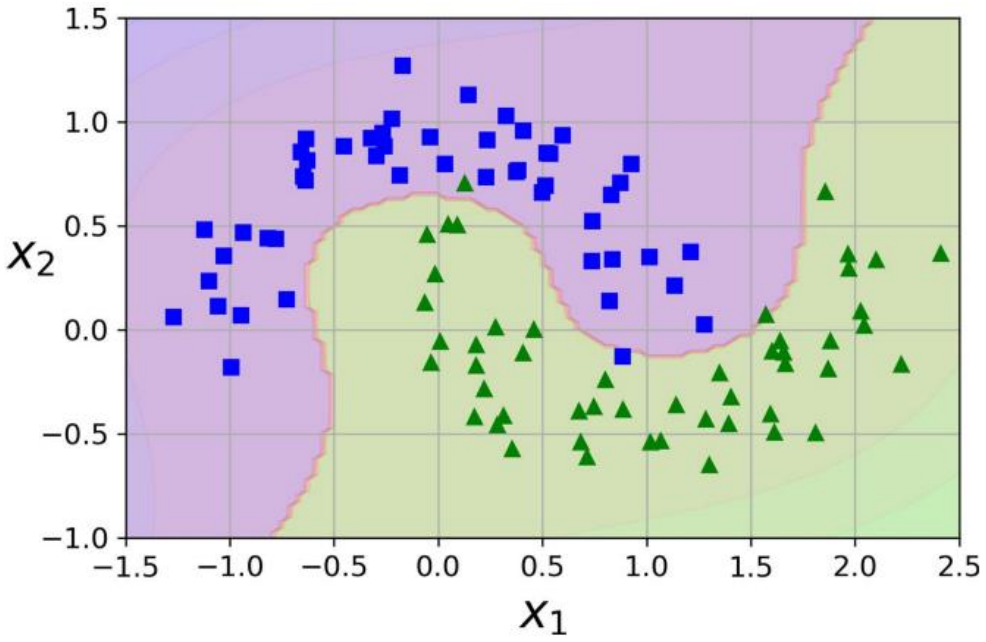
Scikit-Learn으로 구현

- Moons dataset (번갈아 가며 반원 모양의 형태를 갖는 데이터셋) 테스트:

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

polynomial_svm_clf.fit(X, y)
```



Linear SVM classifier using polynomial features

Polynomial Kernel

Polynomial 특성 추가 => (SVM에서만이 아니라) 모든 종류의 ML 알고리즘에서도 잘 동작하게 할 수 있다!

- 저차항만 가지고서는 복잡한 데이터셋을 다룰 수 없음
- 고차항을 추가하면 거대한 개수의 특성이 추가되어 모델을 너무 느리게 만듦

다행히도, SVM 사용 시 거의 기적같은 수학적 기법 적용 가능: ***kernel trick***

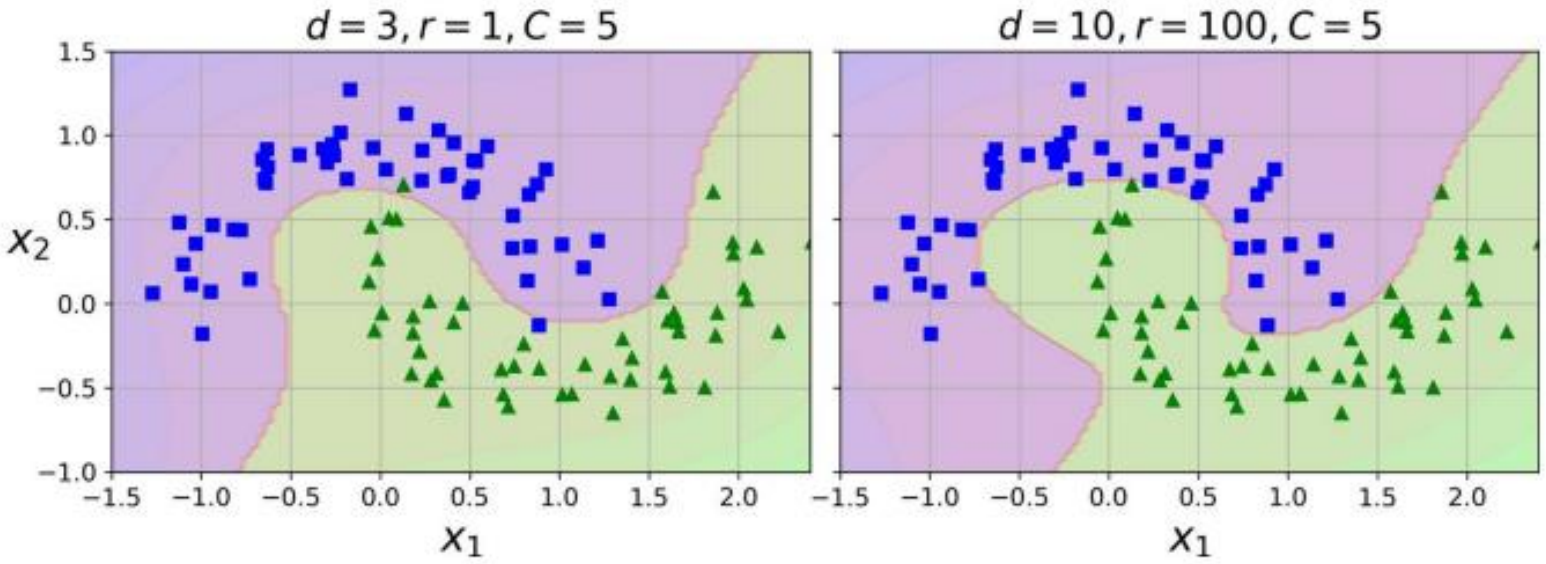
- Kernel trick => polynomial 특성을 추가하지 않고도 매우 큰 고차항을 추가한 것과 같은 효과를 얻도록 해 줌
- 실제 어떤 특성도 추가하지 않기 때문에 특성 개수로 인한 폭주 현상 없음
- SVC 클래스를 이용하여 구현 됨

Polynomial Kernel

3차 polynomial 커널을 이용한 SVM classifier (moons dataset):

```
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)
```

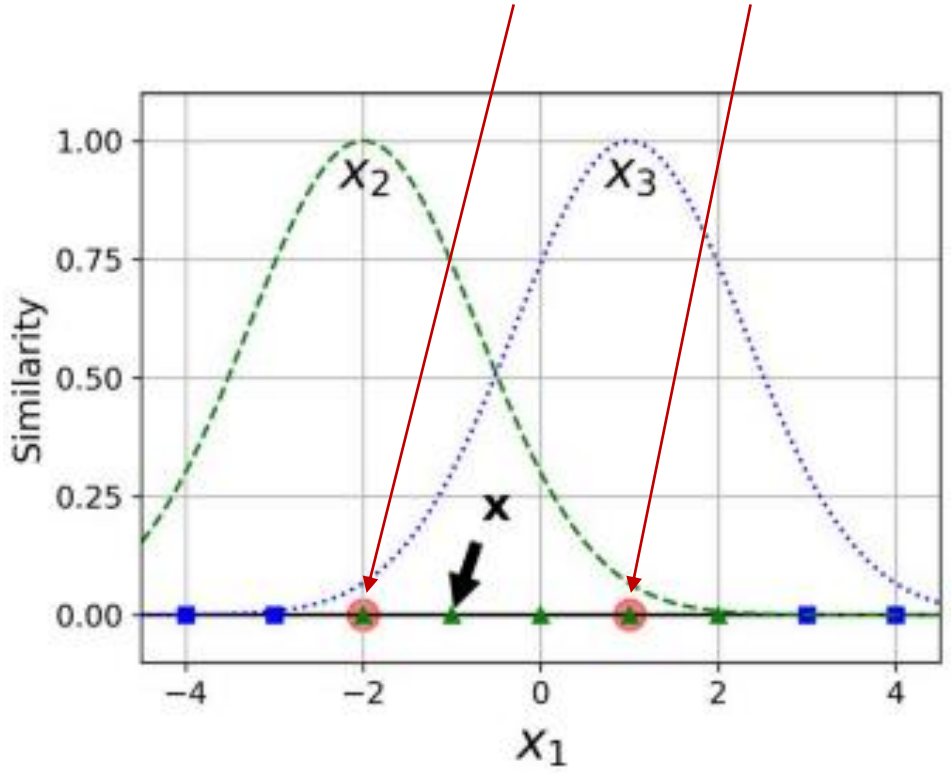
- 오른쪽: 10차 polynomial 커널을 이용한 SVM classifier, Overfitting? => 차수를 낮추면 된다.



Similarity Features

비선형 문제를 다루기 위한 또 다른 기법 => **similarity 함수**를 이용하여 계산된 특성 추가

- Similarity 함수 => 각 인스턴스가 특정 landmark를 얼마나 많이 닮았는가를 측정
- 예를 들어, 1D 데이터셋에 2개의 landmark($x_1 = -2\ and\ x_1 = 1$)를 추가하자.



Similarity 함수 정의:

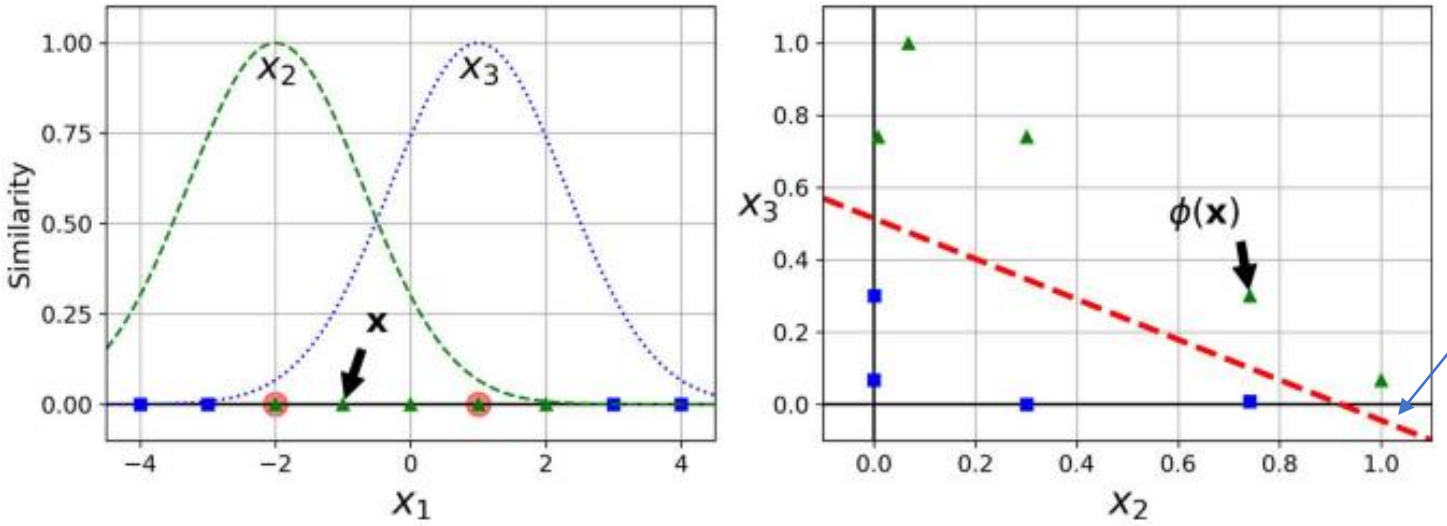
- $\gamma = 0.3$을 갖는 Gausian Basis Function (RBF)

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

Landmark: $x_1 = -2 \; and \; x_1 = 1$

- 0 (=landmark에서 멀리 떨어진)부터 1 (=landmark)까지 값이 변하는 벨 모양의 함수

  => 새로운 특성 계산 가능: $x_2 = \exp(-0.3 \times 1^2) \approx \mathbf{0.74}$, $x_3 = \exp(-0.3 \times 2^2) \approx \mathbf{0.30}$



Linearly separable!

Landmark 선택하는 방법

- 가장 단순한 방법: 데이터셋의 모든 인스턴스 위치를 landmark로!!
  - ✓ 장점: Training set을 선형적으로 분리 가능한 상태 기회 높여 줌
  - ✓ 단점: m개 인스턴스 & n개 특성으로 구성된 Training set

    => m개 인스턴스 & m개 특성으로 구성된 Training set으로 변형 (오리지널 특성 제거한 경우)
- Training set이 매우 클 경우 => 동일한 개수의 입력 특성을 갖게 됨

Gaussian RBF Kernel

Polynomial 특성 메소드와 마찬가지로 **Similarity 특성 방법** 역시 임의의 ML 알고리즘에서 유용

문제점: 모든 부가적인 특성을 계산하는데 비용 부담 큼 (특히 매우 큰 Training set인 경우)

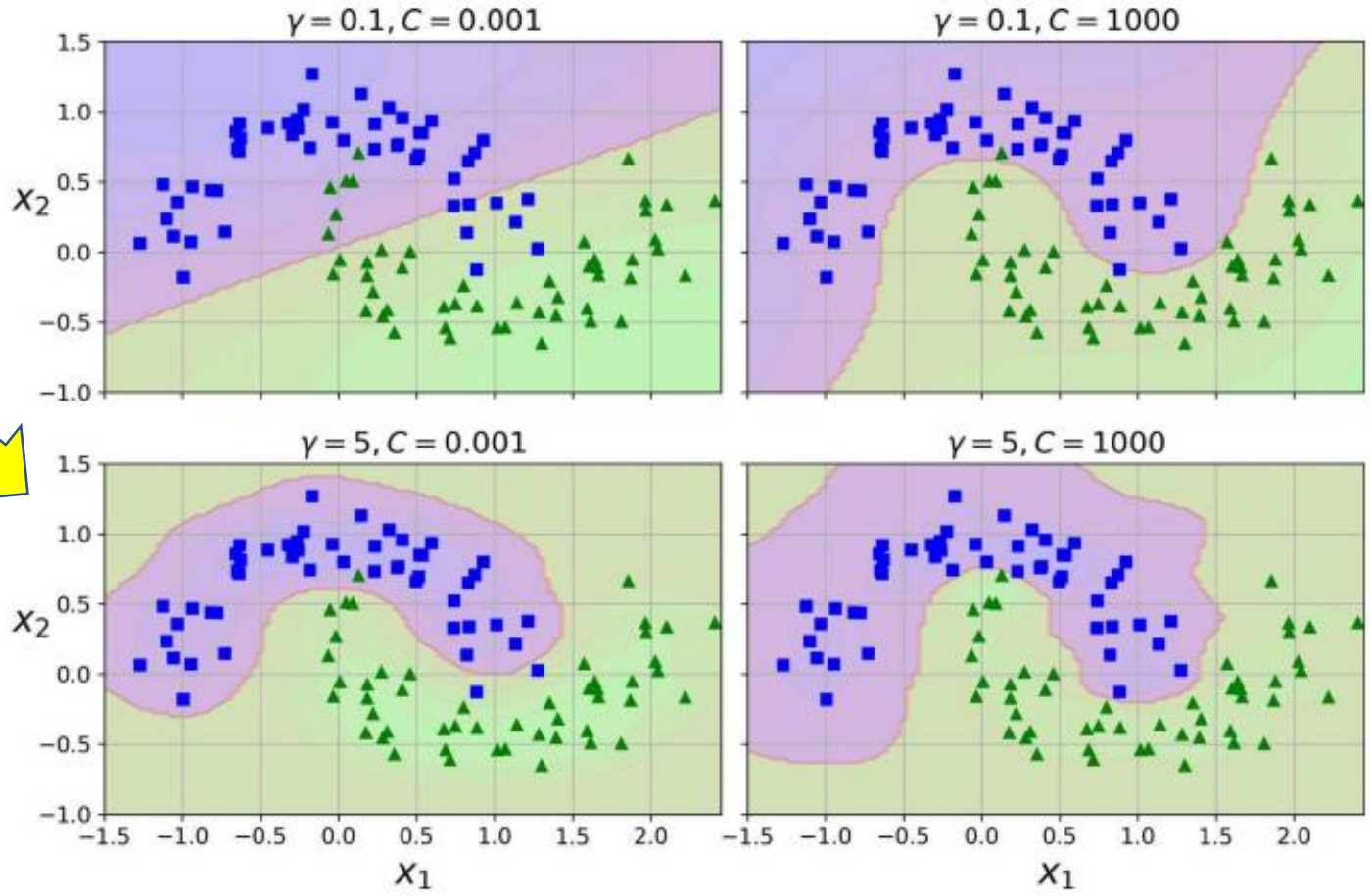
해결책: kernel trick 사용

=> SVM 매직에서 처럼 많은 similarity 특성을 추가한 것과 비슷한 결과를 얻게 해 줌

- SVC class with Gaussian RBF kernel:

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

서로 다른 hyperparameter γ 와 c 값으로 학습된 모델들:



γ : regularization hyperparameter 역할

Overfitting?

⇒ hyperparameter γ 값을 줄여라.

Underfitting?

⇒ hyperparameter γ 값을 높여라.

실습과제 8-1

본문에 나오는 전체 내용 PyCharm에서 실행하기

★ 반드시 결과 값 및 결과로 나온 모든 그래프에 대해 자세한 분석을 하시오.

참고: 제 08강 실습과제 #8 Support Vector Machines [1] - Linear & Nonlinear SVM Classification.pdf

실습과제 8-2

Train a LinearSVC on a linearly separable dataset. Then train an SVC and a SGDClassifier on the same dataset. See if you can get them to produce roughly the same model.

★ 반드시 결과 값 및 결과로 나온 모든 그래프에 대해 자세한 분석을 하시오.

참고: 제 08강 실습과제 8-2 LinearSVC Iris dataset.pdf

실습과제 8-3

Train an SVM classifier on the MNIST dataset. Since SVM classifiers are binary classifiers, you will need to use one-versus-the-rest to classify all 10 digits. You may want to tune the hyperparameters using small validation sets to speed up the process. What accuracy can you reach?

★ 반드시 결과 값 및 결과로 나온 모든 그래프에 대해 자세한 분석을 하시오.

참고: 제 08강 실습과제 8-3 SVM MNIST dataset.pdf

SVM Regression

SVM 알고리즘: 다재다능

- 선형/비선형 Classification 지원
- 선형/비선형 **Regression** 지원

SVM을 Regression 용으로 사용하려면 학습목표를 반대로 해야

- SVM Classification

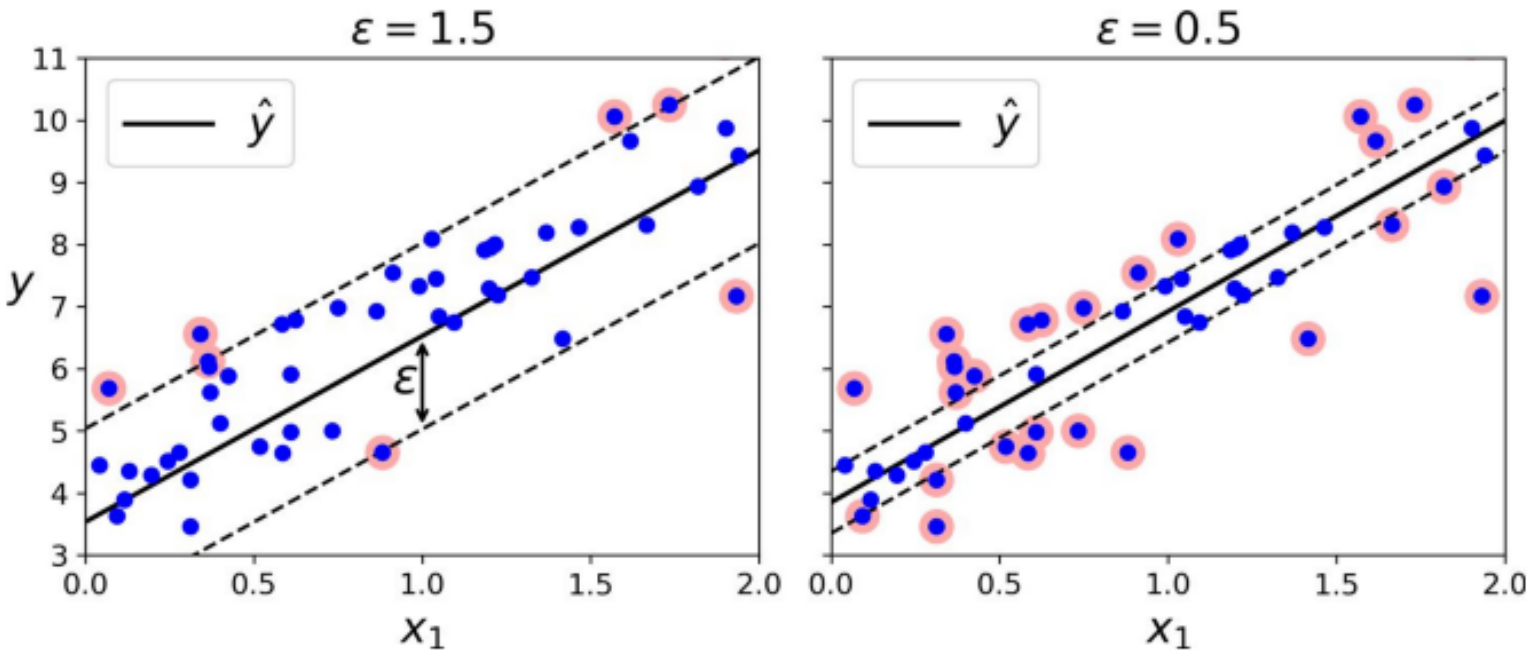
  ⇒ "margin violations" (도로 안의 인스턴스)를 제한하면서 두 클래스 사이에 가장 커다란 도로를 갖도록 학습

- SVM Regression

  ⇒ "margin violations" (도로 밖의 인스턴스)를 제한하면서 **도로 상에** 가급적 많은 인스턴스가 있도록 학습

두 개의 선형 SVM 회귀 모델 => 임의의 선형 데이터에 대해 학습:

- 하나는 큰 마진 (ϵ = 1.5)이고 다른 하나는 작은 마진 (ϵ = 0.5)
- Hyperparameter ϵ => 도로 폭 조절

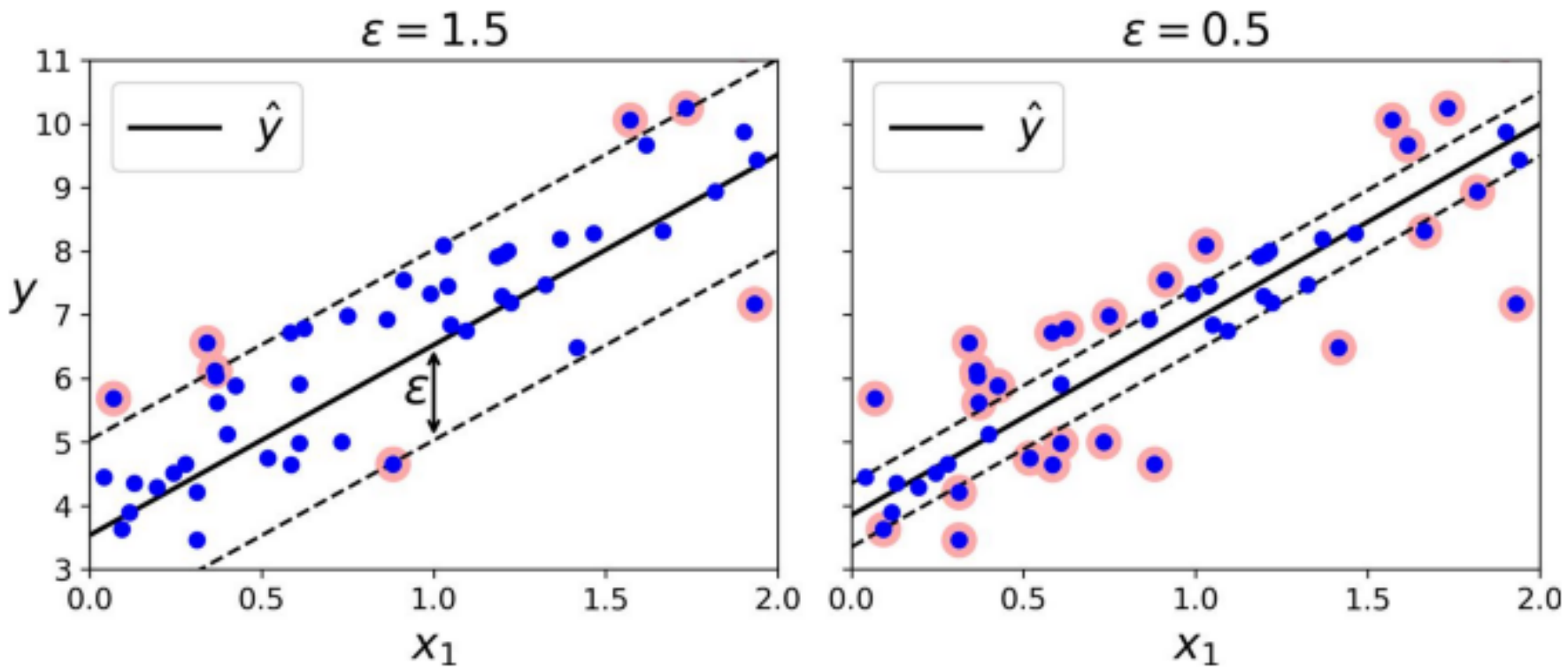


마진 안에 인스턴스를 더 추가한다 해도 모델의 예측에는 전혀 영향을 미치지 못한다!

Scikit-Learn의 LinearSVR 클래스를 이용하여 선형 SVM Regression 수행 가능

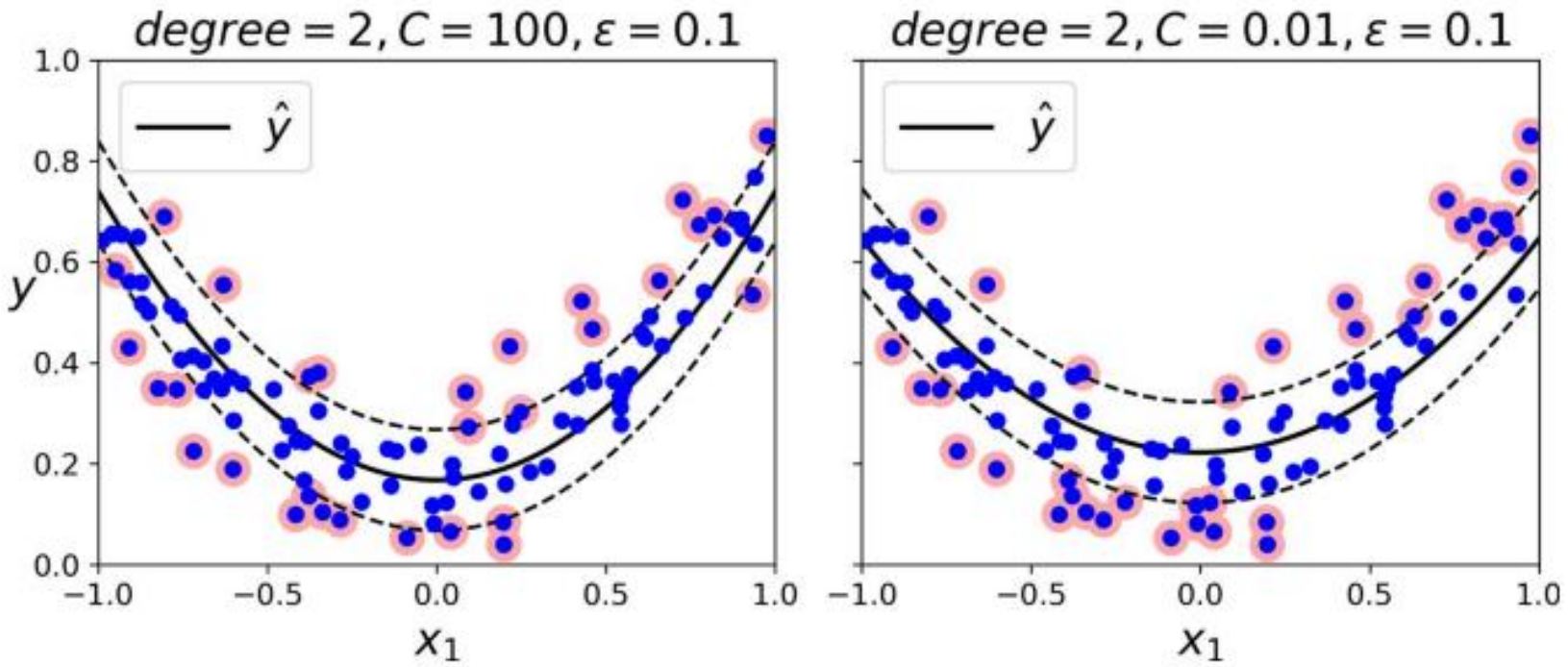
```
from sklearn.svm import LinearSVR

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



비선형 Regression을 다루기 위해 커널 기반의 SVM 모델을 사용할 수 있다.

- 2차 polynomial 커널을 이용한 SVM Regression



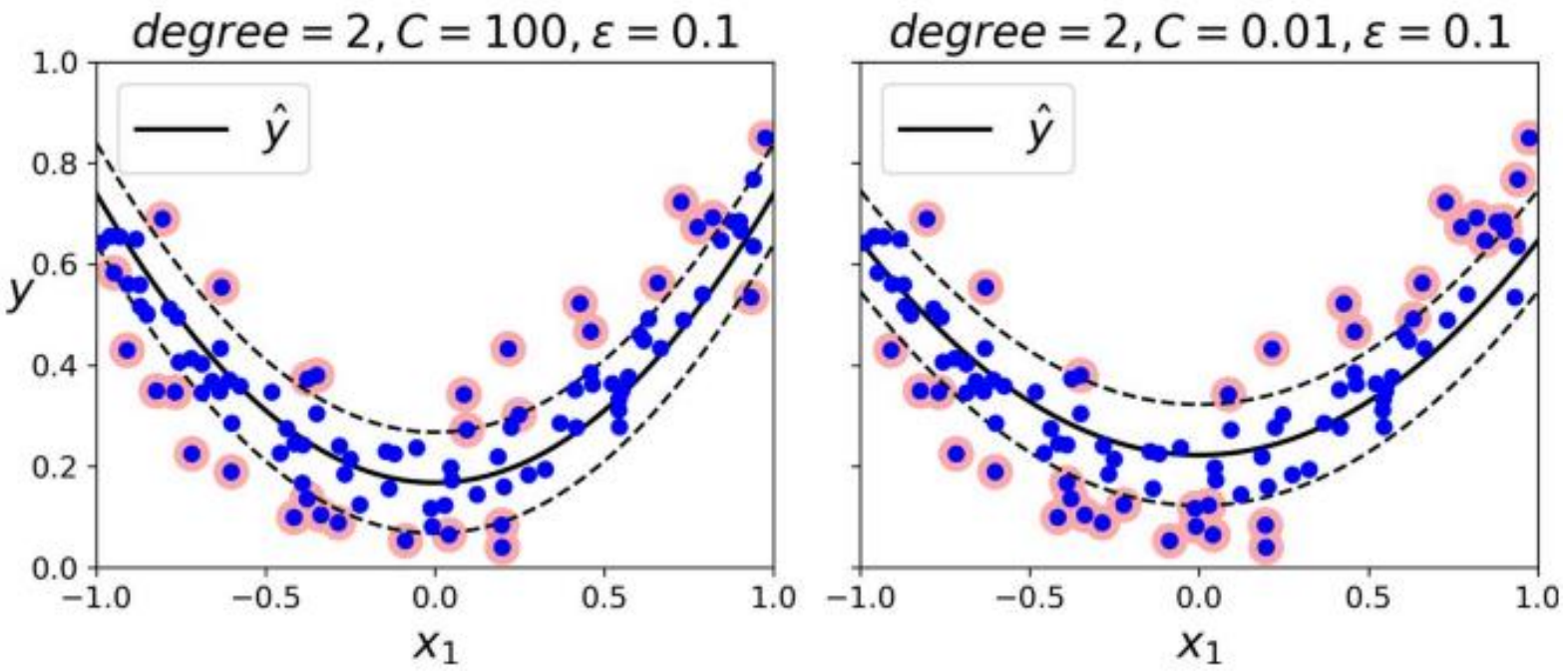
Large c 값

Small c 값

Scikit-Learn의 SVR를 사용하여 아래 왼쪽 그림에 나타낸 모델 생성:

```
from sklearn.svm import SVR

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



실습과제 8-4

본문에 나오는 전체 내용 PyCharm에서 실행하기

★ **반드시 결과 값 및 결과로 나온 모든 그래프에 대해 자세한 분석을 하시오.**

참고: 제 08강 실습과제 8-4 Support Vector Machines - SVM Regression.pdf

실습과제 8-5

Train an SVM regressor on the California housing dataset.

★ 반드시 결과 값 및 결과로 나온 모든 그래프에 대해 자세한 분석을 하시오.

참고: 제 08강 실습과제 8-5 SVM regressor Housing dataset.pdf

Under the Hood

앞에서 모든 모델 파라메터를 1개의 벡터 $\boldsymbol{\theta}$에 둠

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma\left(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x}\right) = P(y = 1|x;\theta)$$

Suppose predict "y=1" if $\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x} \geq \mathbf{0}$

predict "y=0" if $\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x} < 0$

이제 SVM을 다루기 위한 더 편리한 표기법 사용:

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

b: bias term, $\mathbf{w}$: 가중치 벡터

Decision Function and Predictions

Linear SVM Classifier:

- 단순히 decision 함수 $w^T x + b = w_1 x_1 + \cdots + w_n x_n + b$ 계산

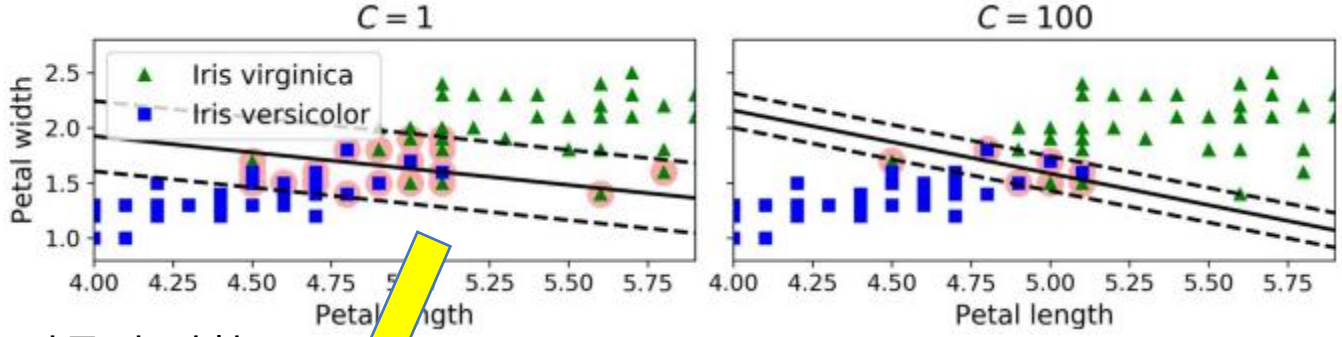
  $\Rightarrow$ 새 인스턴스 $\mathbf{x}$의 클래스 예측 수행

- Linear SVM classifier prediction:

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

b: bias term, $\mathbf{w}$: 가중치 벡터

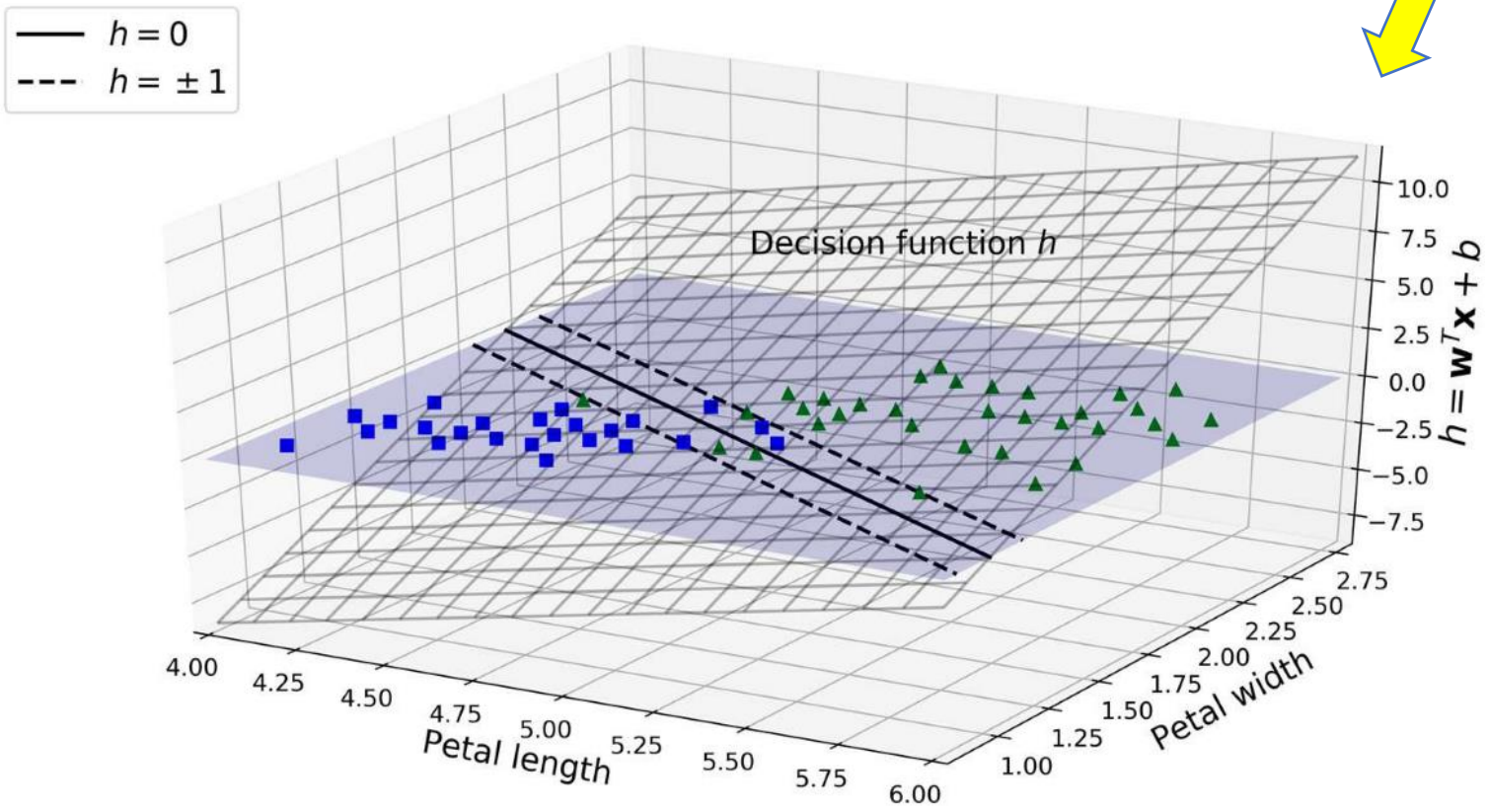
Decision Function and Predictions



Decision function => 2개의 특성 … 2D plane

Decision boundary => Decision function이 0(zero)인 점들의 집합:

- 두 평면이 만나는 교차점, 검은색 굵은 실선



Dashed 라인:

- Decision function이 1 또는 -1인 경우의 점들
- Parallel, decision boundary와 동일 거리
- Margin 형성

Linear SVM Classifier 학습목표:

- 가급적 margin을 넓게 해주는 가중치 벡터 $\mathbf{w}$와 b의 값을 찾는 것!!