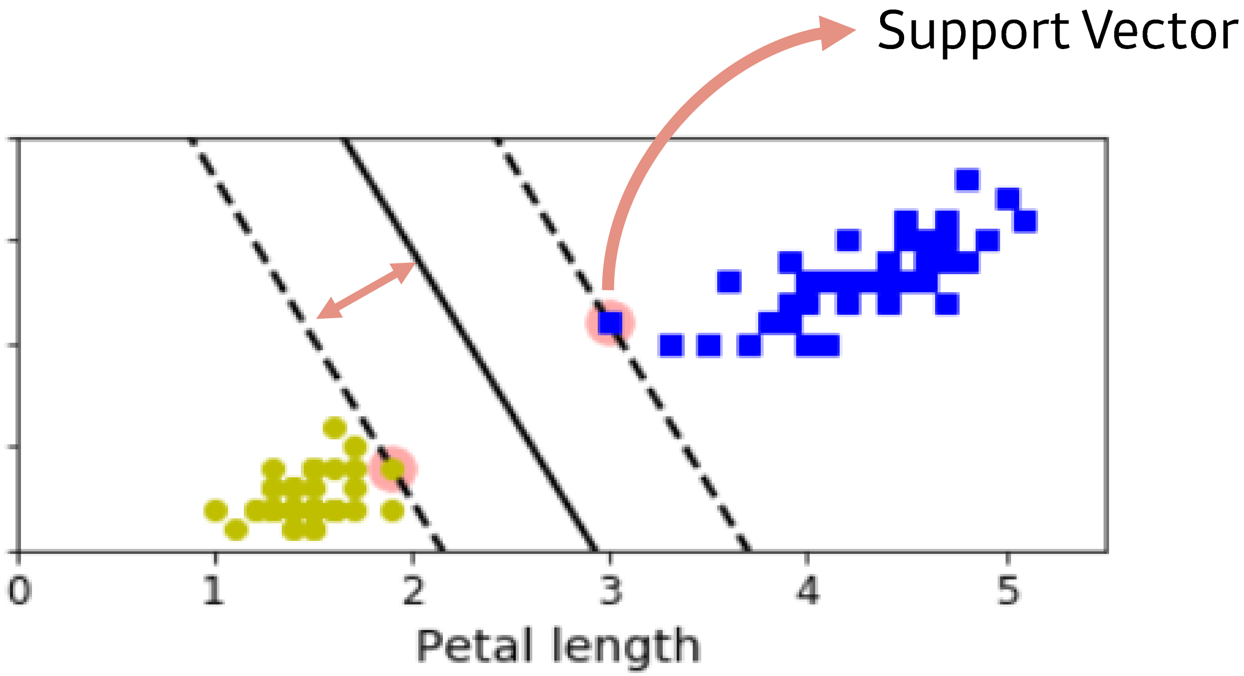
Ch5. SVM

01 Linear SVM Classification

- **Large Margin Classification**



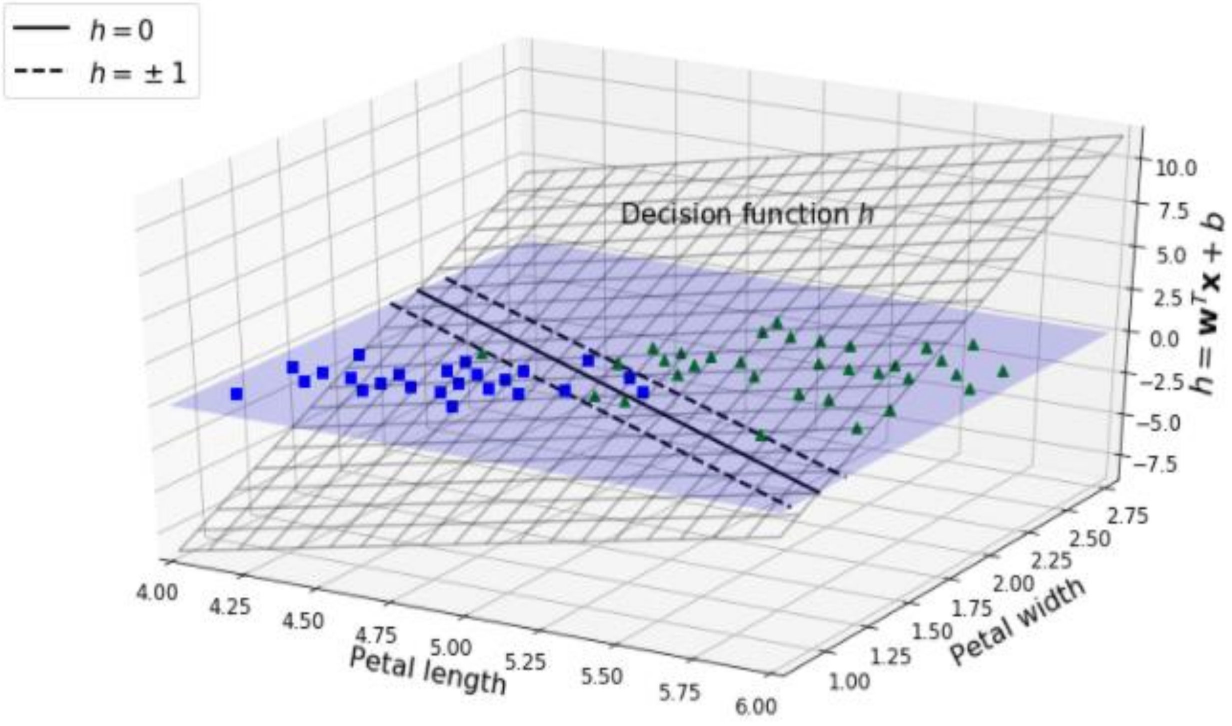
Find the widest margin Between classes with SVM classifier

01 Linear SVM Classification

- **Determination Functions & Predictions**

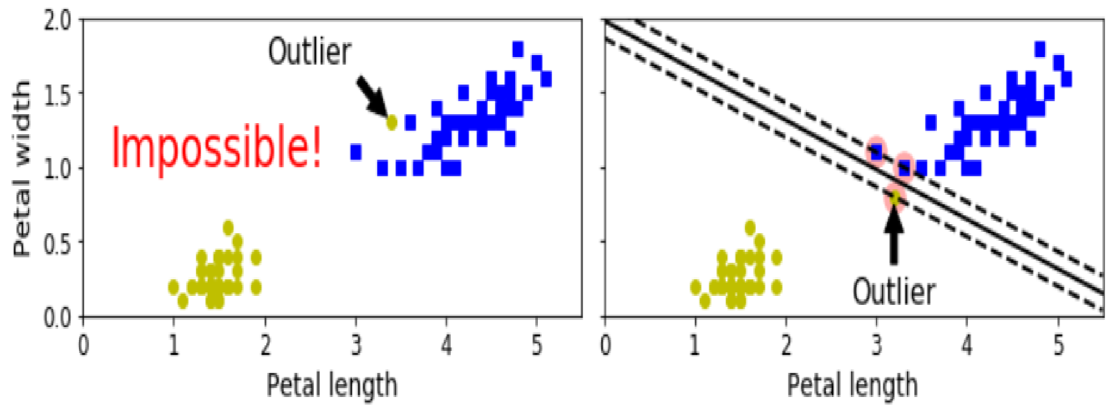
$$\hat{y} = \begin{cases} 0 & \mathbf{w}^T\mathbf{x} + b < 0 \text{ 일 때}, \\ 1 & \mathbf{w}^T\mathbf{x} + b \geq 0 \text{ 일 때} \end{cases}$$

Margin formation at a constant distance
for decision boundaries



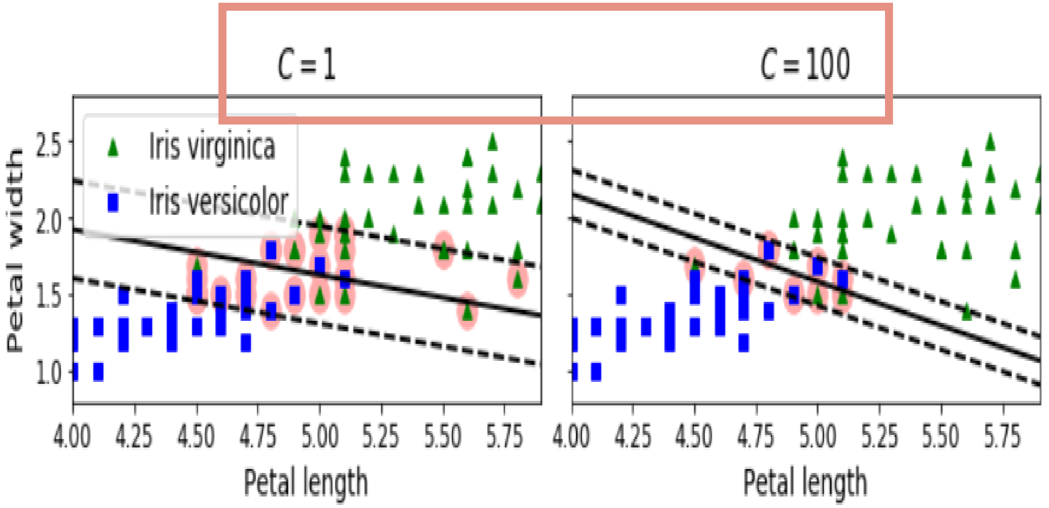
01 Linear SVM Classification

1) Hard Margin Classification



All samples are correctly classified

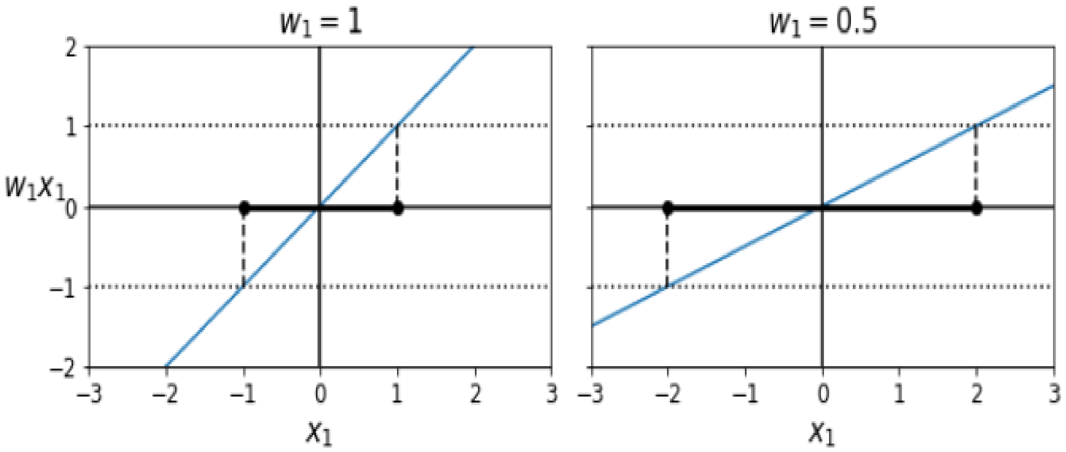
2) Soft Margin Classification



Keep the margin as wide as possible
&
Find the balance between margin errors

01 Linear SVM Classification

- **Objective Function**



1) Hard Margin Classification

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

2) Soft Margin Classification

$$\underset{\mathbf{w},b,\zeta}{\text{minimize}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{m}\zeta^{(i)}$$
$$\text{subject to} \quad t^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1 - \zeta^{(i)} \quad \text{and} \quad \zeta^{(i)} \geq 0 \quad \text{for } i = 1, 2, \dots, m$$

Slope of the decision function = weight vector w
=> Smaller weights result in larger margins

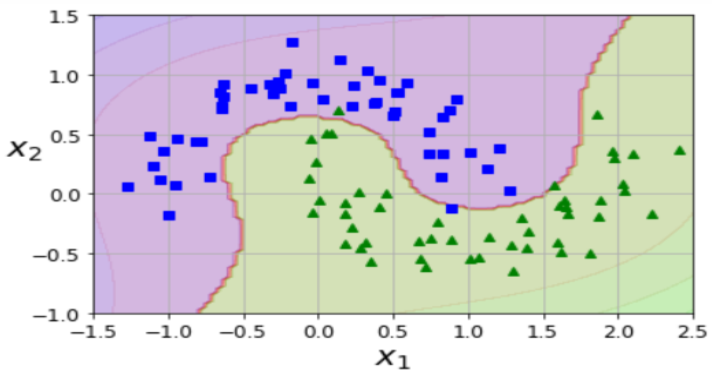
02 Non-Linear SVM Classification

When it is not possible to classify linearly, Add properties to distinguish linearly

- **Polynomial Kernel**

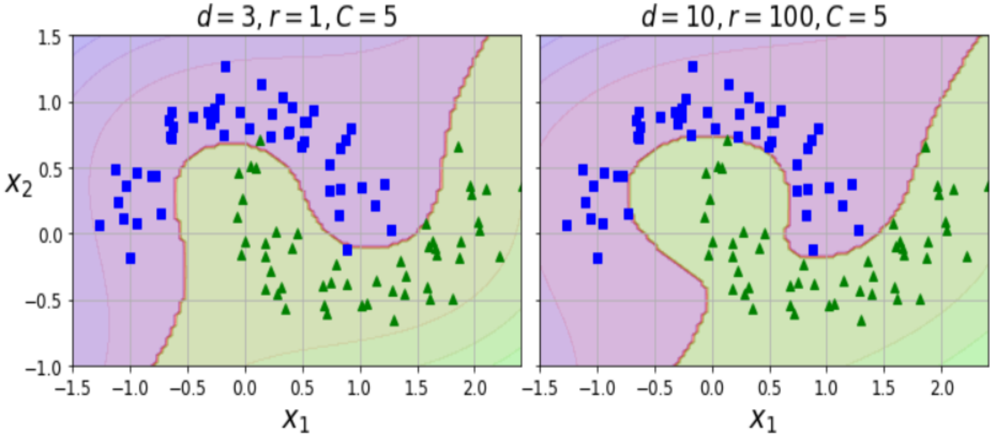
```
from sklearn.datasets import make_moons
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

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)
```



Use PolynomialFeatures

```
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)
```



Use SVM with Kernel Trick

02 Non-Linear SVM Classification

- **Kernel SVM**

Apply a quadratic polynomial transformation & linearSVM classifier to training set

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

=> Transformed Vector : 3 dimensions

Apply polynomial mapping to 2 a, b vectors & Multiply to 2 transformed vectors

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

$$= (a_1 b_1 + a_2 b_2)^2 = \left(\begin{pmatrix} a_1 \\ a_2 \end{pmatrix}^T \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 \boxed{= (\mathbf{a}^T \mathbf{b})^2}$$

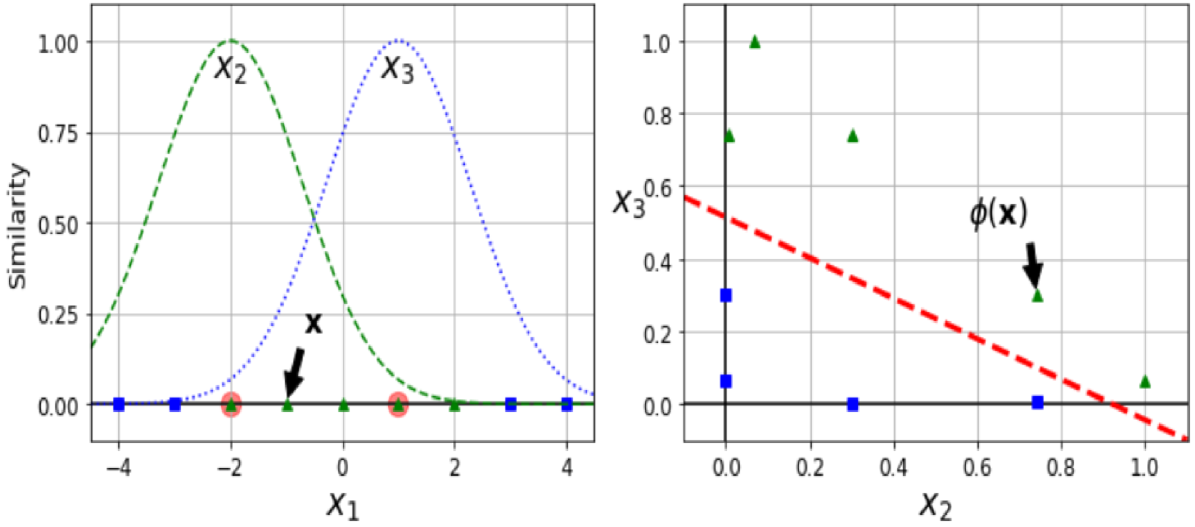
without convert training sample

=> Lower Computation : **Kernel trick**

- **Similarity function**

Add attributes calculated by measuring how much each sample resembles landmark

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

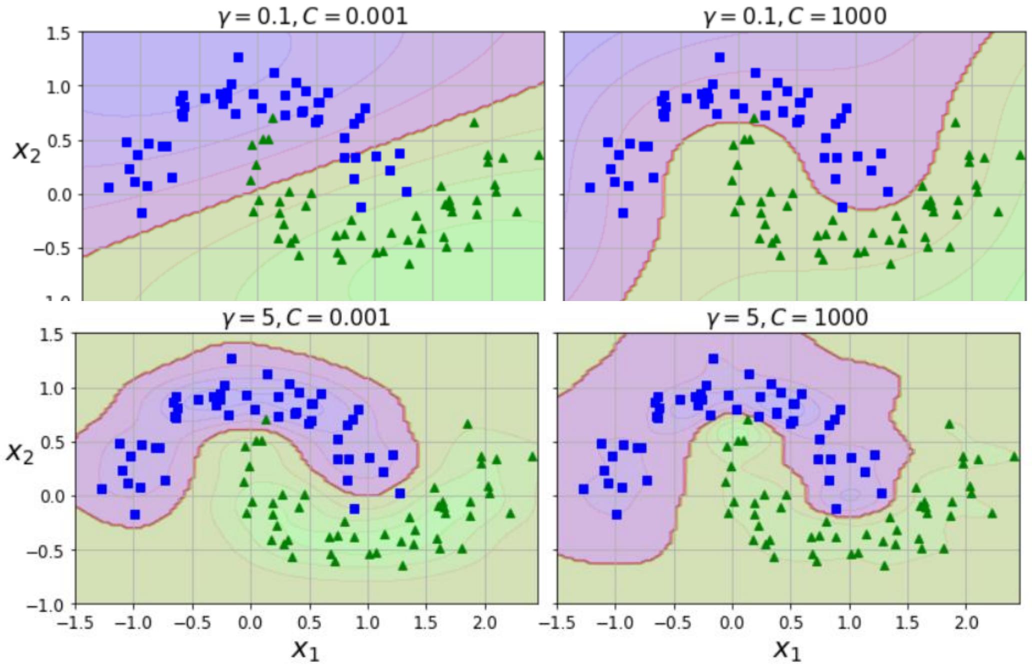


02 Non-Linear SVM Classification

- **RBF Kernel**

Utilize Kernel tricks => Because all additional attributes are computationally expensive

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



- gamma increase

  Narrower bell shape
  Irregular boundaries

- gamma decrease

  Boarder bell shape
  Smoother boundaries

=> gamma : regulation role

- **Computational Complexity**

  - **Linear SVC** : O(m*n) => Increased precision, then increase performance time

  - **SVC** : O(m2*n) ~ O(m3*n) ⇒ The larger number of training samples , the slower calculations
    ⇒ But, it scales well for sparse characteristics

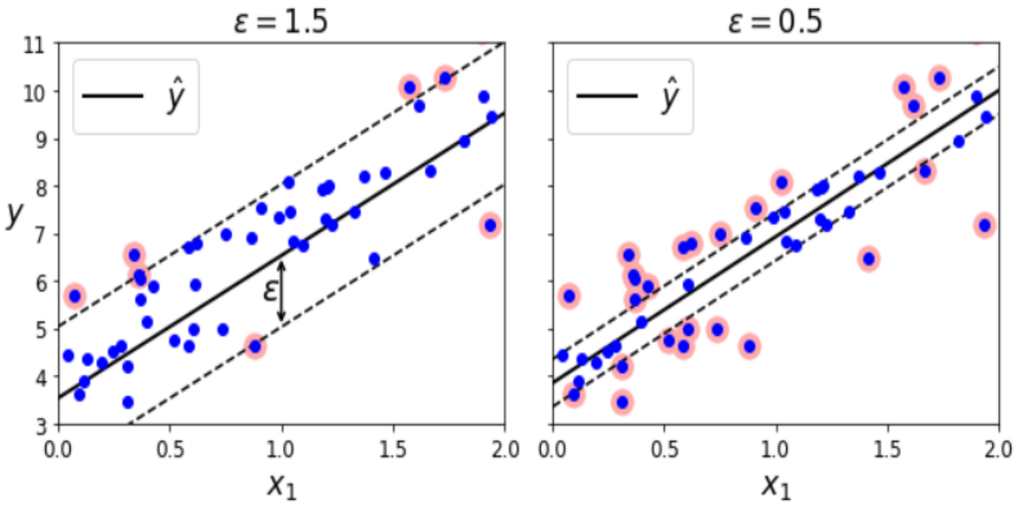
03 SVM Regression

- **Linear Regression : LinearSVR**

Learn to fit as many samples as possible within a limited margin error

```
from sklearn.svm import LinearSVR

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



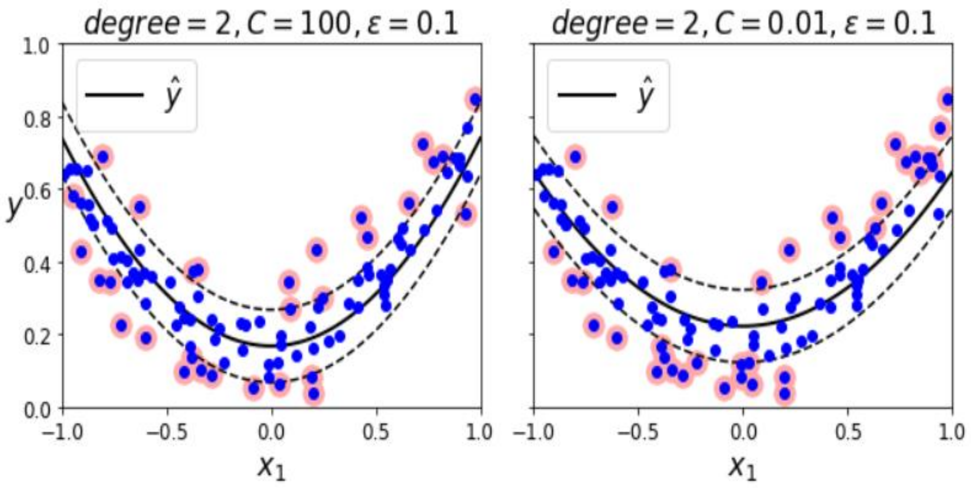
- Epsilon adjust the margin error
- Not sensitive to epsilon

  : Added training samples within margin error does not affect model prediction

03 SVM Regression

- **Non – Linear Regression : SVR**

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



- LinearSVR increases time relative to training set size

- SVR slow down as training set grow bigger

THANK YOU