SVM (Support Vector Machine)

2021. 8

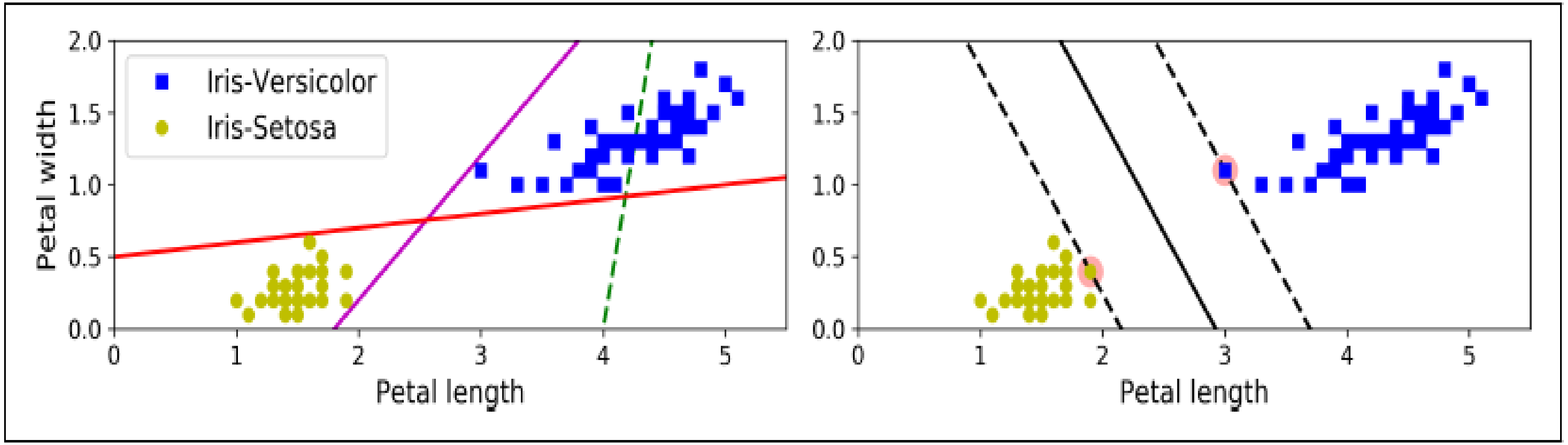
Yongjin Jeong, KwangWoon University

[참고] 본 자료에는 책이나 인터넷, 또는 외부 강의자료에서 인용하여 사용한 그림이나 수식들이 있으니 다른 용도로 사용하거나 외부로 유출을 금해 주시기 바랍니다.

[1] Aurellian Geron, Hands-on Machine Learning with Scikit, Keras, and Tensorflow
[2] https://www.robots.ox.ac.uk/~az/lectures (Prof. Zisserman's lecture slide)

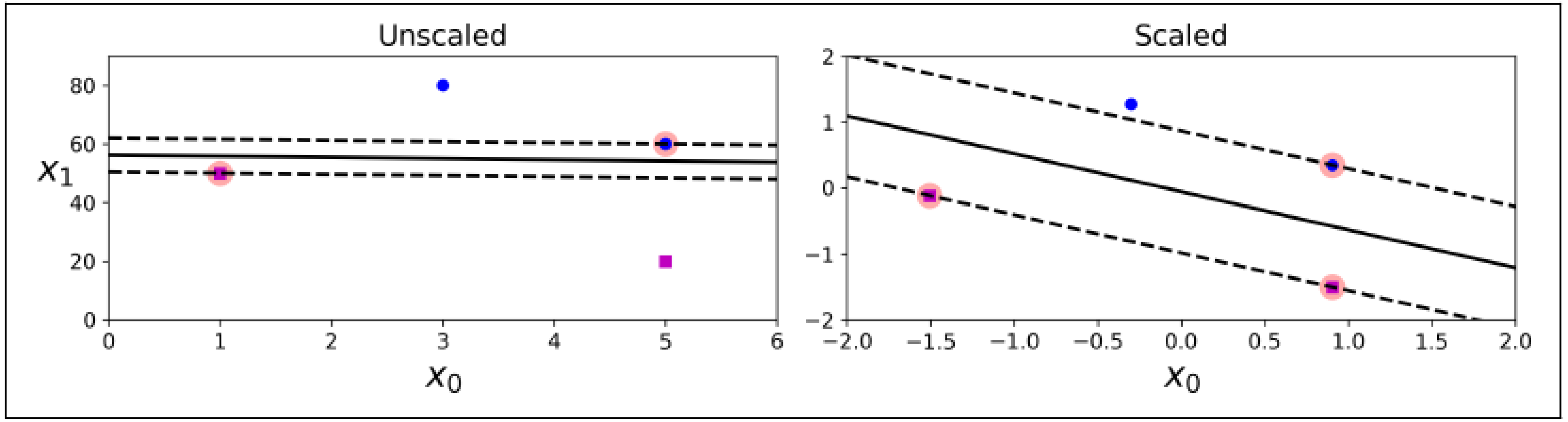
SVM Classifier

- **Linear classification and SVM Classifier**



SVM Classifier

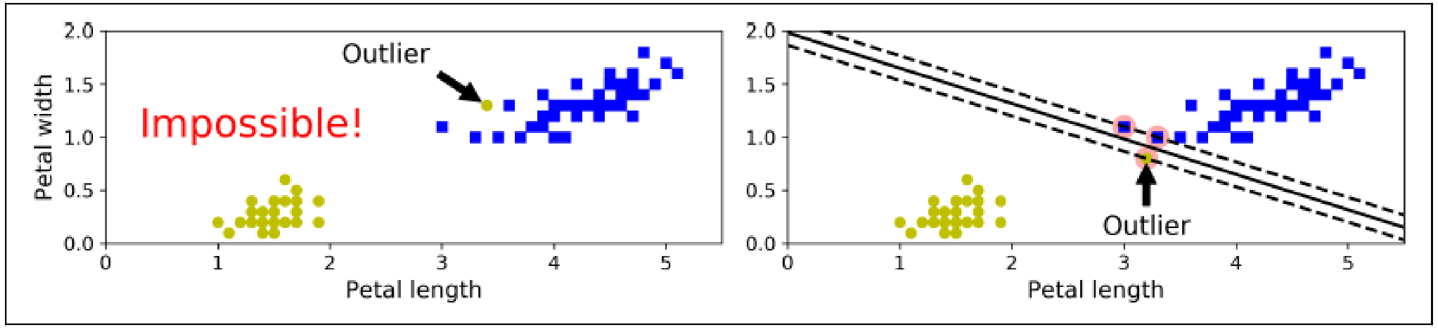
- **Sensitivity to feature scales**



Before Scaling **After Scaling**

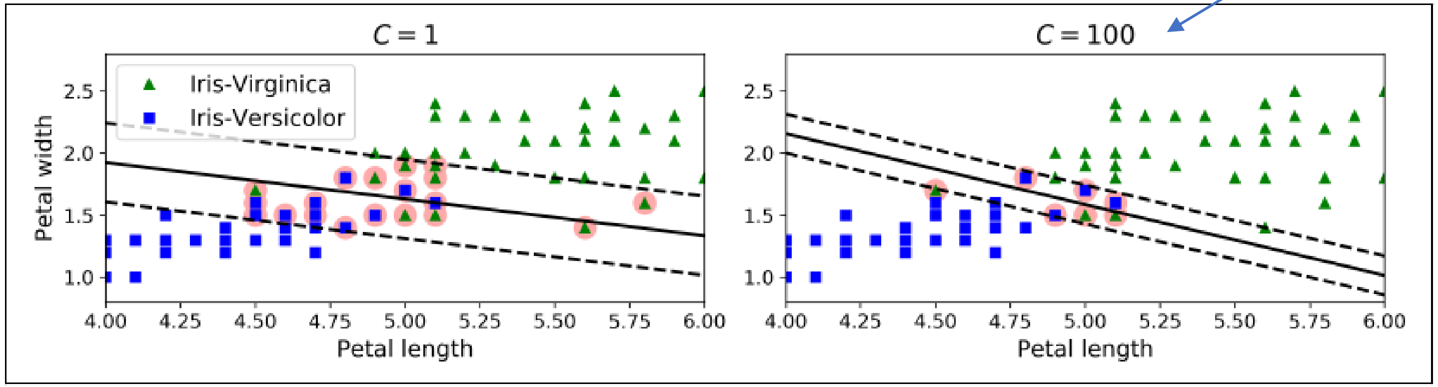
SVM Classifier

- **Hard margin and Soft margin**
  - **Hard margin**



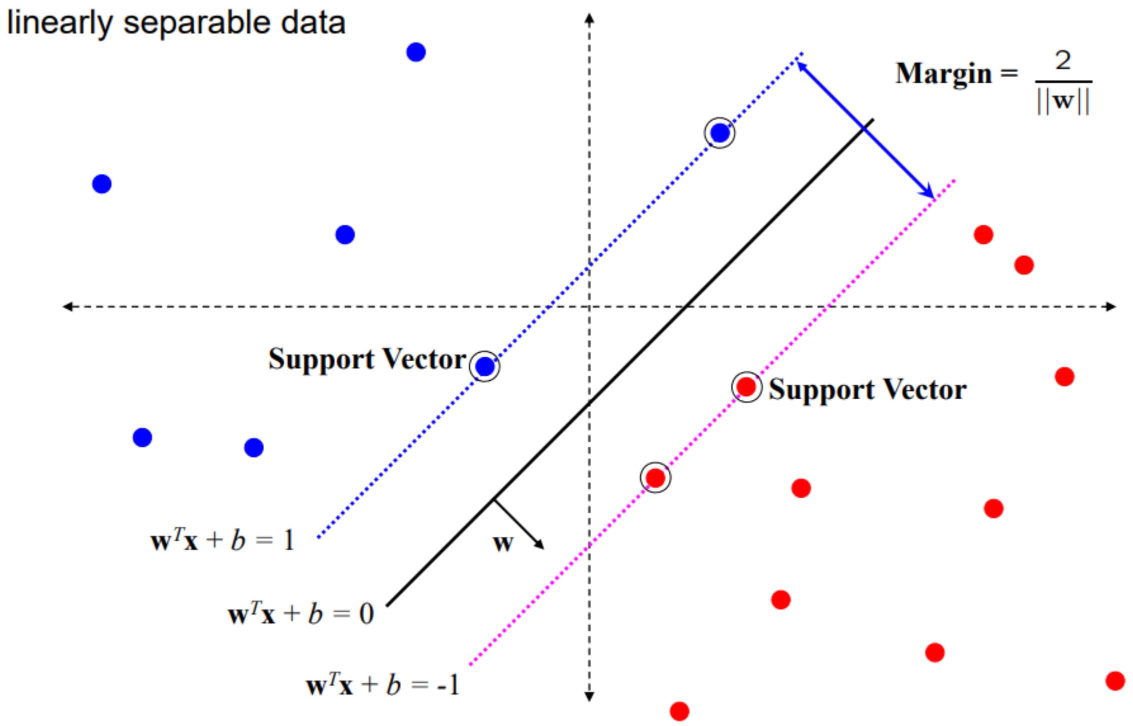
- **Soft margin**
  - Large margin and fewer margin violations

Overfitting 가능성



SVM Classifier

- **Quadratic optimization problem subject to linear constraints**
  - **There is a unique minimum.**



SVM is formulated as an optimization:

$$\max_{\mathbf{w}} \frac{2}{||\mathbf{w}||} \text{ subject to } \mathbf{w}^\top \mathbf{x}_i + b \begin{array}{l} \geq 1 \\ \leq -1 \end{array} \quad \begin{array}{l} \text{if } y_i = +1 \\ \text{if } y_i = -1 \end{array} \quad \text{for } i = 1 \ldots N$$

Or equivalently

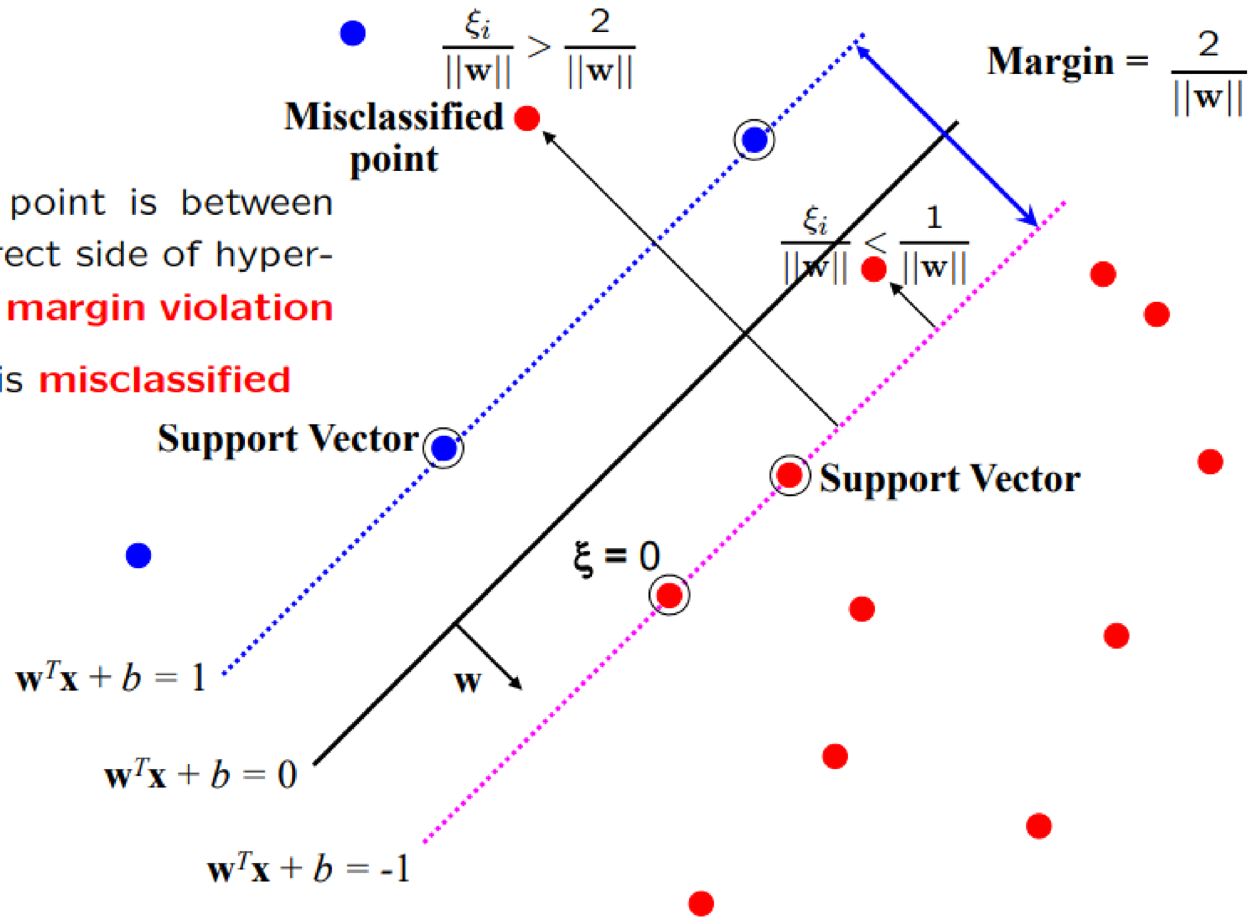
$$\min_{\mathbf{w}} ||\mathbf{w}||^2 \text{ subject to } y_i \left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1 \text{ for } i = 1 \ldots N$$

SVM Classifier

- **Introduce Slack variables**
  - amount of error (hinge loss) from the correct side of hyperplane ?

$$\xi_i \geq 0$$

- for $0 < \xi \leq 1$ point is between margin and correct side of hyper-plane. This is a **margin violation**
- for $\xi > 1$ point is **misclassified**



SVM Classifier

- **Soft margin**

The optimization problem becomes

$$\min_{\mathbf{w}\in\mathbb{R}^d, \xi_i\in\mathbb{R}^+} ||\mathbf{w}||^2 + C\sum_i^N \xi_i$$

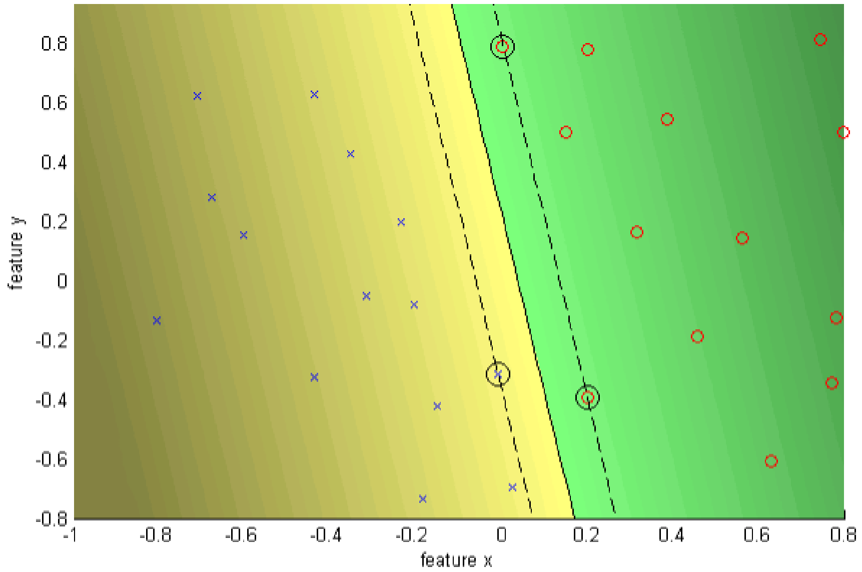
subject to

$$y_i\left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1 - \xi_i \text{ for } i = 1\ldots N$$

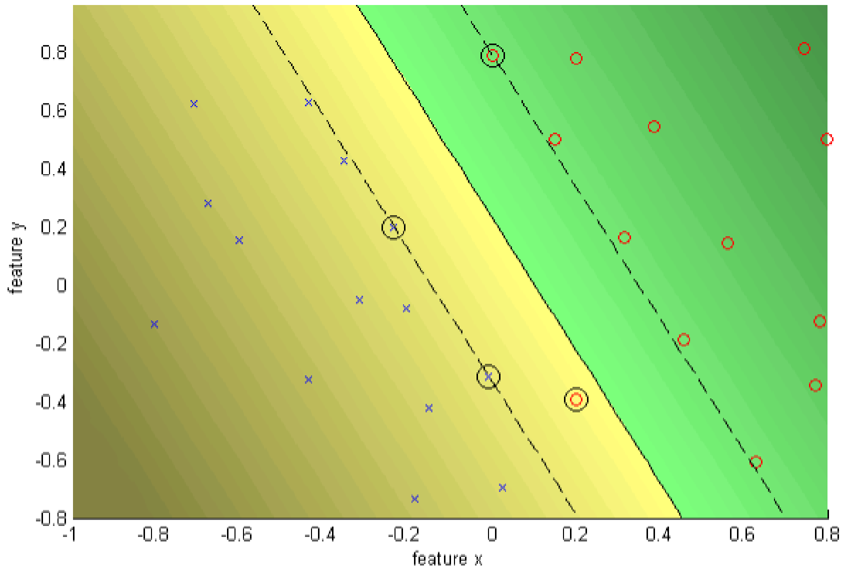
- Every constraint can be satisfied if ξ_i is sufficiently large
- C is a regularization parameter:
  - small C allows constraints to be easily ignored $\rightarrow$ large margin
  - large C makes constraints hard to ignore $\rightarrow$ narrow margin
  - $C = \infty$ enforces all constraints: hard margin
- This is still a quadratic optimization problem and there is a unique minimum. Note, there is only one parameter, C.

SVM Classifier

- **Hard margin and Soft margin**



C = Infinity hard margin



C = 10 Soft margin

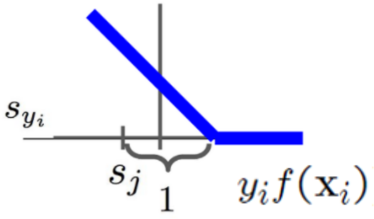
SVM Optimization

- **Constrained optimization problem**

$$\min_{\mathbf{w}\in\mathbb{R}^d, \xi_i \in \mathbb{R}^+} ||\mathbf{w}||^2 + C \sum_i^N \xi_i \text{ subject to } y_i \left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1 - \xi_i \text{ for } i = 1 \dots N$$

$$y_i \left(\mathbf{w}^\top \mathbf{x}_i + b\right) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$

$$\Rightarrow \quad \xi_i = \max\left(0, 1 - y_i f(\mathbf{x}_i)\right)$$



- **The learning problem is now equivalent to the unconstrained optimization problem over w**

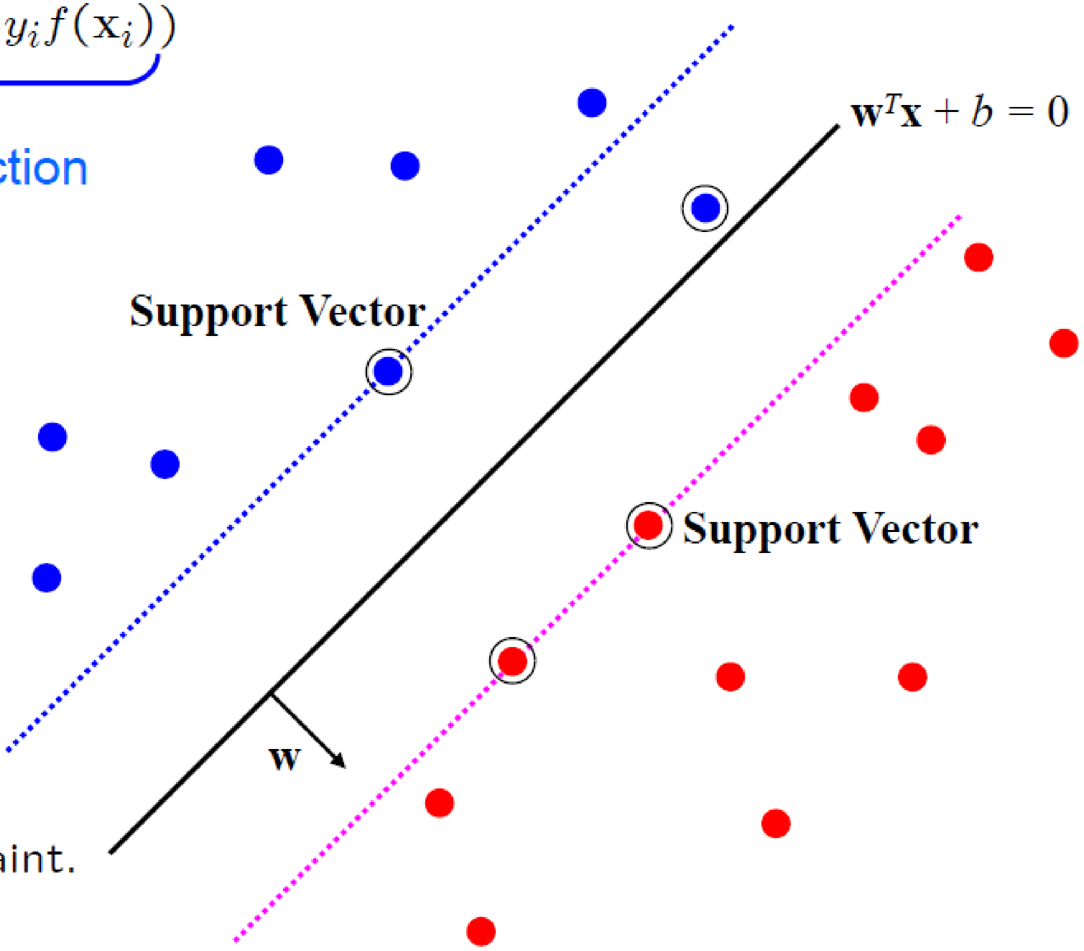
$$\min_{\mathbf{w}\in\mathbb{R}^d} \underbrace{||\mathbf{w}||^2}_{\text{regularization}} + C \sum_i^N \underbrace{\max\left(0, 1 - y_i f(\mathbf{x}_i)\right)}_{\text{loss function}}$$

Loss Function

$$\min_{\mathbf{w}\in\mathbb{R}^d} ||\mathbf{w}||^2 + C\sum_{i}^{N} \underbrace{\max\left(0, 1 - y_i f(\mathbf{x}_i)\right)}_{\text{loss function}}$$

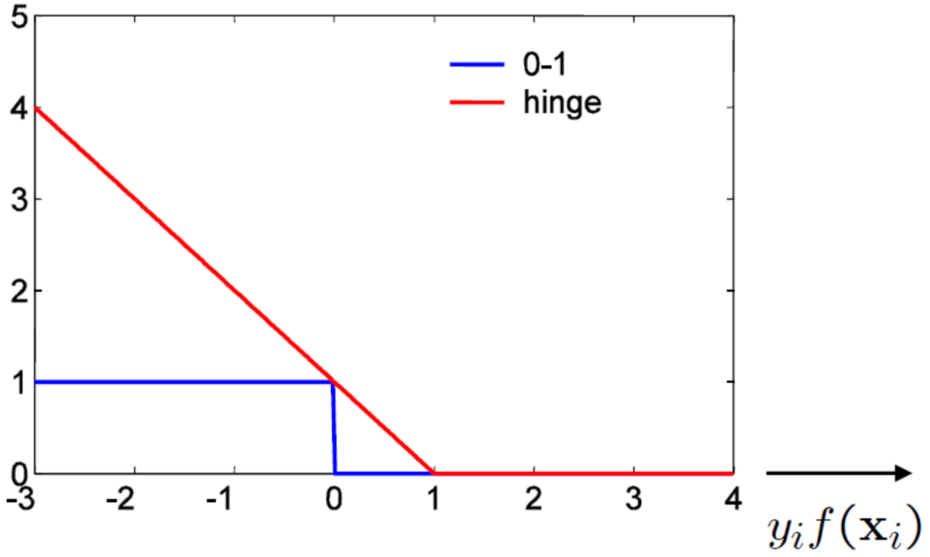
Points are in three categories:

1. $y_i f(x_i) > 1$
   Point is outside margin.
   No contribution to loss
2. $y_i f(x_i) = 1$
   Point is on margin.
   No contribution to loss.
   As in hard margin case.
3. $y_i f(x_i) < 1$
   Point violates margin constraint.
   Contributes to loss



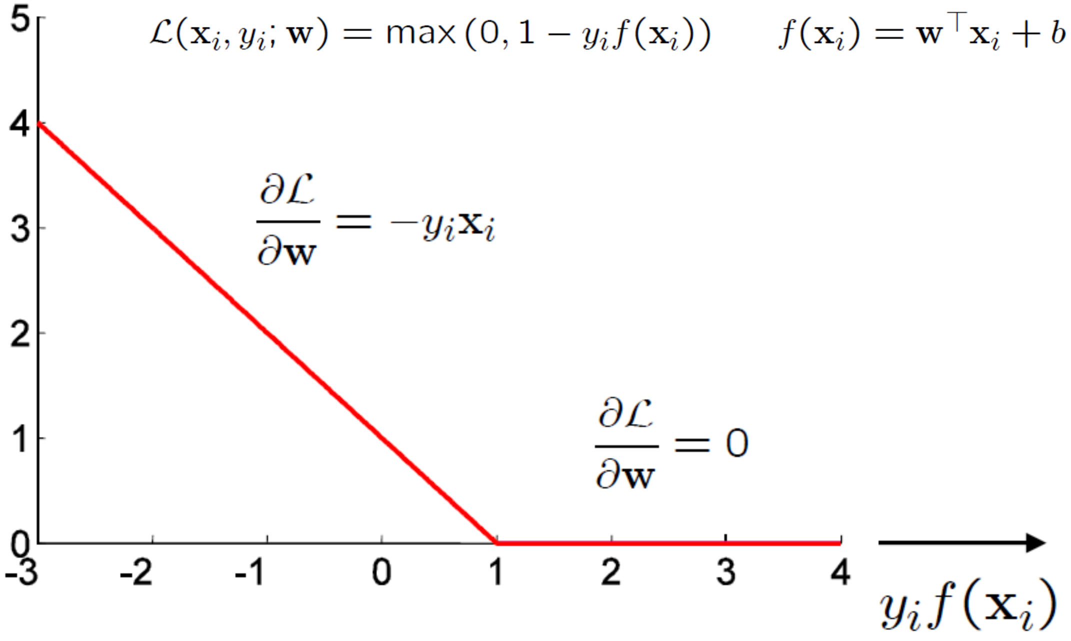
Hinge Loss

- **Hinge loss**



- SVM uses "hinge" loss $\max(0, 1 - y_i f(\mathbf{x}_i))$
- an approximation to the 0-1 loss

- **Sub-gradient for Hinge loss**



Sub-gradient descent algorithm for SVM

$$\mathcal{C}(\mathbf{w}) = \frac{1}{N}\sum_{i}^{N}\left(\frac{\lambda}{2}||\mathbf{w}||^2 + \mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w})\right)$$

The iterative update is

$$\begin{aligned}
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta \nabla_{\mathbf{w}_t}\mathcal{C}(\mathbf{w}_t) \\
&\leftarrow \mathbf{w}_t - \eta \frac{1}{N}\sum_{i}^{N}(\lambda \mathbf{w}_t + \nabla_{\mathbf{w}}\mathcal{L}(\mathbf{x}_i, y_i; \mathbf{w}_t))
\end{aligned}$$

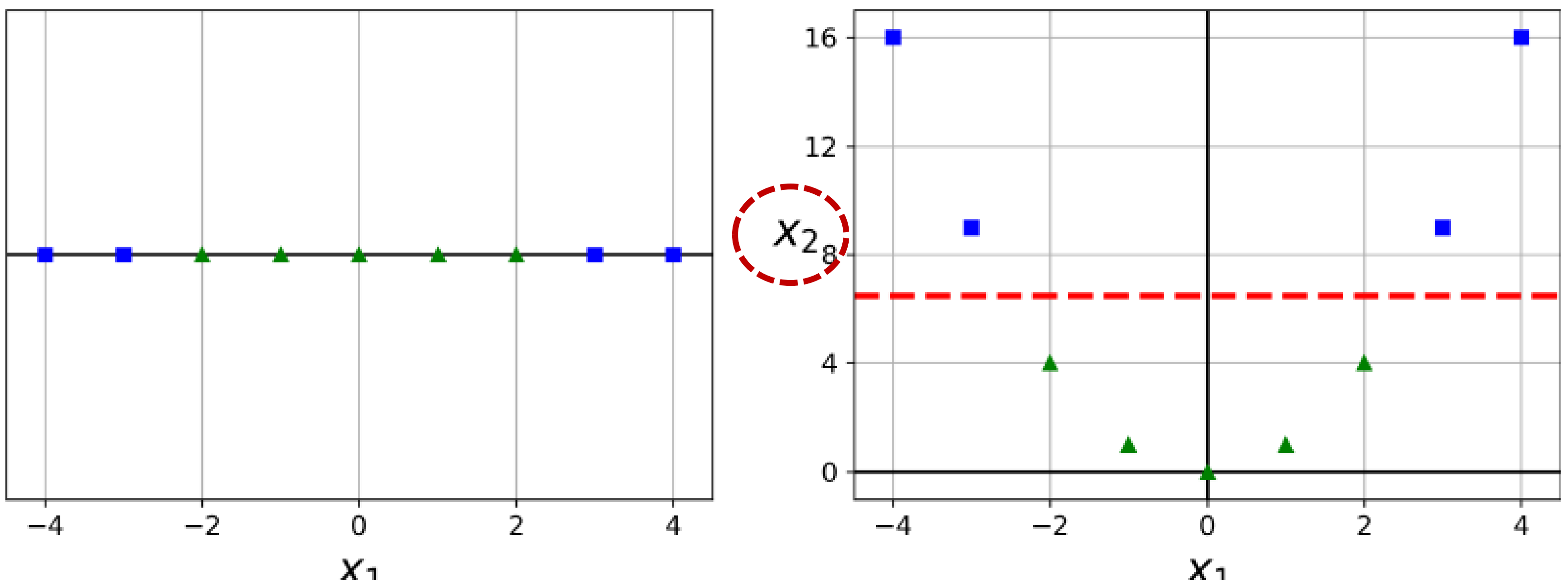
where η is the learning rate.

Then each iteration t involves cycling through the training data with the updates:

$$\begin{aligned}
\mathbf{w}_{t+1} &\leftarrow \mathbf{w}_t - \eta(\lambda \mathbf{w}_t - y_i \mathbf{x}_i) && \text{if } y_i f(\mathbf{x}_i) < 1 \\
&\leftarrow \mathbf{w}_t - \eta \lambda \mathbf{w}_t && \text{otherwise}
\end{aligned}$$

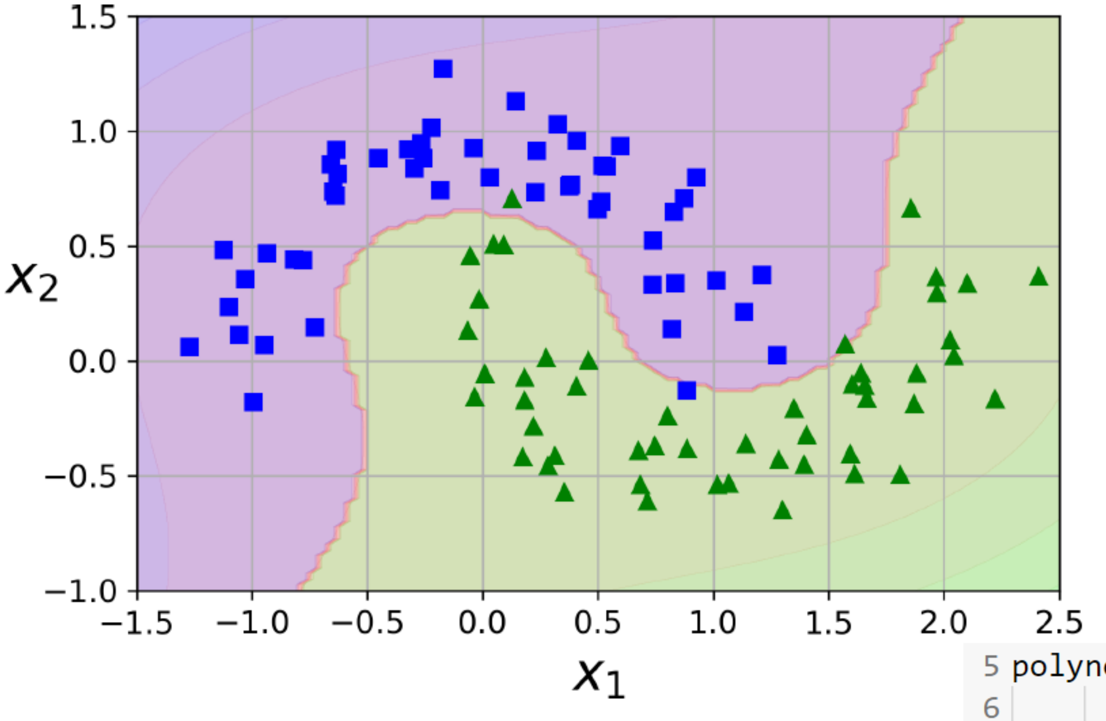
Nonlinear SVM Classifier

- **Adding features to make a dataset linearly separable**
  - Add a second feature $x2 = (x1)^2$



Nonlinear SVM Classifier

- **Linear SVM classifier using polynomial features**



10 features: c, x1, x2, $x1^2$, $x2^2$, x1x2, $x1^3$, $x2^3$, $x1^2x2$, $x1x2^2$

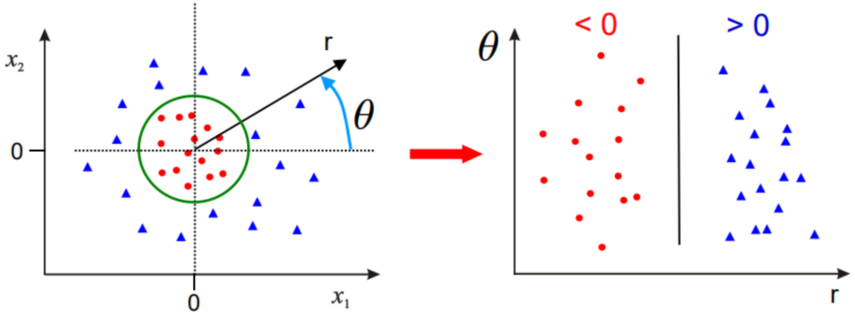
```python
polynomial_svm_clf = Pipeline([
        ("poly_features", PolynomialFeatures(degree=3)),
        ("scaler", StandardScaler()),
        ("svm_clf", LinearSVC(C=10, loss="hinge", random_state=42))
    ])

polynomial_svm_clf.fit(X, y)
```

Nonlinear SVM Classifier

- **General (linearly) non-separable data**

Use polar coordinates

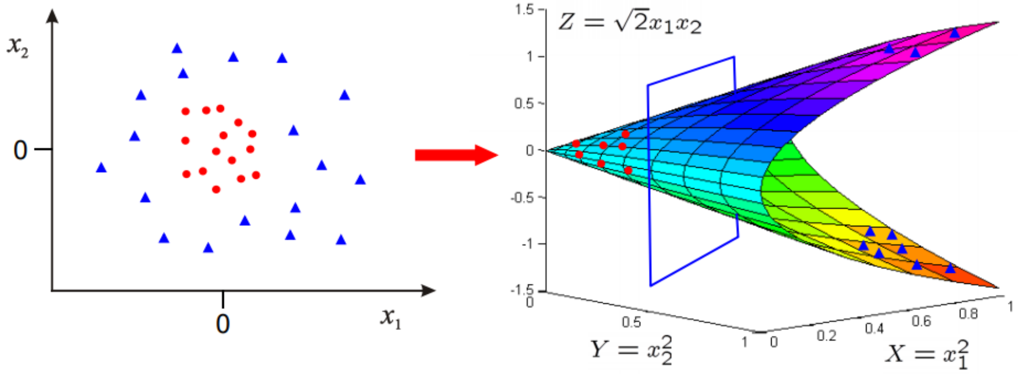


- Data is linearly separable in polar coordinates
- Acts non-linearly in original space

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} r \\ \theta \end{pmatrix} \qquad \mathbb{R}^2 \rightarrow \mathbb{R}^2$$

Map data to higher dimension

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \qquad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$



Feature space

Primal and Dual formulations

N is number of training points, and d is dimension of feature vector $\mathbf{x}$.

Primal problem: for $\mathbf{w} \in \mathbb{R}^d$

$$\min_{\mathbf{w}\in\mathbb{R}^d} ||\mathbf{w}||^2 + C\sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

Dual problem: for $\boldsymbol{\alpha} \in \mathbb{R}^N$ (stated without proof):

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2}\sum_{jk} \alpha_j \alpha_k y_j y_k (\mathbf{x}_j{}^\top \mathbf{x}_k) \text{ subject to } 0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

- Need to learn d parameters for primal, and N for dual
- If $N << d$ then more efficient to solve for α than $\mathbf{w}$
- Dual form only involves $(\mathbf{x}_j{}^\top \mathbf{x}_k)$.

Primal and Dual in transformed Feature space

Primal Classifier in transformed feature space

Classifier, with $\mathbf{w} \in \mathbb{R}^D$:

$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b$$

Learning, for $\mathbf{w} \in \mathbb{R}^D$

$$\min_{\mathbf{w} \in \mathbb{R}^D} ||\mathbf{w}||^2 + C \sum_i^N \max(0, 1 - y_i f(\mathbf{x}_i))$$

- Simply map $\mathbf{x}$ to $\Phi(\mathbf{x})$ where data is separable
- Solve for $\mathbf{w}$ in high dimensional space $\mathbb{R}^D$
- If $D >> d$ then there are many more parameters to learn for $\mathbf{w}$. Can this be avoided?

Dual Classifier in transformed feature space

Classifier:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i \mathbf{x}_i^\top \mathbf{x} + b$$

$$\rightarrow f(\mathbf{x}) = \sum_i^N \alpha_i y_i \Phi(\mathbf{x}_i)^\top \Phi(\mathbf{x}) + b$$

Learning:

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \mathbf{x}_j^\top \mathbf{x}_k$$

$$\rightarrow \max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \Phi(\mathbf{x}_j)^\top \Phi(\mathbf{x}_k)$$

subject to

$$0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

SVM Kernels

Dual Classifier in transformed feature space

- Note, that $\Phi(\mathbf{x})$ only occurs in pairs $\Phi(\mathbf{x}_j)^\top\Phi(\mathbf{x}_i)$
- Once the scalar products are computed, only the N dimensiona vector $\boldsymbol{\alpha}$ needs to be learnt; it is not necessary to learn in the D dimensional space, as it is for the primal
- Write $k(\mathbf{x}_j, \mathbf{x}_i) = \Phi(\mathbf{x}_j)^\top\Phi(\mathbf{x}_i)$. This is known as a Kernel

Classifier:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i \, k(\mathbf{x}_i, \mathbf{x}) + b$$

Learning:

$$\max_{\alpha_i \geq 0} \sum_i \alpha_i - \frac{1}{2} \sum_{jk} \alpha_j \alpha_k y_j y_k \, k(\mathbf{x}_j, \mathbf{x}_k)$$

subject to

$$0 \leq \alpha_i \leq C \text{ for } \forall i, \text{ and } \sum_i \alpha_i y_i = 0$$

Special transformations

$$\Phi : \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \rightarrow \begin{pmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{pmatrix} \qquad \mathbb{R}^2 \rightarrow \mathbb{R}^3$$

$$\begin{aligned} \Phi(\mathbf{x})^\top\Phi(\mathbf{z}) &= \left(x_1^2, x_2^2, \sqrt{2}x_1x_2\right) \begin{pmatrix} z_1^2 \\ z_2^2 \\ \sqrt{2}z_1z_2 \end{pmatrix} \\ &= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \\ &= (x_1z_1 + x_2z_2)^2 \\ &= (\mathbf{x}^\top\mathbf{z})^2 \end{aligned}$$

Kernel Trick

- Classifier can be learnt and applied without explicitly computing $\Phi(\mathbf{x})$
- All that is required is the kernel $k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top\mathbf{z})^2$
- Complexity of learning depends on N (typically it is $O(N^3)$) not on D

Common SVM Kernels

Mercer's Theorem

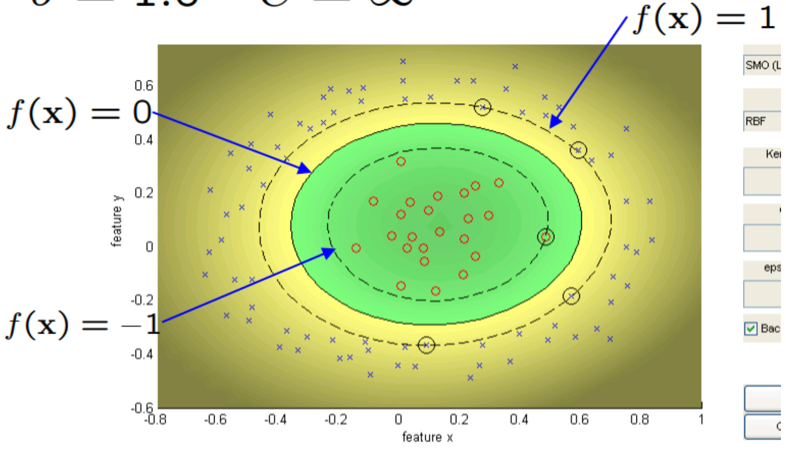
According to *Mercer's theorem*, if a function $K(\mathbf{a}, \mathbf{b})$ respects a few mathematical conditions called *Mercer's conditions* (K must be continuous, symmetric in its arguments so $K(\mathbf{a}, \mathbf{b}) = K(\mathbf{b}, \mathbf{a})$, etc.), then there exists a function ϕ that maps $\mathbf{a}$ and $\mathbf{b}$ into another space (possibly with much higher dimensions) such that $K(\mathbf{a}, \mathbf{b}) = \phi(\mathbf{a})^T \phi(\mathbf{b})$. So you can use K as a kernel since you know ϕ exists, even if you don't know what ϕ is. In the case of the Gaussian RBF kernel, it can be shown that ϕ actually maps each training instance to an infinite-dimensional space, so it's a good thing you don't need to actually perform the mapping!

$$\begin{aligned}
&\text{Linear:} && K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T\mathbf{b} \\
&\text{Polynomial:} && K(\mathbf{a}, \mathbf{b}) = \left(\gamma\mathbf{a}^T\mathbf{b} + r\right)^d \\
&\text{Gaussian RBF:} && K(\mathbf{a}, \mathbf{b}) = \exp\left(-\gamma \| \mathbf{a} - \mathbf{b} \|^2\right) \\
&\text{Sigmoid:} && K(\mathbf{a}, \mathbf{b}) = \tanh\left(\gamma\mathbf{a}^T\mathbf{b} + r\right)
\end{aligned}$$

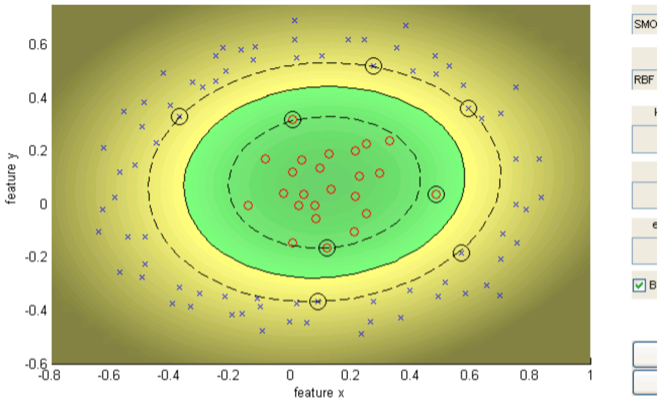
RBF Kernel SVM Example

$$f(\mathbf{x}) = \sum_{i}^{N} \alpha_i y_i \exp\left(-||\mathbf{x} - \mathbf{x}_i||^2 / 2\sigma^2\right) + b$$

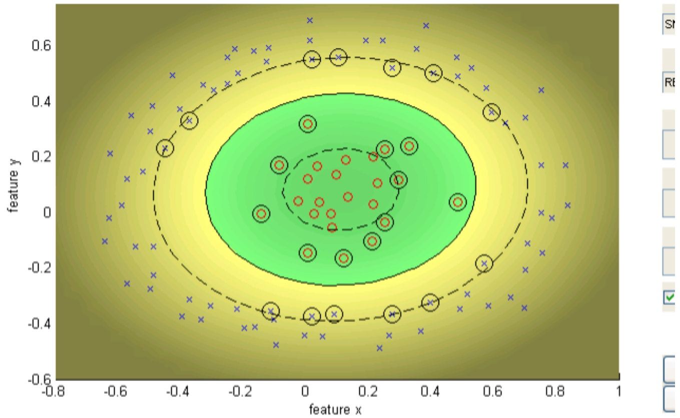
$\sigma = 1.0 \quad C = \infty$



$\sigma = 1.0 \quad C = 100$

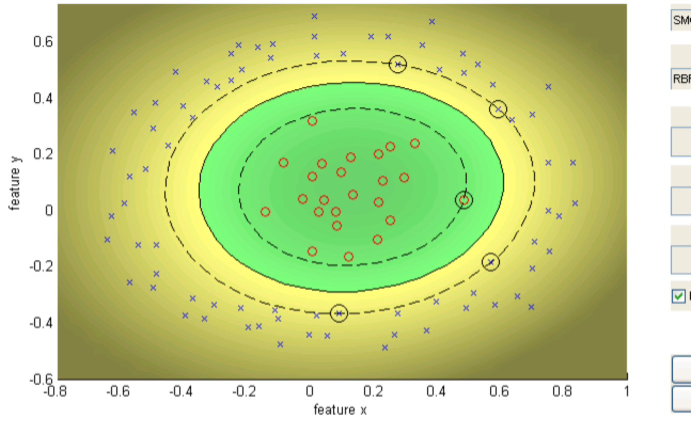


$\sigma = 1.0 \quad C = 10$

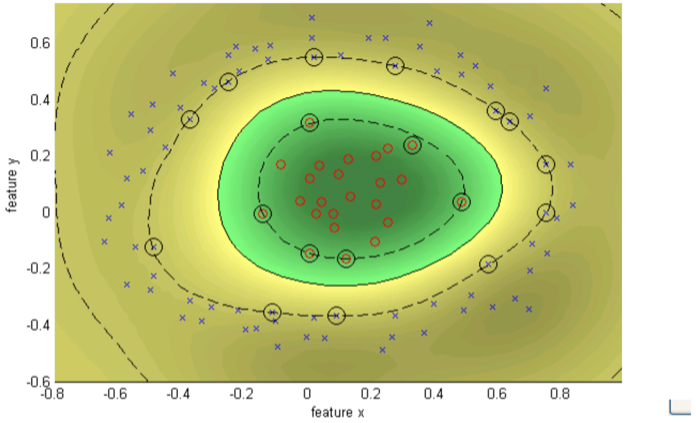


Decrease C -> wider (soft) margin

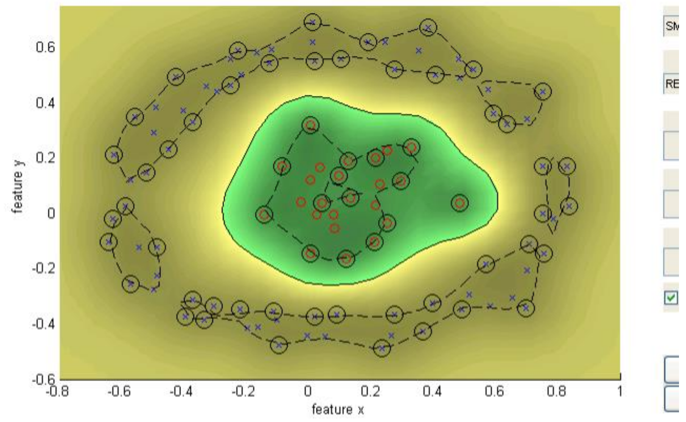
$\sigma = 1.0 \quad C = \infty$



$\sigma = 0.25 \quad C = \infty$



$\sigma = 0.1 \quad C = \infty$



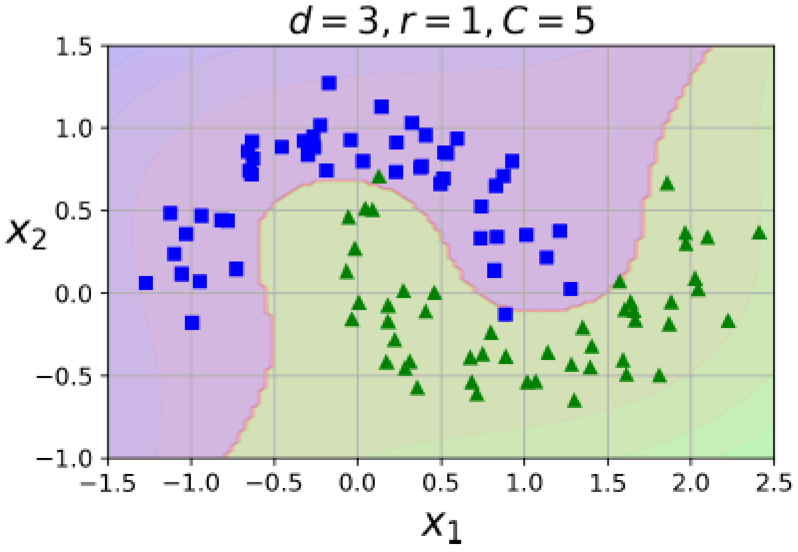
Decrease sigma (increase gamma) -> move towards nearest neighbor classifier

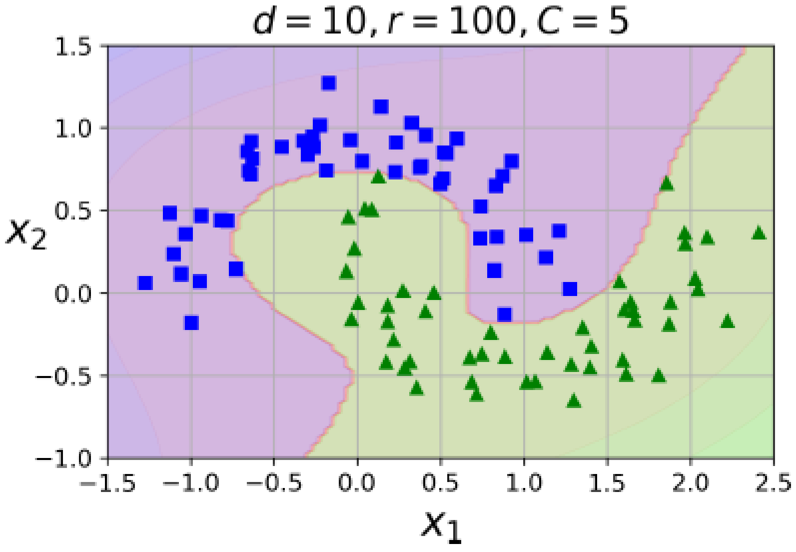
SVM Kernels

- **SVM classifiers with a polynomial kernel**

K(a,b) = $(a \times b + r)^d$

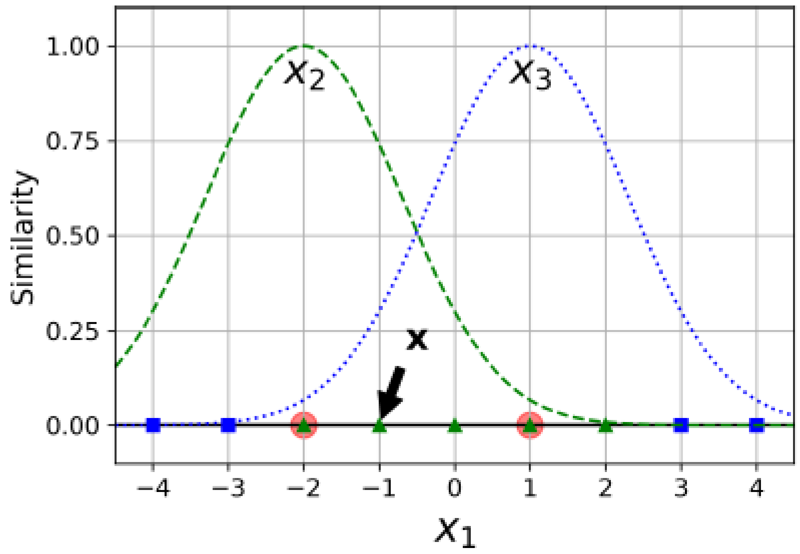
- a, b: 서로 다른 데이터
- r: polynomial의 coefficient를 결정
- d: polynomial의 차수

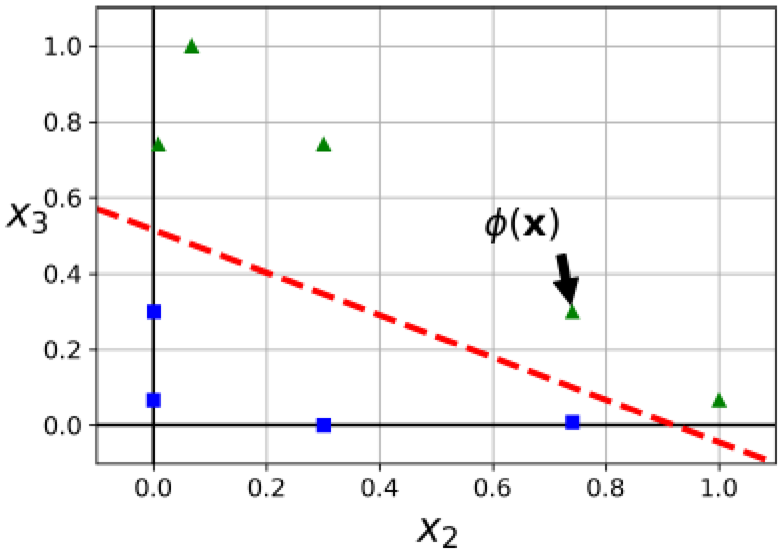




SVM Kernels

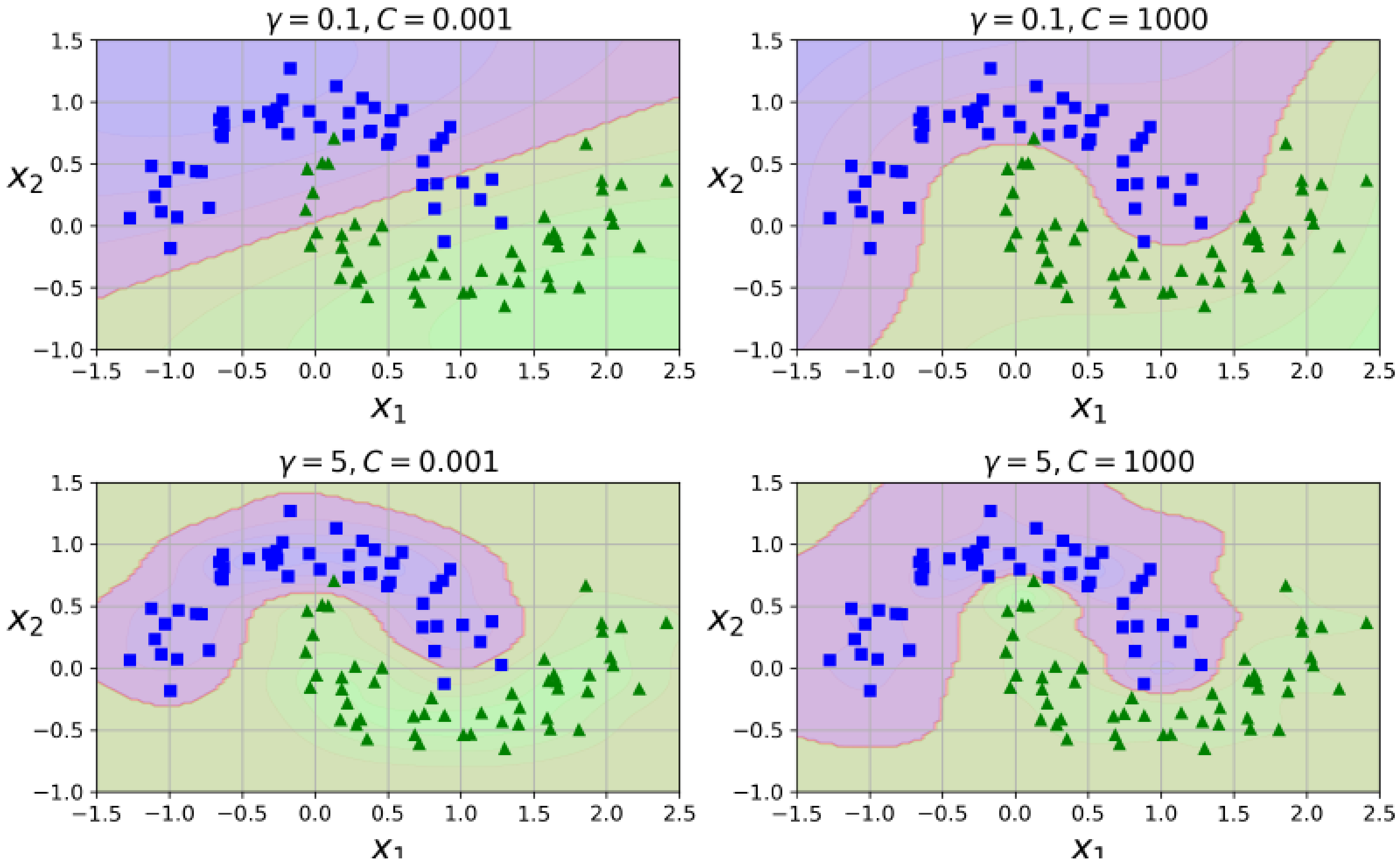
- **Adding Similarity features**
  - measures how much each instance resembles a particular *landmark*.
  - define the similarity function: **Gaussian Radial Basis Function (*RBF*)**





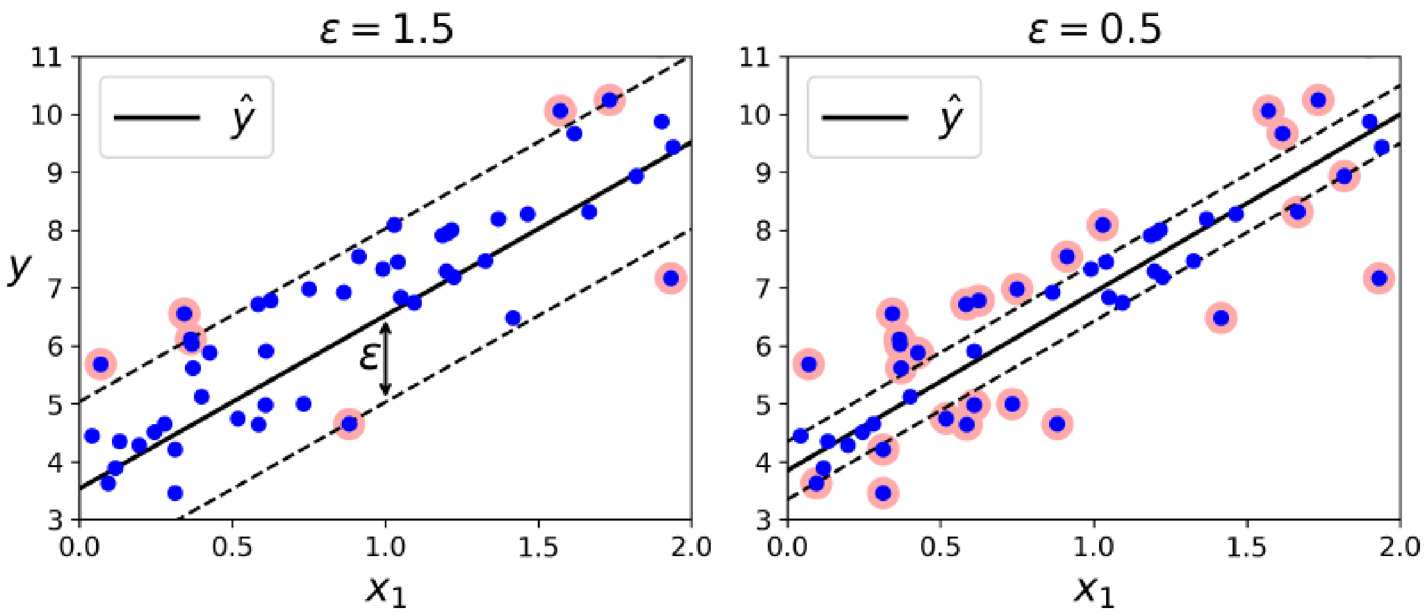
SVM Kernels

- **Gaussian RBF Kernel**



SVM Regression

- **It also supports linear and nonlinear regression.**
- **Reverse the object:**
  - instead of trying to fit the largest possible street between two classes while limiting margin violations,
  - SVM Regression tries to **fit as many instances as possible *on* the street** while limiting margin violations (i.e., instances *off* the street).
  - The width of the street is controlled by a hyperparameter ϵ.



SVM Regression

- **SVM regression using 2nd degree polynomial kernel**