SVM (Support Vector Machine)

2021. 8

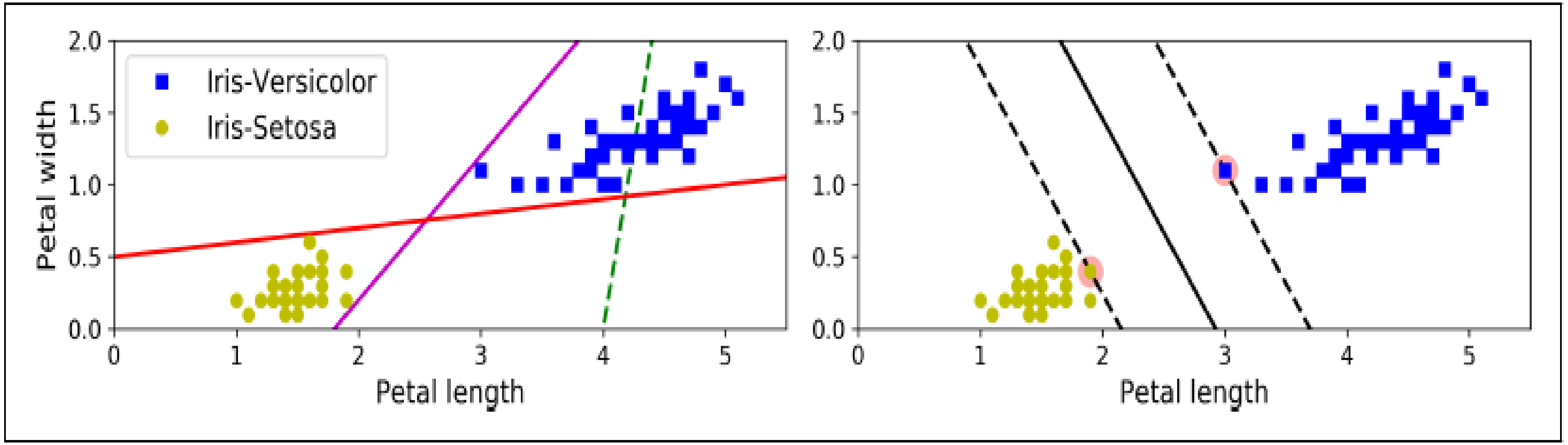
Yongjin Jeong, KwangWoon University

[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)

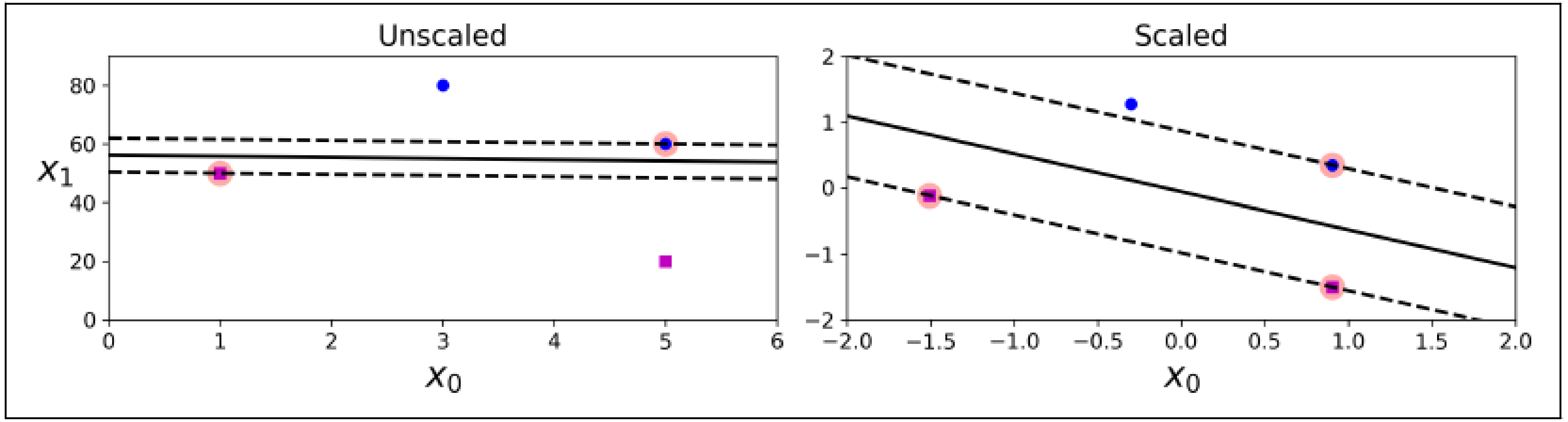
SVM Classifier

- **Linear classification and SVM Classifier**



SVM Classifier

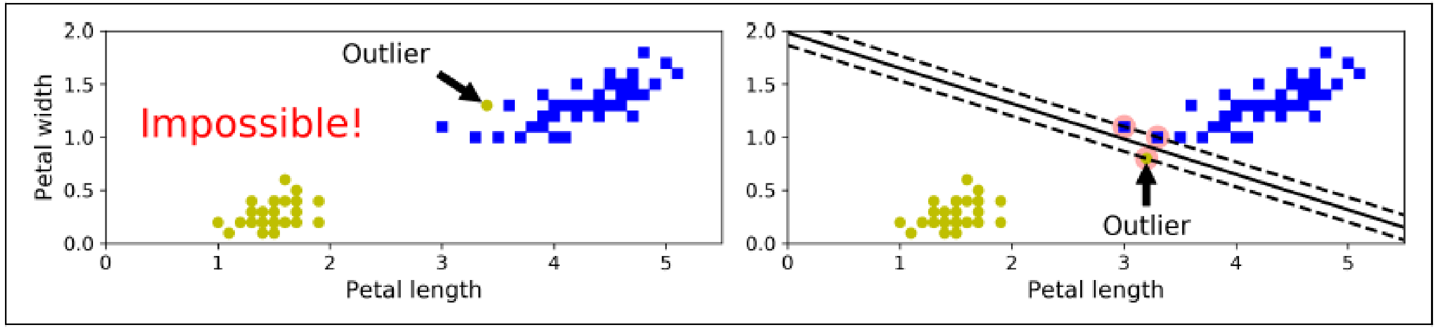
- **Sensitivity to feature scales**



Before Scaling **After Scaling**

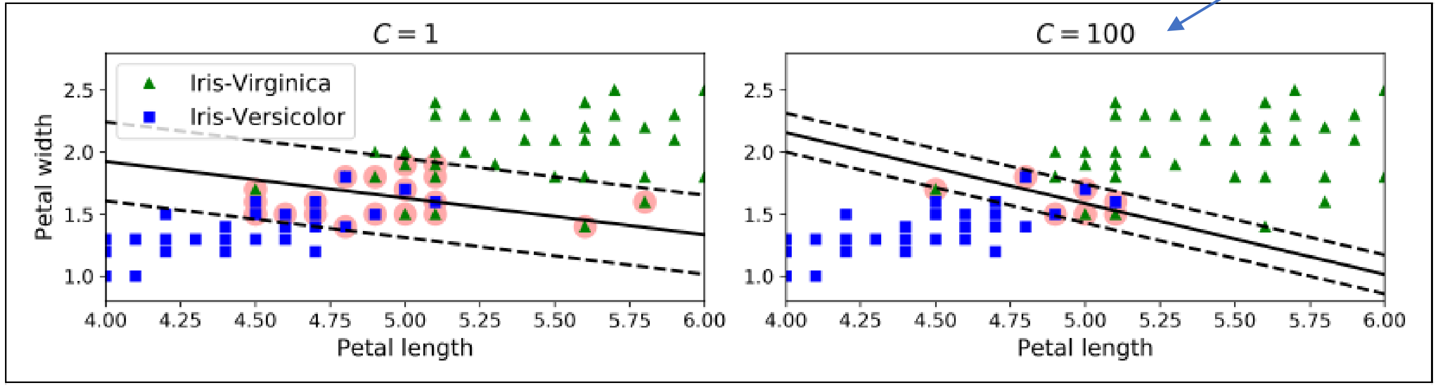
SVM Classifier

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



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

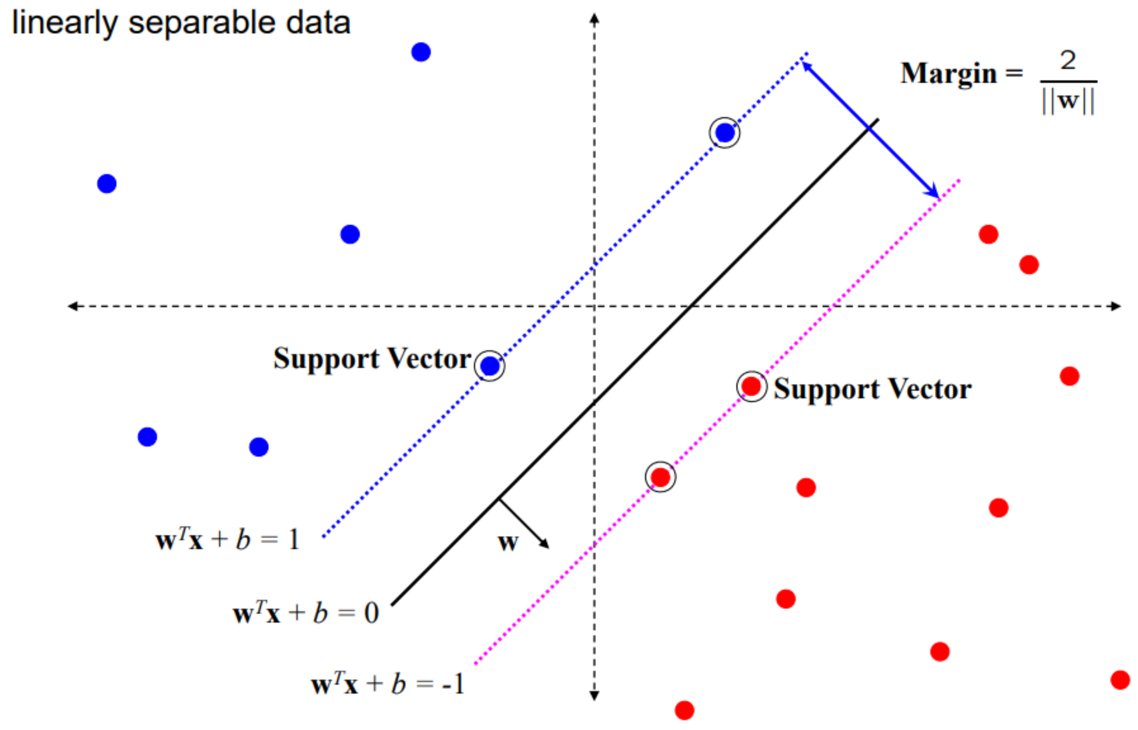
Overfitting 가능성



SVM Classifier

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

$$\vec{w} \bullet \vec{x_+} + b \geq \delta \qquad \vec{w} \bullet \vec{x_+} + b \geq 1$$
normalize
$$\vec{w} \bullet \vec{x_-} + b \leq -\delta \qquad \vec{w} \bullet \vec{x_-} + b \leq -1$$



- 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

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

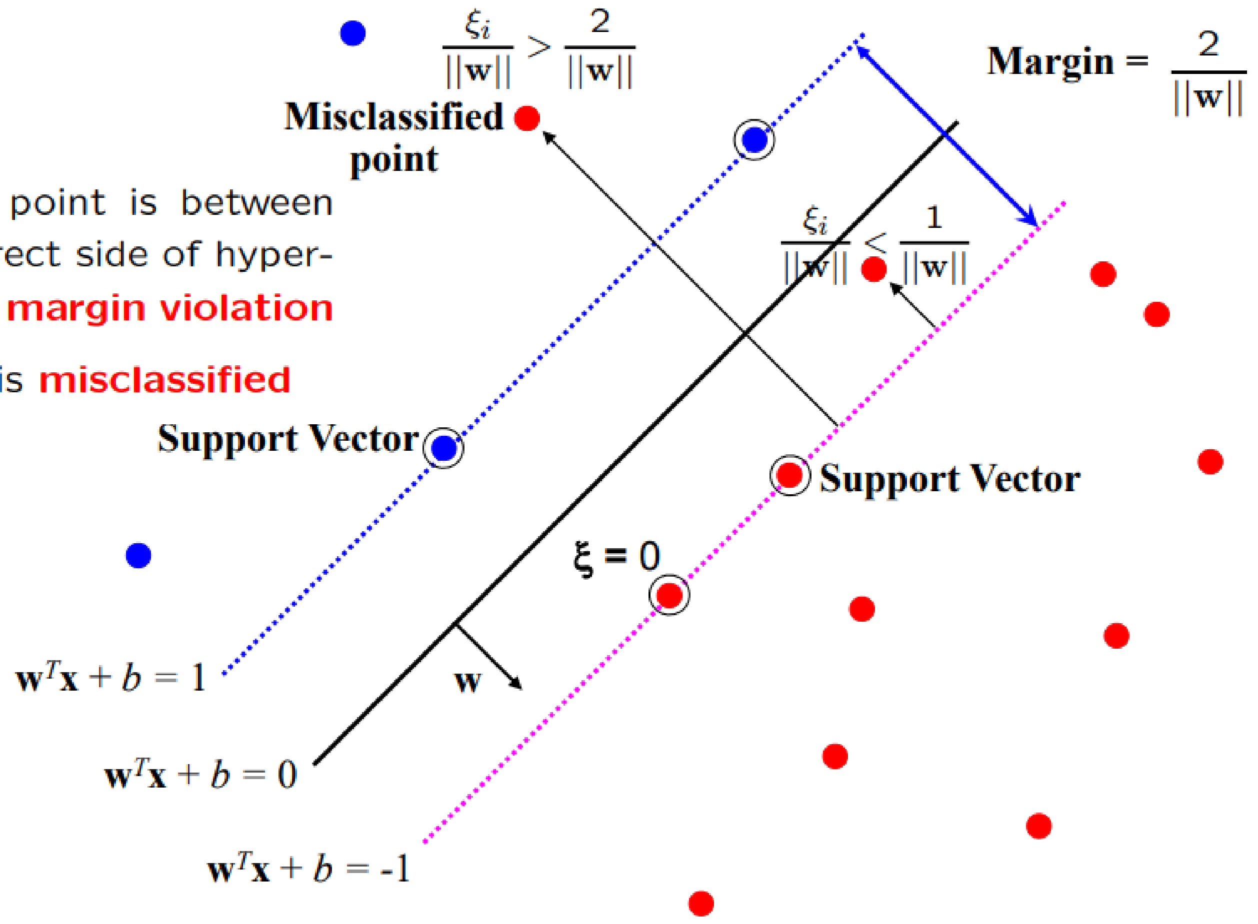
$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x >= 0 \\ 0 & \text{otherwise} \end{cases}$$

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

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

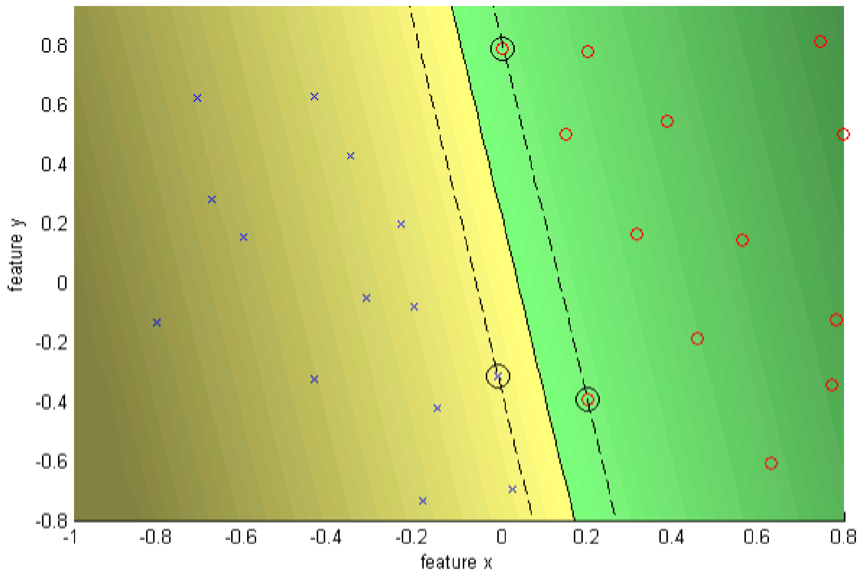
subject to

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

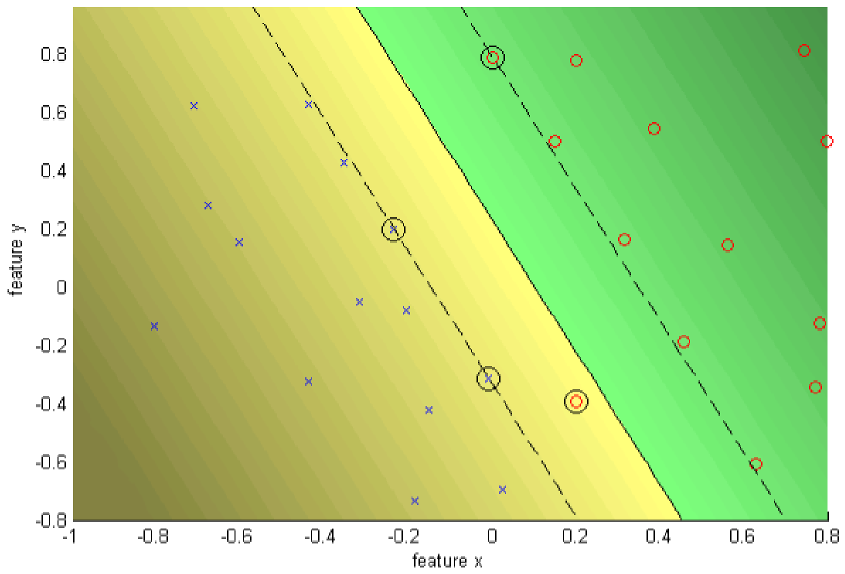
- Every constraint can be satisfied if ξ_i is sufficiently large
- C is a regularization parameter:
  - small C allows constraints to be easily ignored $\rightarrow$ large margin
  - large C makes constraints hard to ignore $\rightarrow$ narrow margin
  - $C = \infty$ enforces all constraints: hard margin
- 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

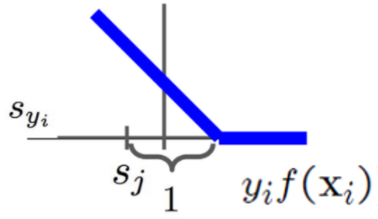
- **Constrained optimization problem**

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

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

$$\xi_i \geq 0$$

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



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

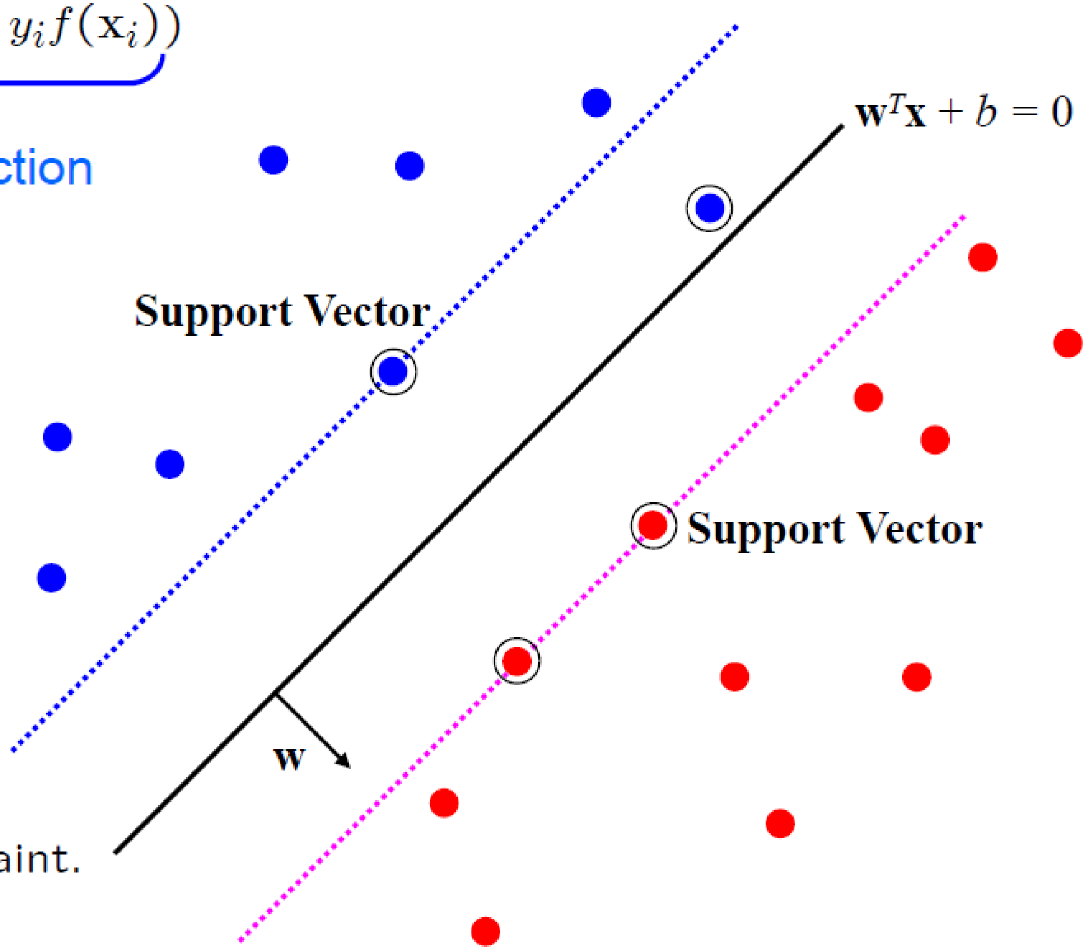
$$\min_{\mathbf{w}\in\mathbb{R}^d} \underbrace{||\mathbf{w}||^2}_{\text{regularization}} + C\sum_i^N \underbrace{\max\left(0, 1 - y_i f(\mathbf{x}_i)\right)}_{\text{loss function}}$$

Loss Function

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

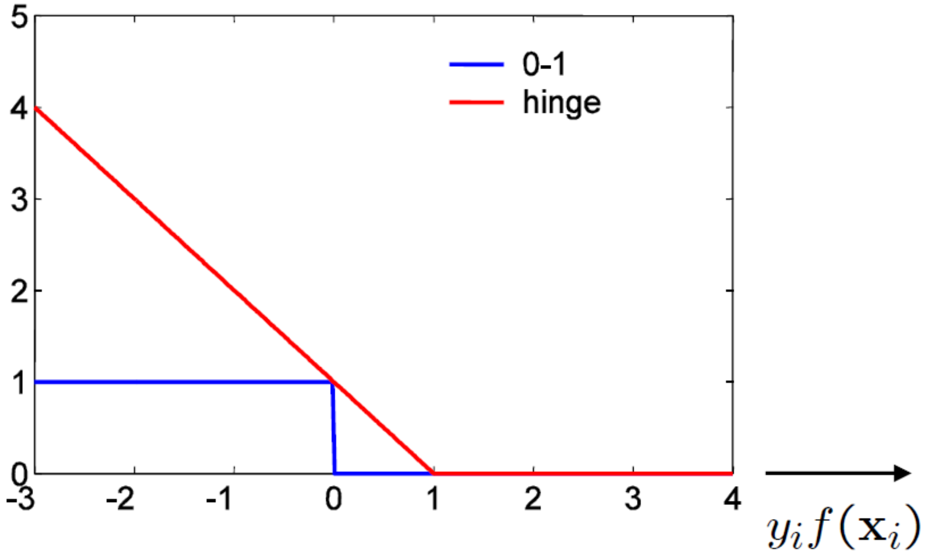
Points are in three categories:

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



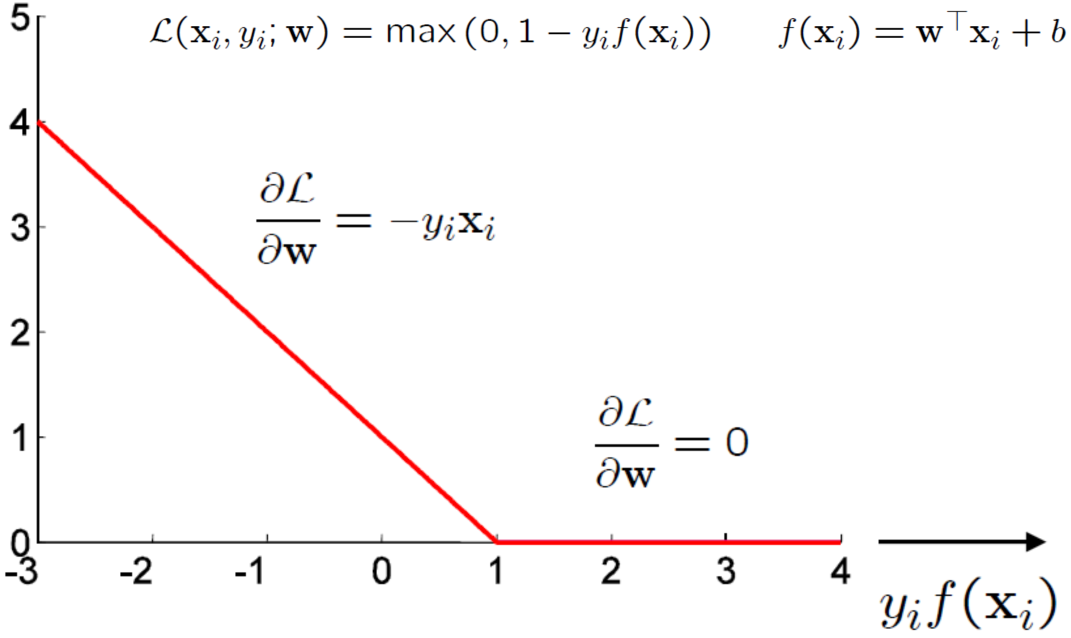
Hinge Loss

- **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

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

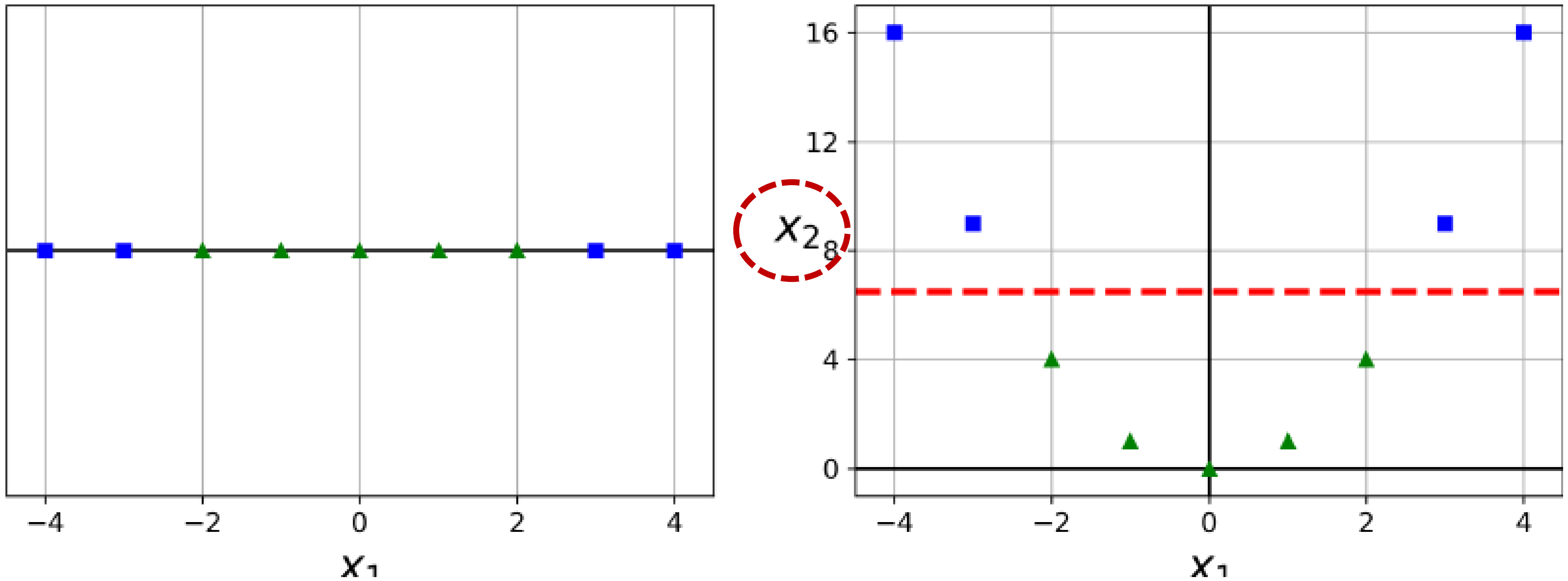
where η is the learning rate.

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

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

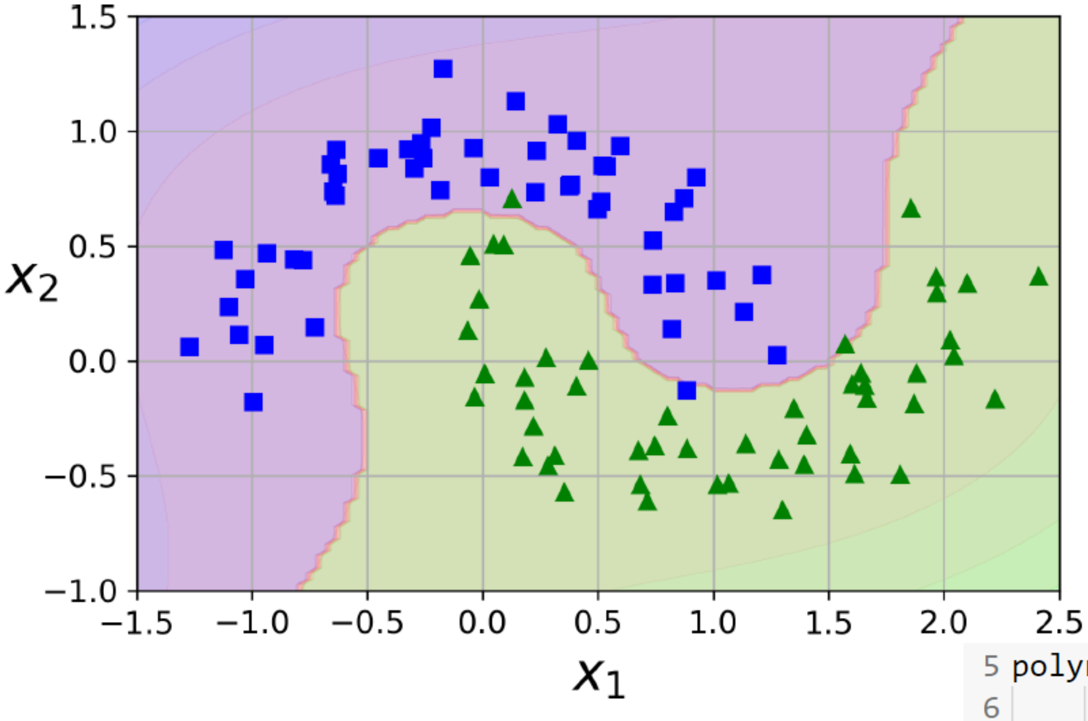
Nonlinear SVM Classifier

- **Adding features to make a dataset linearly separable**
  - Add a second feature $x_2 = (x_1)^2$



Nonlinear SVM Classifier

- **Linear SVM classifier using polynomial features**



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

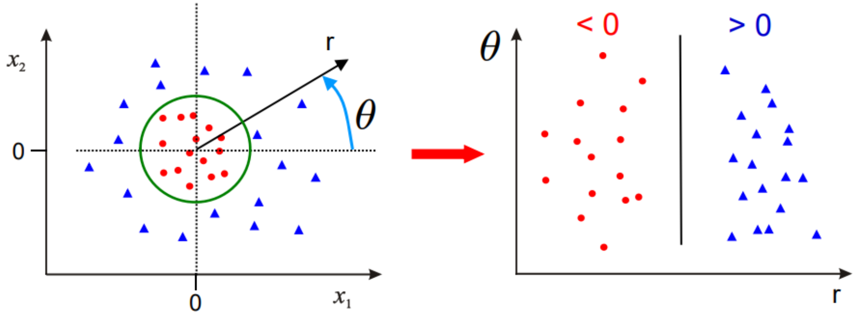
```python
polynomial_svm_clf = Pipeline([
        ("poly_features", PolynomialFeatures(degree=3)),
        ("scaler", StandardScaler()),
        ("svm_clf", LinearSVC(C=10, loss="hinge", random_state=42))
    ])

polynomial_svm_clf.fit(X, y)
```

Nonlinear SVM Classifier

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

Use polar coordinates

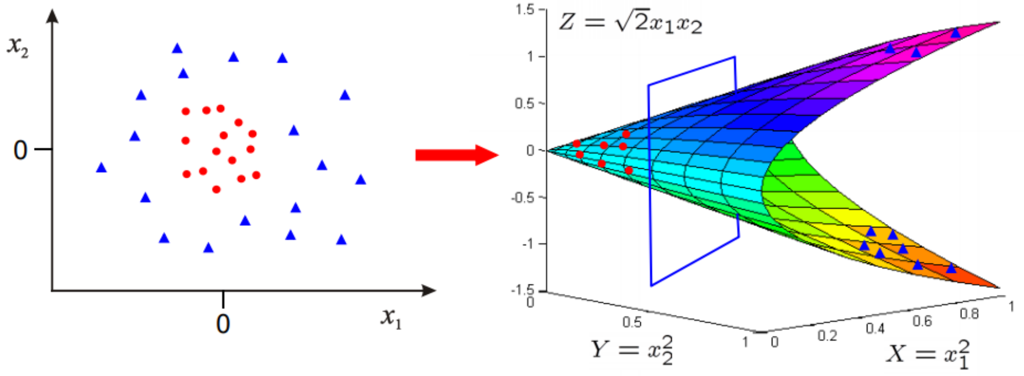


- Data is linearly separable in polar coordinates
- Acts non-linearly in original space

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

Map data to higher dimension

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



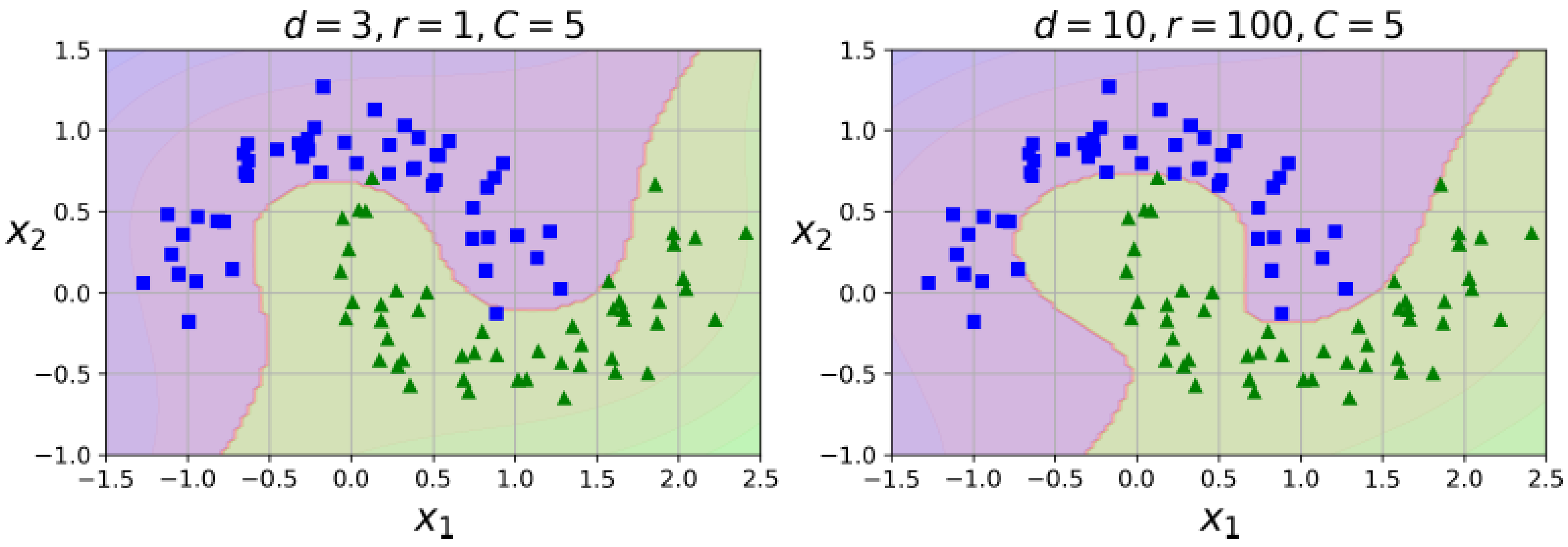
Feature space

Non-linear Classification - 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

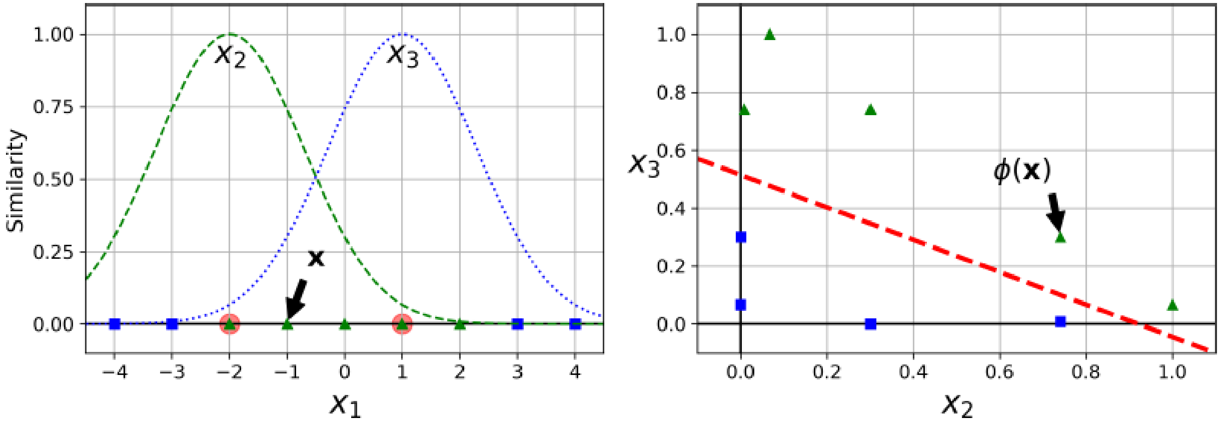


SVM Kernels

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

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

original feature:
x_1 = -1

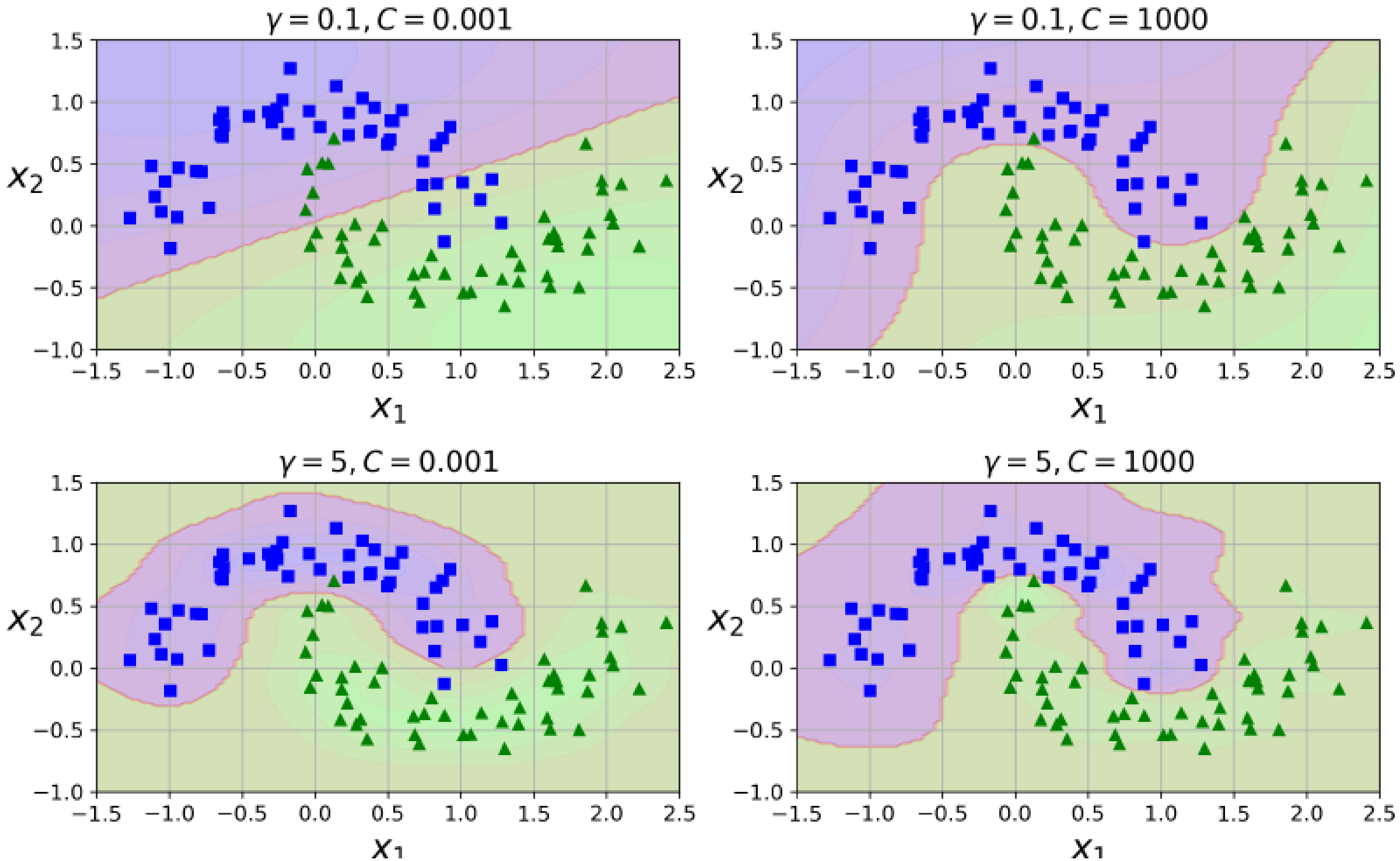


Take two landmarks at x_1=-2, x_1=1
then, new features are:
x_2 = exp (–0.3 × 1^2) ≈ 0.74
x_3 = exp (–0.3 × 2^2) ≈ 0.30
now, linearly separable.

- Create landmarks at the location of each instance, then (assuming drop of the original features)
  m instances, *n* features -> *m* instances, *m* features (large features !)

SVM Kernels

- **Gaussian RBF Kernel**



Optimization – Lagrangian multiplier

<u>Optimization using Lagrange multiplier</u>

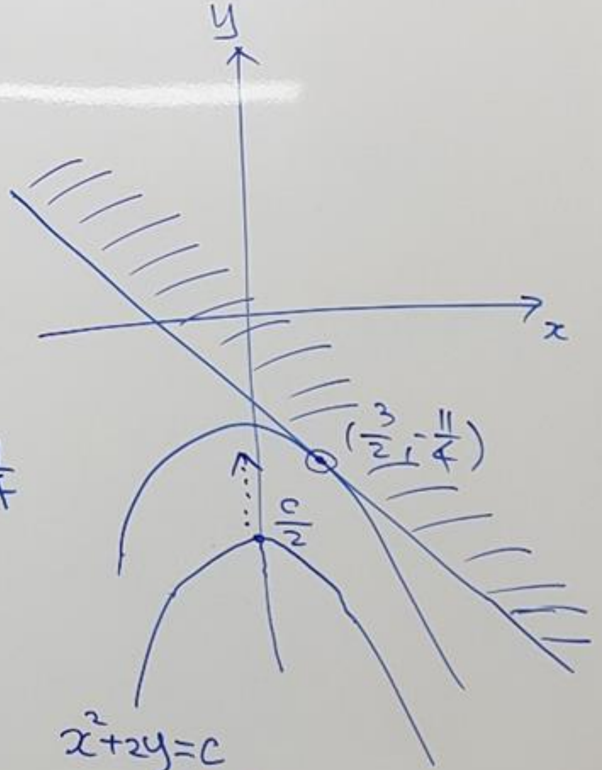
prob: $\min f(x,y) = x^2 + 2y$

s.t. $3x + 2y + 1 = 0$

$\Rightarrow$ define Lagrange function

$$g(x,y,\alpha) \triangleq f(x,y) - \alpha(3x+2y+1)$$
$$= x^2 + 2y - \alpha(3x+2y+1)$$

$$\left.\begin{aligned} \frac{\partial g}{\partial x} &= 2x - 3\alpha = 0 \\ \frac{\partial g}{\partial y} &= 2 - 2\alpha = 0 \\ \frac{\partial g}{\partial \alpha} &= 3x + 2y + 1 = 0 \end{aligned}\right\} \begin{aligned} & x = \frac{3}{2},\ y = -\frac{11}{4} \\ & \alpha = 1 \end{aligned}$$



$x^2 + 2y = c$

$y = -\frac{1}{2}x^2 + \frac{c}{2}$

$\Rightarrow$ what about inequality?

(ex) $3x + 2y + 1 \geq 0$

$\rightarrow$ In some conditions, it can be generalized.

Optimization – SVM Hard margin

$$t_i = \begin{cases} +1 & (y_i = 1) \\ -1 & (y_i = 0) \end{cases}$$

SVM.

Hard margin SVM problem

$$\min_{w,b} \frac{1}{2} w^T \cdot w \quad \left(= \min_{w,b} \|w\|^2\right)$$

subject to $t_i(w^T x_i + b) \geq 1$, for $i = 1, 2 \cdots m$

⇒ Generalized Lagrangian

$$\mathcal{L}(w,b,\alpha) \triangleq \frac{1}{2} w^T w - \sum_{i=1}^{m} \alpha_i \left(t_i(w^T x_i + b) - 1\right)$$

(Luckily, SVM problem meets the generalization condition.)

$$\begin{pmatrix} \nabla_w \mathcal{L}(w,b,\alpha) = w - \sum_{i=1}^{m} \alpha_i t_i x_i \\ \frac{\partial}{\partial b} \mathcal{L}(w,b,\alpha) = - \sum_{i=1}^{m} \alpha_i t_i \end{pmatrix}$$

⇒ $\hat{w} = \sum_{i=1}^{m} \hat{\alpha}_i t_i x_i$, $\sum_{i=1}^{m} \hat{\alpha}_i t_i = 0$

⇒ Generalized Lagrange function is

$$\mathcal{L}(\hat{w}, \hat{b}, \alpha) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j t_i t_j x_i^T x_j - \sum_{i=1}^{m} \alpha_i$$

with $\alpha_i \geq 0$ for $i = 1, 2, \cdots m$

Now, the goal is to find the vector $\hat{\alpha}$ that minimizes this fuction, with $\hat{\alpha}_i \geq 0$ for all instances.

Primal and Dual formulations

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

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

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

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:

$$f(\mathbf{x}) = \sum_i^N \alpha_i y_i \underline{\mathbf{x}_i{}^\top \mathbf{x}} + b \quad \longleftarrow \quad \text{wx + b}$$

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

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 dimensional 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

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

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

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!

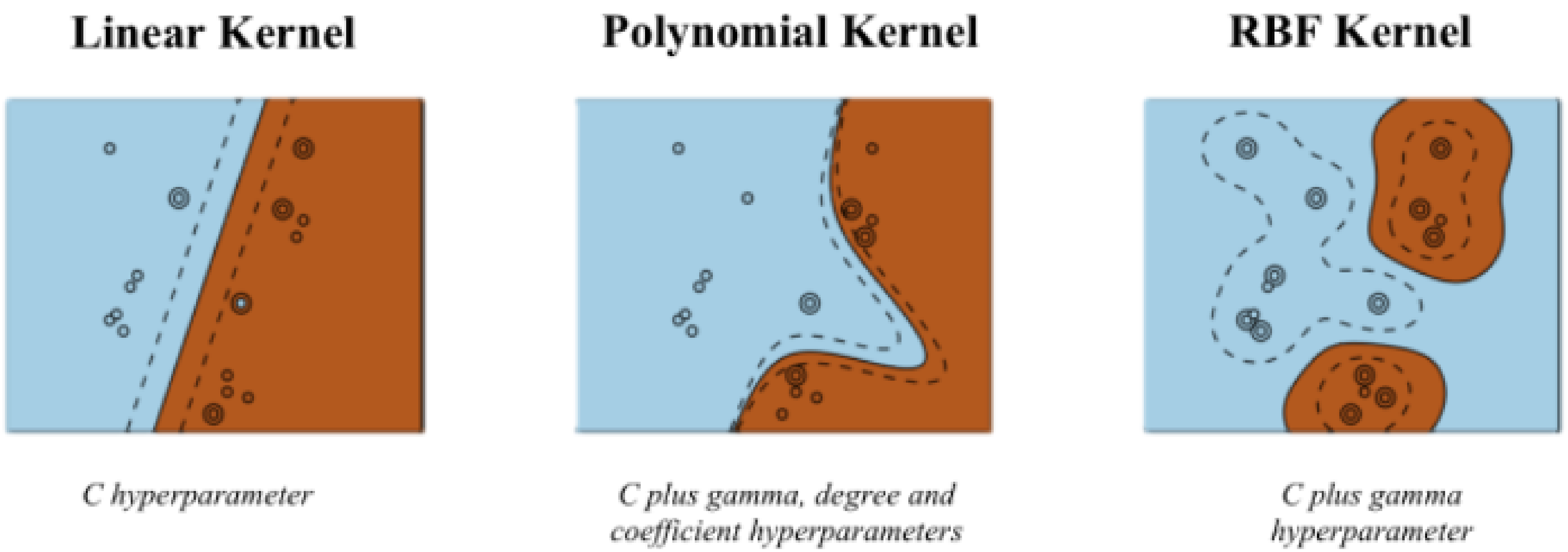
$$\text{Linear:} \quad K(\mathbf{a}, \mathbf{b}) = \mathbf{a}^T\mathbf{b}$$

$$\text{Polynomial:} \quad K(\mathbf{a}, \mathbf{b}) = \left(\gamma\mathbf{a}^T\mathbf{b} + r\right)^d$$

$$\text{Gaussian RBF:} \quad K(\mathbf{a}, \mathbf{b}) = \exp\left(-\gamma\| \mathbf{a} - \mathbf{b} \|^2\right)$$

$$\text{Sigmoid:} \quad K(\mathbf{a}, \mathbf{b}) = \tanh\left(\gamma\mathbf{a}^T\mathbf{b} + r\right)$$

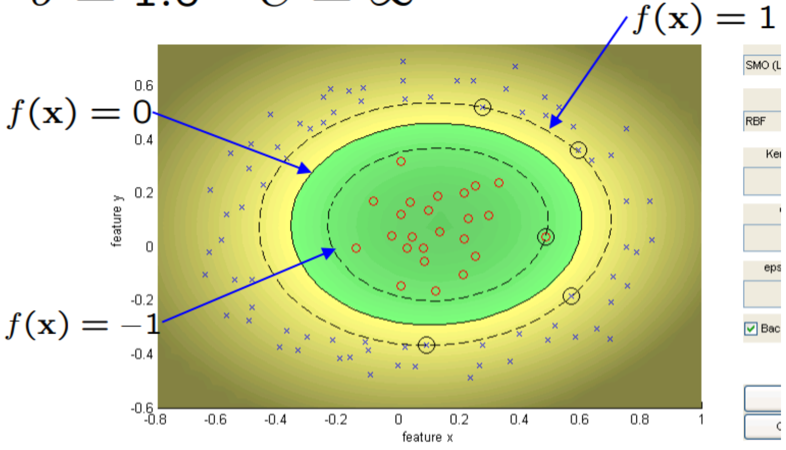
Common SVM Kernels



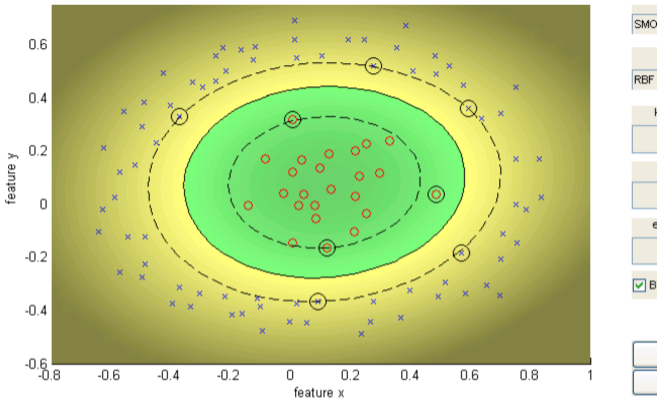
RBF Kernel SVM Example

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

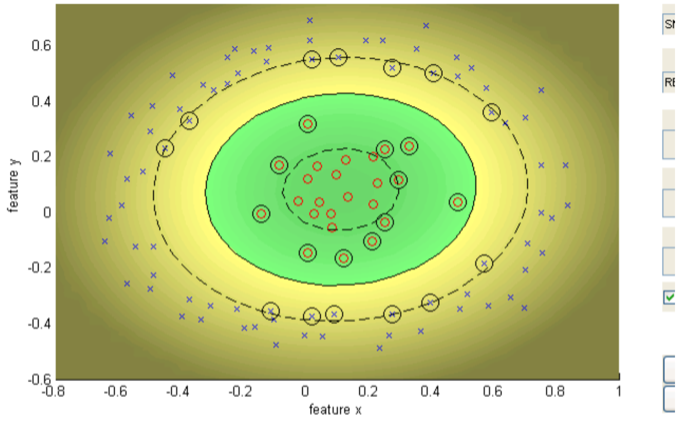
$\sigma = 1.0 \quad C = \infty$



$\sigma = 1.0 \quad C = 100$

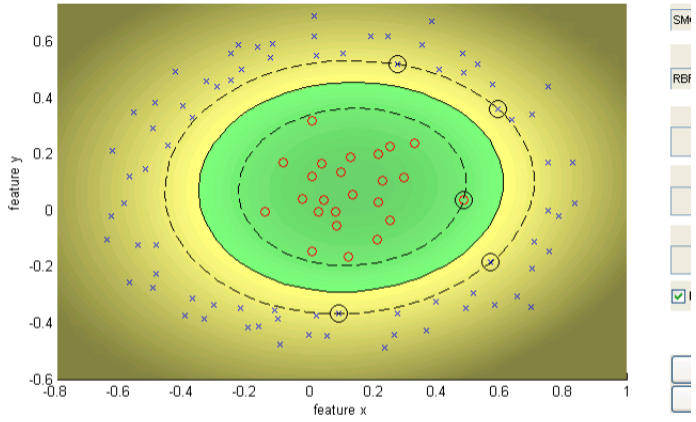


$\sigma = 1.0 \quad C = 10$

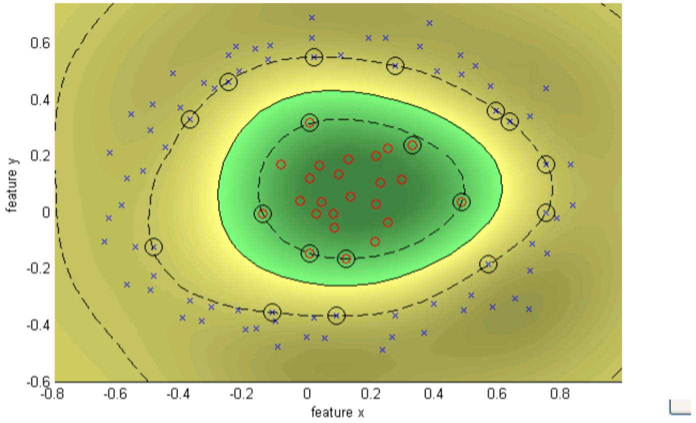


Decrease C -> wider (soft) margin

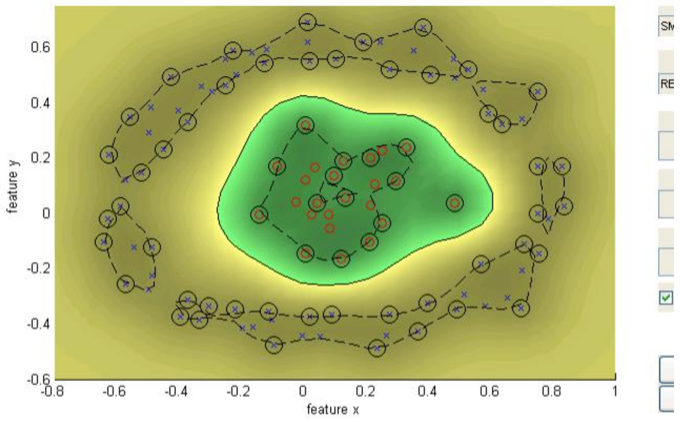
$\sigma = 1.0 \quad C = \infty$



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



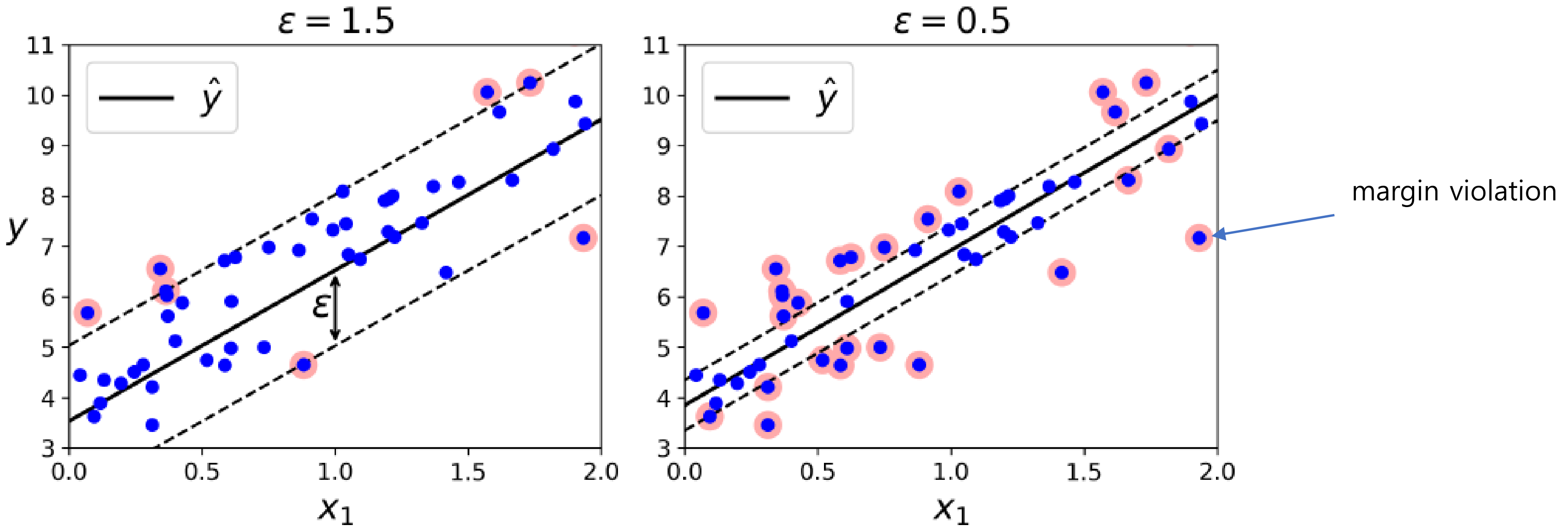
$\sigma = 0.1 \quad C = \infty$



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

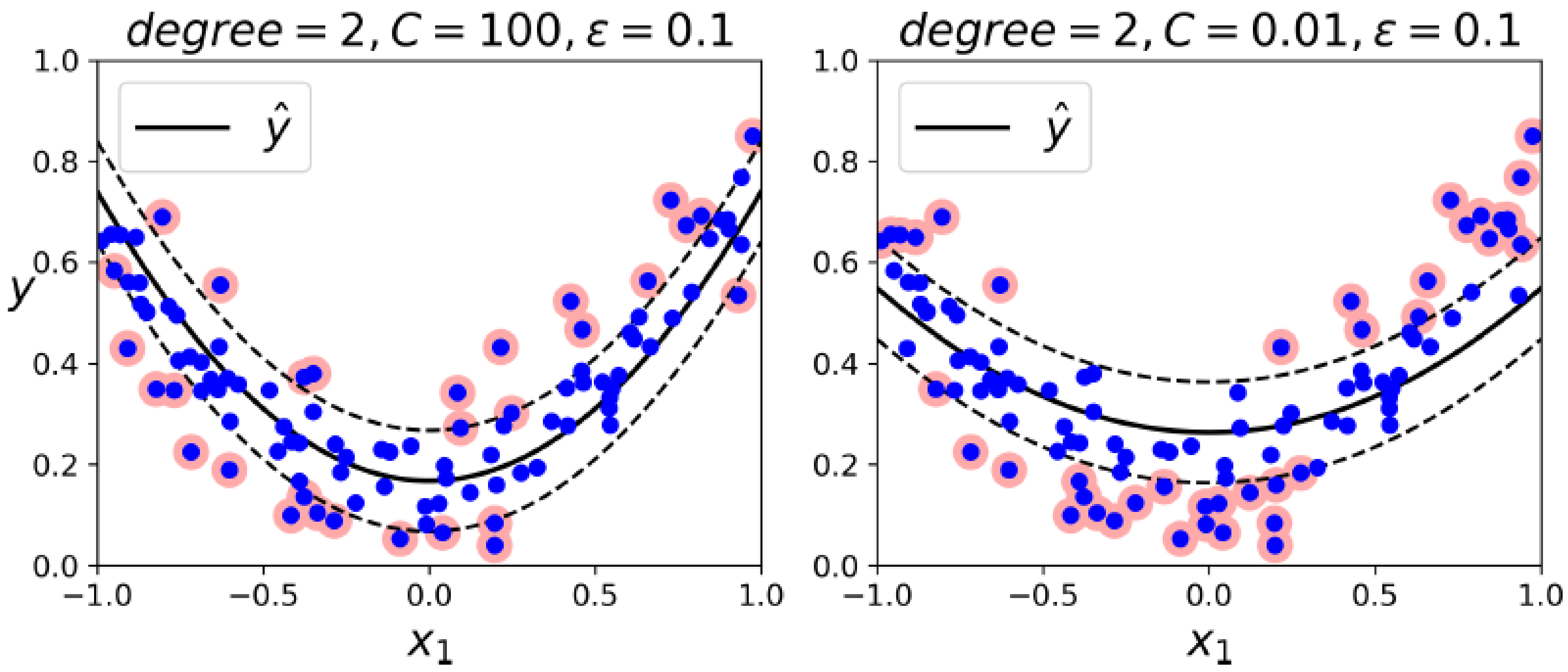
SVM Regression

- **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 hyper-parameter ϵ.



SVM Regression

- **SVM regression using 2^{nd} degree polynomial kernel**



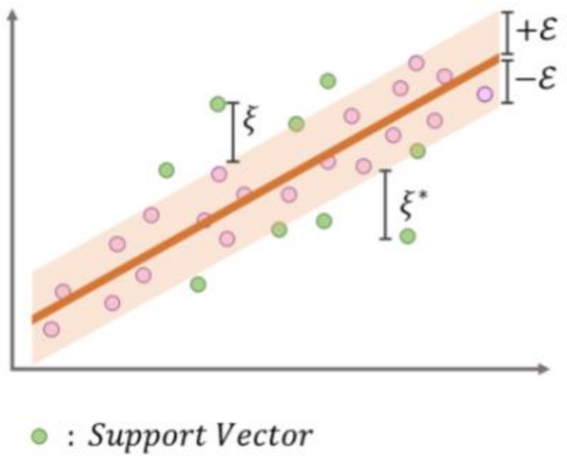
SVM Regression

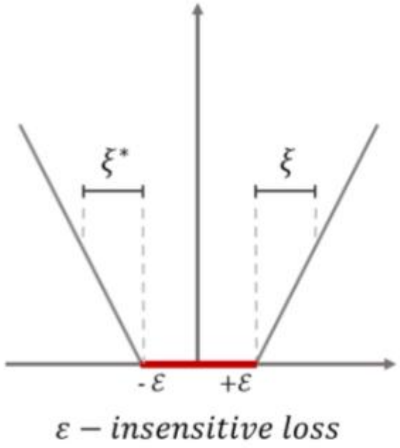
- **Loss function: ε-insensitive loss:**
  - Ignores errors that are within ε distance by treating them as zero
  - Measured based on the distance between observed value y and the ε boundary

$$L_\varepsilon = \begin{cases} 0 & \text{if } |y - f(x)| \le \varepsilon \\ |y - f(x)| - \varepsilon & \text{otherwise} \end{cases}$$

$$L_{SVR} = \min \overbrace{\frac{1}{2}\|w\|^2}^{\text{Robustness}} + \underbrace{C\sum_{i=1}^{n}(\xi_i + \xi_i^*)}_{\text{loss funciton}}$$

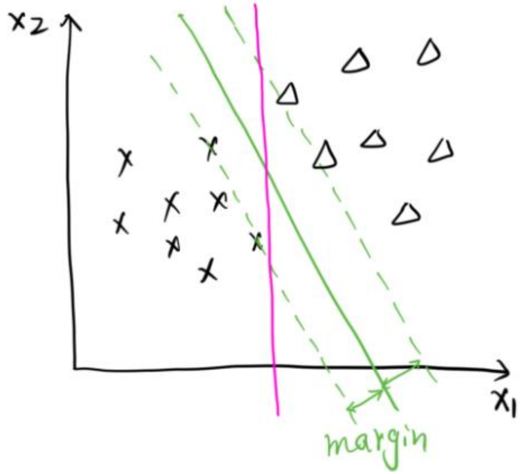
$$\begin{aligned} s.t. \quad & (w^T x_i + b) - y_i \le \epsilon + \xi_i \\ & y_i - (w^T x_i + b) \le \epsilon + \xi_i^* \\ & \xi_i, \xi_i^* \ge 0 \end{aligned}$$

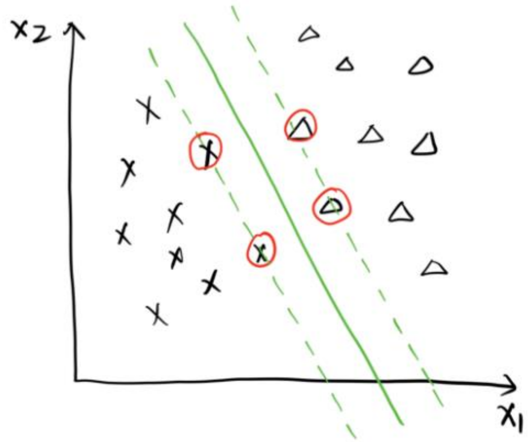


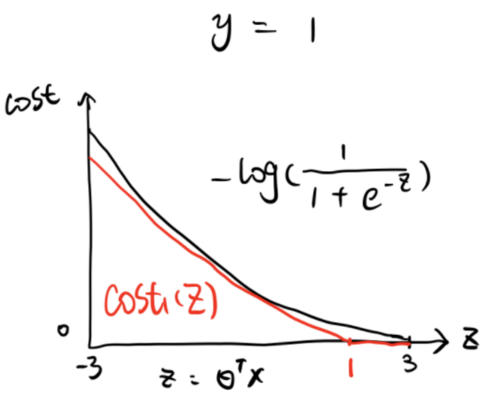


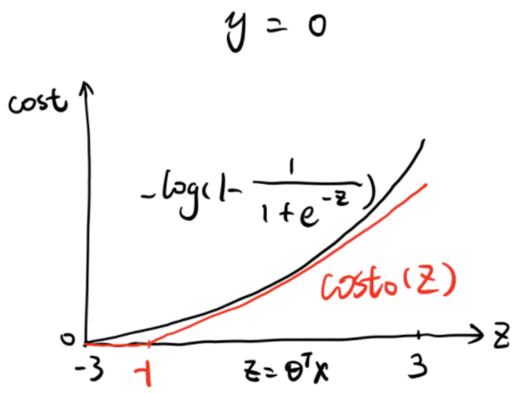
SVM Summary

- **Linear SVM**
  - Concept
  - support vectors
  - Loss function
  - Hypothesis









$$h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T x >= 0 \\ 0 & \text{otherwise} \end{cases}$$

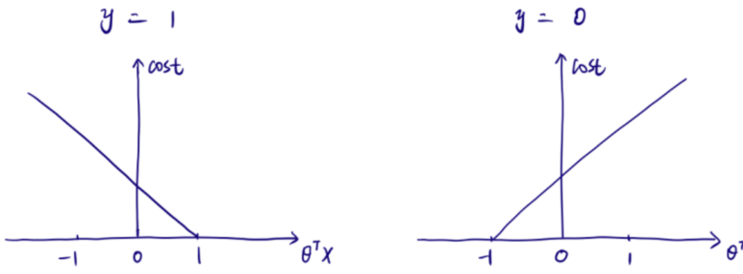
https://towardsdatascience.com/optimization-loss-function-under-the-hood-part-iii-5dff33fa015d

SVM Summary

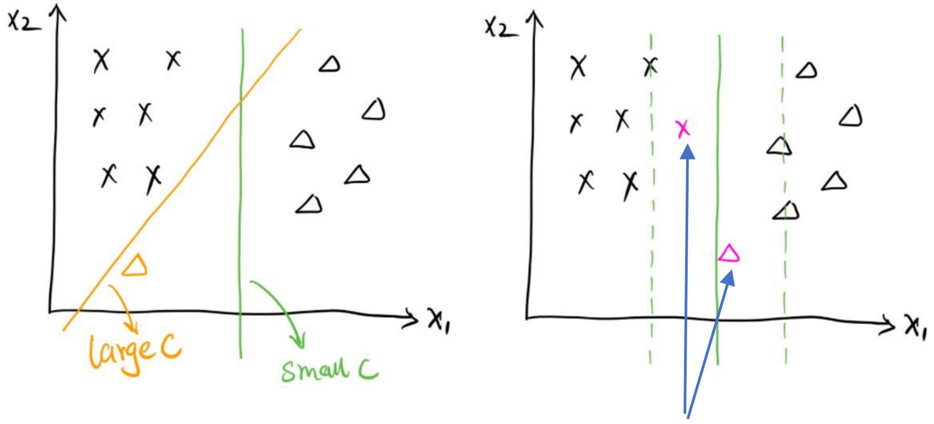
- **SVM cost function**

$$J(\theta) = C\left[\sum_{i=1}^{m} y^{(i)} Cost_1(\theta^T(x^{(i)}) + (1 - y^{(i)})Cost_0(\theta^T(x^{(i)})\right] + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$$

$$m = number\ of\ samples, \quad n = number\ of\ features$$



$$Cost(h_\theta(x), y) = \begin{cases} max(0, 1 - \theta^T x) & \text{if } y = 1 \\ max(0, 1 + \theta^T x) & \text{if } y = 0 \end{cases}$$

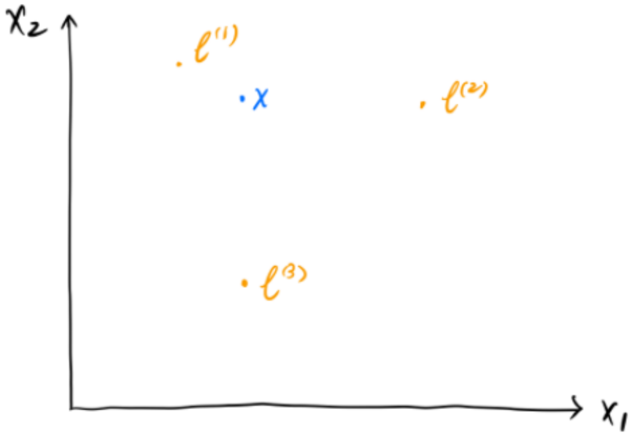


Violations when C is small

https://towardsdatascience.com/optimization-loss-function-under-the-hood-part-iii-5dff33fa015d

SVM Summary

- **Non-linear (rbf) SVM**
  - Hypothesis and cost function are almost the same (x -> f)
  - Define landmarks to see how close x is to them (similarity) – kernel function
  - Gaussian kernel (RBF: radial basis function) – basically the same (use γ to represent $1/2\sigma^2$)
  - Now we have new features (f1,f2,f3) instead of x1 and x2.
  - Prediction: $\theta^T f = \theta 0 + \theta 1 f1 + \theta 2 f2 + \theta 3 f3$



$$f1 = \text{Similarity}(x, l^{(1)}) \text{ or } k(x, l^{(1)})$$

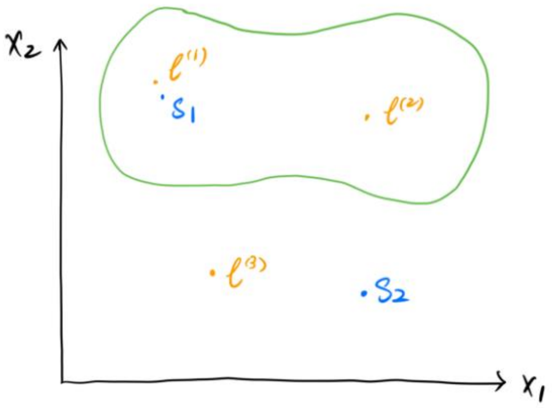
$$f2 = \text{Similarity}(x, l^{(2)}) \text{ or } k(x, l^{(2)})$$

$$f3 = \text{Similarity}(x, l^{(3)}) \text{ or } k(x, l^{(3)})$$

$$f_1 = Similarity(x, l^{(1)}) = exp(\frac{\|x - l^{(1)}\|^2}{2\sigma^2})$$

$$f_2 = Similarity(x, l^{(2)}) = exp(\frac{\|x - l^{(2)}\|^2}{2\sigma^2})$$

$$f_3 = Similarity(x, l^{(3)}) = exp(\frac{\|x - l^{(3)}\|^2}{2\sigma^2})$$



https://towardsdatascience.com/optimization-loss-function-under-the-hood-part-iii-5dff33fa015d

SVM Summary

- **Non-linear (rbf) SVM**

$$Given\ (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots\ldots (x^{(m)}, y^{(m)})$$

$$Choose\ l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots\ldots l^{(m)} = x^{(m)}$$

$$m\ =\ number\ of\ samples$$

$$Given\ the\ i^{th}\ sample\ x^{(i)}\ :$$

$$f_1^{(i)} = k(x^{(i)}, l^{(1)})$$

$$f_2^{(i)} = k(x^{(i)}, l^{(2)})$$

$$\ldots\ldots$$

$$f_i^{(i)} = k(x^{(i)}, l^{(i)})$$

$$\ldots\ldots$$

$$f_m^{(i)} = k(x^{(i)}, l^{(m)})$$

$$where\ x^{(i)} = l^{(i)},\ f_i^{(i)} = 1$$

$$Hypothesis:\quad h_\theta(x) = \begin{cases} 1 & \text{if } \theta^T f >= 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\theta^T f = \theta_0 f_0 + \theta_1 f_1 + \ldots + \theta_m f_m$$

$$Regularized\ Cost\ Function:$$

$$J(\theta) = C[\sum_{i=1}^{m}[y^{(i)} Cost_1(\theta^T(f^{(i)}) + (1 - y^{(i)}) Cost_0(\theta^T(f^{(i)})] + \frac{1}{2}\sum_{j=1}^{m}\theta_j^2$$

https://towardsdatascience.com/optimization-loss-function-under-the-hood-part-iii-5dff33fa015d