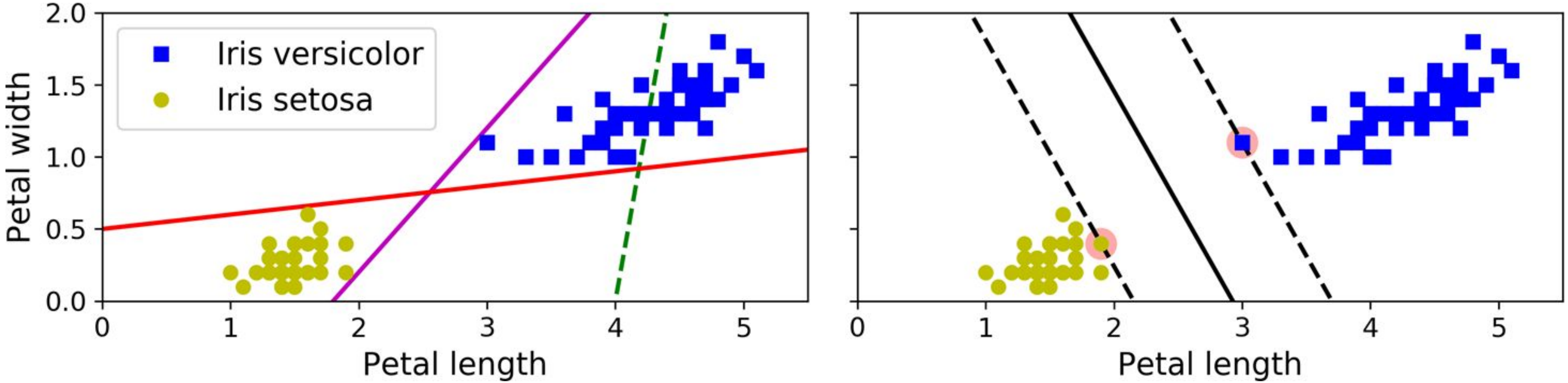
Support Vector Machine

Neil Zhang

ECE 208/408 – The Art of Machine Learning

Linear SVM classification

SVM finds the hyperplane that separates the classes with the **widest margin**



(Figure from Géron figure 5-1)

Optimal margin classifier

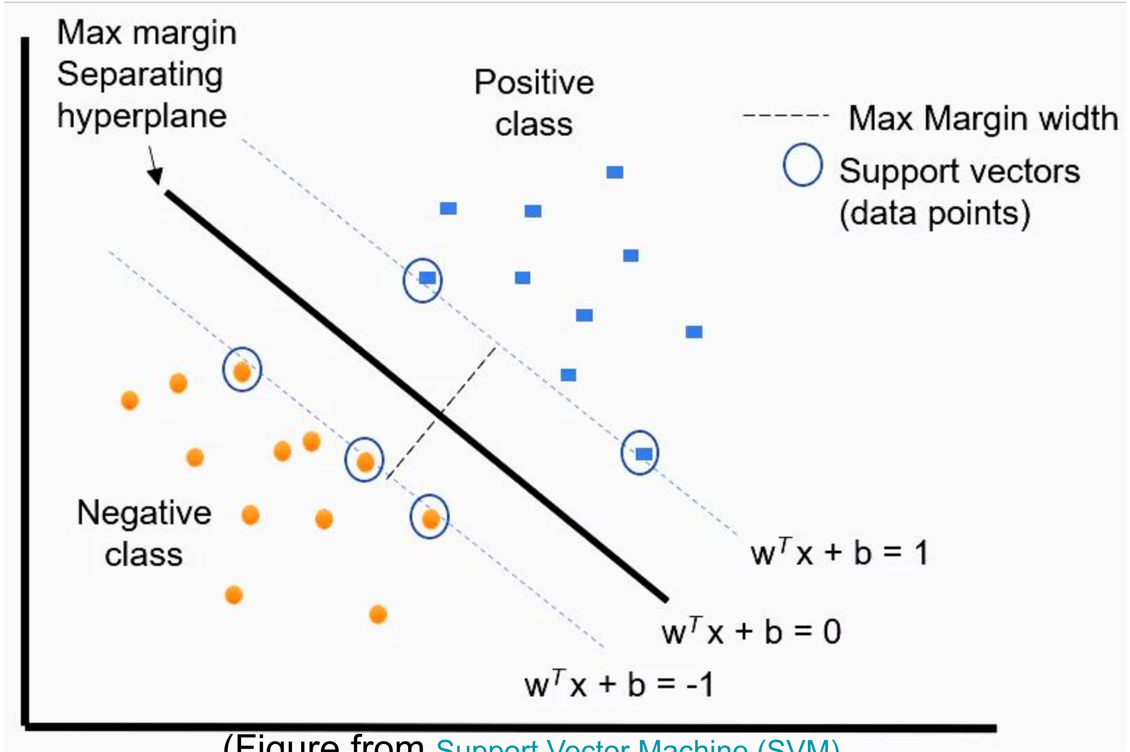
Objective function

$$\min_{\mathbf{w}} \ \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$\text{s.t.} \ \ y_n(\mathbf{w}^T\mathbf{x}^{(n)} + b) \geq 1$$

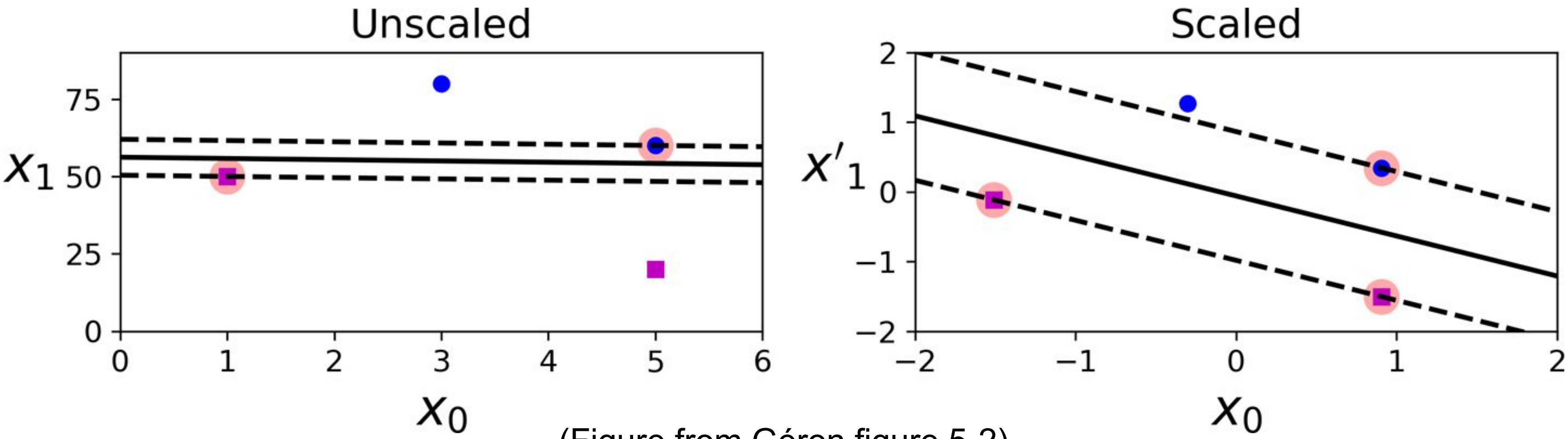
$$\forall n.$$

Derive on blackboard



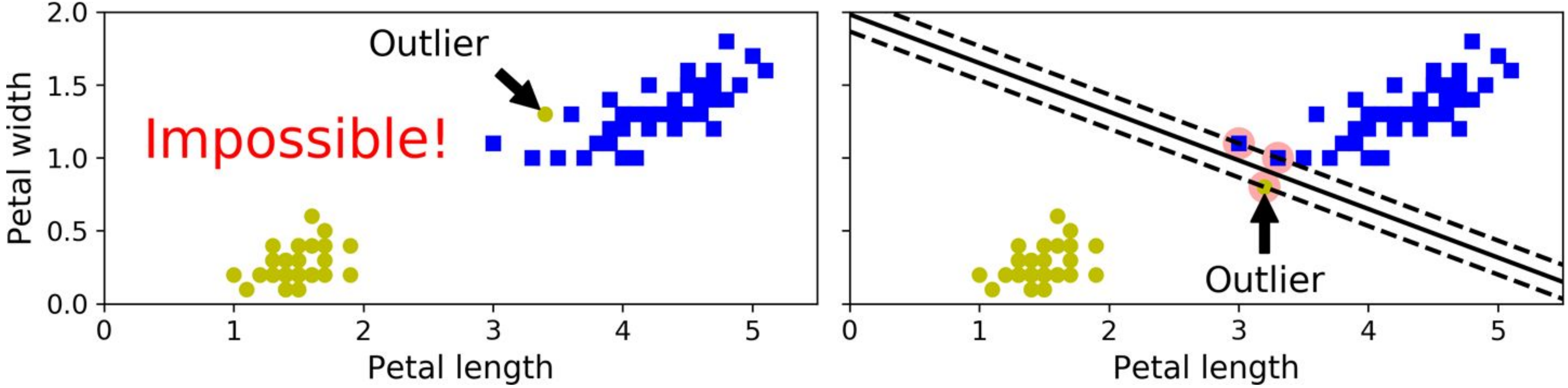
(Figure from Support Vector Machine (SVM) basics and implementation in Python)

Sensitive to feature scales



(Figure from Géron figure 5-2)

Sensitive to outliers



(Figure from Géron figure 5-3)

Soft margin classification

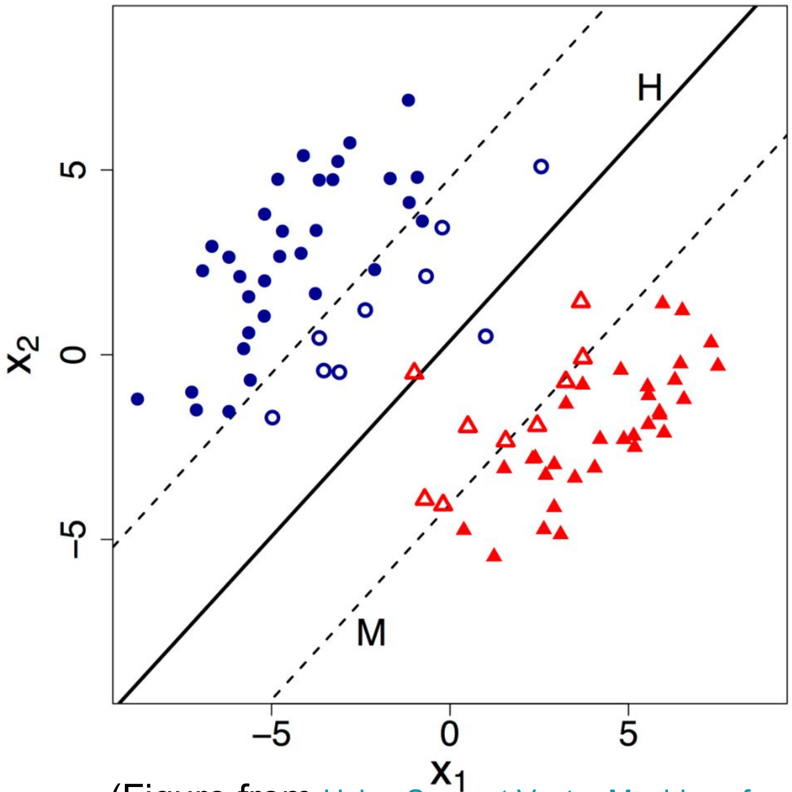
Introduce a slack variable ξ_n to allow margin violations

$$\min_{\mathbf{w},b,\xi} \ \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}\xi_n$$

$$\text{s.t.} \ \ y_n(\mathbf{w}^T\mathbf{x}^{(n)} + b) + \xi_n \geq 1,$$

$$\xi_n \geq 0, \ \ \forall n.$$

Derive on blackboard

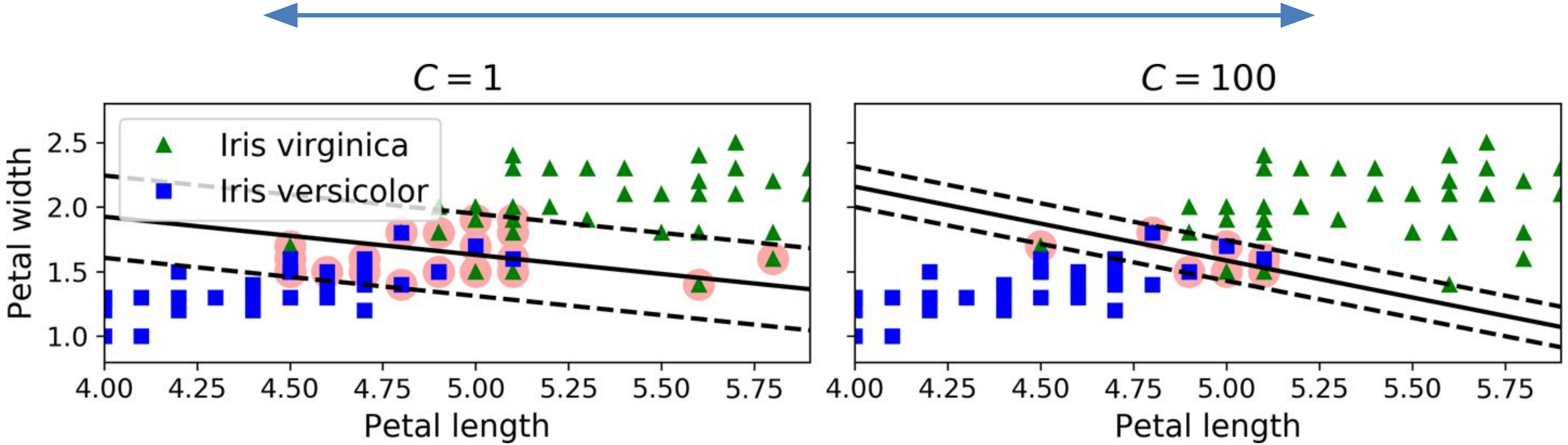


(Figure from Using Support Vector Machines for Survey Research | Published in Survey Practice)

Soft margin classification

More margin violations
Underfitting

Less margin violations
Overfitting



(Figure from Géron figure 5-4)

Summary of linear SVM classifier

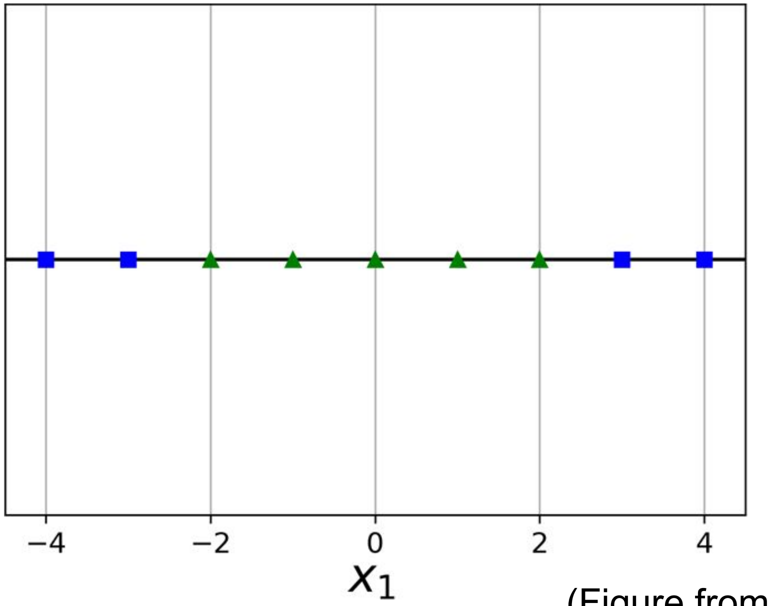
Find the maximized margin between two classes

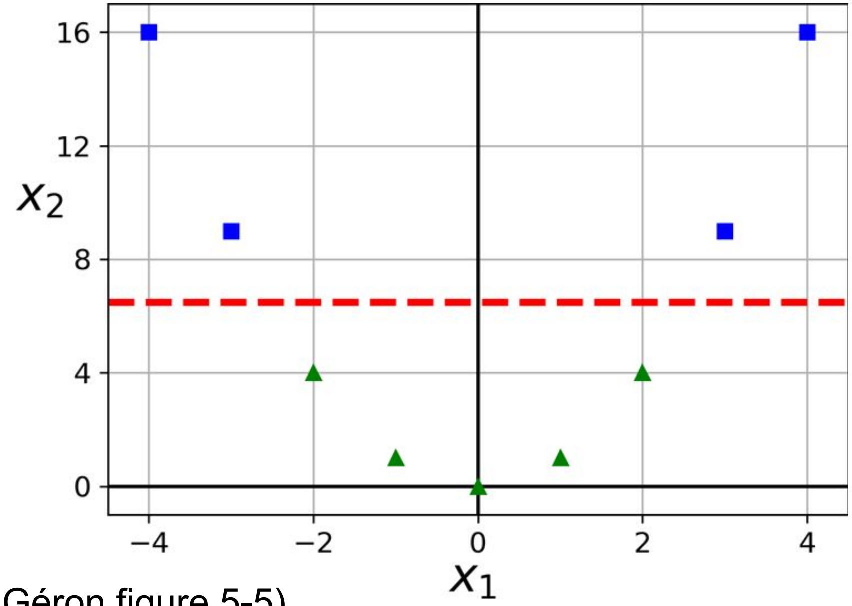
Soft margin classification allows margin violations controlled by a hyperparameter C

https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

Non-linearly separable

Add a non-linear feature $x_2 = (x_1)^2$

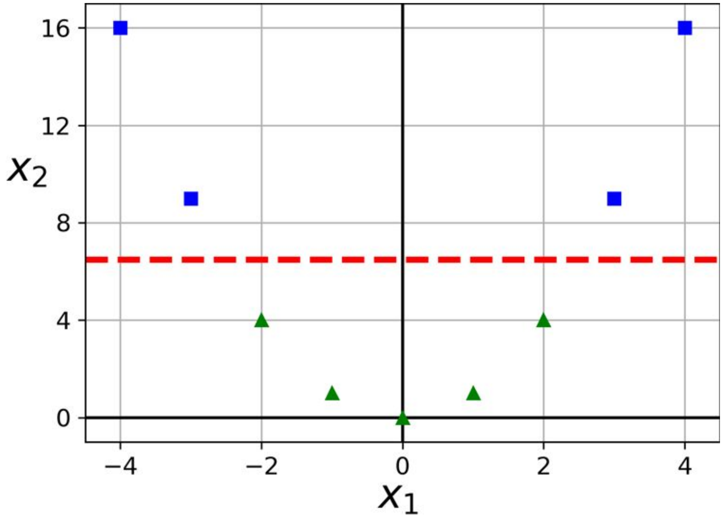




(Figure from Géron figure 5-5)

Adding polynomial features

- Adding polynomial features works with many ML models
  - High polynomial degree creates a huge number of features



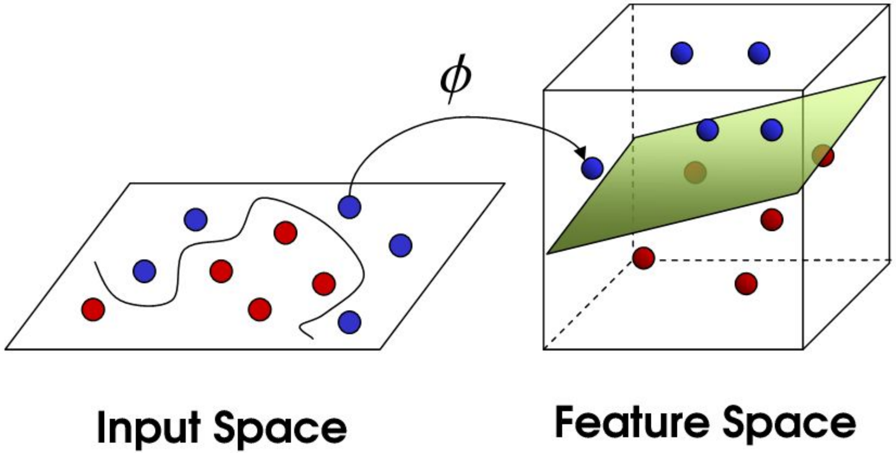
How we handle non-linearity?

Feature mapping

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}\xi_n$$

$$\text{s.t. } y_n\left(\mathbf{w}^T\phi(\mathbf{x}^{(n)}) + b\right) + \xi_n \geq 1$$

$$\xi_n \geq 0, \quad \forall n$$



(Figure from The Kernel Trick in Support Vector Classification | by Drew Wilimitis | Towards Data Science)

Inner product is computationally expensive

To solve the optimization problem, we need to calculate the dot products of the transformed features.

$$
\begin{aligned}
w^T x + b &= \left(\sum_{i=1}^{n} \alpha_i y^{(i)} x^{(i)} \right)^T x + b \\
&= \sum_{i=1}^{n} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b.
\end{aligned}
$$

Kernel trick

Get the same result without adding the polynomial features.

$$k(x, x') = \boldsymbol{\phi}(x)^{\mathrm{T}} \boldsymbol{\phi}(x') = \sum_{i=1}^{M} \phi_i(x) \phi_i(x')$$

We don’t need to apply the underlying transformations to the features, as long as the kernel preserves the inner product.

An example kernel

$$K(x,z) = (x^T z)^2.$$

$$\begin{aligned} K(x,z) &= \left(\sum_{i=1}^{d} x_i z_i\right)\left(\sum_{j=1}^{d} x_j z_j\right) \\ &= \sum_{i=1}^{d}\sum_{j=1}^{d} x_i x_j z_i z_j \\ &= \sum_{i,j=1}^{d} (x_i x_j)(z_i z_j) \end{aligned}$$

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix}$$

Why applying kernel trick is better?

Reduce computational complexity

In this case,

$O(d^2)$ -> $O(d)$

$$\phi(x) = \begin{bmatrix} x_1x_1 \\ x_1x_2 \\ x_1x_3 \\ x_2x_1 \\ x_2x_2 \\ x_2x_3 \\ x_3x_1 \\ x_3x_2 \\ x_3x_3 \end{bmatrix}$$

$$K(x, z) = \sum_{i,j=1}^{d} (x_i x_j)(z_i z_j)$$

Test a kennel is valid or not

Find the underlying transformation ϕ

$$k(x, x') = \boldsymbol{\phi}(x)^{\mathrm{T}} \boldsymbol{\phi}(x')$$

A better way (Out of scope):

Gram matrix should be positive semi-definite.

Polynomial kernel

degree-*M* polynomials

$$K(x, x') = \left(x^T x' + c\right)^M$$

Gaussian Radial Basis Function (RBF) Kernel

$$K(x, x') = e^{-\gamma \|x - x'\|^2}$$

Gaussian kernel has infinite dimensionality

Taylor's series expansion

$$e^x$$

Exponential Function

$$1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

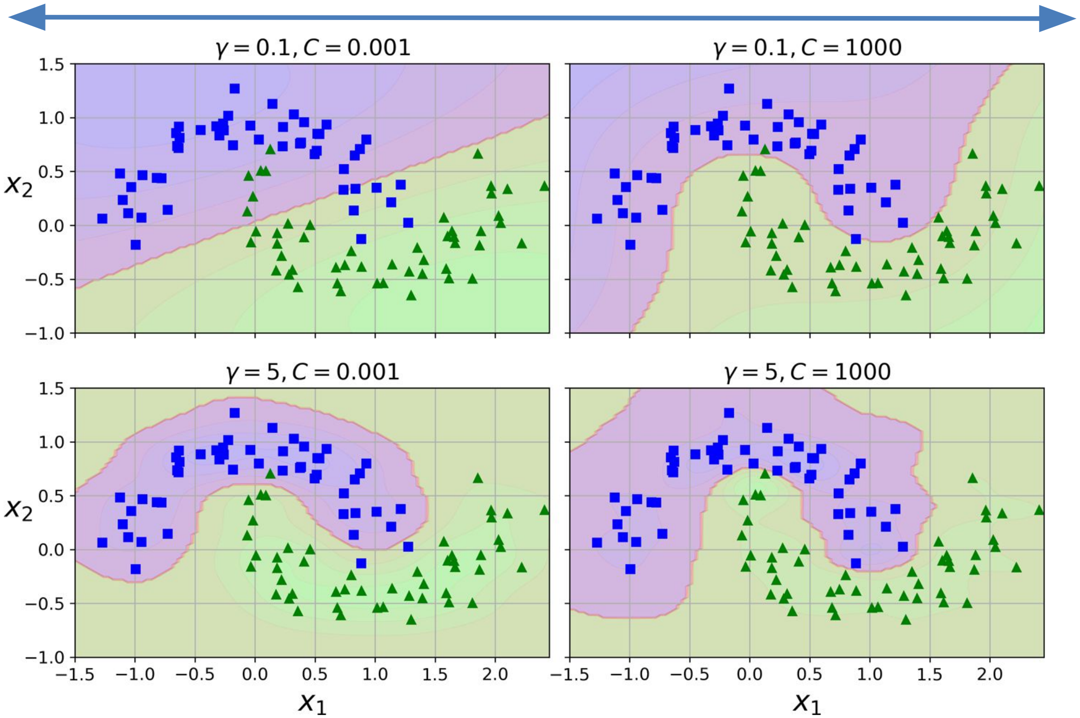
Exponential Function
(Taylor's Version)

Gaussian RBF kernel trick

More regularized
Underfitting

Less regularized
Overfitting

Less regularized
Overfitting



(Figure from Géron figure 5-9)

Summary: Why SVM?

Optimal margin classifier

Kernel trick for non-linearly separable classes

Note: Kernel trick is not limited to SVM, it can be applied to any algorithm that involves inner products.

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

Question

- Can an SVM classifier output a confidence score when it classifies an instance? What about a probability?
- An SVM classifier can output the distance between the test instance and the decision boundary, and you can use this as a confidence score. However, this score cannot be directly converted into an estimation of the class probability.
- If you set probability=True when creating an SVM in Scikit-Learn, then after training it will calibrate the probabilities using Logistic Regression on the SVM's scores (trained by an additional five-fold cross-validation on the training data).

Some concepts we did not cover

Representer theorem

Lagrange duality

Karush–Kuhn–Tucker conditions

Dual form

Gram matrix