Aprendizaje automático

Máquinas de vectores de soporte (SVM)

FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Agenda

- Máquinas de vectores de soporte lineales
  - *Hard margin*
  - *Soft margin*
- Máquinas de vectores de soporte lineales
  - Características polinómicas
  - *Kernel* polinómico
  - Características de similitud
  - *Kernel Gaussian RBF*
- Regresión con SVM
- Algunos detalles (si hay tiempo)
  - Función de decisión
  - Frontera de decisión
  - Función objetivo

Máquinas de vectores de soporte

FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Máquinas de vectores de soporte

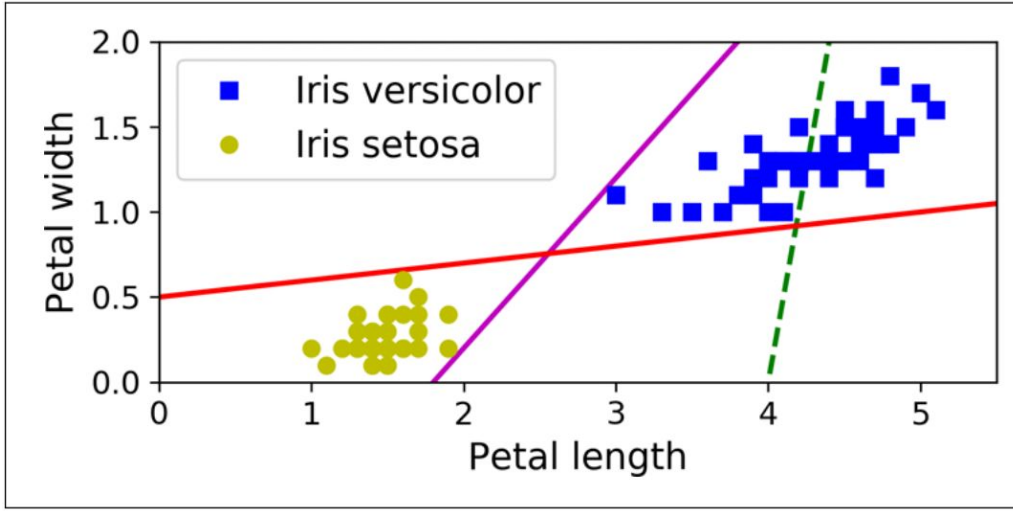
- Máquinas de vectores de soporte o Support Vector Machine (SVM)
- Clasificador *large margin*
- Clasificación, regresión, detección de *outliers*
- SVM lineal y no-lineal
- *Kernel trick*

Máquinas de vectores de soporte lineales

FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

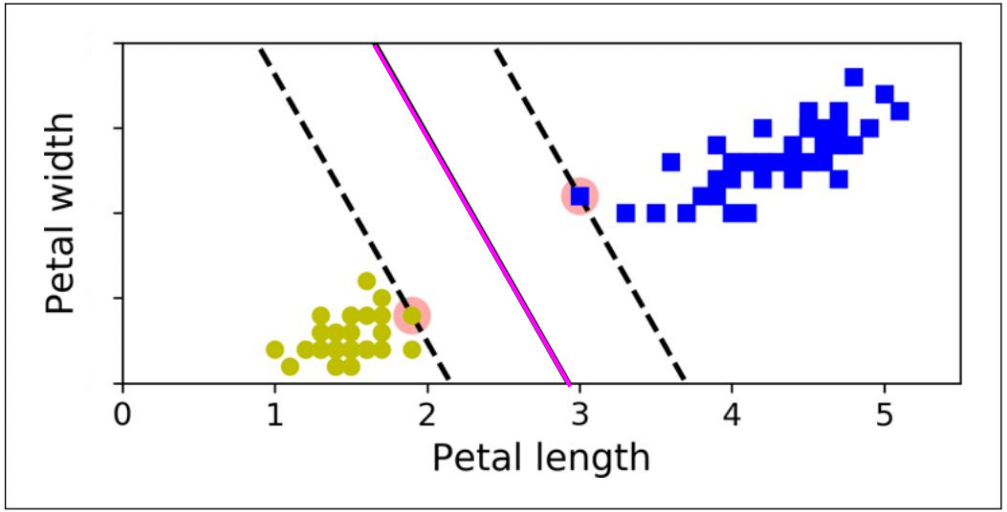
Motivación

- Dos clases linealmente separables
- Tres posibles clasificadores lineales
- Clasificador verde: no funciona
- Clasificadores rojo y lila: se ajustan perfectamente a los datos de entrenamiento → **sobreajuste**



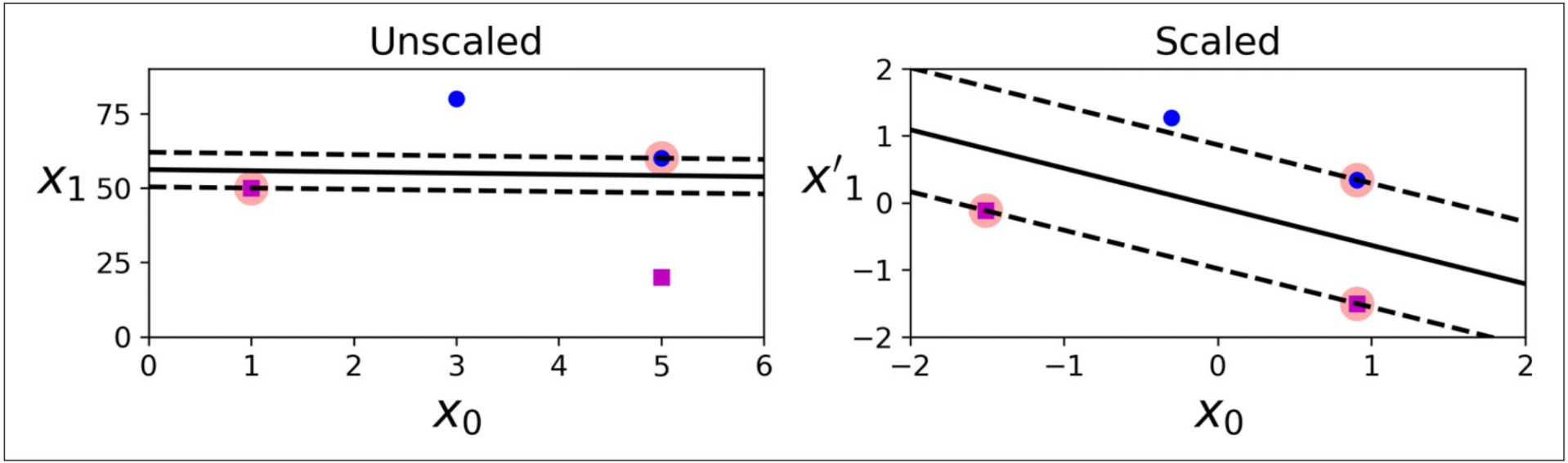
- Clasificador SVM
- Frontera de decisión lo más lejos posible de las muestras de entrenamiento más cercanas
- Totalmente determinado por las muestras que están en la frontera y dentro del margen → vectores de soporte

vectores de soporte



Escala de las características

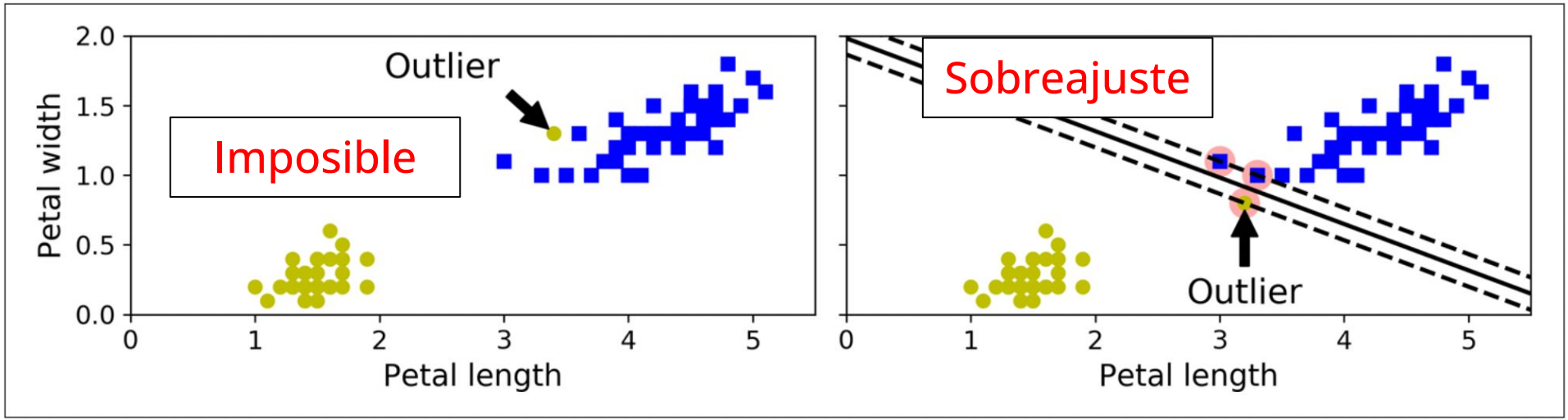
- Los clasificadores SVM son sensibles a la escala de las características



- Se puede escalar usando `StandardScaler` de Scikit-Learn

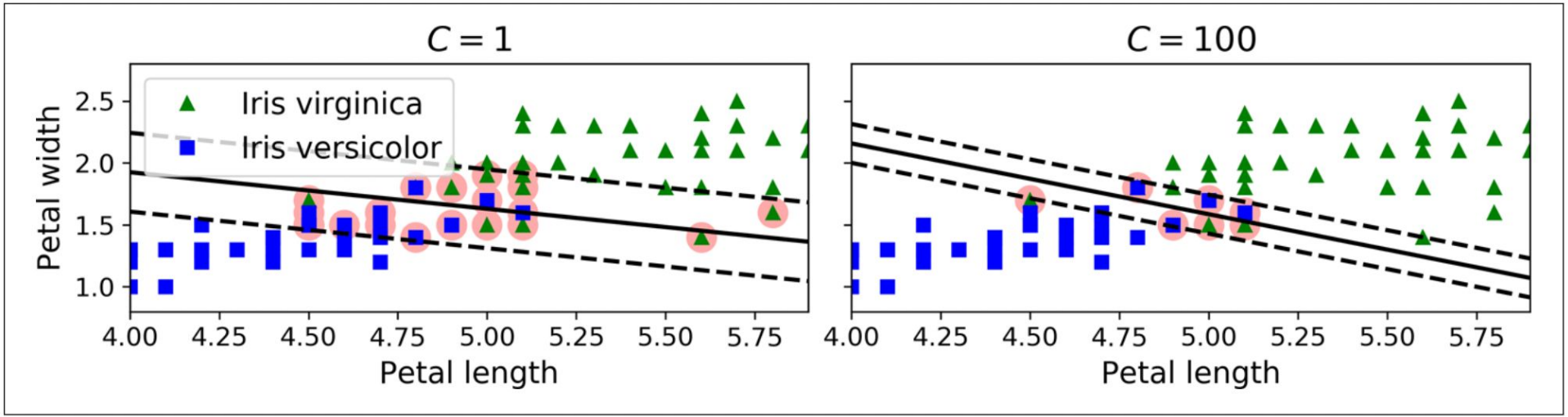
Clasificación *hard margin*

- Todas las muestras tienen que estar fuera del margen SVM y de un lado de la frontera
- Dos problemas:
  - Solo funciona si los datos son linealmente separables
  - Sensible a valores anómalos
- Agregamos un dato anómalo a los datos de iris



Clasificación *soft margin*

- Modelo mas flexible
- Se busca un buen equilibrio entre **mantener los márgenes lo más amplios posible** y **limitar violaciones de los mismos**
- Para controlar dicho equilibrio existe el **hiperparámetro C**
    - Si hay sobreajuste, se puede intentar controlar reduciendo el valor de C



LinearSVC de Scikit-Learn

```
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,
    ])

svm_clf.fit(X, y)
```

También se puede usar:

- La clase SVC

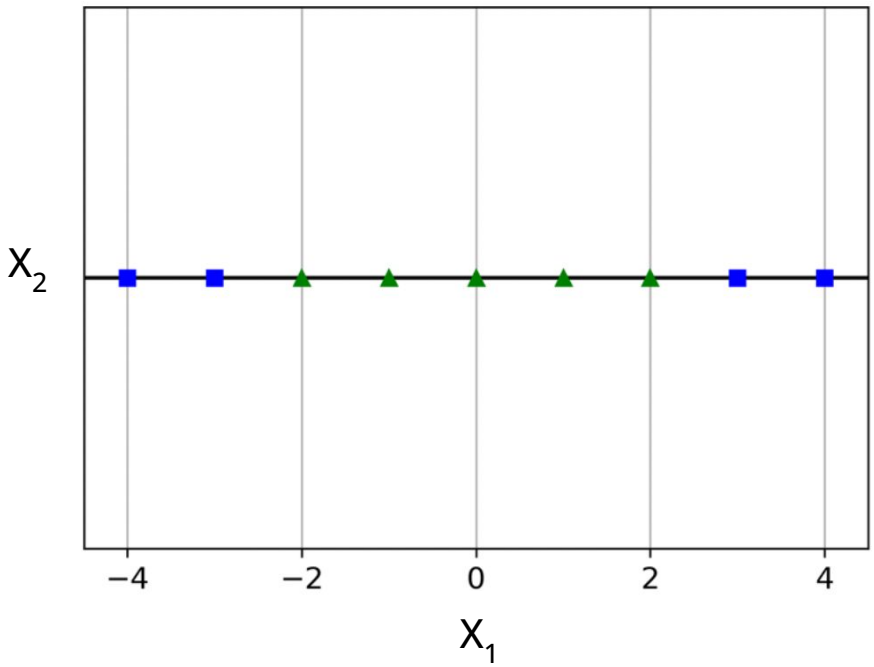
  `SVC(kernel="linear", C=1)`

Cuál es la mayor restricción del modelo hasta ahora?

Los datos tienen que ser linealmente separables.

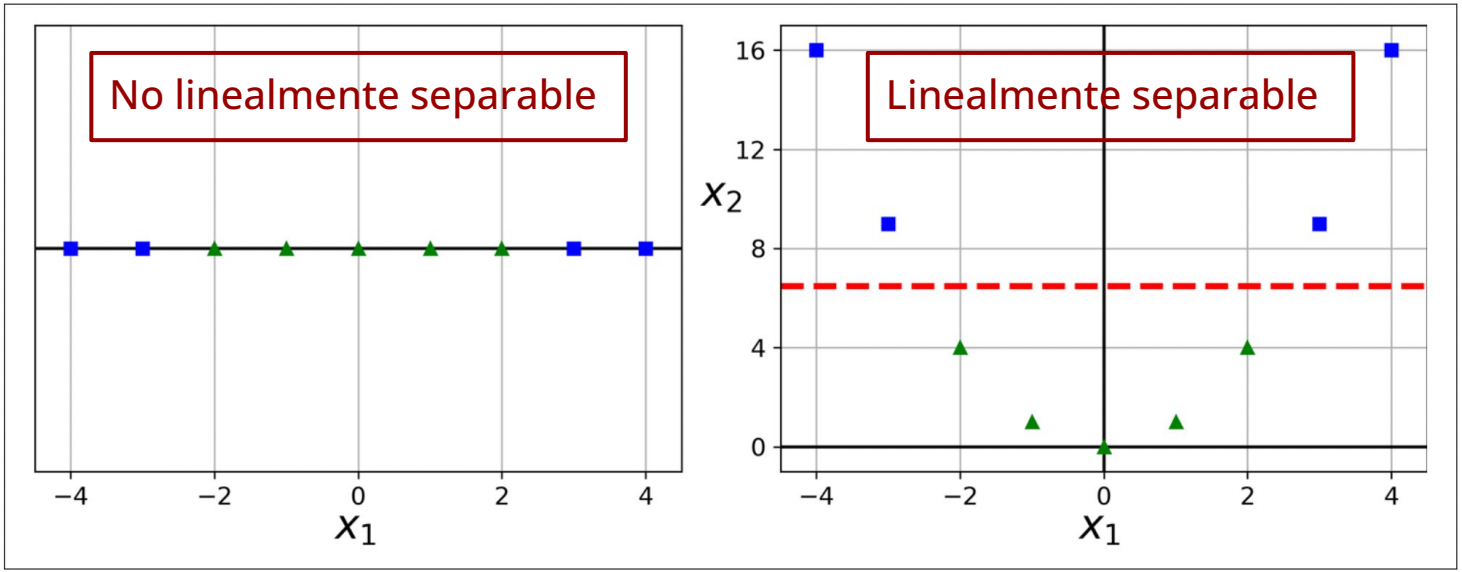
Máquinas de vectores de soporte no lineales

- La mayoría de los datos están muy lejos de ser linealmente separables.



Características polinomiales

- Agregamos más características con las potencias de las entradas
- Queda un problema de mayor dimensión
- En el ejemplo:
  - Característica 1: x_1
  - Característica 2: $x_2 = x_1^2$ (nueva)



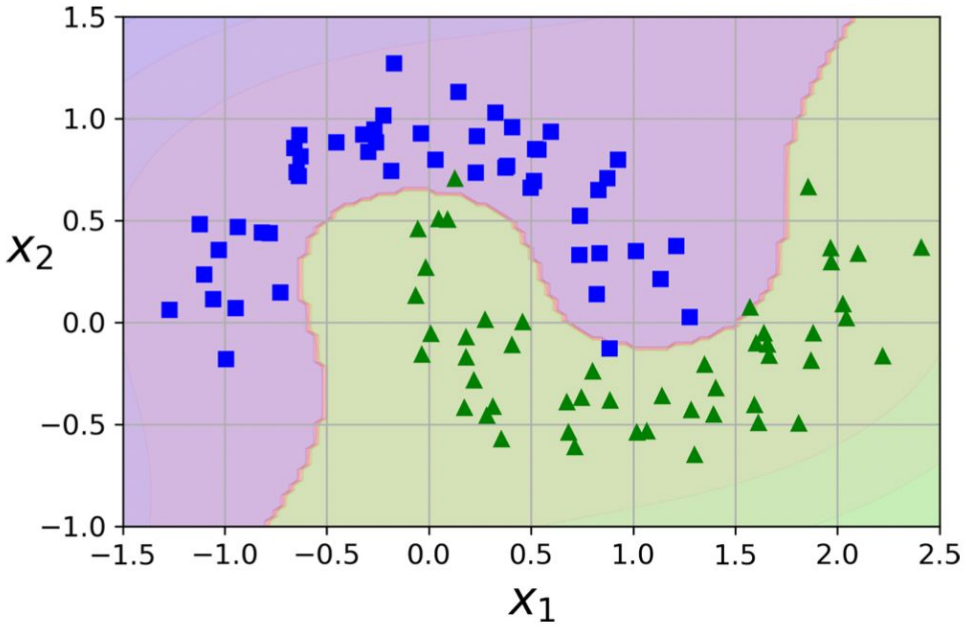
Clasificación lineal SVM con características polinomiales

```
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)
```

Clasificación lineal SVM con características polinomiales



Clasificación lineal SVM con características polinomiales

- Se hizo un mapeo explícito, se calcularon las características nuevas y se calculó el hiperplano sobre dichas características de mayor dimensión.
- Pero no es necesario realizar el mapeo explícito de las características.
- Se puede hacer el cálculo usando los características en el espacio original.
- Esto permite mapear a espacios de altas dimensiones eventualmente infinitas dimensiones.
- Esto es lo que se conoce como el *kernel trick*

Kernel

- Los *kernels* se utilizan para calcular el producto interno entre pares de puntos en el espacio de características transformado sin calcular **explícitamente** la propia transformación.
- Esto hace que sea **computacionalmente eficiente** tratar con espacios de características de alta dimensión.
- Los *kernels* más utilizados en SVM son el *kernel* lineal, el kernel polinómico y el *kernel* gaussiano RBF.
- El *kernel trick* es una técnica que permite a los SVM resolver problemas de clasificación no lineales mediante el mapeo **implícito** de los datos de entrada a un espacio de características de mayor dimensión.
- Esto permite encontrar un hiperplano que separa los datos.

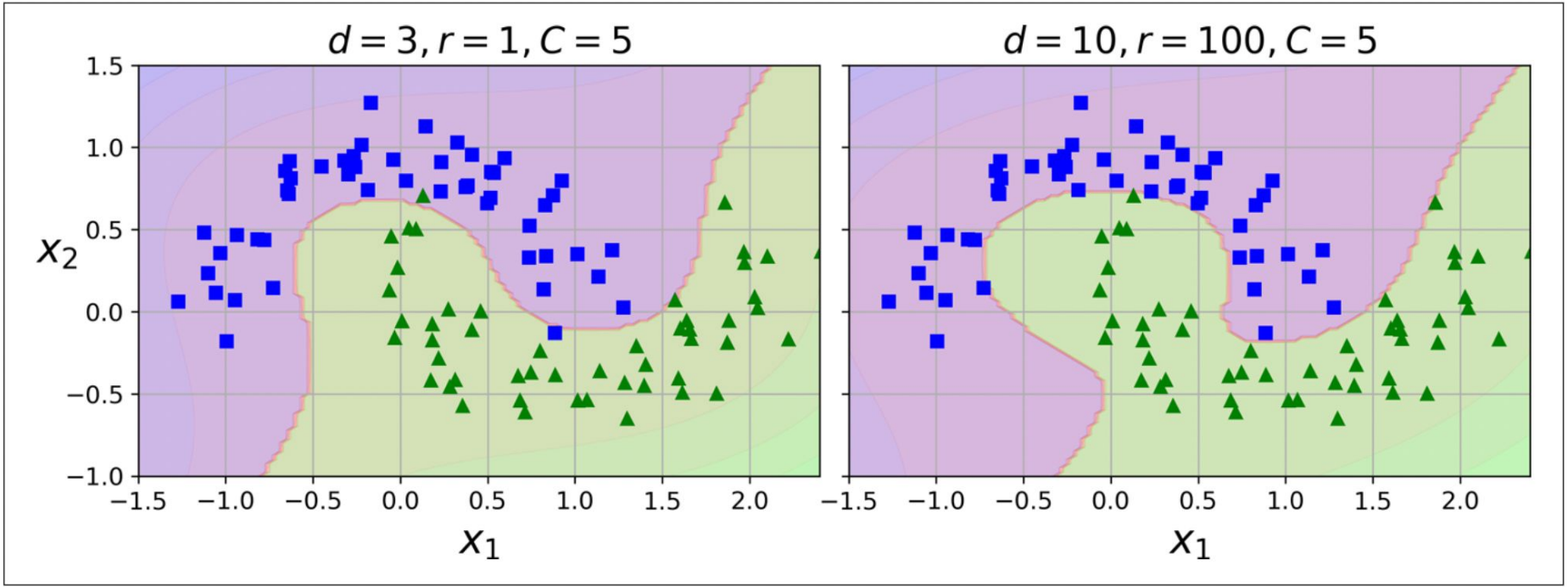
Kernel polinómico

- Las características polinómicas son simples de implementar
- Pero
  - Características de bajo grado polinómico: limitación con datos muy complejos
  - Características de alto grado polinómico: muchas características ((n+d)!/(d!n!)) → modelo muy lento
- Solución con SVM el *kernel trick*: permite obtener el mismo resultado que con el mapeo explícito pero sin tener que calcular explícitamente las nuevas características
- Se puede para esto usar la clase SVC de Scikit-Learn e indicar que se usa un *kernel* polinómico K

$$K(\mathbf{a},\mathbf{b}) = (\gamma \mathbf{a}^T \mathbf{b} + r)^d$$

```
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)
```

Kernel polinómico



- Hiperparámetro `d`: grado del polinomio
- Si hay sobreajuste, aumentar o reducir `d`?

Características de similitud

- Agregamos más características usando la función de similitud
- Mide cuanto un dato se parece a otro de referencia
- Una función de similitud típica es la *Gaussian Radial Basis Function* (Gaussian RBF)

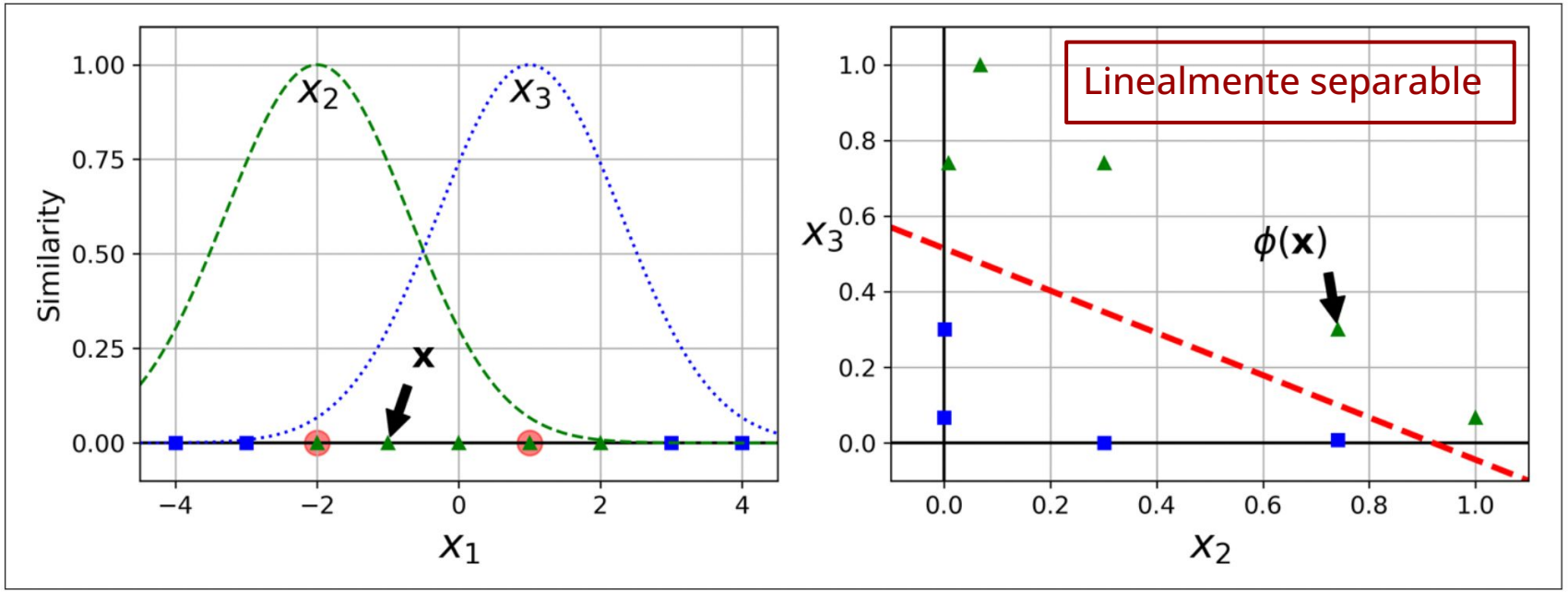
$$\phi_\gamma(\mathbf{x}, \ell) = \exp\left(-\gamma \| \mathbf{x} - \ell \|^2\right)$$

donde l es la referencia.

- Esta medida de similitud va de 0 (lejos de la referencia) a 1 (en la referencia).

Características de similitud

- Tomemos el ejemplo anterior agregando dos referencias l_1=-2 y l_2=1 y γ=0.3
- Para x_1=-1
  - $x_1 - l_1 = 1$ para $l_1 = -2$ y $x_2 = \phi_{0.3}(x_1, l_1) = \exp(-0.3*1^2) \approx 0.74$
  - $x_1 - l_2 = 2$ para $l_2 = 1$ y $x_3 = \phi_{0.3}(x_1, l_2) = \exp(-0.3*2^2) \approx 0.30$



Kernel Gaussiano RBF

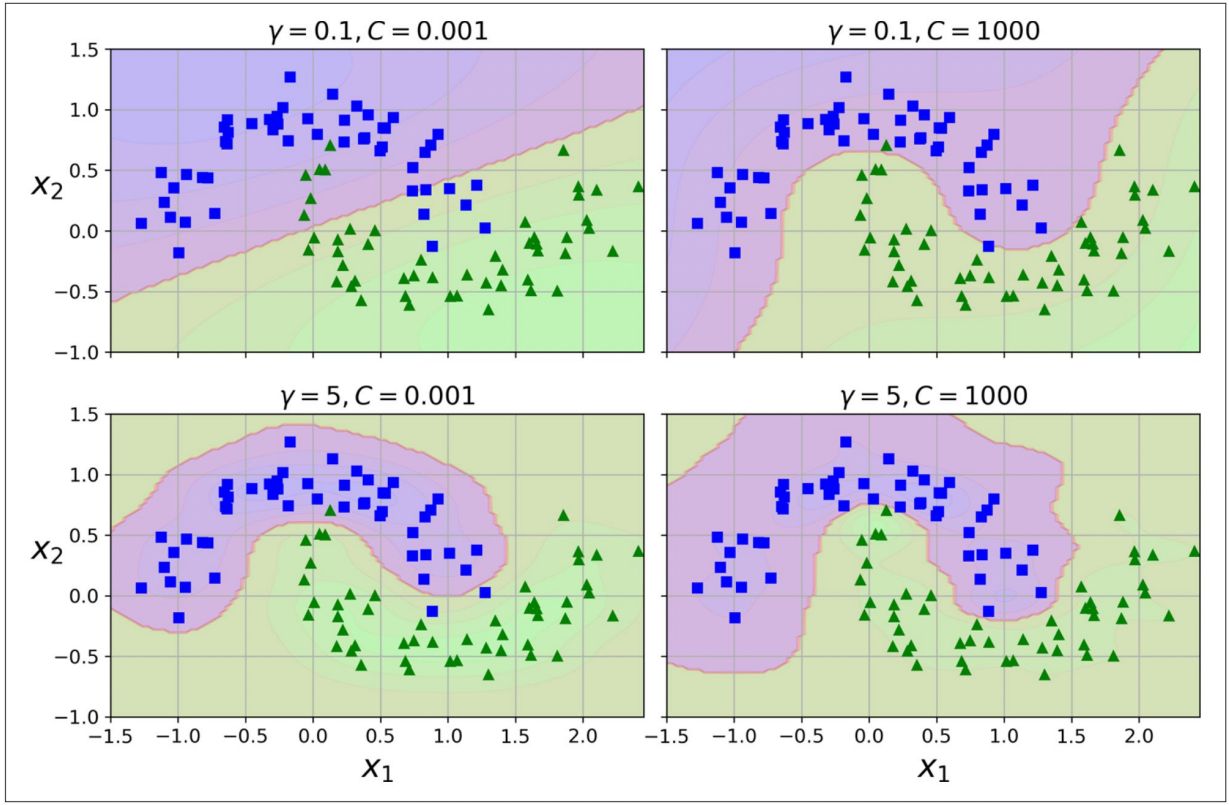
- Al igual que para el caso polinómico, puede ser muy costoso computacionalmente
  - m muestras, n características → m muestras, m características
- Solución con SVM el *kernel trick*: permite obtener el mismo resultado que con el mapeo explícito pero sin tener que calcular explícitamente las nuevas características
- Se puede para esto usar la clase SVC de Scikit-Learn e indicar que se usa un *kernel Gaussian RBF* (para la función de similitud Gaussian RBF) K

$$K(\mathbf{a},\mathbf{b}) = \exp(-\gamma||\mathbf{a}-\mathbf{b}||^2)$$

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

Kernel Gaussiano RBF

- Hiperparámetros γ y C
- Mayor γ → Gaussiana más estrecha → más " local" → puede llevar a sobreajuste
- Menor γ →Gaussiana más dispersa → menos "local" → puede llevar a subajuste
- γ actúa como un regularizador
  - reducir si sobreajuste
  - augmentar si subajuste



Complejidad computacional

Table 5-1. Comparison of Scikit-Learn classes for SVM classification

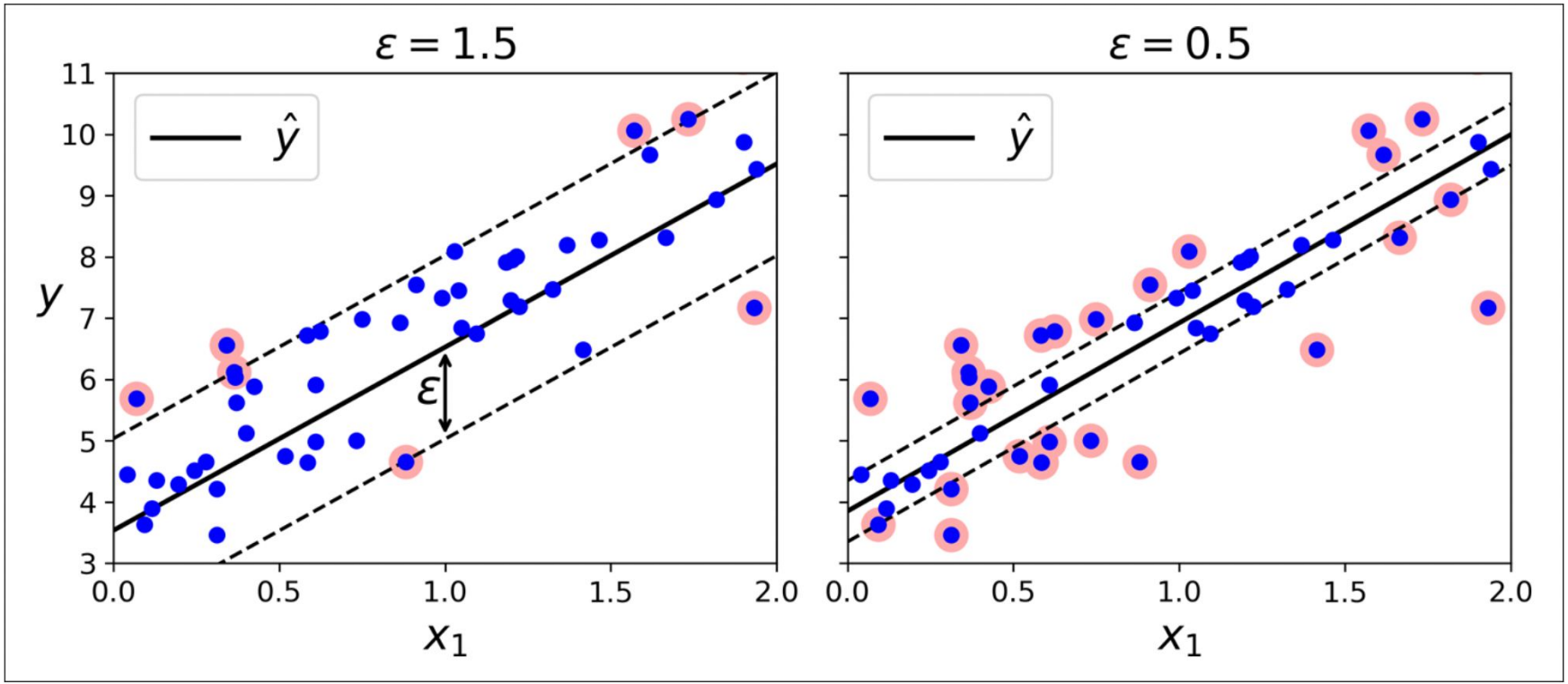
Class	Time complexity	Out-of-core support	Scaling required	Kernel trick
`LinearSVC`	$O(m \times n)$	No	Yes	No
`SGDClassifier`	$O(m \times n)$	Yes	Yes	No
`SVC`	$O(m^2 \times n)$ to $O(m^3 \times n)$	No	Yes	Yes

Regresión SVM

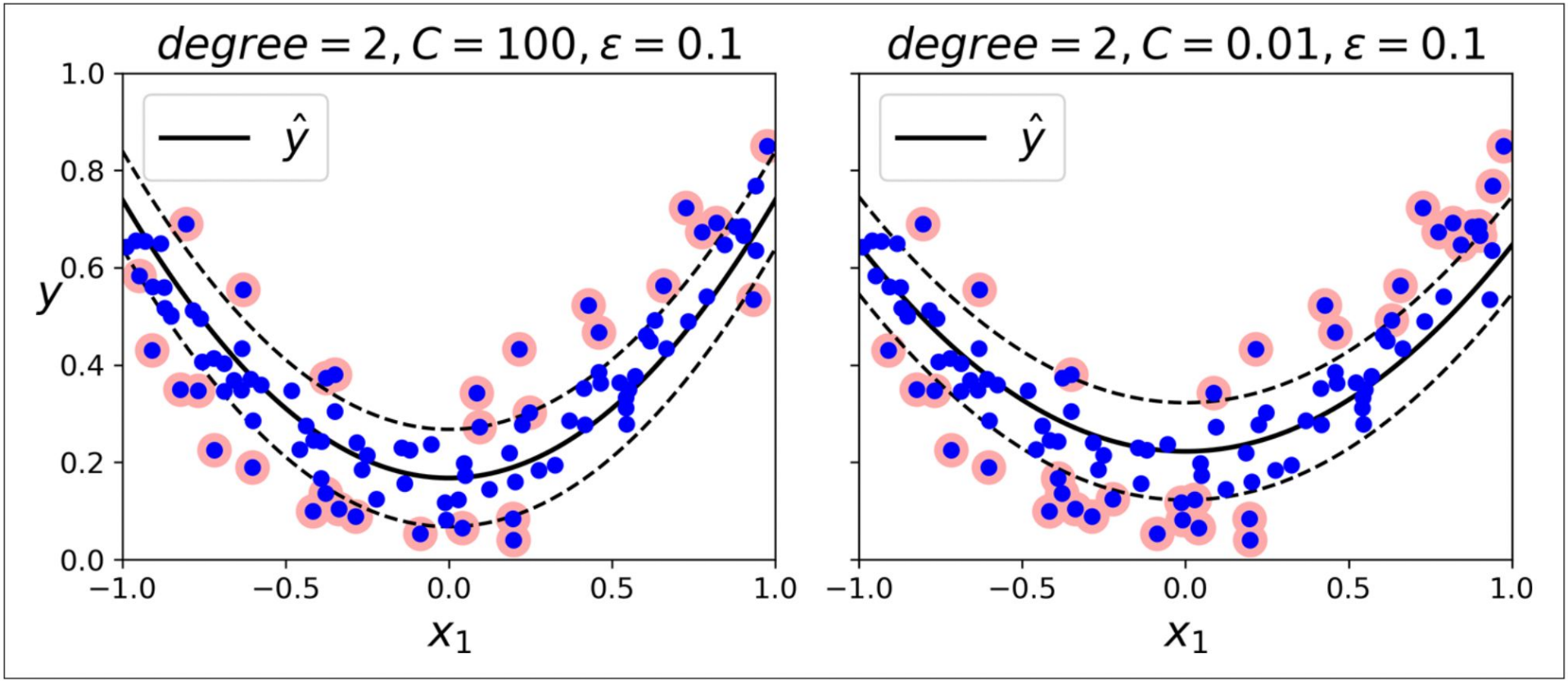
Regresión SVM

- Se busca tener la máxima cantidad de instancias dentro del margen y minimizar violaciones de dicho margen (que caigan fuera)
- El ancho del margen se controla con el hiperparámetro ε
- Las muestras que están en la frontera y fuera del margen son los vectores de soporte
- El modelo sólo depende de los vectores de soporte, los que están en del margen SVM
- Cuanto mayor ε, más se ajustará a *outliers* por lo que puede haber un subajuste
- Cuanto menor ε, más se ajustará a la mayoría de los datos entrenamiento por lo que puede haber un sobrebajuste

Regresión SVM lineal



Regresión SVM no lineal



Algunos detalles

Notación

$$h(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n = \theta^T x$$

$$h(x) = b + w_1 x_1 + \cdots + w_n x_n = w^T x + b$$

bias

pesos

$$w^T = [w_1, w_2, \cdots, w_n] \in \mathbb{R}^n$$

$$x^T = [x_1, x_2, \cdots, x_n] \in \mathbb{R}^n$$

$$b \in \mathbb{R}$$

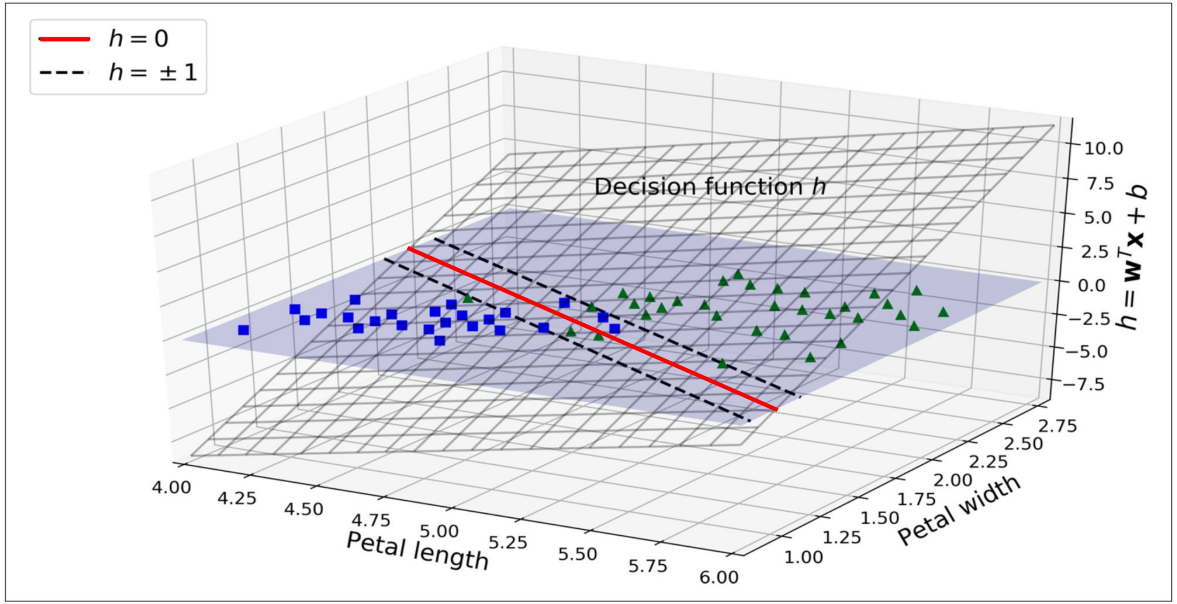
Función de decisión

- La función de decisión h(**x**) = $\mathbf{w}^T\mathbf{x}$+b de un clasificador lineal SVM predice la clase de un nuevo dato **x** siguiendo la ecuación:

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

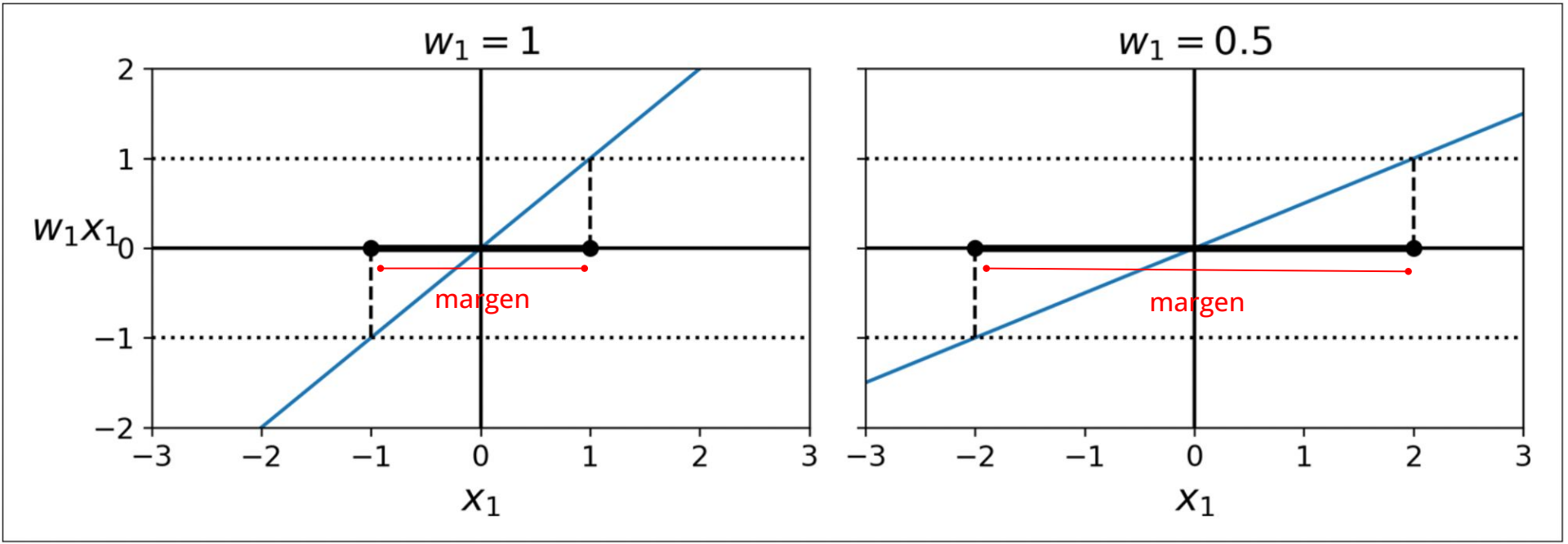
Frontera de decisión

- La función de decisión h es un hiperplano de $\mathbb{R}^n$ para **x** en $\mathbb{R}^n$
  - Plano 2D para n=2 características en el ejemplo
- La frontera de decisión es la intersección de dos hiperplanos
- Entrenar un clasificador lineal SVM implica encontrar los valores **w** y b que maximiza el margen al mismo tiempo que impide (*hard*) o limita (*soft*) las violaciones del mismo



Función objetivo

- Cuanto menor es la norma de ||**w**||, mayor es el margen SVM
- Queremos minimizar ||**w**||



Función objetivo - *hard margin*

- Además, para evitar que instancias de entrenamiento caigan dentro de los márgenes SVM hay que restringir la función h

$$h(x) \leq -1, \forall x \mid y(x) = 0$$

$$h(x) \geq 1, \forall x \mid y(x) = 1$$

$$t^{(i)} = \begin{cases} -1 & \text{si } y^{(i)} = 0 \\ 1 & \text{si } y^{(i)} = 1 \end{cases} \longrightarrow t^{(i)}\left(w^T x^{(i)} + b\right) \geq 1$$

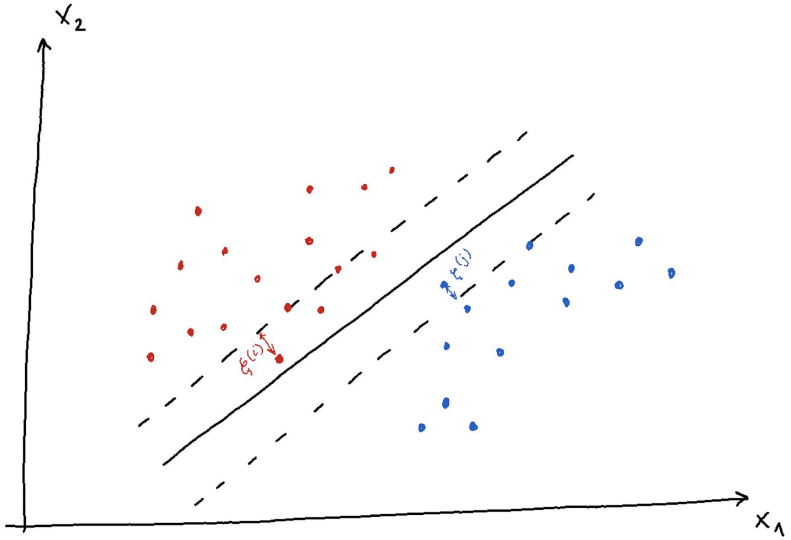
$$\underset{\mathbf{w}, b}{\text{minimize}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$\text{subject to} \quad t^{(i)}\left(\mathbf{w}^T\mathbf{x}^{(i)} + b\right) \geq 1 \quad \text{for } i = 1, 2, \cdots, m$$

Función objetivo - *soft margin*

- Para el *soft margin*, se introduce una variable auxiliar $\zeta^{(i)} \geq 0$
- $\zeta^{(i)}$ mide cuanto la instancia i puede violar la restricción del margen SVM

- Se crea un compromiso entre dos términos
  - Minimizar la variable auxiliar
  - Minimizar la norma al cuadrado del vector de pesos
- Para dicho compromiso → hiperparámetro C



$$\underset{\mathbf{w}, b, \zeta}{\text{minimize}} \quad \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C \sum_{i=1}^{m} \zeta^{(i)}$$

$$\text{subject to} \quad t^{(i)}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b\right) \geq 1 - \zeta^{(i)} \quad \text{and} \quad \zeta^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m$$

Problema dual

- Para un problema de optimización con restricciones, es posible expresar este problema (primario) como otro problema cercano, el dual.
- Bajo algunas condiciones, el problema dual y primario tienen la misma solución.
- Es el caso para SVM: se puede resolver tanto el primario como el dual.
- Forma dual del problema SVM lineal

$$\underset{\alpha}{\text{minimize}} \; \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha^{(i)} \alpha^{(j)} t^{(i)} t^{(j)} \mathbf{x}^{(i)\mathsf{T}} \mathbf{x}^{(j)} \; - \; \sum_{i=1}^{m} \alpha^{(i)}$$

$$\text{subject to} \quad \alpha^{(i)} \geq 0 \quad \text{for } i = 1, 2, \cdots, m$$

- Determinando la solución α del dual, se obtiene la solución del primal como

$$\widehat{\mathbf{w}} = \sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \mathbf{x}^{(i)}$$

$$\hat{b} = \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \widehat{\mathbf{w}}^{\mathsf{T}} \mathbf{x}^{(i)} \right)$$

Kernelized SVMs

- Supongamos que se mapean las características de un problema con un polinomio de segundo orden

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

- Para dos vectores **a** y **b**, el producto escalar del mapeo de dichos vectores es

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

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

Kernelized SVMs

- El producto escalar del mapeo de dos vectores es simplemente el cuadrado de dicho producto

$$\phi(\mathbf{a})^\top \phi(\mathbf{b}) = (\mathbf{a}^\top \mathbf{b})^2$$

How about that? The dot product of the transformed vectors is equal to the square of the dot product of the original vectors: $\phi(\mathbf{a})^T \phi(\mathbf{b}) = (\mathbf{a}^T \mathbf{b})^2$.

The function $K(\mathbf{a}, \mathbf{b}) = (\mathbf{a}^T \mathbf{b})^2$ is a second-degree polynomial kernel. In Machine Learning, a *kernel* is a function capable of computing the dot product $\phi(\mathbf{a})^T \phi(\mathbf{b})$, based only on the original vectors **a** and **b**, without having to compute (or even to know about) the transformation ϕ. Equation 5-10 lists some of the most commonly used kernels.

Common kernels

Equation 5-10. Common kernels

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

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

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

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

Predictions

Equation 5-11. Making predictions with a kernelized SVM

$$h_{\hat{\mathbf{w}}, \hat{b}}\left(\phi\left(\mathbf{x}^{(n)}\right)\right) = \hat{\mathbf{w}}^{\mathsf{T}}\phi\left(\mathbf{x}^{(n)}\right) + \hat{b} = \left(\sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \phi\left(\mathbf{x}^{(i)}\right)\right)^{\mathsf{T}} \phi\left(\mathbf{x}^{(n)}\right) + \hat{b}$$

$$= \sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \left(\phi\left(\mathbf{x}^{(i)}\right)^{\mathsf{T}} \phi\left(\mathbf{x}^{(n)}\right)\right) + \hat{b}$$

$$= \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \hat{\alpha}^{(i)} t^{(i)} K\left(\mathbf{x}^{(i)}, \mathbf{x}^{(n)}\right) + \hat{b}$$

Bias term

Equation 5-12. Using the kernel trick to compute the bias term

$$\hat{b} = \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \hat{\mathbf{w}}^{\mathsf{T}} \phi\left(\mathbf{x}^{(i)}\right) \right) = \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \left(\sum_{j=1}^{m} \hat{\alpha}^{(j)} t^{(j)} \phi\left(\mathbf{x}^{(j)}\right) \right)^{\mathsf{T}} \phi\left(\mathbf{x}^{(i)}\right) \right)$$

$$= \frac{1}{n_s} \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \left(t^{(i)} - \sum_{\substack{j=1 \\ \hat{\alpha}^{(j)} > 0}}^{m} \hat{\alpha}^{(j)} t^{(j)} K\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) \right)$$

Online SVMs

Equation 5-13. Linear SVM classifier cost function

$$J(\mathbf{w}, b) = \frac{1}{2}\mathbf{w}^\intercal\mathbf{w} \quad + \quad C\sum_{i=1}^{m} max\left(0, 1 - t^{(i)}\left(\mathbf{w}^\intercal\mathbf{x}^{(i)} + b\right)\right)$$

Hinge Loss

The function $max(0, 1 - t)$ is called the *hinge loss* function (see the following image). It is equal to 0 when $t \geq 1$. Its derivative (slope) is equal to -1 if $t < 1$ and 0 if $t > 1$. It is not differentiable at $t = 1$, but just like for Lasso Regression (see "Lasso Regression" on page 137), you can still use Gradient Descent using any *subderivative* at $t = 1$ (i.e., any value between -1 and 0).