Support Vector Machine

Esquema de

- Aprendizaje supervisado

- Clasificadores binarios

- Aspectos teóricos:

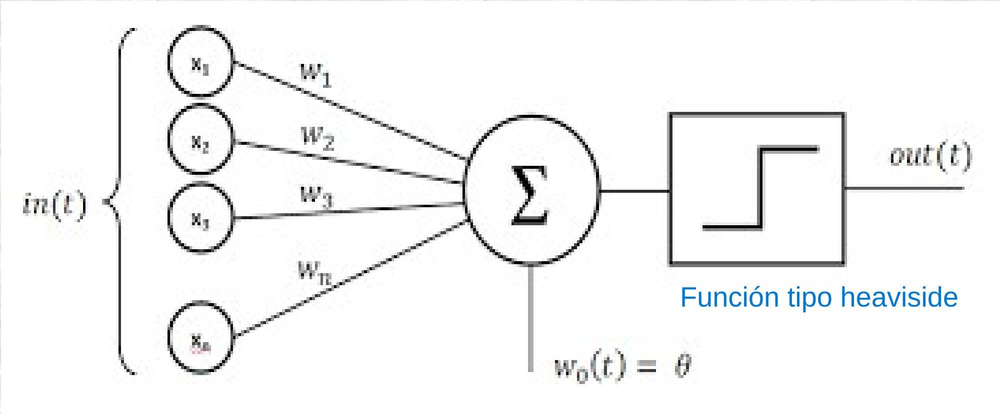
  ¿Es una red neuronal? Es un sistema híbrido A veces se considera de una forma o de la otra dependiendo del enfoque

  ¿Es un sistema lineal o no lineal?

  ¿Que clase de problemas podemos encarar?

Support Vector Machine

La red neuronal mas simple posible: el *perceptrón*



Support Vector Machine

Si la entrada es el vector $\mathbf{x}_i$ y la salida deseada es y_i:

$$\mathbf{w} \cdot \mathbf{x}_i + b \geq 0 \quad \text{para } y_i = 1$$

$$\mathbf{w} \cdot \mathbf{x}_i + b < 0 \quad \text{para } y_i = -1$$

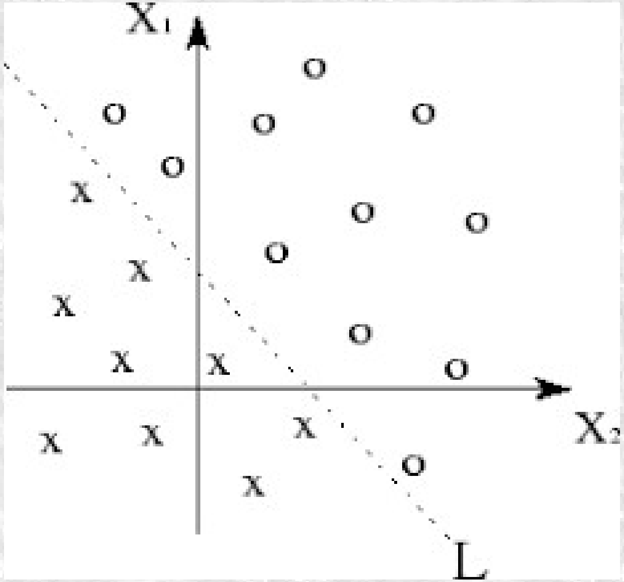
O

$$y_i (\mathbf{w} \cdot \mathbf{x}_i + b) \geq 0$$

Este sistema resuelve problemas de clasificación que son *linealmente separables*

Support Vector Machine

Problemas *linealmente separables*

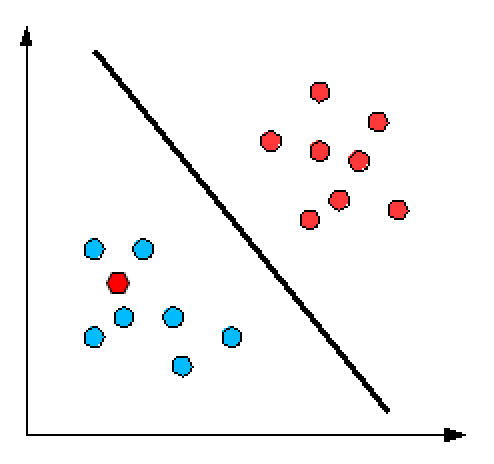


Si el problema es linealmente separable hay un algoritmo (*algoritmo del perceptrón)* que encuentra una solución en tiempo finito

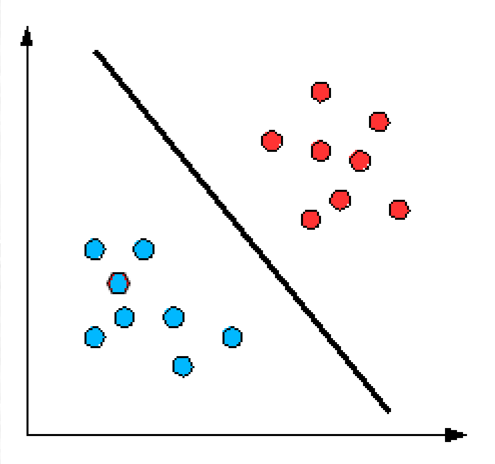
Support Vector Machine

Problemas *linealmente separables: p* entradas en N dimensiones

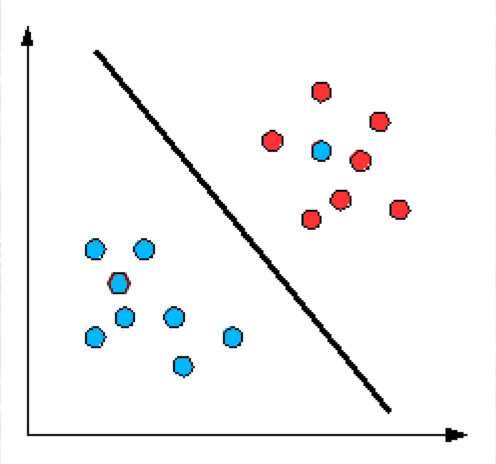
No



Si



No



¿Cual es el número total de problemas que se pueden definir si tengo p entradas?

Support Vector Machine

Problemas *linealmente separables:*

¿Cual es el número total de problemas que se pueden definir si tengo p entradas? $\rightarrow$ 2^p

Si estoy p entradas en dimensión N, el número de problemas linealmente separables es denotado por C(p,N)

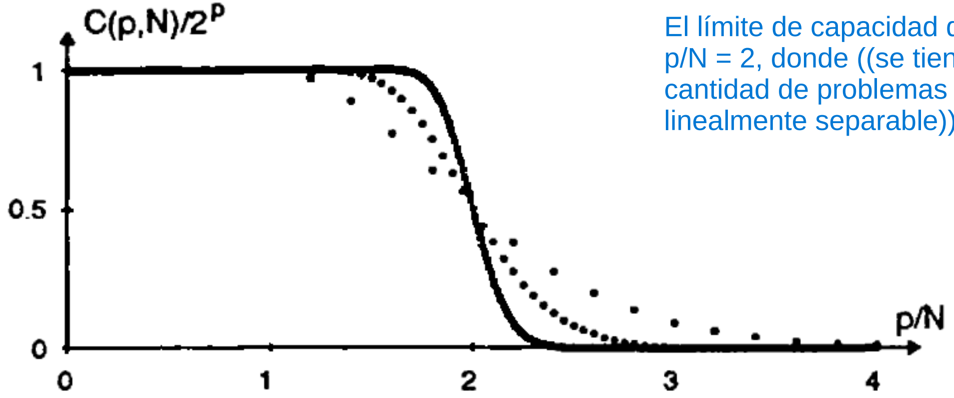
La fracción de problemas linealmente separables es $C(p,N)/2^p$

Esta cantidad se puede calcular con métodos de geometría combinatoria (**para puntos en posición general**)

Support Vector Machine

Problemas *linealmente separables:*

Esta cantidad se puede calcular con métodos de geometría combinatoria (Hertz p. 112):



El límite de capacidad del perceptrón es p/N = 2, donde ((se tiene la mayor cantidad de problemas que pueden ser linealmente separable))

FIGURE 5.11 The function $C(p, N)/2^p$ given by (5.67) plotted versus p/N for $N =$ 5, 20, and 100.

Support Vector Machine

Problemas *linealmente separables:*

Para dimensión grande TODO problema es linealmente separable si $p<2N$ Entonces uno podría aumentar N todo lo posible y podría resolver el problema

Otra forma de encararlo es:

Si un problema no es linealmente separable se lo puede transformar en uno mapeándolo a dimensión alta

$\mathbf{x} \rightarrow \varphi(\mathbf{x})$ Función no lineal fija sin parámetros

$\varphi: R^N \rightarrow R^{N'}$ transformación no lineal, $N' > N$ (quizás $N' >> N$)

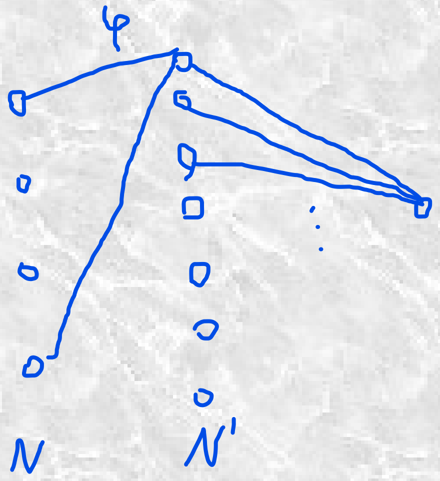
Como se incrementó la dimensión del sistema, es muy probable que el problema ahora sí sea linealmente separable

Support Vector Machine

Una vez que le problema es *linealmente separable* se puede utilizar algún método de aprendizaje tipo algoritmo del perceptrón

Sin embargo,

El número de parámetros es tan grande que voy a tener una situación de *overfitting*



Support Vector Machine

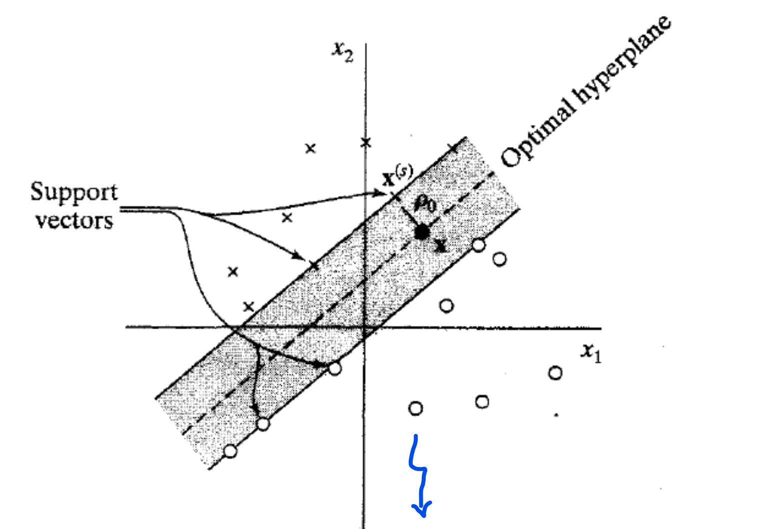
La solución no es única

¿Que solución tendrá menor error de generalización?

Se puede probar que

Es la que tiene el mayor *margen de separación:*

Esto es lógico porque los nuevos datos con los que voy a calcular el error de generalización siguen la distribución de los datos ya presentados, de modo que es poco probable que aparezca una x en el área de los círculos, por ejemplo



Una vez definido el hiperplano, los demás puntos que no pertenecen a la región en negrita, se vuelven irrelevantes para determinar el hiperplano óptimo

Support Vector Machine

Supongamos que los hiperplanos están definidos por

Versor   bias

$\hat{\mathbf{w}} \,.\, \mathbf{x} + \beta = c$ [1] $\hat{\mathbf{w}} \,.\, \Delta\mathbf{x} = 2\,c$

$\hat{\mathbf{w}} \,.\, \mathbf{x} + \beta = -c$ [2]

Vector ortogonal que define el hiperplano

O equivalentemente:

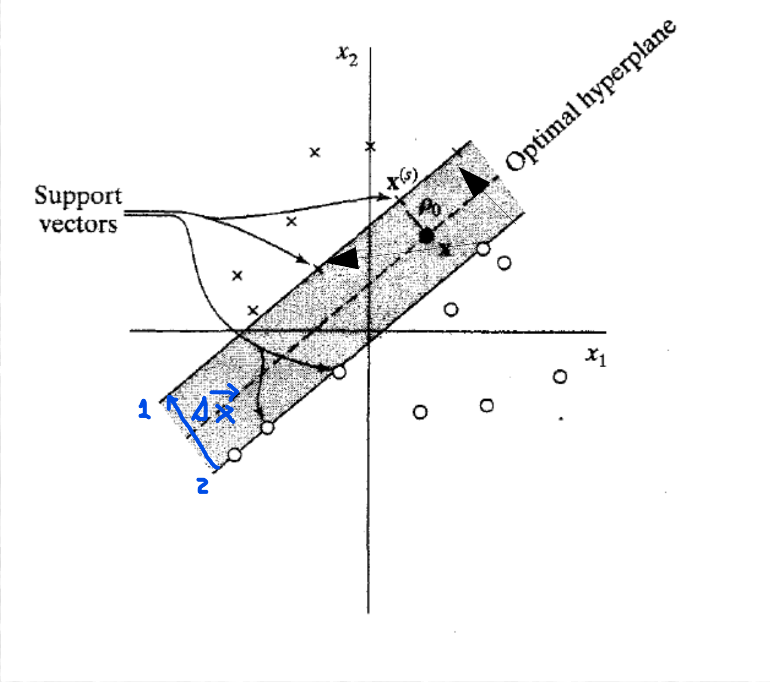
Renormalizando

$\mathbf{w} \,.\, \mathbf{x} + b = 1$ $\mathbf{w} \,.\, \Delta\mathbf{x} = 2$

$\mathbf{w} \,.\, \mathbf{x} + b = -1$ donde $\mathbf{w}=\hat{\mathbf{w}}/c$, $b=\beta/c$

La distancia entre los hiperplanos es

$2\,\rho_0 = 2/|\mathbf{w}|$



Support Vector Machine

Es decir que el mejor hiperplano es el que minimiza $|\mathbf{w}|$

Esto vale únicamente en esta representación de w

En este contexto el clasificador óptimo es que se obtiene de mimimizar $|\mathbf{w}|$ (o $|\mathbf{w}|^2$) con los constraints (que son desigualdades lineales)

$$y_i\,(\mathbf{w}\ .\ \mathbf{x}_i + b) \geq 1 \qquad (\text{o } y_i\,(\mathbf{w}\ .\ \varphi(\mathbf{x}_i) + b) \geq 1) \qquad \text{para } i=1,\ldots,p$$

Resulta que este problema particular ya era un problema conocido denominado como

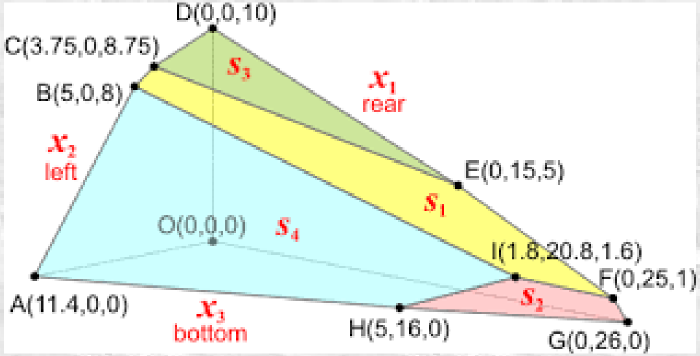
Problema de *programación cuadrática.*

Se puede ver que si |x|^2 << D, entonces se puede calcular d_VC (diapositiva no incluida en este power point)

Support Vector Machine

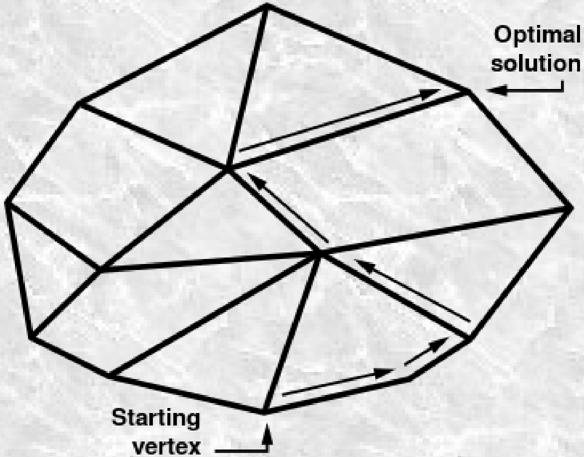
Los constraints definen un poliedro irregular

En el espacio w cada constraint define en su igualdad un hiperplano y en su desigualdad, de qué lado quedarme



La solución involucra una búsqueda por las aristas de este objeto

en los cuales se cumplen las igualdades de los constraint (vectores de apoyo) y por lo tanto son justamente los hiperplanos 1 y 2 que estamos buscando



Support Vector Machine

Se puede pasar al problema *dual* introduciendo multiplicadores de Lagrange α_i $(1 \leq i \leq p)$ (ver libro Haykin p. 345-346).

No es exactamente multiplicadores de Lagrange porque uno tiene desigualdades en lugar de igualdades

El vector **w** puede ser escrito como

$$\mathbf{w} = \Sigma_{1 \leq i \leq p} \underbrace{\alpha_i\, y_i}_{\text{es como el delta}}\, \mathbf{x}_i \quad (\text{o} \quad \mathbf{w} = \Sigma_{1 \leq i \leq p}\, \alpha_i\, y_i\, \boldsymbol{\varphi}(\mathbf{x}_i))$$

Donde los α_i maximizan la función

$$Q(\{\alpha_i\}) = \Sigma_{1 \leq i \leq p}\, \alpha_i - \tfrac{1}{2}\, \Sigma_{1 \leq i,j \leq p}\, \alpha_i\, \alpha_j\, y_i\, y_j\, \boldsymbol{\varphi}(\mathbf{x}_i) \,\boldsymbol{.}\, \boldsymbol{\varphi}(\mathbf{x}_j)$$

Con los constraints $\alpha_i \geq 0$, $\Sigma_{1 \leq i \leq p}\, \alpha_i\, y_i = 0$ (Haykin, p. 344-346)

Support Vector Machine

Notar que la función $Q(\{\alpha_i\})$ solo depende de los datos de entrenamiento a través de pxp productos escalares $\mathbf{x}_i \cdot \mathbf{x}_j$ (o $\boldsymbol{\varphi}(\mathbf{x}_i).\boldsymbol{\varphi}(\mathbf{x}_j)$) $(1 \leq i,j \leq p)$

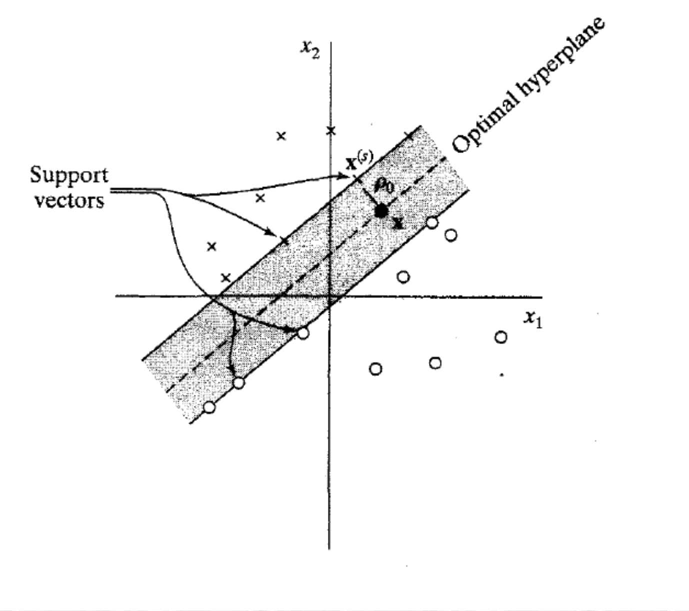
Se define el *kernel* $k(\mathbf{x}_i,\mathbf{x}_j) = \boldsymbol{\varphi}(\mathbf{x}_i).\boldsymbol{\varphi}(\mathbf{x}_j)$ es un producto escalar

dados p datos de entrada, es una matriz pxp que es simétrica

Se puede probar que

El vector de peso óptimo es una combinación lineal de los $\mathbf{x}$ para los cuales los α son no nulos $\rightarrow$ vectores de apoyo

Habrá muchos alpha = 0 que corresponden a los x_i que están lejos del hiperplano. Una vez conocidos los alpha distintos de cero, usando la fórmula para w ya tenemos el problema resuelto



Support Vector Machine

$k(\mathbf{x}_i, \mathbf{x}_j)$ es el kernel

En la práctica se toma en cuenta un kernel más general

Kernels usuales:

- Polynomial (homogeneous): $k(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j})^d$.
- Polynomial (inhomogeneous): $k(\vec{x_i}, \vec{x_j}) = (\vec{x_i} \cdot \vec{x_j} + 1)^d$.
- Gaussian radial basis function: $k(\vec{x_i}, \vec{x_j}) = \exp(-\gamma \|\vec{x_i} - \vec{x_j}\|^2)$ for $\gamma > 0$. Sometimes parametrized using $\gamma = 1/(2\sigma^2)$.
- Hyperbolic tangent: $k(\vec{x_i}, \vec{x_j}) = \tanh(\kappa \vec{x_i} \cdot \vec{x_j} + c)$ for some (not every) $\kappa > 0$ and $c < 0$.

Support Vector Machine

Una vez que los coeficientes α_i han sido evaluados la predicción de la red en el punto **x** esta dada por

$$\mathbf{w} \cdot \varphi(\mathbf{x}) + b = \sum_{1\leq i\leq p} \alpha_i\, y_i\, \boldsymbol{\varphi}(\mathbf{x}_i) \cdot \boldsymbol{\varphi}(\mathbf{x}) + b = \sum_{1\leq i\leq p} \alpha_i\, y_i\, k(\mathbf{x}_i, \mathbf{x}) + b$$

Es decir, la predicción uno la puede hacer solamente mediante el kernel

Observar que:

La suma no es sobre todos los p, sino sobre un conjunto reducido!

-α_i son no nulos SOLO para los vectores de apoyo: predicción eficiente

-**NO** es necesario dar una forma explícita de la transformación no-lineal **φ**. La dimensionalidad N' es potencialmente infinita (ver teorema de Mercer, Haykin p. 354)

Tanto el aprendizaje como la predicción, se pueden hacer directamente a partir del kernel! sin necesidad de especificar el phi

Puedo trabajar con infinitos parámetros sin que la generalización sea nula porque estoy buscando minimizar |w|

Support Vector Machine

A veces queremos tolerarar tener cierto número de errores, si eso mejora el error de generalización

Podemos buscar cual es la solución que minimiza el número de errores:

Soft-margin SVM linear (o ni linear) (Support Vector Machine con bordes blandos)

Término de error:

$$E_i = \max(0, 1 - y_i (\mathbf{w} \cdot \mathbf{x}_i + b))$$

$$(\text{o } \max(0, 1 - y_i (\mathbf{w} \cdot \boldsymbol{\varphi}(\mathbf{x}_i) + b)))$$

Support Vector Machine

Todo esto ya se encuentra implementado en librerías estándar

Soft-margin SVM linear (o no linear)

Queremos minimizar

$E = 1/p \, \Sigma_{1 \leq i \leq p} \, E_i + \lambda \, |\mathbf{w}|^2$

El parámetro λ controla cuan fuertemente controlamos el tamaño de los pesos

Support Vector Machine

Soft-margin SVM linear (o no linear)

Pasando al problema dual tenemos:

$\mathbf{w} = \Sigma_{1\leq i\leq p}\ \alpha_i\, y_i\, \mathbf{x}_i$ (o $\mathbf{w} = \Sigma_{1\leq i\leq p}\ \alpha_i\, y_i\, \boldsymbol{\varphi}(\mathbf{x}_i)$)

Donde α_i maximizan la función

$Q(\{\alpha_i\}) = \Sigma_{1\leq i\leq p}\ \alpha_i\ -\ ½\ \Sigma_{1\leq i,j\leq p}\ \alpha_i\, \alpha_j\, y_i\, y_j\ \boldsymbol{\varphi}(\mathbf{x}_i)\ \mathbf{.}\ \boldsymbol{\varphi}(\mathbf{x}_j)$

Con los constraints $\mathbf{(2p\lambda)^{-1}} \geq \alpha_i \geq 0$, $\Sigma_{1\leq i\leq p}\ \alpha_i\, y_i = 0$

Support Vector Machine

Comparación *SVM* vs. redes multicapa

SVM	Redes Multicapa
(paraboloide con constraints) Funcion a optimizar cuadrática ($t=O(p^3)$) Hay teorema de convergencia!	Función a optimizar extremadamente complicada, múltiples mínimos locales
Generalización óptima garantizada	Generalización óptima determinada empíricamente
Requiere memoria $O(p^2)$ Hay que almacenar el kernel	Requiere memoria O(batch size)

y no tiene el concepto de batch. No es un aprendizaje del tipo incremental en el que se va aprendiendo con los ejemplos, sino que se manda todo junto

Support Vector Machine

Implementación: scikit-learn

https://scikit-learn.org/stable/modules/svm.html#svm-mathematical-formulation

https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python

-Que kernel utilizar

-Parámetros del kernel: γ, κ, c, etc

-Parámetro de regularización: $C = (p\lambda)^{-1}$ para soft-margin

-Optativo: clases desbalanceadas Por si tuviéramos muchas más salidas 1 que -1