Support Vector Machine

- Aprendizaje supervisado
- Clasificadores binarios
- Aspectos teóricos:

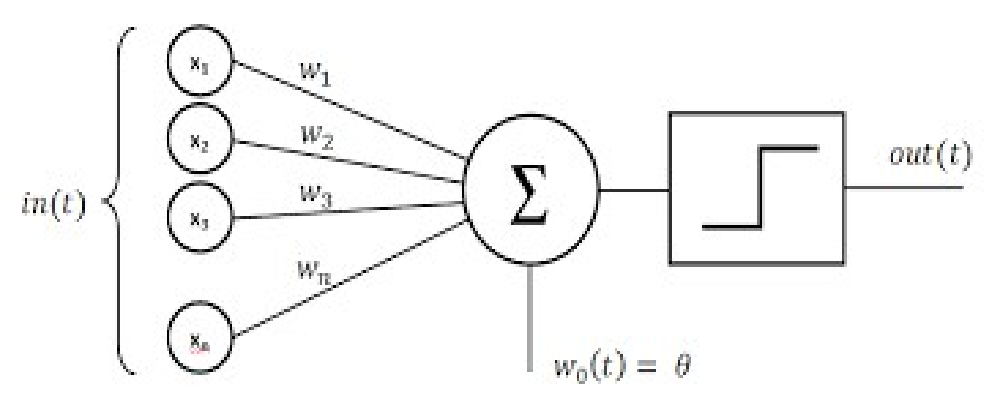
  ¿Es una red neuronal?

  ¿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:

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

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

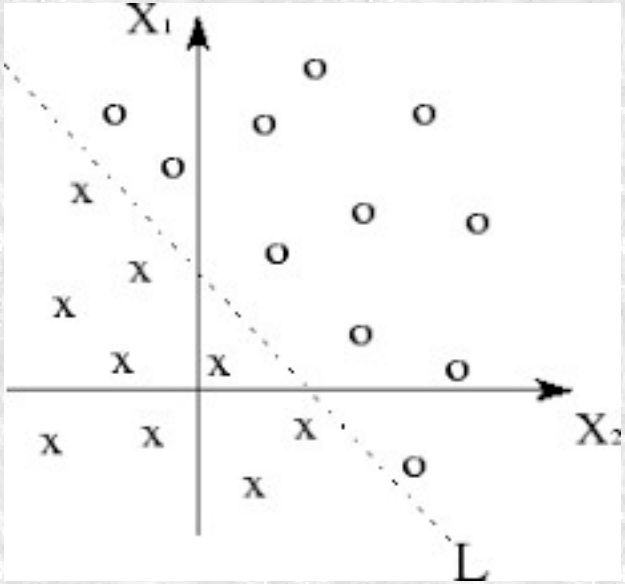
O

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

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

Support Vector Machine

Problemas *linealmente separables*

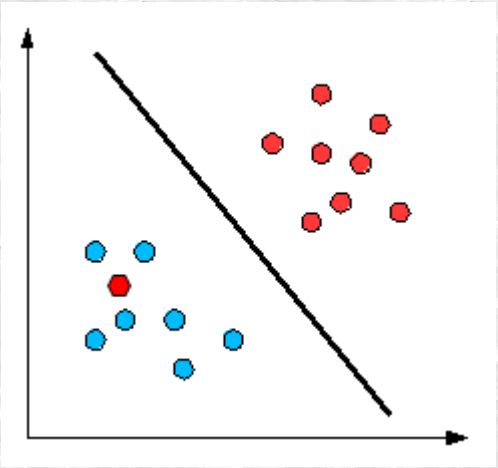


Si el problema es linealmente separable hay un algoritmo (*algoritmo del perceptrón)* que encuentra una solución en tiempo finito

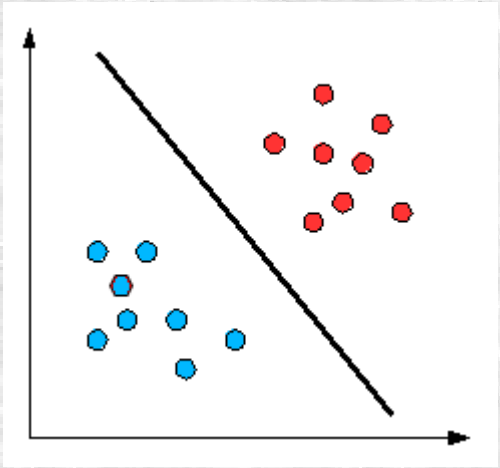
Support Vector Machine

Problemas *linealmente separables: p* entradas en N dimensiones

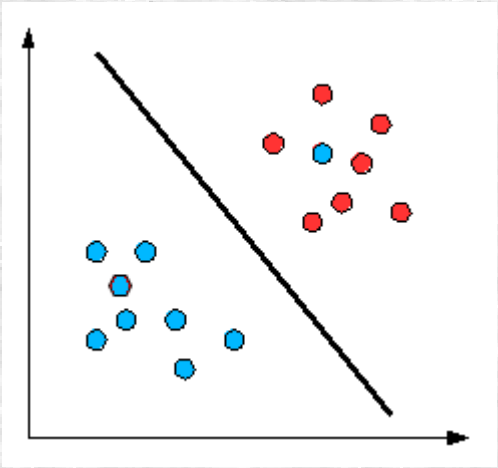
No



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? → 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):

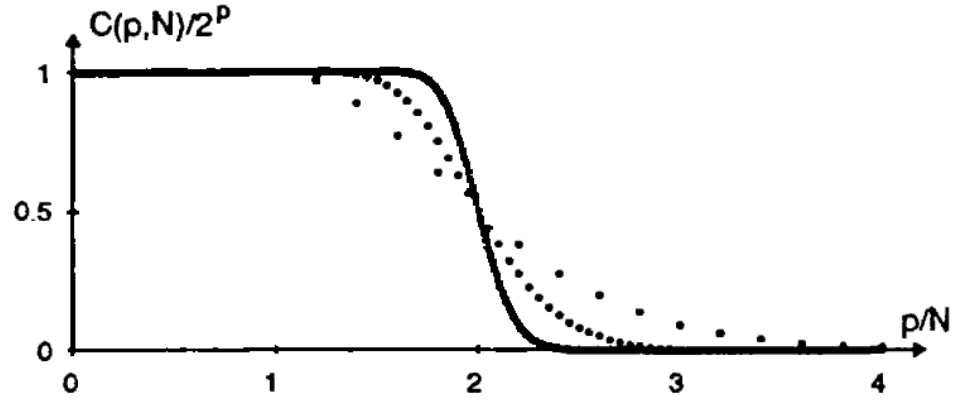


FIGURE 5.11 The function $C(p, N)/2^p$ given by (5.67) plotted versus p/N for $N =$ 5, 20, and 100.

Support Vector Machine

Problemas *linealmente separables:*

Para dimensión grande TODO problema es linealmente separable si $p<2N$

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

$\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x})$

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

Support Vector Machine

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

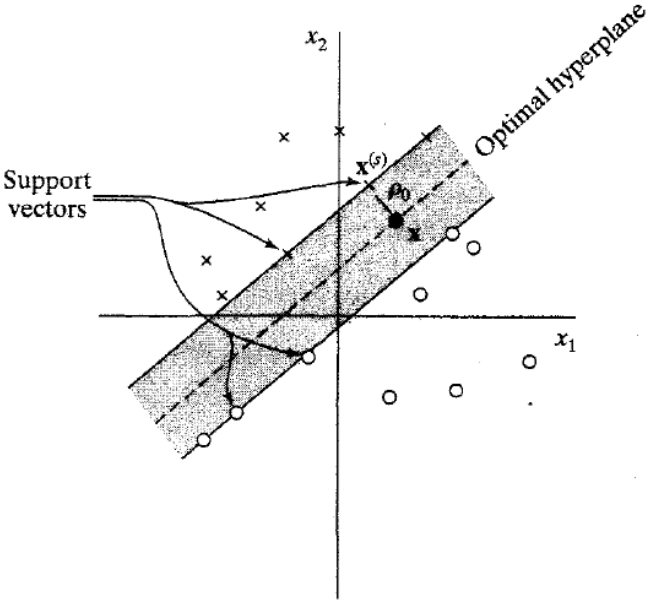
El número de parámetros es tan grande que voy a tener una situación de *overfitting*

Support Vector Machine

La solución no es única

¿Que solución tendrá menor error de generalización?

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



Support Vector Machine

Supongamos que los hiperplanos están definidos por

$\hat{\mathbf{w}} \cdot \mathbf{x} + ß = c$ $\qquad$ $\hat{\mathbf{w}} \cdot \Delta\mathbf{x} = 2\,c$

$\hat{\mathbf{w}} \cdot \mathbf{x} + ß = -c$

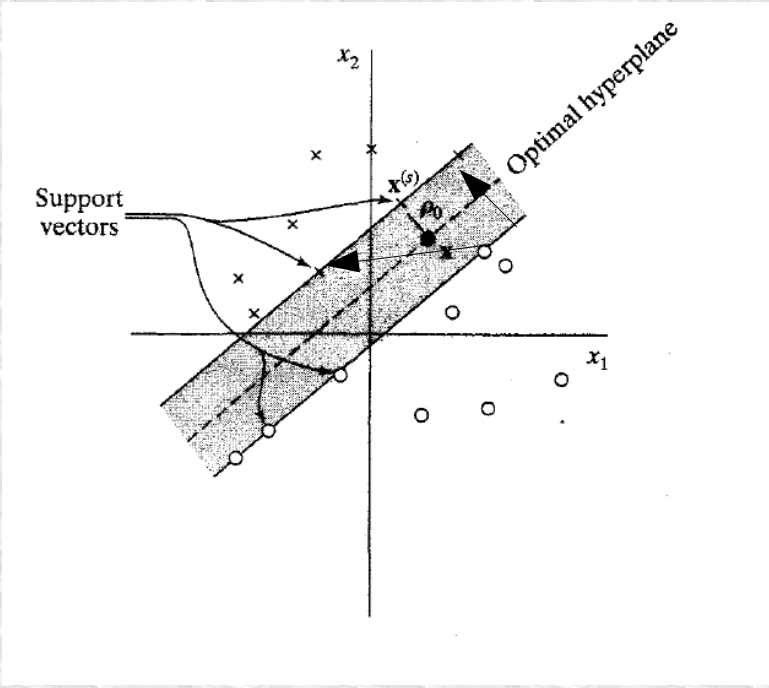
O equivalentemente:

$\mathbf{w} \cdot \mathbf{x} + b = 1$ $\qquad$ $\mathbf{w} \cdot \Delta\mathbf{x} = 2$

$\mathbf{w} \cdot \mathbf{x} + b = -1$ $\qquad$ donde $\mathbf{w}=\hat{\mathbf{w}}/c$, $b=ß/c$

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 $|\mathbf{w}|$

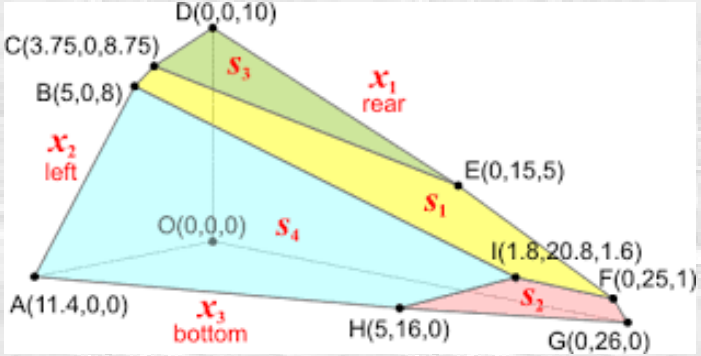
En este contexto el clasificador óptimo es que se obtiene de mimimizar $|\mathbf{w}|$ (o $|\mathbf{w}|^2$) con los constraints

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

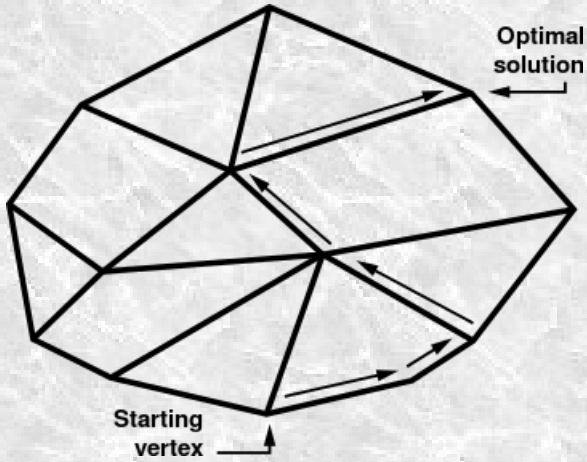
Problema de *programación cuadrática.*

Support Vector Machine

Los constraints definen un poliedro irregular



La solución involucra una búsqueda por las aristas de este objeto



Support Vector Machine

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

El vector $\mathbf{w}$ puede ser escrito como

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

Donde los α_i maximizan la función

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

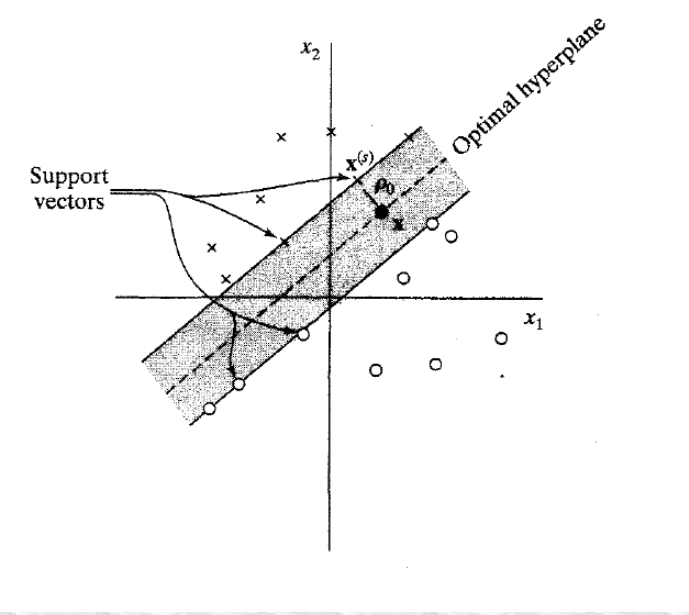
Con los constraints $\alpha_i \ge 0$, $\Sigma_{1 \le i \le p} \; \alpha_i \, y_i = 0$ (Haykin, p. 344-346)

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* $k(\mathbf{x}_i,\mathbf{x}_j) = \boldsymbol{\varphi}(\mathbf{x}_i).\boldsymbol{\varphi}(\mathbf{x}_j)$

El vector de peso óptimo es una combinación lineal de los **x** para los cuales los α son no nulos
→ vectores de apoyo



Support Vector Machine

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

Kernels usuales:

- Polynomial (homogeneous): $k(\overrightarrow{x_i}, \overrightarrow{x_j}) = (\overrightarrow{x_i} \cdot \overrightarrow{x_j})^d$.
- Polynomial (inhomogeneous): $k(\overrightarrow{x_i}, \overrightarrow{x_j}) = (\overrightarrow{x_i} \cdot \overrightarrow{x_j} + 1)^d$.
- Gaussian radial basis function: $k(\overrightarrow{x_i}, \overrightarrow{x_j}) = \exp(-\gamma \|\overrightarrow{x_i} - \overrightarrow{x_j}\|^2)$ for $\gamma > 0$. Sometimes parametrized using $\gamma = 1/(2\sigma^2)$.
- Hyperbolic tangent: $k(\overrightarrow{x_i}, \overrightarrow{x_j}) = \tanh(\kappa \overrightarrow{x_i} \cdot \overrightarrow{x_j} + c)$ 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

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

Observar que:

-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)

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)

Término de error:

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

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

Support Vector Machine

Soft-margin SVM linear (o no linear)

Queremos minimizar

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

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:

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

Donde α_i maximizan la función

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

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

Support Vector Machine

Comparación *SVM* vs. redes multicapa

SVM	Redes Multicapa
Funcion a optimizar cuadrática ($t=O(p^3)$)	Función a optimizar extremadamente complicada, múltiples mínimos locales
Generalización óptima garantizada	Generalización óptima determinada empíricamente
Requiere memoria $O(p^2)$	Requiere memoria O(batch size)

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: $C= (p\lambda)^{-1}$

-Optativo: clases desbalanceadas