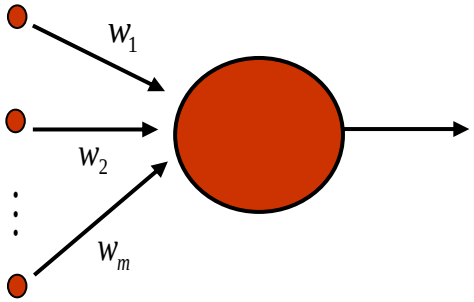
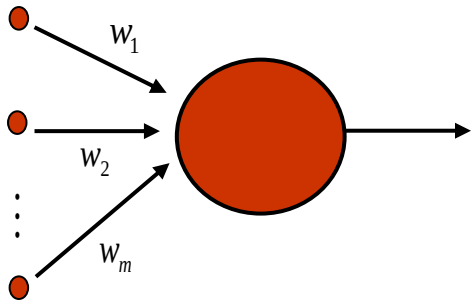
ALGUNOS MODELOS DE UNA NEURONA

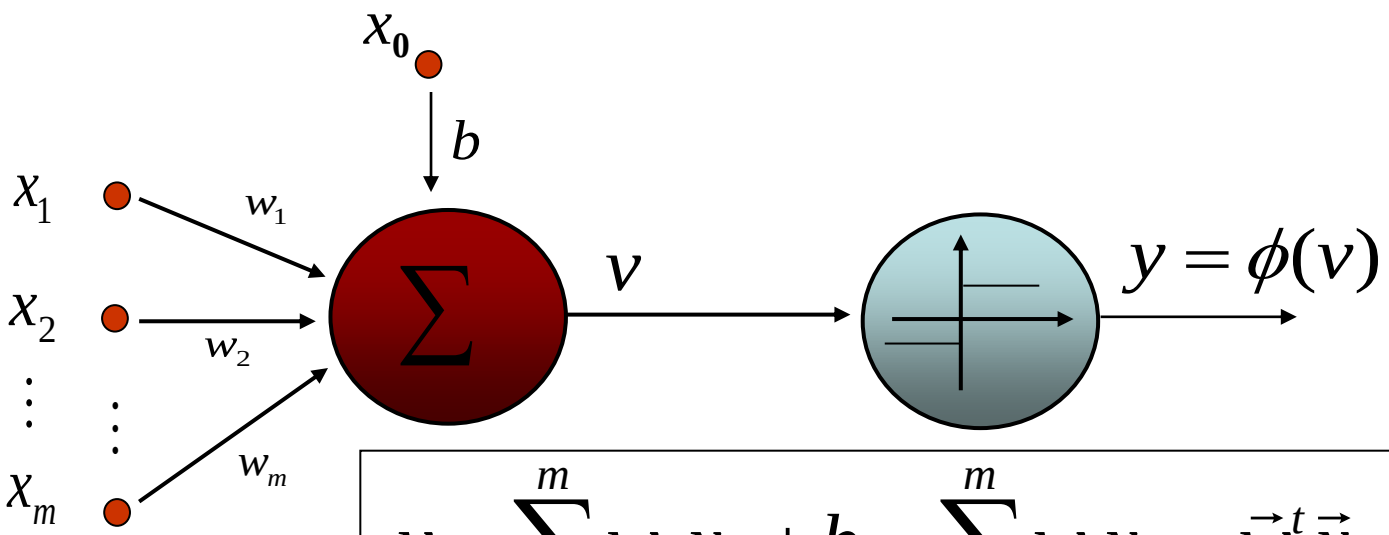


Neuronas de McCulloch-Pitts (El Perceptrón)

Rosenblatt, F. (1958), *The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain*, Psychological Review, Vol. 65, pp. 386-408

Rosenblatt, F. (1962), *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, Washington, DC.





$$v = \sum_{j=1}^{m} w_j x_j + b = \sum_{i=0}^{m} w_i x_i = \vec{w}^t \vec{x}$$

con

$$\vec{w} = \begin{bmatrix} b & w_1 & w_2 & \cdots & w_m \end{bmatrix}^t$$

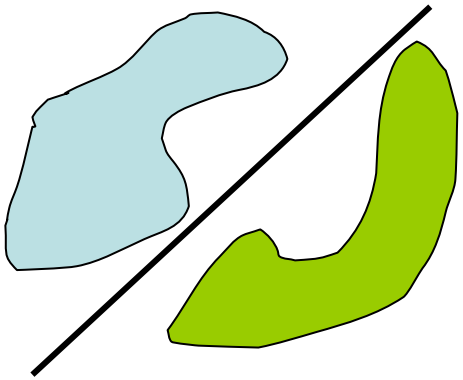
$$\vec{x} = \begin{bmatrix} 1 & x_1 & x_2 & \cdots & x_m \end{bmatrix}^t$$

$$\Rightarrow \quad y = \begin{cases} 1 & \vec{w}^t \vec{x} > 0 \\ -1 & \vec{w}^t \vec{x} \leq 0 \end{cases}$$

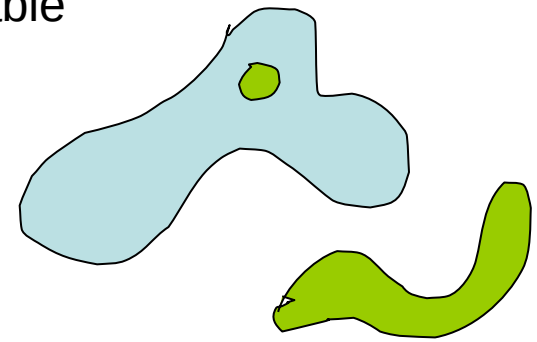
El Perceptrón divide al hiperplano en dos clases siempre y cuando estas sean linealmente separables.

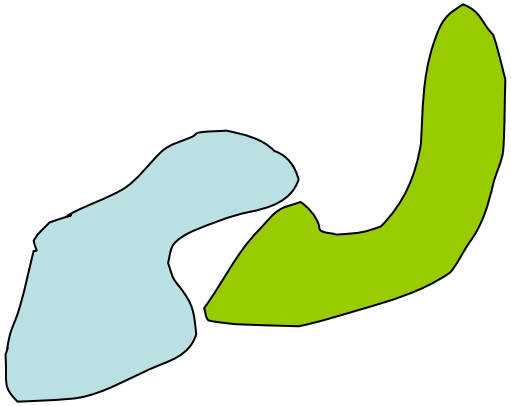
En 2D:

Si es separable



No es separable





No es linealmente separable

Entrenamiento: Actualizar los pesos sinápticos para minimizar el error de clasificación.

Algoritmo

Entrada: $X(n)=[1, x_1(n), x_2(n), \ldots\ldots, x_m(n)]^t$

$$d(n)=\begin{cases} 1 \text{ si } X(n) \text{ pertenece a } C_1 \\ -1 \text{ si } X(n) \text{ pertenece a } C_2 \end{cases}$$

η, constante positiva (tasa de aprendizaje)

Salida: $W(n)=[b(n), w_1(n), w_2(n), \ldots\ldots, w_m(n)]^t$

1) Inicializar $W(1)=0$, $n=1$. También se puede inicializar aleatorimente.

2) En el paso n, activamos el perceptrón con X(n), $y(n)=sgn[W^t(n) * X(n)]$;

3) Actualizar los pesos según la regla $W(n+1)= W(n) + \eta[d(n)-y(n)]X(n)$

4) $n= n+1$, retornar a 2) hasta alcanzar la condición de parada

Definición: Una época es la presentación del conjunto completo de datos.

OBSERVACIONES

1. **Definimos Δ(n) = d(n) –y(n). Δ(n) es el error de la clasificación en la iteración n.**
2. **Cuando el error de clasificación es 0, los pesos no se actualizan.**
3. **Condición de parada: Realizar tantas épocas como hagan falta hasta lograr que todos los Δ(n) de una época sean 0.**
4. **La inicialización de los pesos iniciales puede ser a cualquier valor.**
5. **Se puede demostrar que el algoritmo termina con Δ(n)=0 para cualquier valor de η positivo (OJO)**

Interpretación Geométrica

Queremos encontrar pesos W tales que

sgn($\vec{w}^t \vec{x}$) = d ⟵ La proyección del patrón **X** sobre **W** tenga el mismo signo que d

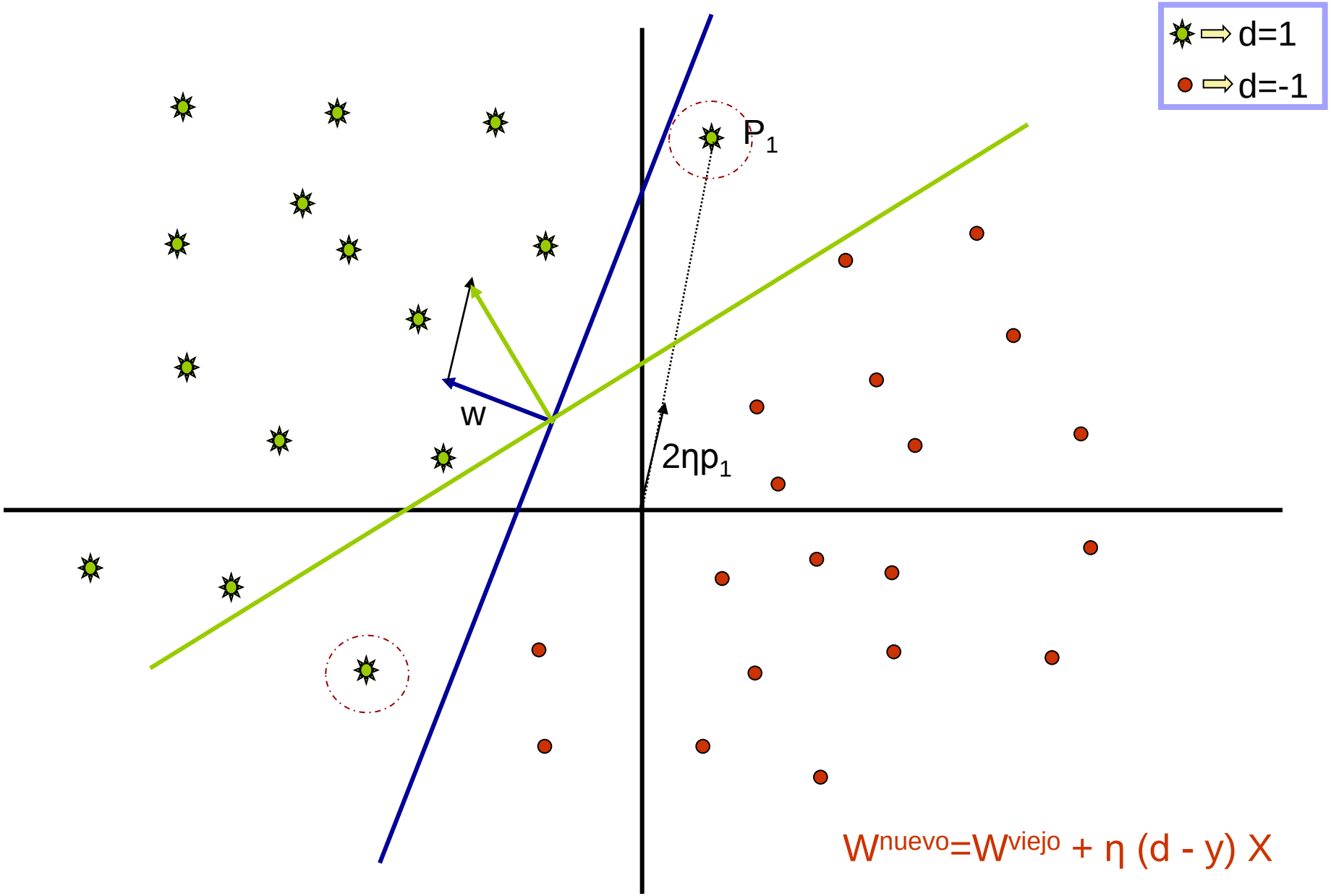
La frontera entre proyecciones positivas y negativas es.....

El hiperplano $\vec{w}^t \vec{x}=0$ ⟵ En 2D, es la ec. de la recta Con vector perpendicular **W**

Si y=d no cambio W

Si y≠ d actualizo según la regla

$W^{nuevo}=W^{viejo} + \eta (d - y) X$, **para todo d**



$$W^{nuevo} = W^{viejo} + \eta\,(d - y)\,X$$

Qué pasa si η es muy grande/pequeño?

Tardaremos MUCHO en converger

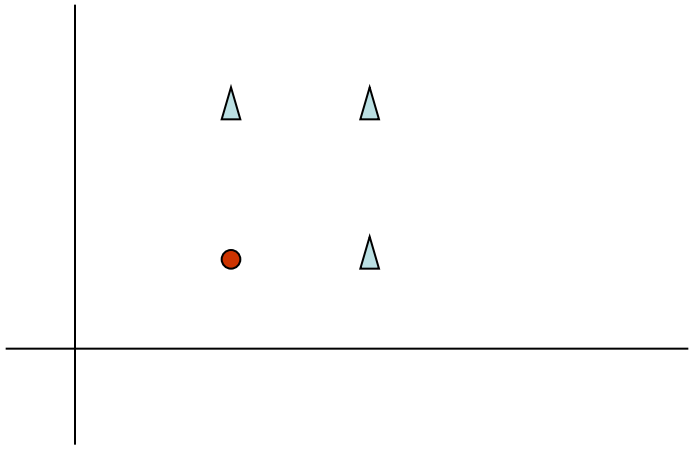
Qué pasa si P_1 es un punto atípico?

Tardaremos MUCHO en converger

Mejora es este caso: $W^{nuevo} = W^{viejo} + \eta (d - y) X / ||X||$

Ejemplo:

x_1	x_2	d(**X**)
1	1	-1
1	2	+1
2	1	+1
2	2	+1



Es linealmente separable? Si

Hay algún punto atípico? No

```
w = [.1,-.1,.1]';
Epoca = 0;
error = ones(1,4);
dibujarecta(w,X,d,Epoca)
while any(error)
      for i = 1 : 4,  %una epoca
        suma(i) = X(i,:)*w;
        if suma(i) > 0,
            y(i) = 1;
        else
            y(i) = -1;
        end;
        error(i) = d(i) - y(i);
       if error(i) ~= 0, % no converge, actualizo los pesos.
            w(1) = w(1) + Eta*error(i);
            w(2) = w(2) + Eta*error(i)*X(i,2);
            w(3) = w(3) + Eta*error(i)*X(i,3);
        end;
    end;
    Epoca = Epoca + 1;
    dibujarecta(w,X,d,Epoca)
end
```

TEOREMA DE CONVERGENCIA (Lippmann, 1987)

●Considere la entrada: $x_1, \quad x_2, \quad \cdots$ que representan muestras de dos clases linealmente separables, C_1 y C_2 , asuma que existe un vector $\boldsymbol{w}$ tal que

$$\vec{w}^t \vec{x} > 0 \quad \text{si} \quad \vec{x} \in C_1$$
$$\vec{w}^t \vec{x} \leq 0 \quad \text{si} \quad \vec{x} \in C_2$$

•Sea H_1 el subconjunto de entrenamiento que pertenecen a la clase C_1 .
•Sea H_2 el subconjunto de entrenamiento que pertenecen a la clase C_2

Si H es un conjunto de entrenamiento linealmente separable, entonces para η positivo, el algoritmo termina (converge).

Demostración:

Queremos encontrar los pesos tales que se satisfaga:

$$\vec{w}^t \vec{x} > 0 \quad \text{para } \vec{x} \in C_1$$
$$\vec{w}^t \vec{x} \leq 0 \quad \text{para } \vec{x} \in C_2 \qquad (1)$$

(Les quitamos las flechitas)

Si $x(n)$ no es clasificado correctamente actualizo los pesos según la regla:

$$w(n+1) = w(n) - \eta x(n) \text{ si } w^t(n)x(n) > 0 \text{ y } x(n) \in C_2$$
$$w(n+1) = w(n) + \eta x(n) \text{ si } w^t(n)x(n) \leq 0 \text{ y } x(n) \in C_1 \qquad (2)$$

η Controla el ajuste que se aplica en la iteración n

$\eta > 0$ Supongamos que :

- la constante es igual a 1, el valor no es importante, solo debe ser positivo. Valores distintos a 1 escalan el patrón sin afectar la separabilidad.
- w=0 inicialmente
- para $n = 1,2,3, ...,$ $w^t(n)x(n) < 0$ y $x(n)$ pertenece a H_1 (está mal clasificado)

⇨ $w(n+1) = x(1) + x(2) + \cdots + x(n)$ (3) Utilizando la regla (2) e iterando

Existe una solución w_0 para la cual $w^t(n)x(n) < 0$ Para $x(1), x(2), x(3), \ldots$ en H_1

Definimos $\mathbf{w^t(n)x(n) > 0}$ Multiplicando a (3) por w_o^t

⇨ $\alpha = \min w_0^t x(n) \text{ sobre } x(n) \in H_1$ (4)

Ahora,

$$\|w_o\|^2 \|w(n+1)\|^2 \geq \left[w_0^t w(n+1)\right]^2 \geq n^2\alpha^2$$

Des. Cauchy Schwartz (4)

$$\|w(n+1)\|^2 \geq \frac{n^2\alpha^2}{\|w_0\|^2}$$

Note que $w(i+1) = w(i) + x(i)$ para $x(i)$, una muestra de la clase C_1 con i = 1, 2, 3, …,n. Tomamos norma a ambos lados y obtenemos

$$\|w(i+1)\|^2 = \|w(i)\|^2 + \|x(i)\|^2 + 2w^t(i)x(i)$$

Pero el perceptrón clasifica incorrectamente (como asumimos al inicio), $w^t(i)x(i) < 0$ por lo tanto,

$$\|w(i+1)\|^2 \leq \|w(i)\|^2 + \|x(i)\|^2$$

$\|w(i+1)\|^2 \leq \|w(i)\|^2 + \|x(i)\|^2$ para $i = 1,2,3,\cdots,n$

entonces

$$(\|w(2)\|^2 - \|w(1)\|^2 \leq \|x(1)\|^2) +$$
$$(\|w(3)\|^2 - \|w(2)\|^2 \leq \|x(2)\|^2) +$$

$(\|w(n+1)\|^2 - \|w(n)\|^2 \leq \|x(n)\|^2)$ y usando que inicialmente w=0

$$\|w(n+1)\|^2 \leq \sum_{i=1}^{n} \|x(i)\|^2 \leq n\beta \text{ con } \beta = \max_{x(i) en H_1} \|\vec{x}(i)\|^2$$

Tenemos que: $$\frac{n^2\alpha^2}{\|w_0\|^2} \leq \|w(n+1)\|^2 \leq n\beta$$

Qué pasa con n grandes?

Debe existir un n tal que ambas desigualdades se satisfagan con igualdad, n_{max}

$$\frac{n_{\max}{}^2 \alpha^2}{\|w_0\|^2} = n_{\max}\beta \quad \Rightarrow \quad n_{\max} = \frac{\beta\|w_0\|^2}{\alpha^2}$$

Hemos probado que el proceso termina luego de un número finito de iteraciones

Como motiva esto al algoritmo propuesto?

$$w(n+1) = w(n) + \alpha\left[d(n) - y(n)\right]x(n)$$

Supongamos que x(n) está mal clasificado, por ejemplo:

Caso 1: X(n) pertenece a C_2 (d(n)=-1),pero y(n) =1 ⇒ d(n)-y(n)=-2

⇒ $w(n+1) = w(n) - \eta x(n)$

Caso 2: X(n) pertenece a C_1 (d(n)=1),pero y(n) =-1 ⇒ d(n)-y(n)=2

⇒ $w(n+1) = w(n) + \eta x(n)$

$$\eta = 2\alpha$$

Esta versión (o un equivalente) del algoritmo se conoce como la versión primal del algoritmo del perceptrón, muchas veces descrito como:

2) En el paso n , activamos el perceptrón con la entrada si $y(n)d(n) \leq 0$ entonces se actualizan los pesos según la regla $w(n+1) = w(n) + \eta d(n)x(n)$ y $b = b + \eta d(n)R^2$ ($R^2 = max \|x\_i\|$)

Pueden ver la equivalencia entre éste y la propuesta de algoritmo anterior?

El teorema de Novikoff dice que si existe un vector óptimo $\mathbf{w}_{opt}$ tal que

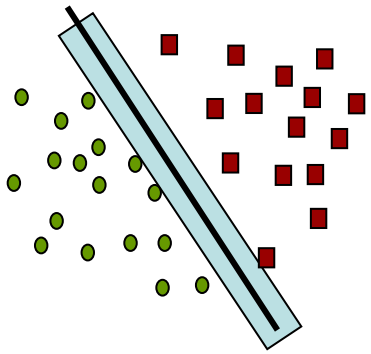
$$\|\mathbf{w}_{opt}\| = 1 \quad \text{y} \quad d_i \cdot \left(\langle \mathbf{w}_{opt} \cdot x_i \rangle + b_{opt} \right) \geq \gamma$$ **para todo i. Entonces**

el número de errores cometidos por el algortimo del perceptrón es a lo sumo

$$\left(\frac{2R}{\gamma} \right)^2$$

para un conjunto de entrenamiento no trivial (hay elementos de dos clases) y linealmente separable, con

$$R = \max_{1 \leq i \leq l} \|x_i\| \quad \text{y} \quad \gamma$$ **el margen del conjunto de entrenamiento**



- **El teorema de Novikoff dice que no importa cuan pequeño sea el márgen que separa la data, si esta es linealmente separable, entonces el preceptrón encontrará una solución que separa a las dos clases en un número finito de pasos.**

- **El número de pasos (tiempos de corrida) dependerá del márgen de separación y es inversamente proporcional al cuadrado del márgen.**

Recordar que la función de activación para el perceptrón es

$$y_i = \operatorname{sgn}\left(\langle w \cdot x \rangle + b\right)$$

Es importante notar que los pesos del perceptrón no son otra cosa que:

$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i d_i \mathbf{x}_i$$

(en escencia lo que hacemos es sumar los patrones mal clasificados en virtud de la regla de actualización $w(n+1) = w(n) + \eta d(n) x(n)$ y que inicializamos en 0)

Notar que los α_i son valores positivos proporcionales al número de veces que el patrón $x(i)$ dio origen a una actualización del vector de pesos.

El vector de α puede darnos información importante!

Una representación alterna (dual), viene de la siguiente relación:

$$
\begin{aligned}
y(\mathbf{x}) &= \operatorname{sgn}\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right) \\
&= \operatorname{sgn}\left(\left\langle \sum_{j=1}^{l} \alpha_j d_j \mathbf{x}_j, \mathbf{x} \right\rangle + b\right) \\
&= \operatorname{sgn}\left(\sum_{j=1}^{l} \alpha_j d_j \langle \mathbf{x}_j, \mathbf{x} \rangle + b\right)
\end{aligned}
$$

El algortimo del perceptrón puede ser re-escrito en forma dual:

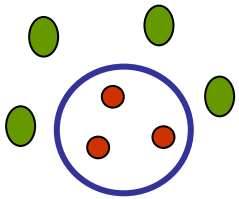
1) **Inicializar $\alpha = 0$; $b = 0$; Calcular R.**
2) **Para cada dato (patrón) si** $d_i\left(\sum_{j=1}^{l} \alpha_j d_j \langle x_j \cdot x_i \rangle + b\right) \leq 0$

   Actualizo $\alpha_i = \alpha_i + 1$; $b = b + d(i)R^2$
3) **Repetir el paso 2 hasta que todos los patrones estén correctamente clasificados.**

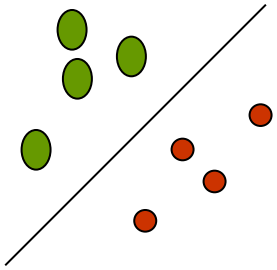
Hasta ahora hemos asumido la premisa de la separabilidad lineal, pero cómo podemos saber si una data es linealmente separable?

Hay resultados que reducen la existencia de la separabilidad lineal a la existencia de una solución a un problema de optimización lineal. Ver artículo en material de lectura.

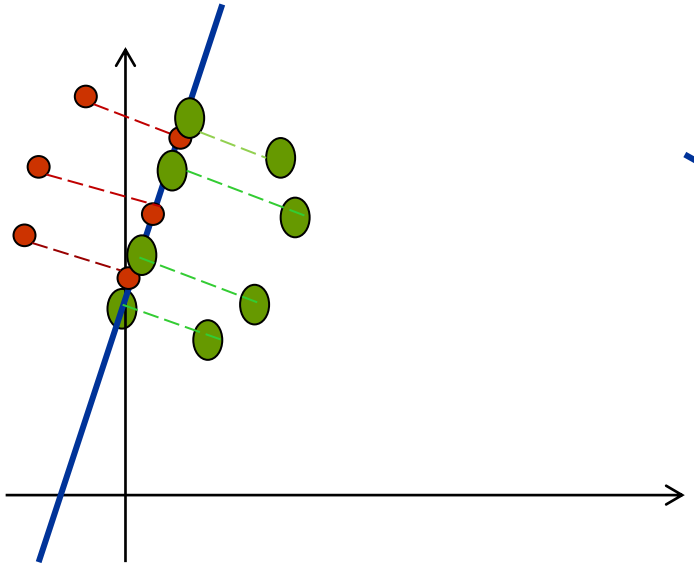
También hemos asumido el problema de separar entre dos clases distintas. En lo que veremos ahora vamos a tratar de relajar estas restricciones un poco.



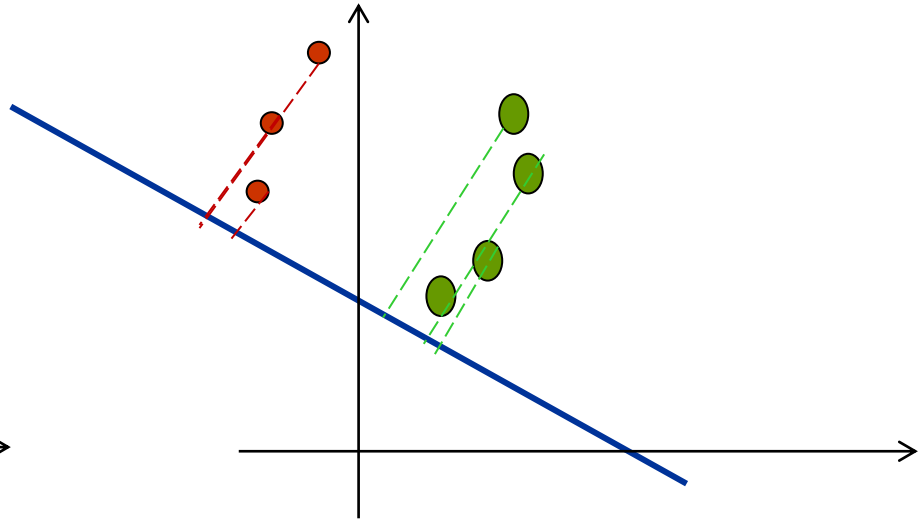
$$Z(P) = \{(x, y, x^2 + y\text{^}2)\}$$



$y = w^t x$ y decimos que clasificamos a C_1 si $y \geq -b$ o la otra clase en caso contrario

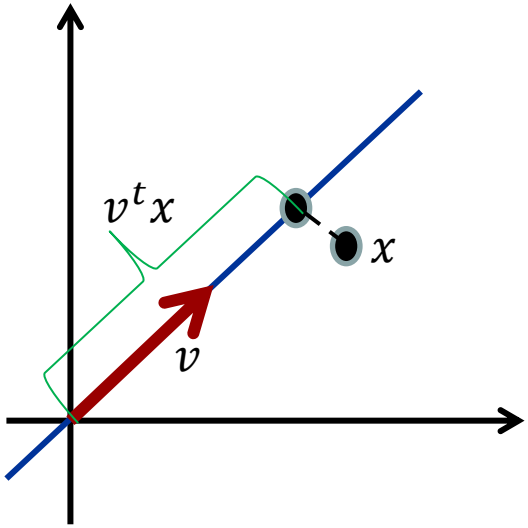


Esta recta es una mala escogencia para proyectar



Esta recta es una mejor elección, las clases están separadas.

La idea es elegir el vector de pesos que maximiza la separabilidad de las clases.

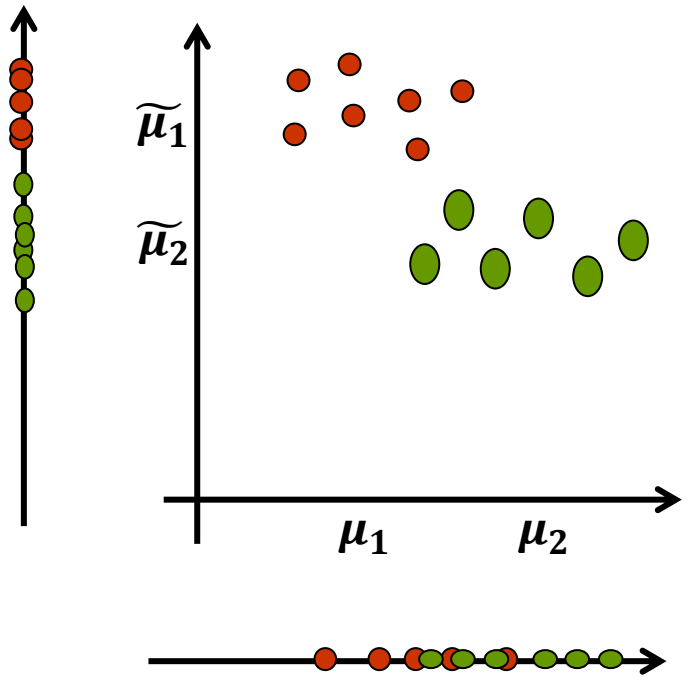


El escalar $v^t x$ es la distancia de la proyección de x del origen. Es la proyección de x a un espacio unidimensional.

Sean $\widetilde{\mu_1}$ y $\widetilde{\mu_2}$ las medias de las proyecciones de las clases 1 y 2. Sean μ_1 y μ_2 las medias de las clases.

$$\widetilde{\mu_1} = \frac{1}{n_1}\sum_{x_i \in C_1}^{n_1} v^t x_i = v^t\left(\frac{1}{n_1}\sum_{x_i \in C_1}^{n_1} x_i\right) = v^t \mu_1$$

De qué manera podríamos medir la separación entre las clases?



La cantidad $|\widetilde{\mu_1} - \widetilde{\mu_2}|$ parece una buena idea. Mientras mayor sea $|\widetilde{\mu_1} - \widetilde{\mu_2}|$ mejor será la separabilidad esperada…..

Aquí $|\widetilde{\mu_1} - \widetilde{\mu_2}| > |\mu_1 - \mu_2|$. El problema es que esta medida no considera la varianza en la clase.

Idea: Normalizamos por un factor proporcional a las varianzas (dispersión).

Sean $z_1, z_2, \cdots, z_n$ muestras, su media está dada por μ_z. Definimos

$$S_z = \sum_{i=1}^{n} (z_i - \mu_z)^2$$

Sean $Y_i = v^t x_i$ las proyecciones de las muestras. Definimos

$$\tilde{S}_j^{\,2} = \sum_{y_i \in C_j} (y_i - \tilde{\mu}_2)^2$$

La idea de Fisher es encontrar una dirección tal que maximize

$$J(v) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\left(\tilde{S}_1^{\,2} + \tilde{S}_2^{\,2}\right)}$$

Las medias proyectadas lo más separadas posible

La menor dispersión en la clase proyectada.

Luego de algo de álgebra y cálculo (Revisar Bishop, cap 4), se puede demostrar que la dirección que maximiza $J(v)$ es proporcional a la diferencia entre las medias de las clases!!

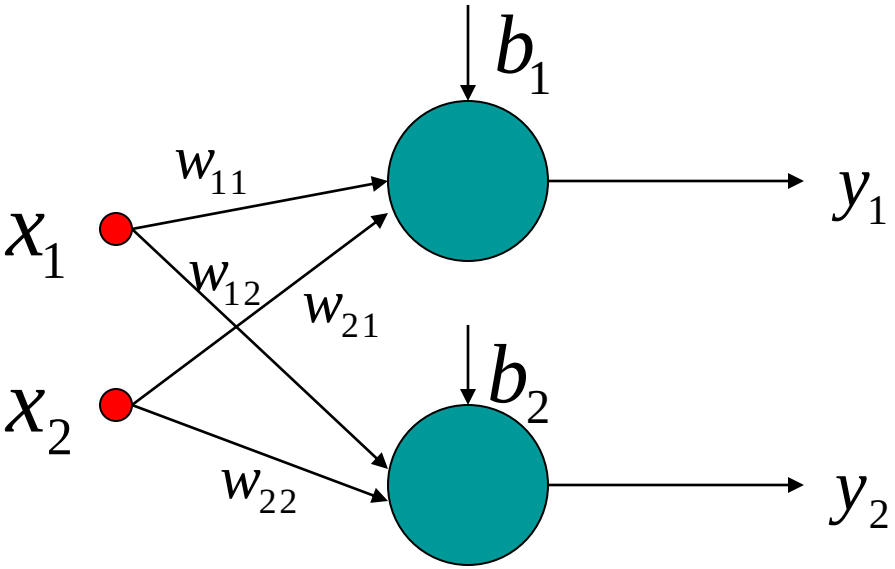
$$v \propto S_w^{-1}(\mu_1 - \mu_2) \text{ con } S_w^{-1} = S_1 + S_2.$$

Este resultado se conoce como el discriminante de Fisher que en si mismo no discrimina, pero que puede ser útil discriminado si lo acoplamos con un umbral (si $y(x) \geq y_0$ entonces pertecene a una clase).

Resumen:

- **Definimos el perceptrón y le dimos una interpretación geométrica.**
- **Demostramos el teorema de convergencia para el perceptrón**
- **Definimos las versiones primales y duales del perceptrón (útil más adelante cuando veamos SVM)**
- **Vimos que el preceptrón hace un buen trabajo solo cuando la data el liealmente separable.**
- **La separabilidad lineal se puede lograr si transformamos el espacio (más sobre este punto en semanas venideras), pero podemos tratar de encontrar un clasificador que pueda maximizar la separabilidad entre las clases (discriminante de Fisher).**

Volviendo al perceptrón…….



Una entrada

$$\begin{pmatrix} x_1(k) \\ x_2(k) \end{pmatrix}$$

tiene valor deseado

$$\begin{pmatrix} d_1(k) \\ d_2(k) \end{pmatrix} = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$$

$$y_1 = \varphi_1\left(x_1 w_{11} + x_2 w_{21} + b_1\right)$$

$$y_2 = \varphi_1\left(x_1 w_{12} + x_2 w_{22} + b_2\right)$$

En general, cuando hay k neuronas (perceptrón múltiple) la salida correspondiente a la red se puede escribir vectorialmente como

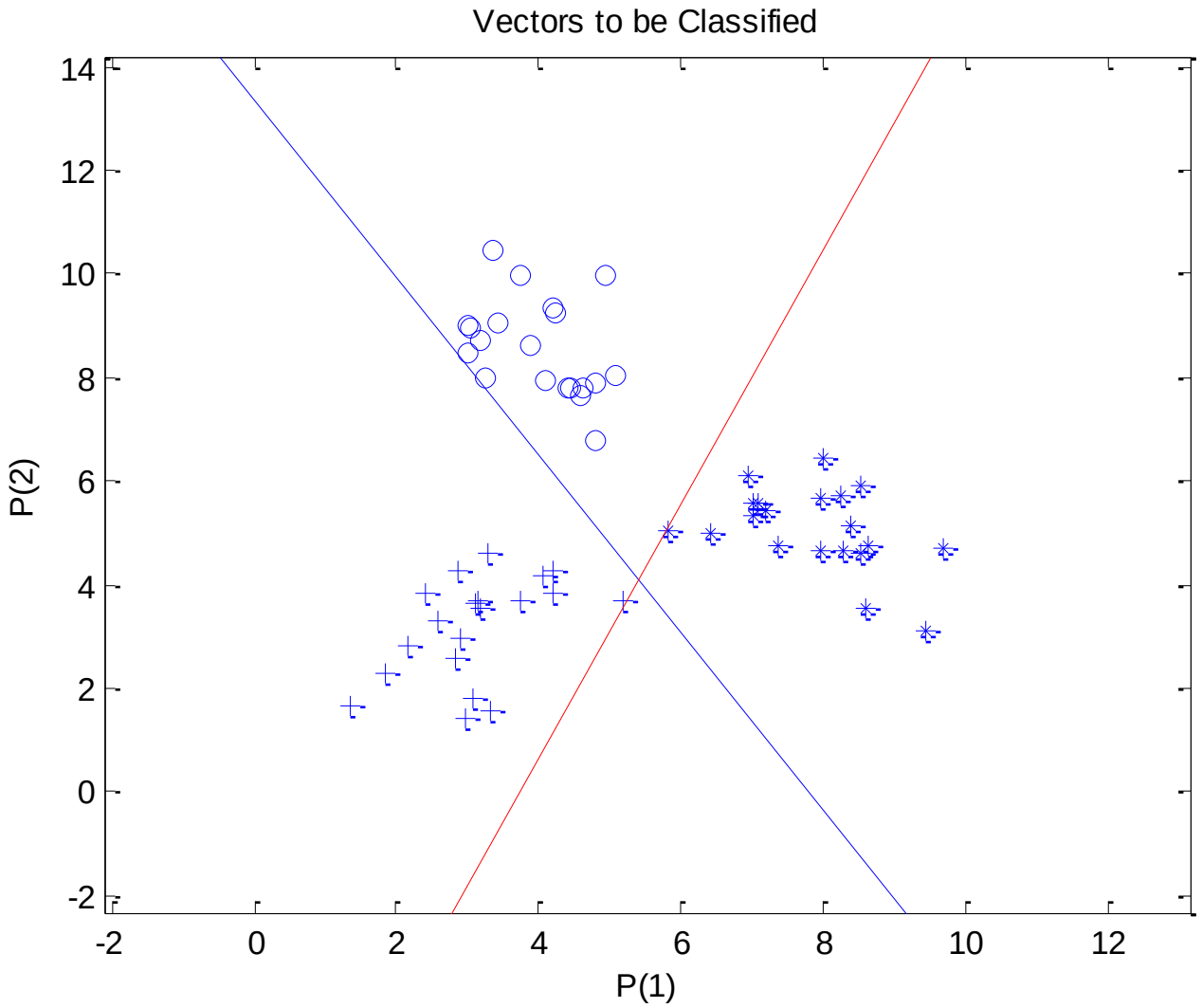
$$\phi(W^T \cdot \vec{x} + \vec{b}) = \vec{\phi}\left\{ \begin{bmatrix} w_{11} & w_{21} & \cdots & w_{k1} \\ w_{12} & w_{22} & \cdots & w_{k2} \\ \vdots & \vdots & \cdots & \vdots \\ w_{1m} & w_{2m} & \cdots & w_{km} \end{bmatrix}^T \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} \right\}$$

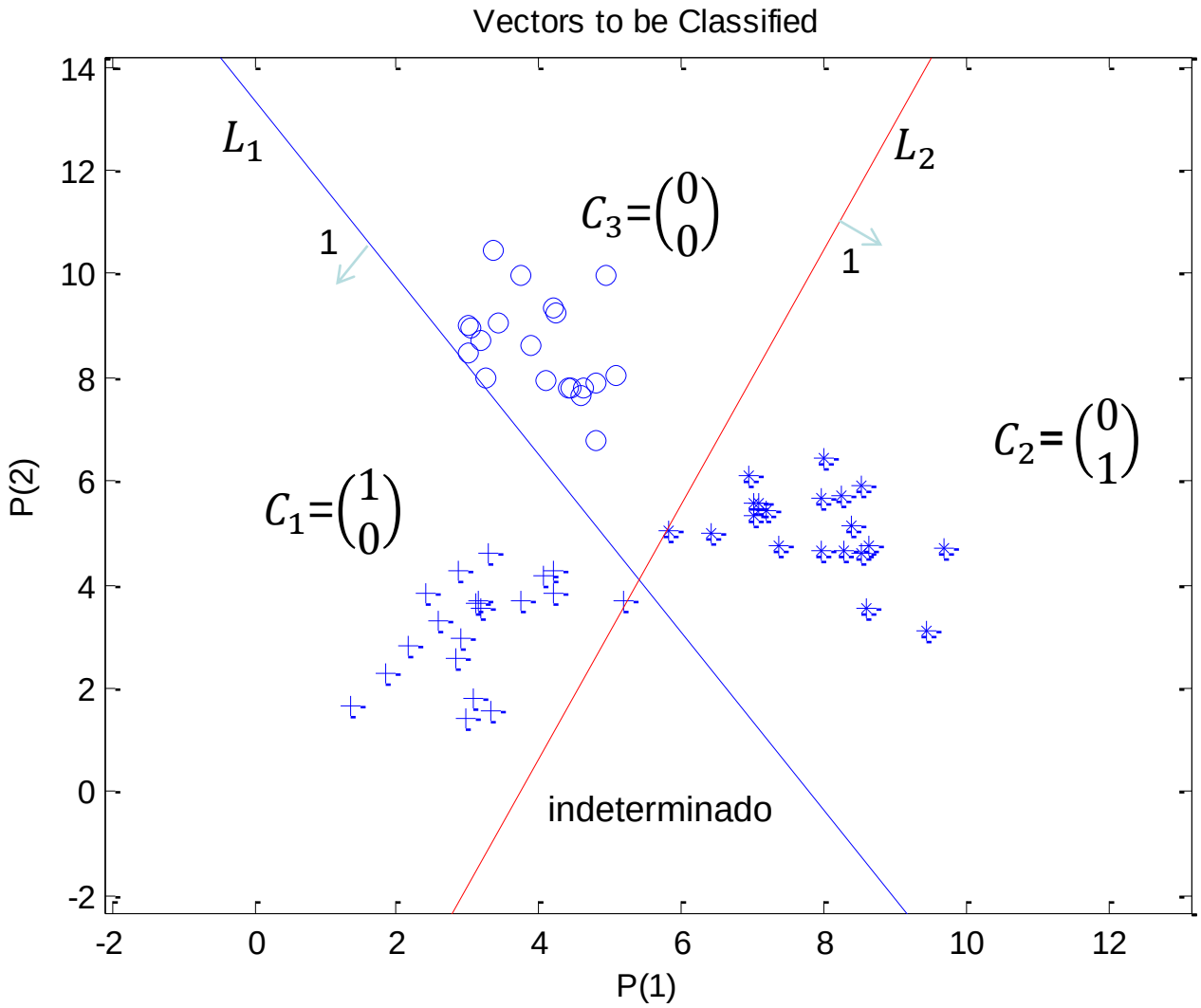
Los pesos se actualizan según la regla

$$\begin{bmatrix} w_{11} & w_{21} & \cdots & w_{k1} \\ w_{12} & w_{22} & \cdots & w_{k2} \\ \vdots & \vdots & \cdots & \vdots \\ w_{1m} & w_{2m} & \cdots & w_{km} \end{bmatrix} = \left[\begin{bmatrix} w_{11} & w_{21} & \cdots & w_{k1} \\ w_{12} & w_{22} & \cdots & w_{k2} \\ \vdots & \vdots & \cdots & \vdots \\ w_{1m} & w_{2m} & \cdots & w_{km} \end{bmatrix}^T + \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_k \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \right]^T$$

$$\begin{bmatrix} w_{11} & w_{21} & \cdots & w_{k1} \\ w_{12} & w_{22} & \cdots & w_{k2} \\ \vdots & \vdots & \cdots & \vdots \\ w_{1m} & w_{2m} & \cdots & w_{km} \end{bmatrix} = \left[\begin{bmatrix} w_{11} & w_{21} & \cdots & w_{k1} \\ w_{12} & w_{22} & \cdots & w_{k2} \\ \vdots & \vdots & \cdots & \vdots \\ w_{1m} & w_{2m} & \cdots & w_{km} \end{bmatrix}^T + \begin{bmatrix} e_1 x_1 & e_1 x_2 & \cdots & e_1 x_m \\ e_2 x_1 & e_2 x_2 & \cdots & e_2 x_m \\ \vdots & \vdots & \cdots & \vdots \\ e_k x_1 & e_k x_2 & \cdots & e_k x_m \end{bmatrix} \right]^T$$

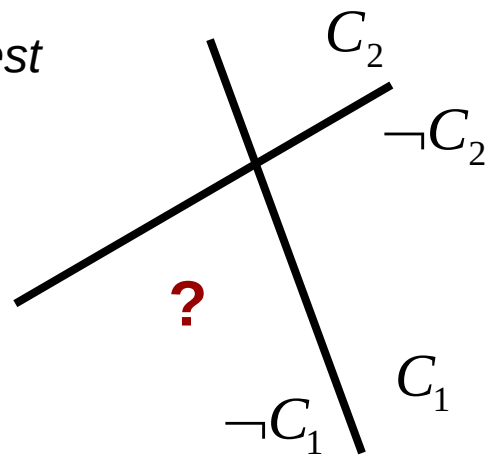
$$\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_k \end{bmatrix}$$


Vectors to be Classified
P(2)
P(1)
14
12
10
8
6
4
2
0
-2
-2
0
2
4
6
8
10
12

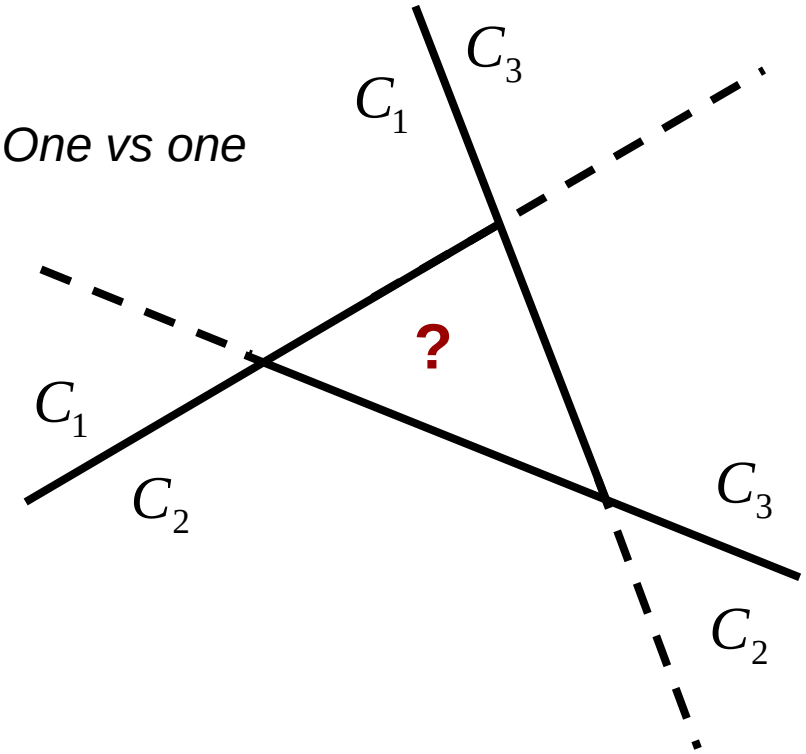

Vectors to be Classified
L_1
L_2
1
1
$C_3=\begin{pmatrix}0\\0\end{pmatrix}$
$C_1=\begin{pmatrix}1\\0\end{pmatrix}$
$C_2=\begin{pmatrix}0\\1\end{pmatrix}$
indeterminado
P(1)
P(2)
14
12
10
8
6
4
2
0
-2
-2
0
2
4
6
8
10
12

Si tenemos más clases, la cosa no es tan sencilla.

One vs rest



One vs one

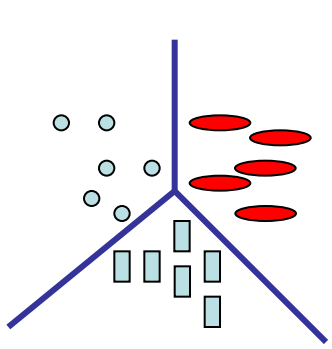


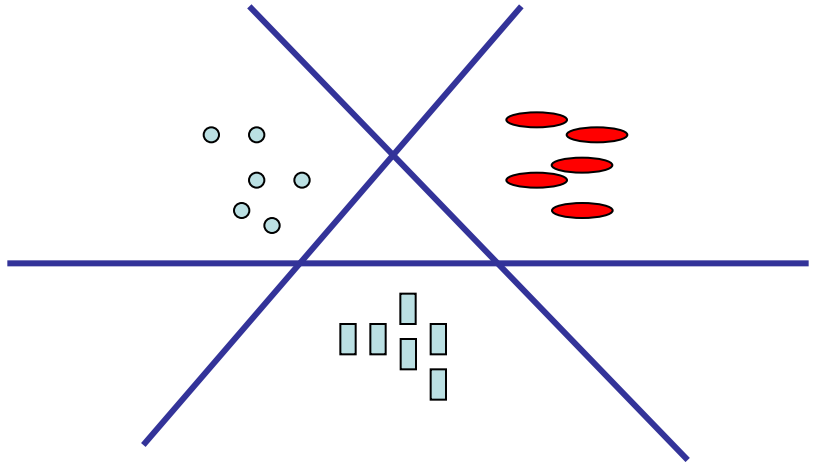
Usar K(k-1)/2 discriminantes (por par)

Suponga que se tienen multiples clases (L) y se cuentan con M patrones para el entrenamiento ($X^1, X^2, \ldots . . XM$) . Con este postulado de aprendizaje cada vector de peso representa una clase. (*Winner takes all)*

El conjunto se dice linealmente separable si existen vectores de pesos w_i con $i = 1, \ldots, L$ tal que para cualquier patrón x en la i-ésima clase

$$w_i^t \, x \; > \; w_j^t \, x \;\; \forall j \neq i$$





En esencia estamos usando K funciones lineales de la forma:

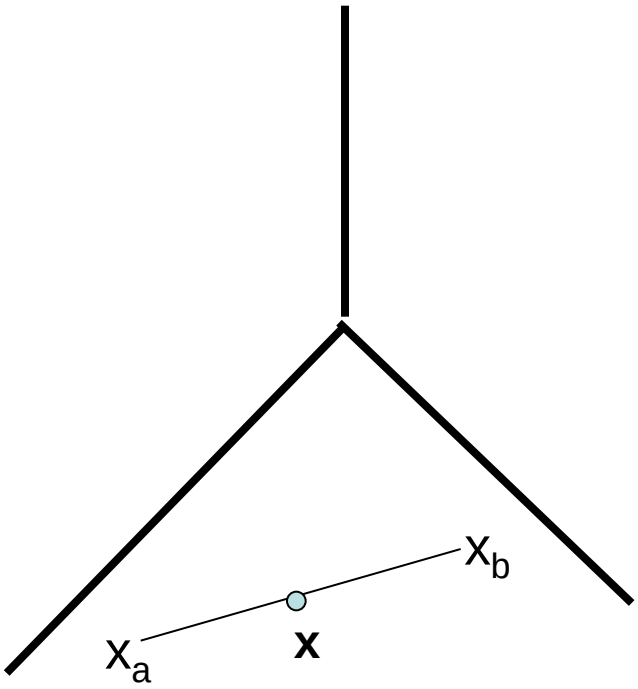
$$y_k(x) = w_k^t x + w_{k0}$$

$$x \in C_k \text{ si } y_k(x) > y_j(x), \ \forall j \neq k$$

La frontera de decisión está dada por: $y_k(x) = y_j(x)$

Que es un hiperplano definido por: $\left(w_k - w_j\right)^t x + \left(w_{k0} - w_{j0}\right) = 0$

La región de decisión es conexa y convexa



$$x = \lambda x_a + (1-\lambda) x_b$$

$$y_k(x) = \lambda y_k(x_a) + (1-\lambda) y_k(x_b)$$

dado que

$$y_k(x_a) > y_j(x_a), \ \forall j \neq k \ \text{ y}$$

$$y_k(x_b) > y_j(x_b), \ \forall j \neq k,$$

$$y_k(x) > y_j(x)$$

Suponga que el patrón z se sabe pertenece a la clase i y que la neurona ganadora (la clase ganadora) se denota por un entero j, por lo tanto $\forall l \neq j$,

$$w_j^t z > w_l^t z$$

1. Cuando $j = i$ (el patrón está correctamente clasificado), no hay actualizaciones.
2. Cuando $j \neq i$ (z está mal clasificado), se hacen las siguientes actualizaciones:

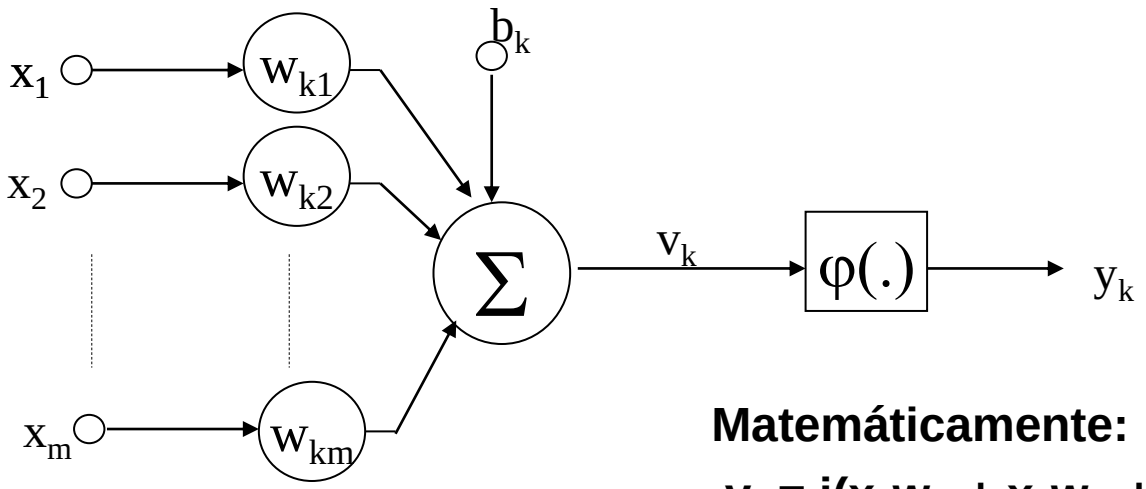
    1. RL: $W_i^{nuevo} = W_i^{viejo} + h\, z$ (i)

    2. ARL: $W_j^{nuevo} = W_j^{viejo} - h\, z$ (j)

    Dejando los demás pesos sin alterar

Este esquema es similar al previsto en el teorema de convergencia, por lo que cuando la data es linealmente separable, se puede probar que converge en un número finito de pasos.

Una neurona con múltiples entradas



Matemáticamente:

$$y_k = j(x_1 w_{k1} + x_2 w_{k2} + \ldots\ldots + x_m w_{km} + b_k)$$
$$= j(XW + b)$$

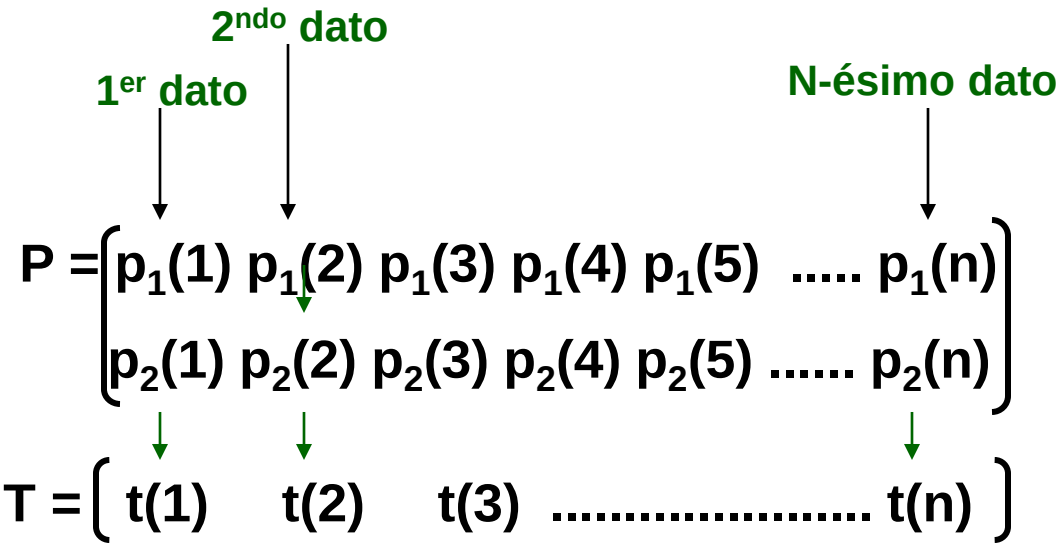
En Matlab la data es representada en forma vectorial:

```
>>w=[2,-2]; b=2;
>>x=[-1,1]; out=w*x'+ b
   out=-2
```

¿Cómo entrenamos en Matlab a una neurona dada una data?

>>load classdata **(Esta data se puede obtener de aula virtual)**

>>who



Este ejemplo tiene dos coordenadas (2D)

t(i) = 1 o 0, dependiendo a qué grupo pertenece.

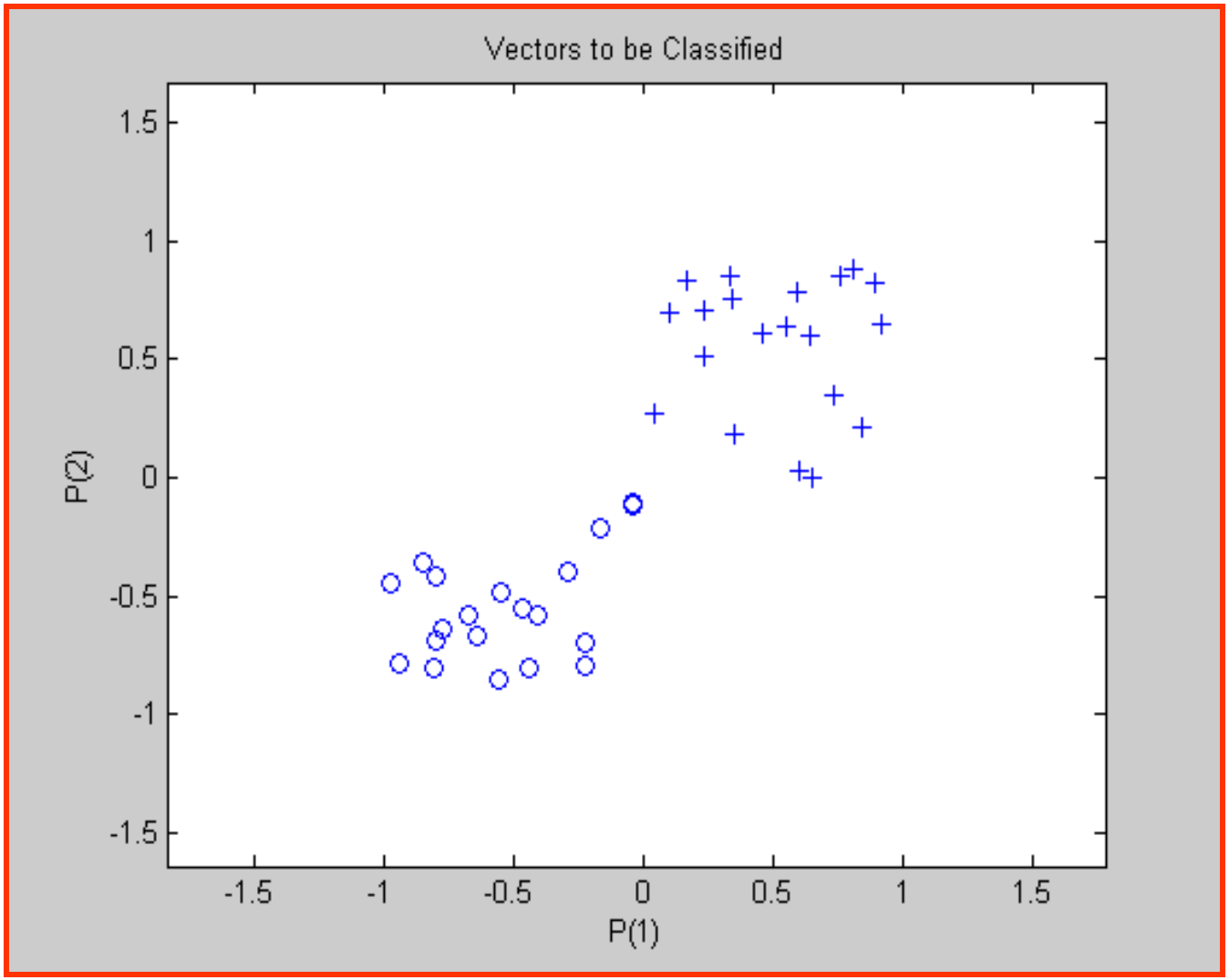
Misión: entrenar un perceptrón para que clasifique la data correctamente.

>>[P' T']

>>help plotpv

>>plotpv(P,T)

Estos comandos Ayudan a visualizar La data



```
>>help newp
>>net = newp([-2 2;-2 2],1);
```

Acabamos de crear un perceptrón con los rangos especificados y con una neurona. Veamos .....

```
>>net
>>net.layers{1}
>>net.IW{1}
>>net.b{1}
```

Exploremos las opciones

```
>>help(net.inputweights{1,1}.learnfcn)
>> net.inputWeights{1,1}.learnFcn='learnwh'
>>net.inputweights{1,1}
```

Recapitulamos

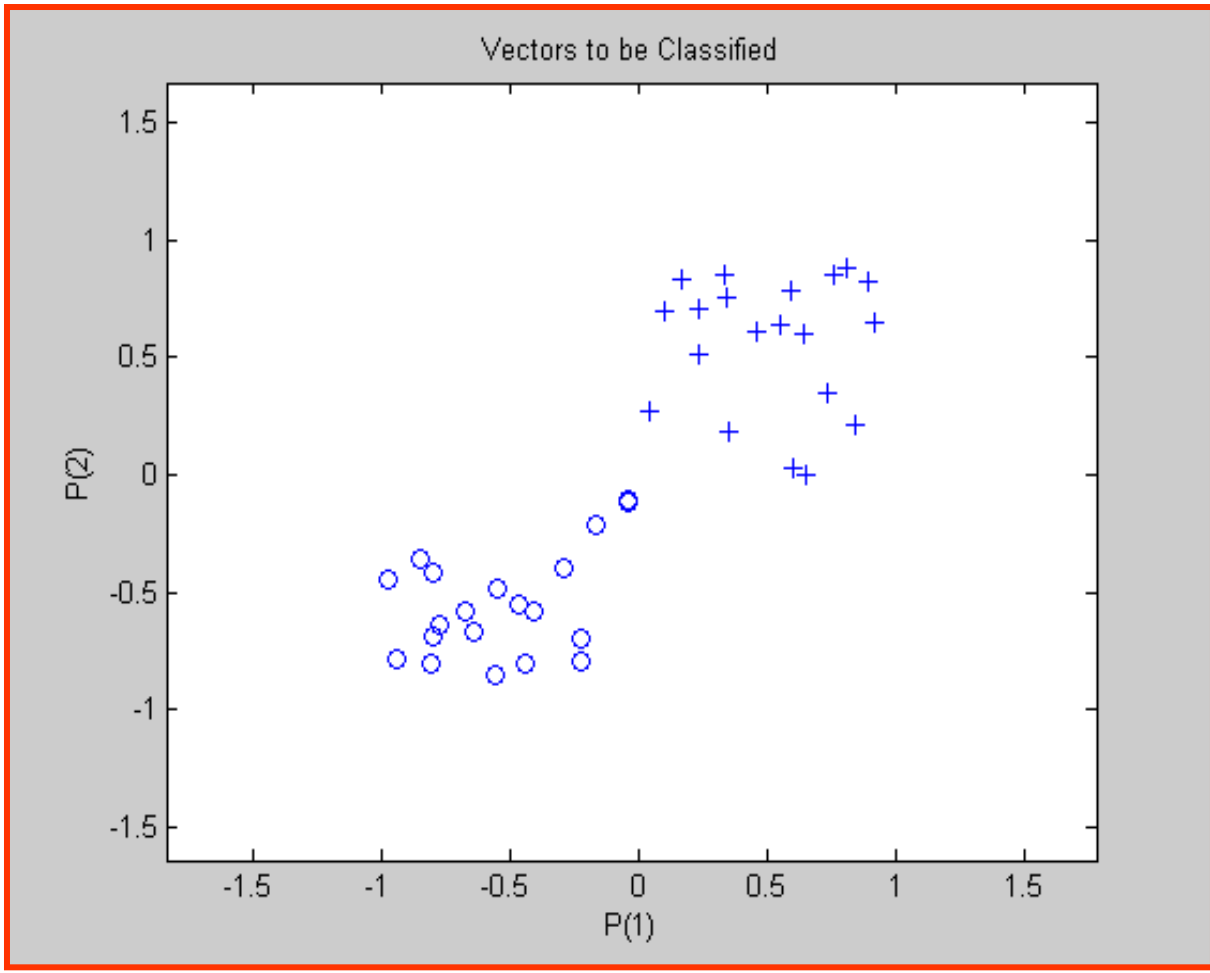
```
>>net=newp([-2 2; -2 2],1)
```

Originalmente los pesos son 0, asi que la neurona no hace nada.

>>help plotpc

>>plotpc(net.IW{1},net.b{1})

Con esto visualizamos el efecto del perceptrón



Automáticamente Matlab coloca marcadores distintos para cada caso

¿Cuál es el algoritmo de entrenamiento?

```
>>help adapt
>>help train
```

La diferencia radica en la forma en que se presentan los datos

Si los datos son presentados en una matriz la actualización es por lote (lo veremos la próxima clase)

```
>>[net1,y,e] = adapt(net,P,T);
>>mse(e)
```

Si los datos son presentados en celdas la actualización es secuencial (teorema)

```
texto2='TS={';
texto='PS={';
for i=1:40,
texto=[texto,'  [',num2str(P(1,i)),';',num2str(P(2,i)), ']'];
texto2=[texto2,'  ',num2str(T(i))];
end;
texto=[texto, '}'];
texto2=[texto2, '}'];
eval(texto)
eval(texto2)
```

```
>>[net2,y,e] = adapt(net,PS,TS);
>>mse(e)
```

Por defecto el número de pases por la data es 1.
Fijamos algunos parámetros del algoritmo

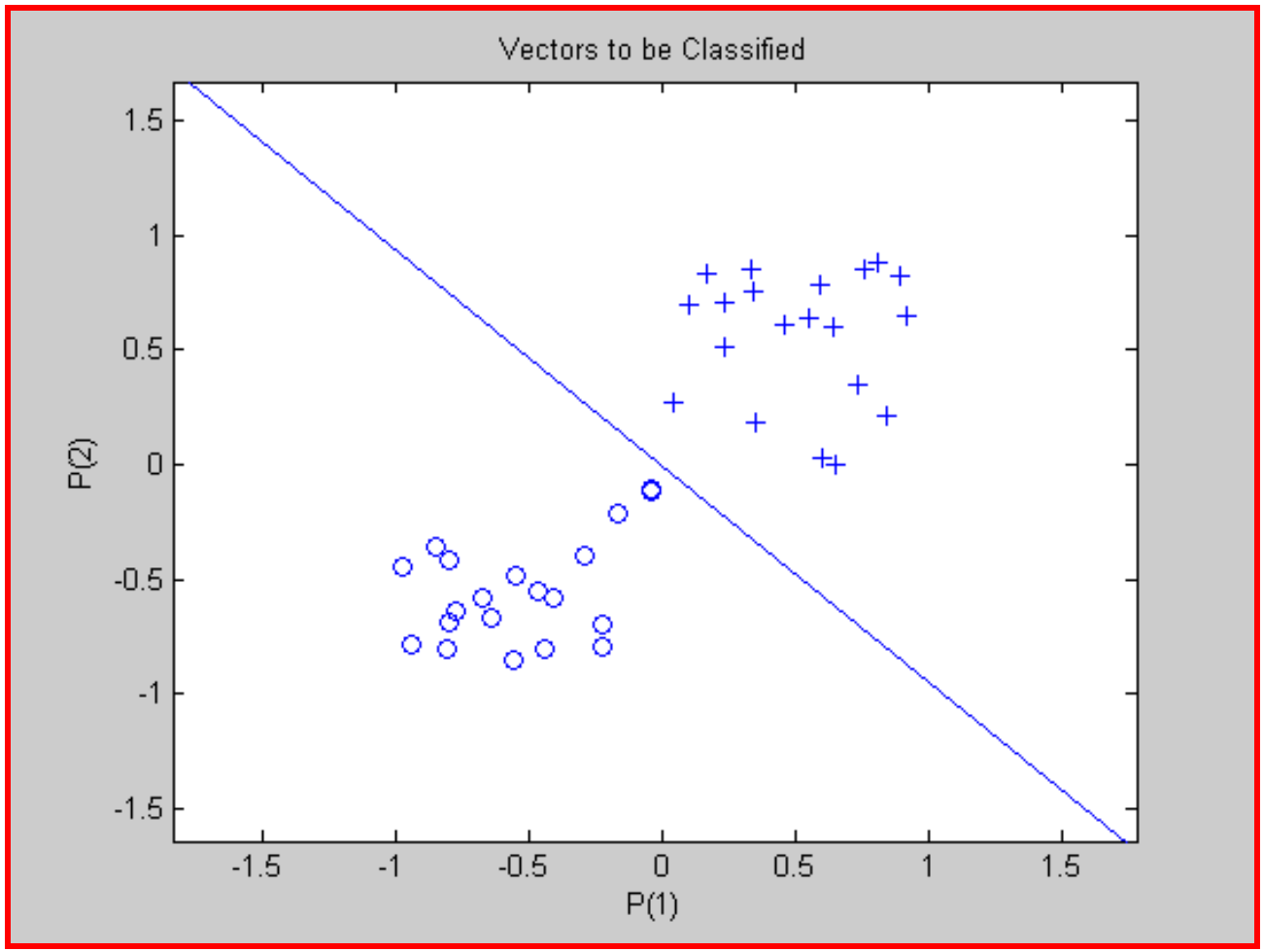
```
>>net.adaptParam.passes = 2;
```

Entrenamos la red y retornamos su valor.

```
>>[net,y,e] = adapt(net,P,T);
>>mse(e)
```

El perceptrón ha sido entrenado, es decir se han modificado los pesos para que la salida del perceptrón concuerde con el vector t.

```
>>plotpc(net2.IW{1},net2.b{1});
```



Hemos clasificado exitosamente!!

Exploramos la data de validación

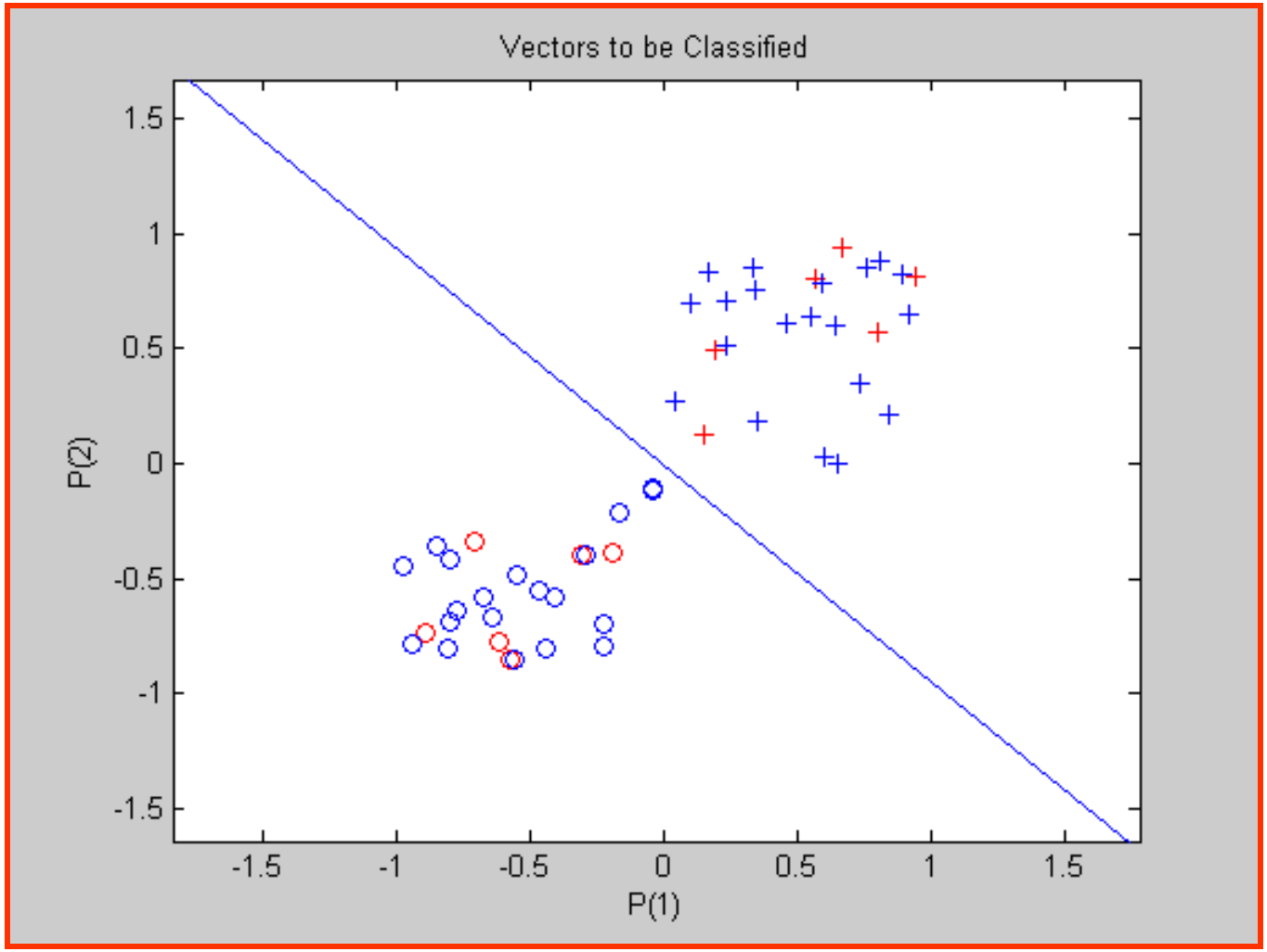
```
>>Val
```

La validación se realiza con la simulación de nueva data en la red (pasamos nueva información por el perceptrón)

```
>>help sim
>>a = sim(net,Val);
```

Ahora queremos mostrar en una misma gráfica la data de entrenamiento y la data que usamos para la validación pero al esta última le cambiamos el color a rojo.

```
>>plotpv(Val,a);
>>point = findobj(gca,'type','line');
>>set(point,'Color','red');
>>hold on;
>>plotpv(P,T);
>>plotpc(net.IW{1},net.b{1});
```


Vectors to be Classified
P(2)
P(1)
1.5
1
0.5
0
-0.5
-1
-1.5
-1.5
-1
-0.5
0
0.5
1
1.5

- **Se puede inicializar el perceptrón con otros valores, como por ejemplo valores aleatorios.**

```
>> net.inputweights{1,1}.initFcn = 'rands'
>> net.biases{1}.initFcn = 'rands'
>> net=init(net)
```

- **Un perceptrón puede aprender utilizando varias reglas de aprendizaje**
  - Learnp: El algoritmo para el cual probamos convergencia con parametro fijo
  - Learnpn: La iteración modificada para ser robusta ante puntos atípicos con parámetros fijos
  - Learnwh: la iteración LMS con parámetro variable. (lo veremos próxima clase)

- **Podemos también controlar la cantidad de "pasadas" del entrenamiento.**

```
>> net.TrainParam.epochs=1;
>>net=train(net,P,T);
>>net.IW{1}
```

De esa forma vemos el cambio de una iteración del algoritmo de entrenamiento.

Los perceptrones no se comportan bien cuando las clases no son linealmente separables. Podemos pensar en un conjunto óptimo de pesos.

optimalidad  **mayor número de clasificados**

- **Un perceptrón podria visitar un conjunto de pesos óptimos en una iteración del algoritmo del perceptrón y luego pasar al peor de todos.**
- **El algoritmo del perceptrón puede nunca estabilizarse en los casos no separables linealmente.**

El algoritmo del bolsillo (pocket) lo que hace es añadirle un refuerzo positivo al aprendizaje de los pesos buenos para estabilizar el algoritmo.

Idea: usar el algoritmo del perceptrón manteniendo un conjunto de pesos en el bolsillo.

Algoritmo

1) Inicializar W(0) a algún valor aleatorio.
2) Definir algún vector w_b (en el bolsillo).
3) Inicializar h_b =0 (un contador de historia de w_b).

4) En el paso n, calcular la actualización W(n+1) según el algoritmo del perceptrón. Usar el nuevo vector de pesos para calcular h (número de muestras clasificados correctamente)

Si $h>h_b$,
$w_b \leftarrow W(n+1)$
$h_b \leftarrow h$
fin

Se puede demostrar que el algoritmo del bolsillo (Pocket Algorithm) converge con probabilidad 1 a la solución óptima.

Gallant, S.I.
Neural Networks, IEEE Transactions on
Volume 1, Issue 2, Jun 1990 Page(s):179 - 191

Otras variantes:

-Thermal Perceptron algorithm

-Loss minimization algorithm

-Barycentric correction procedure

Si se quieren divertir:

- **Para practicar Matlab:Generar data aleatoria que siga ciertas especificaciones y data de validacion. Crear un perceptron y practicar. (ayuda: randn, uso de vectores)**

- **La data de las flores: (iris.dat) Esta es una data interesante, hay tres clases de irises que deben ser clasificadas según 4 atributos: long y ancho de la sepa y long y ancho de los pétalos. Mirar la data por pares y darse cuenta de la no-separabilidad de dos de los subconjuntos. Pueden ustedes segregar entre esta y las anteriores?**

- Vimos que el **perceptrón** (función umbral) puede resolver problemas de clasificación para data linealmente separable.
- Demostramos teorema de **convergencia del perceptrón.**
- Estudiamos la **interpretación geométrica** del algoritmo.
- Vimos formas alternas de formulación del algortimo.
- Estudiamos la interpretación del **discriminante de Fisher.**
- **Problema multiclases** (no siempre K neuronas para K clases es buena opción): formulación con **algoritmo por reforzamiento**.
- **Algoritmo del bolsillo** como una variante del algoritmo cuando la data no es linealmente separable.
- Algunos **ejemplos prácticos** + NNet toolbox