



TECNOLÓGICO
NACIONAL DE MÉXICO



PROGRAMA INSTITUCIONAL DE FORMACIÓN DOCENTE

Fundamentos de Machine Learning

Cómo aprenden las máquinas

Ildeberto de los Santos Ruiz

<https://isantosruiz.github.io/home>

Tecnológico Nacional de México
Instituto Tecnológico de Tuxtla Gutiérrez

Turix-Dynamics Diagnosis and Control Group

Los algoritmos

Una clase para aprendizaje supervisado

◆ className.m

```
classdef className
    properties
        params
    end

    methods
        function obj = className(args)
            obj.params = [];
        end

        function obj = fit(obj, X, y) % also named train()
            obj.params = [];
        end

        function y = predict(obj, X)
            y = [];
        end
    end
end
```

Regresión lineal

$$y = c_1 f_1(x) + c_2 f_2(x) + \cdots + c_n f_n(x)$$

Regresión lineal

$$y = c_1 f_1(x) + c_2 f_2(x) + \cdots + c_n f_n(x)$$

x	y
x_1	y_1
x_2	y_2
\vdots	\vdots
x_m	y_m

Regresión lineal

$$y = c_1 f_1(x) + c_2 f_2(x) + \cdots + c_n f_n(x)$$

x	y
x_1	y_1
x_2	y_2
\vdots	\vdots
x_m	y_m



$$c_1 f_1(x_1) + c_2 f_2(x_1) + \cdots + c_n f_n(x_1) = y_1$$

$$c_1 f_1(x_2) + c_2 f_2(x_2) + \cdots + c_n f_n(x_2) = y_2$$

 \vdots

$$c_1 f_1(x_m) + c_2 f_2(x_m) + \cdots + c_n f_n(x_m) = y_m$$

Regresión lineal

$$y = c_1 f_1(x) + c_2 f_2(x) + \cdots + c_n f_n(x)$$

x	y
x_1	y_1
x_2	y_2
\vdots	\vdots
x_m	y_m



$$c_1 f_1(x_1) + c_2 f_2(x_1) + \cdots + c_n f_n(x_1) = y_1$$

$$c_1 f_1(x_2) + c_2 f_2(x_2) + \cdots + c_n f_n(x_2) = y_2$$

 \vdots

$$c_1 f_1(x_m) + c_2 f_2(x_m) + \cdots + c_n f_n(x_m) = y_m$$

$$\begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_m) & f_2(x_m) & \cdots & f_n(x_m) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

Regresión lineal

$$y = c_1 f_1(x) + c_2 f_2(x) + \cdots + c_n f_n(x)$$

x	y
x_1	y_1
x_2	y_2
\vdots	\vdots
x_m	y_m



$$c_1 f_1(x_1) + c_2 f_2(x_1) + \cdots + c_n f_n(x_1) = y_1$$

$$c_1 f_1(x_2) + c_2 f_2(x_2) + \cdots + c_n f_n(x_2) = y_2$$

 \vdots

$$c_1 f_1(x_m) + c_2 f_2(x_m) + \cdots + c_n f_n(x_m) = y_m$$

$$\begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_m) & f_2(x_m) & \cdots & f_n(x_m) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\mathbf{A}\mathbf{c} = \mathbf{y}$$

Regresión lineal

$$y = c_1 f_1(x) + c_2 f_2(x) + \cdots + c_n f_n(x)$$

x	y
x_1	y_1
x_2	y_2
\vdots	\vdots
x_m	y_m



$$c_1 f_1(x_1) + c_2 f_2(x_1) + \cdots + c_n f_n(x_1) = y_1$$

$$c_1 f_1(x_2) + c_2 f_2(x_2) + \cdots + c_n f_n(x_2) = y_2$$

 \vdots

$$c_1 f_1(x_m) + c_2 f_2(x_m) + \cdots + c_n f_n(x_m) = y_m$$

$$\begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_m) & f_2(x_m) & \cdots & f_n(x_m) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\mathbf{A}\mathbf{c} = \mathbf{y} \quad \Rightarrow \quad \mathbf{c} = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{y}$$

Regresión lineal

$$y = c_1 f_1(x) + c_2 f_2(x) + \cdots + c_n f_n(x)$$

\mathbf{x}	\mathbf{y}
x_1	y_1
x_2	y_2
\vdots	\vdots
x_m	y_m



$$c_1 f_1(x_1) + c_2 f_2(x_1) + \cdots + c_n f_n(x_1) = y_1$$

$$c_1 f_1(x_2) + c_2 f_2(x_2) + \cdots + c_n f_n(x_2) = y_2$$

 \vdots

$$c_1 f_1(x_m) + c_2 f_2(x_m) + \cdots + c_n f_n(x_m) = y_m$$

$$\begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_n(x_1) \\ f_1(x_2) & f_2(x_2) & \cdots & f_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_m) & f_2(x_m) & \cdots & f_n(x_m) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$

$$\mathbf{A}\mathbf{c} = \mathbf{y} \quad \Rightarrow \quad \mathbf{c} = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{y} \quad \Rightarrow \quad \mathbf{c} = \text{linsolve}(\mathbf{A}, \mathbf{y})$$

Codificando la regresión lineal

⚠ MATLAB

```
data = readtable('chiapas_population.csv');
x = data.Year;
y = data.Population;
A = [x.^2,x.^1,x.^0];
c = linsolve(A,y);
polinomio = @(x) c(1)*x.^2 + c(2)*x + c(3);
polinomio(2025)           % ans = 6.1701

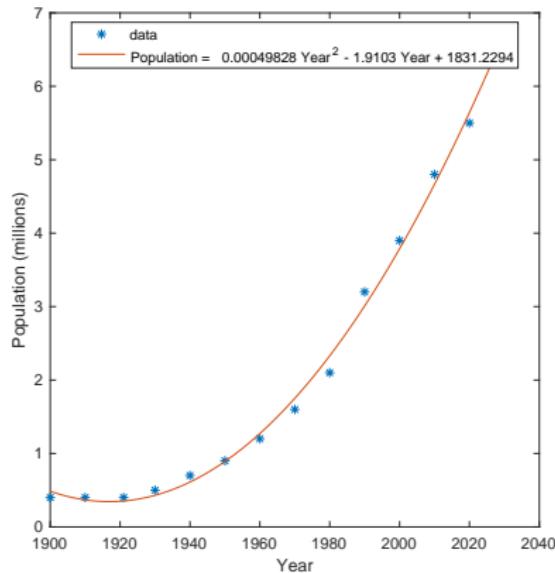
x_test = (1900:2030)';
plot(x,y,'*',x_test,polinomio(x_test))
xlabel('Year')
ylabel('Population (millions)')
legend("data","Population =" + ...
    poly2str(c,'Year'))
yhat = polinomio(x);
r = y - yhat;             % residuos
RMSE = sqrt(mean(r.^2))   % error típico
```

Codificando la regresión lineal

MATLAB

```
data = readtable('chiapas_population.csv');
x = data.Year;
y = data.Population;
A = [x.^2,x.^1,x.^0];
c = linsolve(A,y);
polinomio = @(x) c(1)*x.^2 + c(2)*x + c(3);
polinomio(2025) % ans = 6.1701

x_test = (1900:2030)';
plot(x,y,'*',x_test,polinomio(x_test))
xlabel('Year')
ylabel('Population (millions)')
legend("data","Population =" + ...
    poly2str(c,'Year'))
yhat = polinomio(x);
r = y - yhat; % residuos
RMSE = sqrt(mean(r.^2)) % error típico
```



Una clase para regresión polinomial

polynom.m

```
classdef polynom < handle
    properties
        coeff
    end
    methods
        function obj = fit(obj,x,y,n)
            A = zeros(numel(y),n+1);
            for k = 0:n
                A(:,k+1) = x.^k;
            end
            obj.coeff = linsolve(A,y);
        end
        function y = predict(obj,x)
            y = zeros(size(x));
            n = numel(obj.coeff) - 1;
            for k = 0:n
                y = y + obj.coeff(k+1) * x.^k;
            end
        end
    end
end
```

MATLAB

```
x = data.Year;
y = data.Population;

model = polynom;
model.fit(x,y,2)

model.predict(2025)
xx = (1900:2030)';
yy = model.predict(xx);
plot(x,y,'*',xx,yy)
```

Regresión con las funciones del toolbox

MATLAB

```
% Regresión lineal
model = fitlm(x,y,'quadratic');
model.predict(2025)                                % ans = 6.1701

% Regresión lineal generalizada
model = fitglm(x,y,'quadratic');
model.predict(2025)                                % ans = 6.1701

% Especificando la forma del modelo
modelspec = 'Population ~ 1 + Year + Year^2';
model = fitglm(data,modelspec)
model.predict(2025)                                % ans = 6.1701
```

Regresión con procesos gaussianos

Es una regresión **no paramétrica** que permite hacer **predicciones con una estimación de la incertidumbre** asociada a esas predicciones.

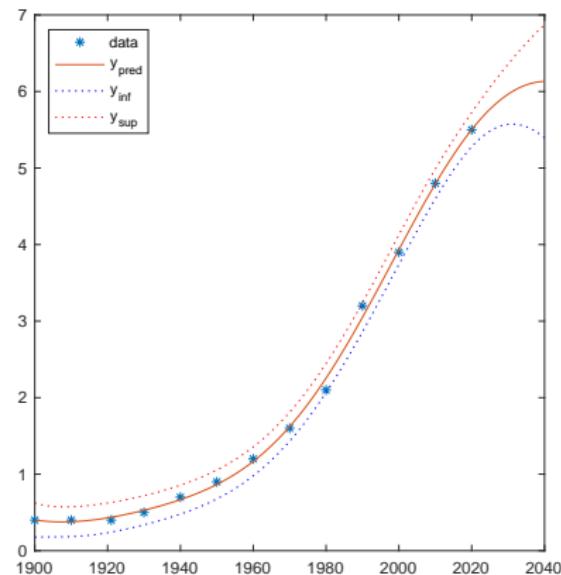
MATLAB

```
data = readtable('chiapas_population.csv');
x = data.Year;
y = data.Population;
model = fitrgp(x,y);

disp("Población estimada para 2025:")
disp(model.predict(2025))
[m,s] = model.predict(2025);
disp("Intervalo de confianza:")
disp([m-1.96*s, m+1.96*s])

x_test = (1900:2040)';
[y_pred,s] = model.predict(x_test);
y_inf = y_pred - 1.96*s;
y_sup = y_pred + 1.96*s;

plot(x,y,'*',x_test,y_pred, ...
      x_test,y_inf,'b:',x_test,y_sup,'r:')
legend('data','y_{pred}',...
      'y_{inf}', 'y_{sup}',Location='best')
```



Regresión multivariada con procesos gaussianos

Considere el dataset **patients.xls**. Interpolando el peso a partir de la edad, la estatura y el género, ¿cuál sería el peso de un hombre de 51 años con una estatura de 180 cm?

◆ MATLAB

```
cm2in = @(cm) cm/2.54;
lb2kg = @(lb) lb*0.454;

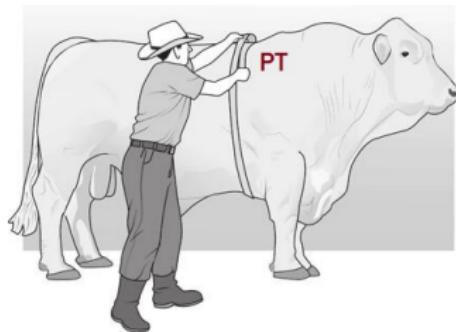
t = readtable('patients.xls');
Manhood = double(t.Gender=="Male"); % masculinidad
model = fitrgp([t.Age,t.Height,Manhood],t.Weight);

lb2kg(model.predict([51,cm2in(180),1])) % ans = 81.7032

[m,s] = model.predict([51,cm2in(180),1])
disp("Peso esperado (kg): " + lb2kg(m))
disp("Intervalo de confianza al 95%:")
disp(lb2kg([m-1.96*s,m+1.96*s]))
```

Actividad 4: Regresión

Con el dataset **bovine.csv** generar un modelo de regresión para estimar el **peso vivo (PV)** en bovinos a partir de las mediciones de **perímetro torácico (PT)**. Elaborar una tabla para PV desde 1.20 m hasta 1.80 m con incrementos de un centímetro en PT.



PT = model.predict(PT)



Clustering



Busca patrones ocultos en conjuntos de datos cuyas respuestas no están etiquetadas y permite explorar los datos cuando no se sabe qué información contienen.

Clustering

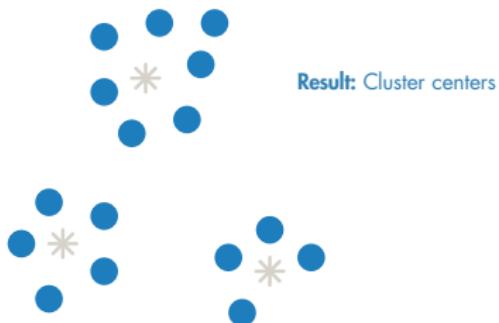
k-Means

How it Works

Partitions data into k number of mutually exclusive clusters.
How well a point fits into a cluster is determined by the distance from that point to the cluster's center.

Best Used...

- When the number of clusters is known
- For fast clustering of large data sets



Result: Cluster centers

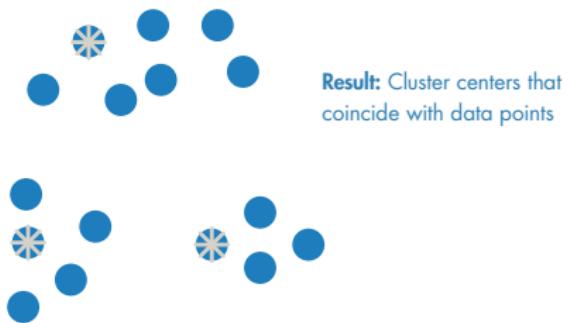
k-Medoids

How It Works

Similar to k-means, but with the requirement that the cluster centers coincide with points in the data.

Best Used...

- When the number of clusters is known
- For fast clustering of categorical data
- To scale to large data sets



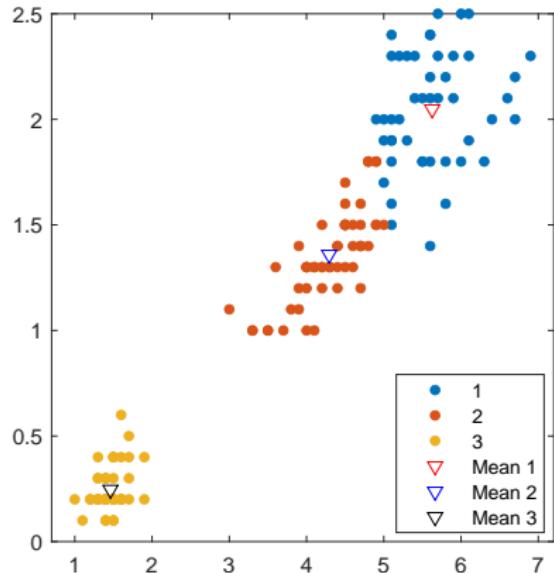
Result: Cluster centers that coincide with data points

Clustering: k -Means

MATLAB

```
data = load('fisheriris.mat');
x = data.meas(:,3:4);
% PetalLength, PetalWidth

[idx,m] = kmeans(x, 3);
gscatter(x(:,1), x(:,2), idx)
hold on
plot(m(1,1), m(1,2), 'rv', ...
    DisplayName='Mean 1')
plot(m(2,1), m(2,2), 'bv', ...
    DisplayName='Mean 2')
plot(m(3,1), m(3,2), 'kv', ...
    DisplayName='Mean 3')
hold off
```

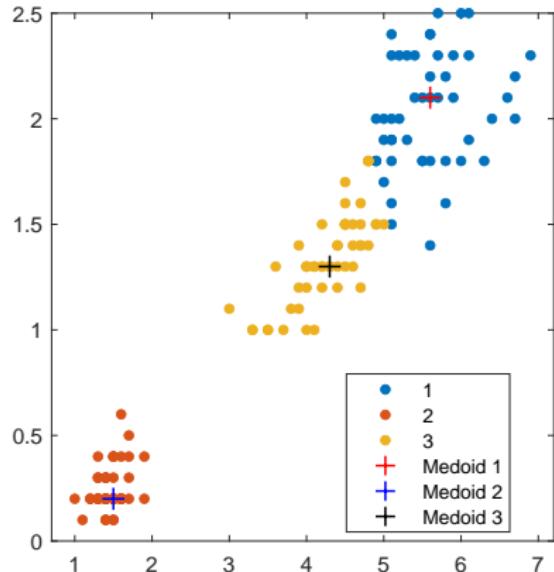


Clustering: k -Medoids

MATLAB

```
data = load('fisheriris.mat');
x = data.meas(:,3:4);
% PetalLength, PetalWidth

[idx,m] = kmedoids(x, 3);
gscatter(x(:,1), x(:,2), idx)
hold on
plot(m(1,1), m(1,2), 'r+', ...
    DisplayName='Medoid 1', ...
    MarkerSize=10, LineWidth=1)
plot(m(2,1), m(2,2), 'b+', ...
    DisplayName='Medoid 2', ...
    MarkerSize=10, LineWidth=1)
plot(m(3,1), m(3,2), 'k+', ...
    DisplayName='Medoid 3', ...
    MarkerSize=10, LineWidth=1)
hold off
```

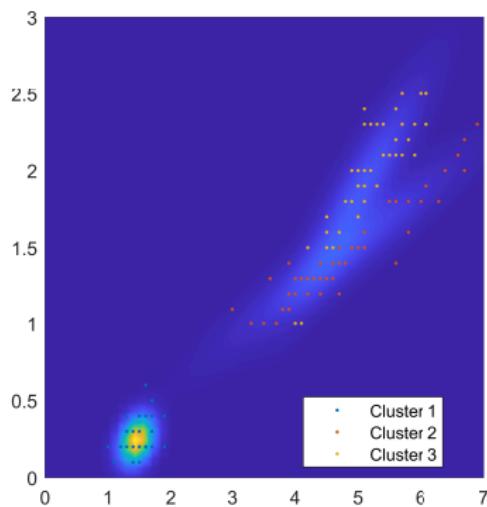


Modelo de mezcla gaussiana

Un modelo de **mezcla gaussiana** (GM) es un enfoque probabilístico que representa la distribución de los datos como una combinación de varias distribuciones gaussianas, cada una con su propia media y covarianza.

MATLAB

```
data = load('fisheriris.mat');
X = data.meas(:,3:4);
gm = fitgmdist(X, 3);
i = gm.cluster(X);
clusterName = "Cluster " + i;
[X1,X2] = meshgrid(linspace(0,7,50), ...
    linspace(0,3,50));
P = gm.pdf([X1(:),X2(:)]);
P = reshape(P, size(X1));
pcolor(X1, X2, P)
hold on
gscatter(X(:,1), X(:,2), clusterName)
shading interp
hold off
l = legend; l.Location = 'best';
axis([0,7,0,3])
```



Clasificación

DUCKS



NOT DUCKS

Clasificación

DUCKS

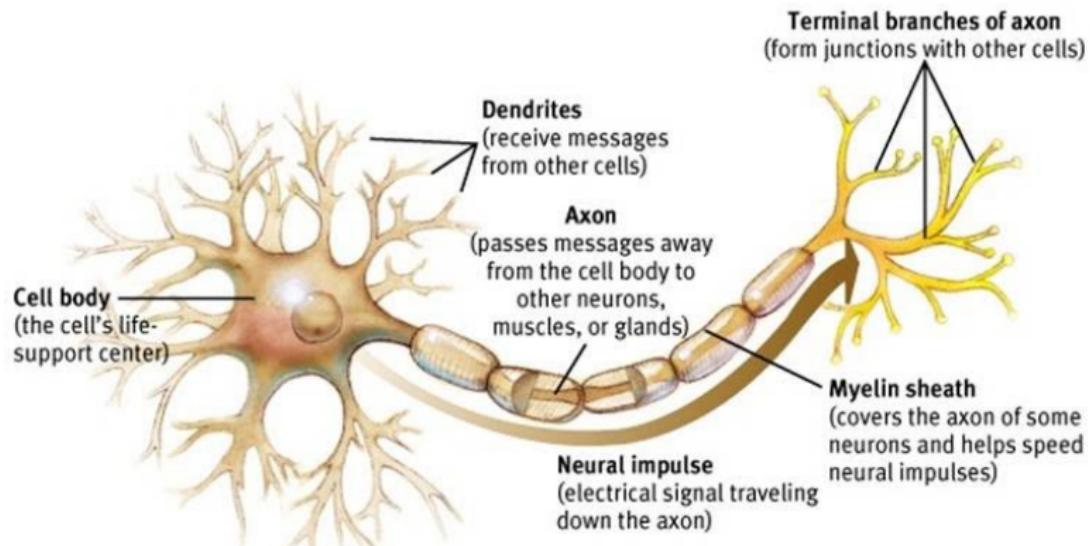


NOT DUCKS

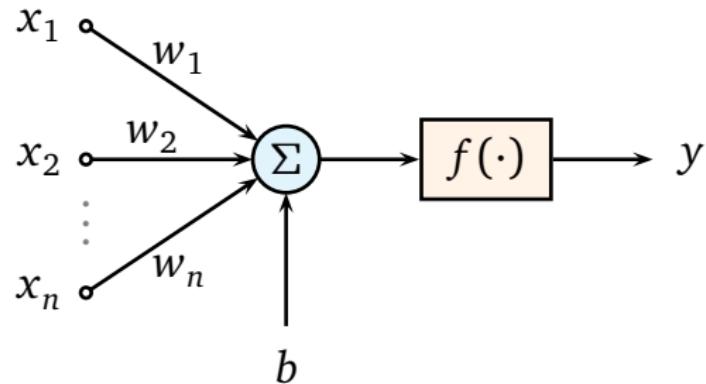
Si grazna como un pato, camina como un pato
y se comporta como un pato, entonces,
¡seguramente es un pato!

Enfoque conexionista: redes neuronales

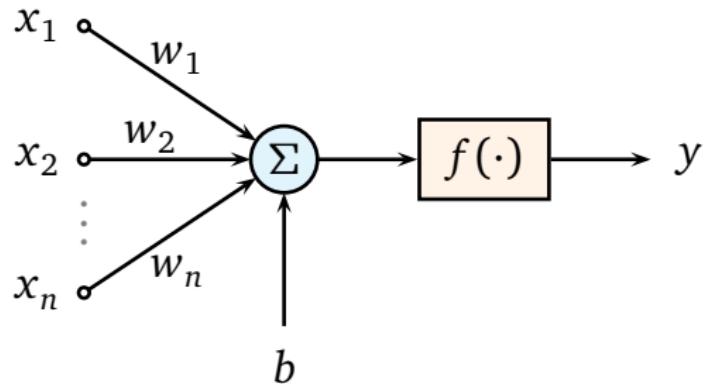
Neurona biológica



La neurona artificial



La neurona artificial



$$y = f(w_1 x_1 + w_2 x_2 + \dots + w_n x_n + b) = f(\mathbf{w}^\top \mathbf{x} + b)$$

\mathbf{w} : pesos

b : sesgo

f : función de activación (de transferencia)

Funciones de activación comunes

- **hardlim** (escalón unitario):

- ▶ Definición: Devuelve 1 si la entrada es mayor o igual a 0, de lo contrario, devuelve 0.
- ▶ Fórmula: $f(x) = \text{heaviside}(x)$
- ▶ Uso: Común en redes neuronales binarias y sistemas de clasificación.

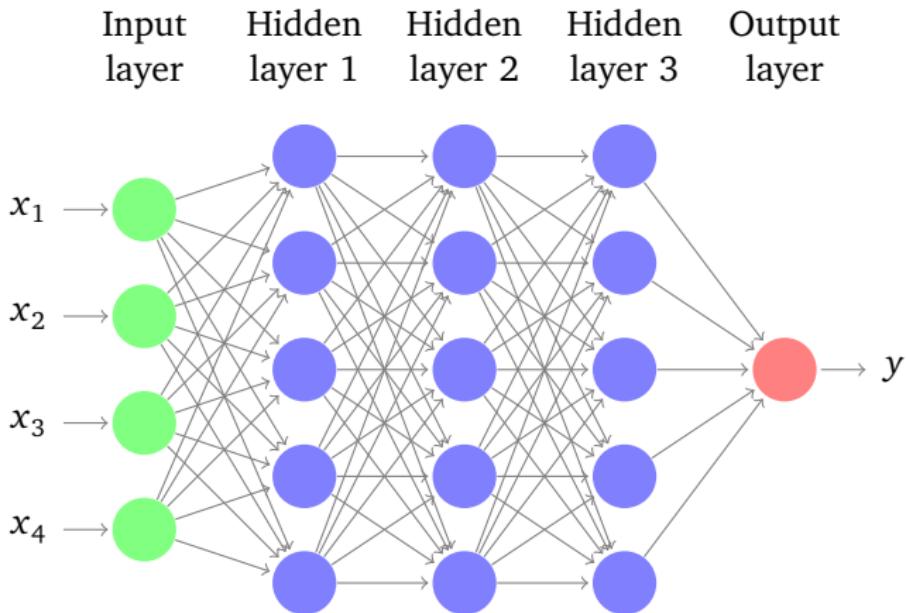
- **purelin** (lineal pura):

- ▶ Definición: Función lineal que devuelve el mismo valor de la entrada.
- ▶ Fórmula: $f(x) = x$
- ▶ Uso: Común en capas de salida para problemas de regresión.

- **tansig** (sigmoide tangente):

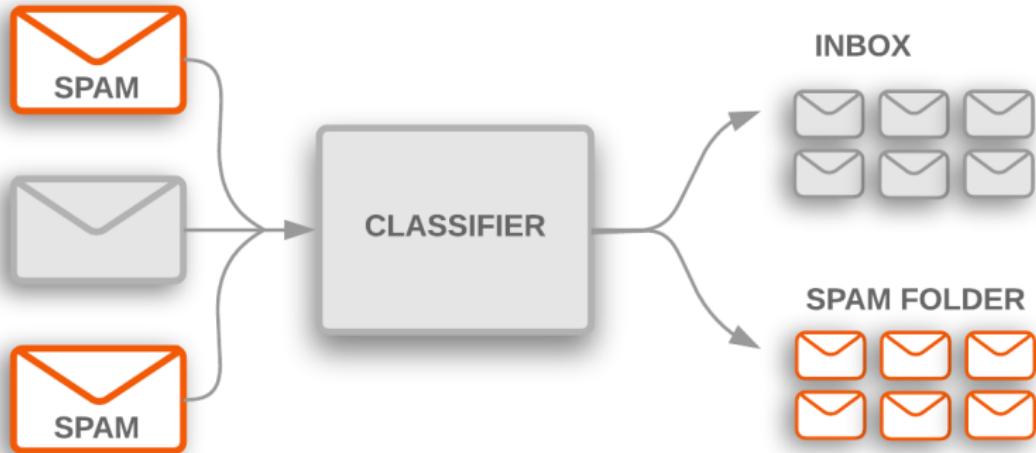
- ▶ Definición: Función sigmoide que devuelve valores en el rango de -1 a 1.
- ▶ Fórmula: $f(x) = \tanh(x)$
- ▶ Uso: Popular en capas ocultas para modelos no lineales.

Redes neuronales artificiales



Red neuronal Feedforward

Clasificación binaria

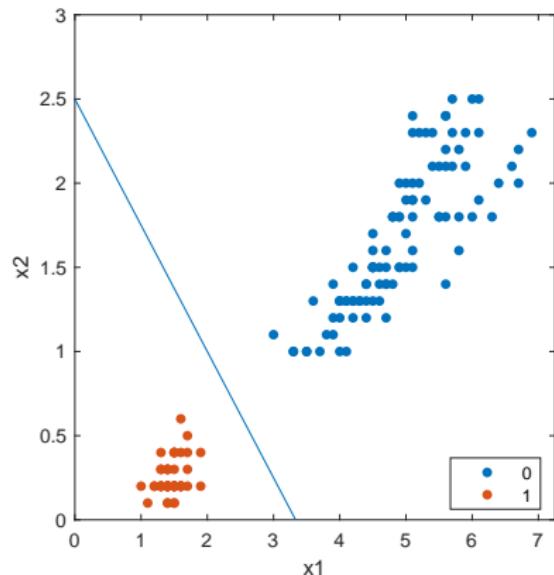


El perceptrón

Un **perceptrón** es el modelo básico de una neurona artificial que toma una combinación lineal de entradas, aplica una función de activación (como el escalón unitario) y produce una **salida binaria**, utilizada principalmente para tareas de **clasificación lineal**.

MATLAB

```
t = readtable('fisheriris');
x1 = t.PetalLength;
x2 = t.PetalWidth;
X = [x1, x2]';
y = double(t.Species == "setosa')";
net = perceptron;
net = train(net,X,y);
yhat = net(X);
C = confusionmat(y, yhat)
w = net.IW{1}
b = net.b{1}
recta = @(x1,x2) w(1)*x1 + w(2)*x2 + b;
fimplicit(recta, [0,7,0,3]); hold on
gscatter(x1, x2, y); hold off
```



Support Vector Machine (SVM)

La **máquina de vectores de soporte** (SVM) es un algoritmo utilizado principalmente para clasificación y, en menor medida, para regresión. Su objetivo es encontrar la recta, plano o hiperplano que mejor separa las clases.

► MATLAB

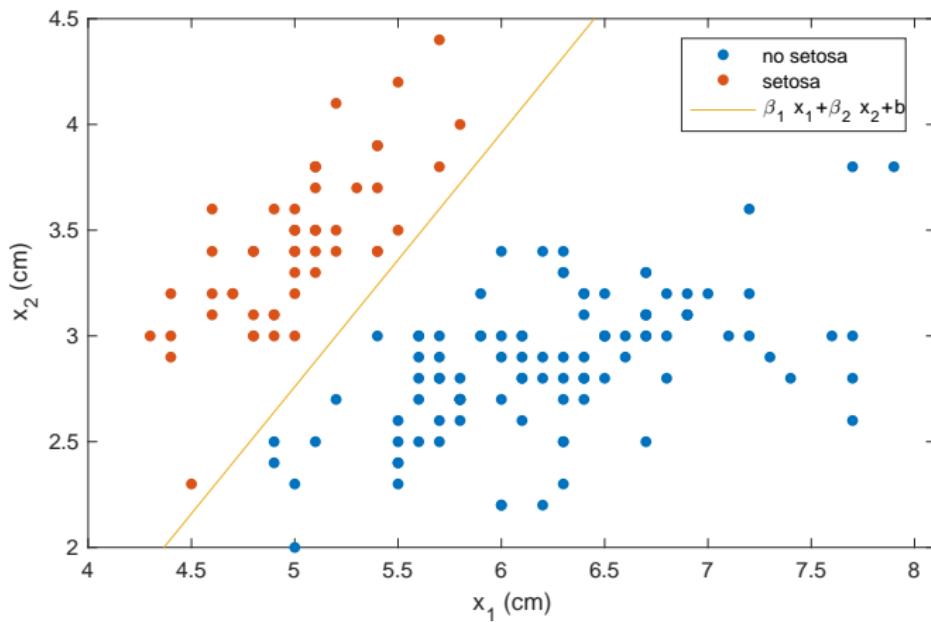
```
load fisheriris
X = meas(:,1:2);
y = string(species) == "setosa";

model = fitcsvm(X, y, "BoxConstraint", inf);
w1 = model.Beta(1);
w2 = model.Beta(2);
b = model.Bias;
boundary = @(x1,x2) w1*x1 + w2*x2 + b;

gscatter(X(:,1), X(:,2), y); hold on
fimplicit(boundary, [gca().XLim, gca().YLim]); hold off
xlabel('x_1 (cm)'); ylabel('x_2 (cm)')
l = legend; l.String{1} = "no setosa"; l.String{2} = "setosa";
```

Support Vector Machine (SVM)

SVM no solo busca un hiperplano que separa las clases, sino que maximiza su distancia (margen) a las dos clases más cercanas al hiperplano. Este margen máximo garantiza que el modelo no solo se ajuste a los datos de entrenamiento, sino que también sea robusto y tenga una mejor capacidad de generalización para datos nuevos.



Datos no linealmente separables

MATLAB

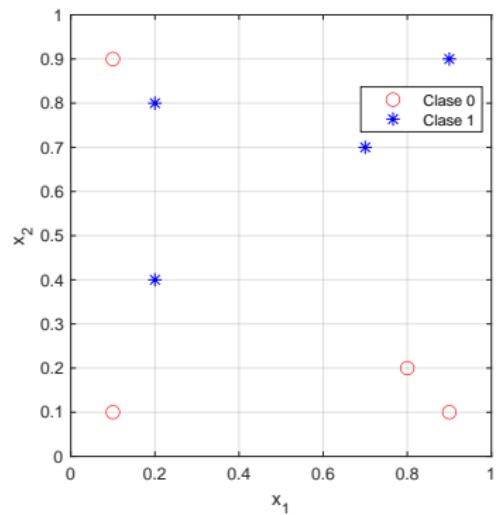
```
% Datos de entrada (inputs)
x1 = [0.1, 0.9, 0.1, 0.9, 0.2, 0.2, 0.8, 0.7];
x2 = [0.1, 0.1, 0.9, 0.9, 0.4, 0.8, 0.2, 0.7];
X = [x1; x2];

% Etiquetas de clase (targets)
y = [0, 0, 0, 1, 1, 1, 0, 1];

gscatter(X(1,:), X(2,:), y, 'rb', 'o*', 8);
legend({'Clase 0', 'Clase 1'})
axis([0, 1, 0, 1])
grid on

net = perceptron
net = train(net, X, y)

yhat = net(X)
C = confusionmat(y, yhat)
% C =
%     3     1
%     1     3
```



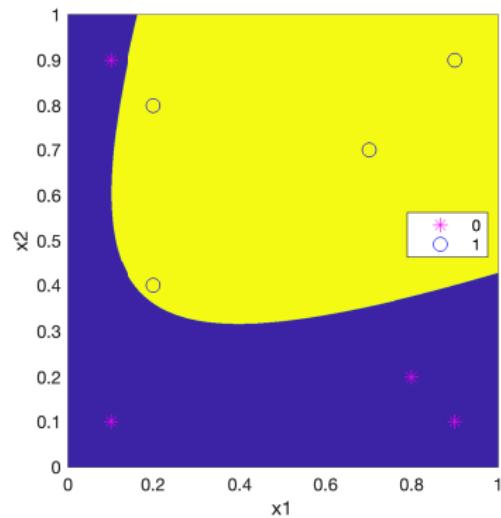
Separando datos no linealmente separables

MATLAB

```
x1 = [0.1, 0.9, 0.1, 0.9, 0.2, 0.2, 0.8, 0.7];
x2 = [0.1, 0.1, 0.9, 0.9, 0.4, 0.8, 0.2, 0.7];
x3 = x1.*x2; x4 = x1.^2; x5 = x2.^2;
X = [x1; x2; x3; x4; x5]; %  $\mathbb{R}^2 \rightarrow \mathbb{R}^5$ 
y = [0, 0, 0, 1, 1, 1, 0, 1];
net = perceptron
net = train(net, X, y)
yhat = net(X)
C = confusionmat(y, yhat)

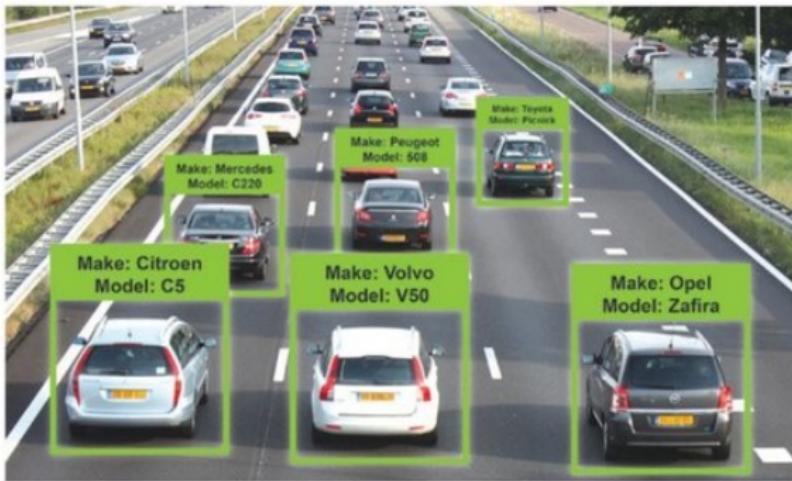
[X1,X2] = meshgrid(0:0.001:1, 0:0.001:1);
x1t = X1(:)'; x2t = X2(:)';
x3t = x1t.*x2t; x4t = x1t.^2; x5t = x2t.^2;
Xt = [x1t; x2t; x3t; x4t; x5t];

Yp = net(Xt);
pcolor(X1, X2, reshape(Yp,size(X1)))
shading interp; hold on
gscatter(x1, x2, y, 'mb', '*o', 8);
hold off
```



La transformación de las muestras desde el espacio original hacia un espacio de mayor dimensionalidad es llamada “kernel trick”.

Clasificación multiclas



Ejemplo: Clasificación de flores



Versicolor



Setosa



Virginica

Ejemplo: Clasificación de flores



Versicolor



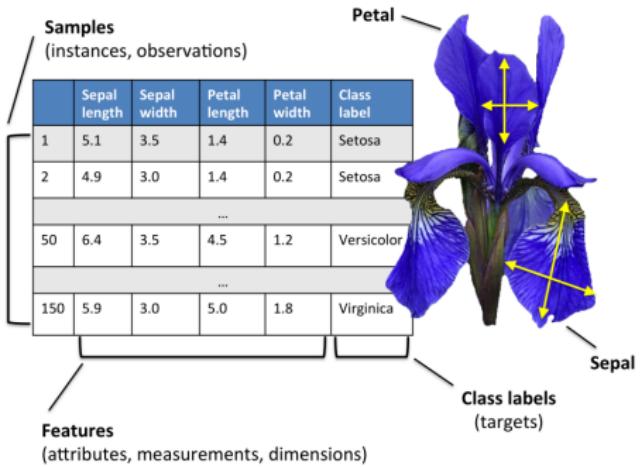
Setosa



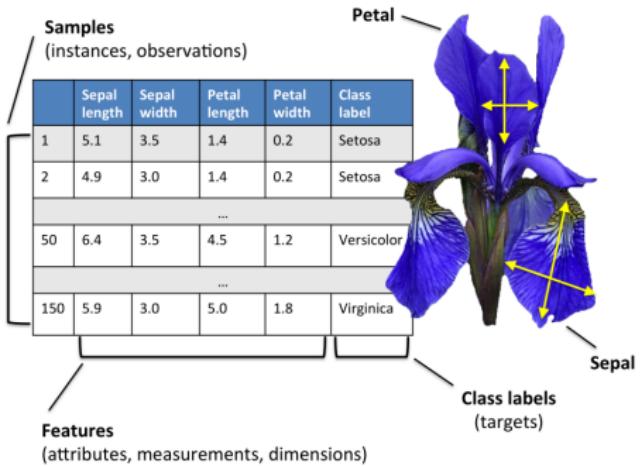
Virginica



Ejemplo: Clasificación de flores

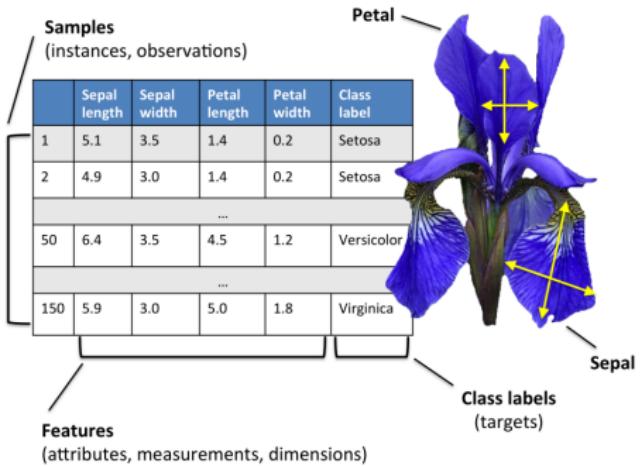


Ejemplo: Clasificación de flores



$$[x_1, x_2, x_3, x_4] \rightarrow \text{Clasificador} \rightarrow y$$

Ejemplo: Clasificación de flores



$$[x_1, x_2, x_3, x_4] \longrightarrow$$

Clasificador

$$\longrightarrow y$$

Atributos

$$\mathbb{R}^4$$

Clase

$$\mathbb{Z}$$

Error Correcting Output Codes (ECOC)

ECOC es una técnica utilizada para abordar problemas de clasificación multiclase mediante la descomposición de estos en múltiples problemas de clasificación binaria.

Error Correcting Output Codes (ECOC)

ECOC es una técnica utilizada para abordar problemas de clasificación multiclase mediante la descomposición de estos en múltiples problemas de clasificación binaria.

Supongamos que tenemos un problema con tres clases: A, B y C. Podríamos usar una matriz de codificación como esta:

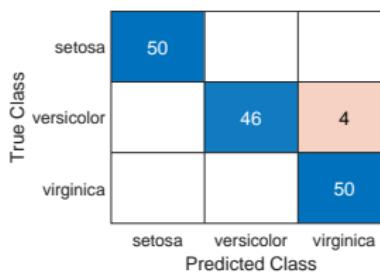
	Clasificador 1	Clasificador 2	Clasificador 3
A	1	1	0
B	-1	0	1
C	0	-1	-1

- **Clasificador 1:** distingue entre A (1) y B (-1), ignorando C (0).
- **Clasificador 2:** distingue entre A (1) y C (-1), ignorando B (0).
- **Clasificador 3:** distingue entre B (1) y C (-1), ignorando A (0).

Clasificación multiclase con ECOC

MATLAB

```
load fisheriris
X = meas;
y = categorical(species);
t = templateSVM("Standardize", true);
model = fitcecoc(X, y, "Learners", t)
yhat = model.predict(X);
C = confusionmat(y, yhat)
confusionchart(y, yhat)
model.resubLoss % 0.0267
X_test = X; y_test = y;
model.loss(X_test, y_test) % 0.0267
```



One *versus* one, One *versus* all

MATLAB

```
model = fitcecoc(X, y, "Coding", "onevsone");
model.CodingMatrix

%      1      1      0
%     -1      0      1
%      0     -1     -1

model = fitcecoc(X, y, "Coding", "onevsall");
model.CodingMatrix

%      1     -1     -1
%     -1      1     -1
%     -1     -1      1
```

Clasificador k -NN

El **k -NN** (k -Nearest Neighbors) es un algoritmo de clasificación que asigna una etiqueta a un punto nuevo basándose en las k muestras más cercanas en el conjunto de datos de entrenamiento. Se clasifica según la mayoría de las etiquetas de estos vecinos cercanos.

► MATLAB

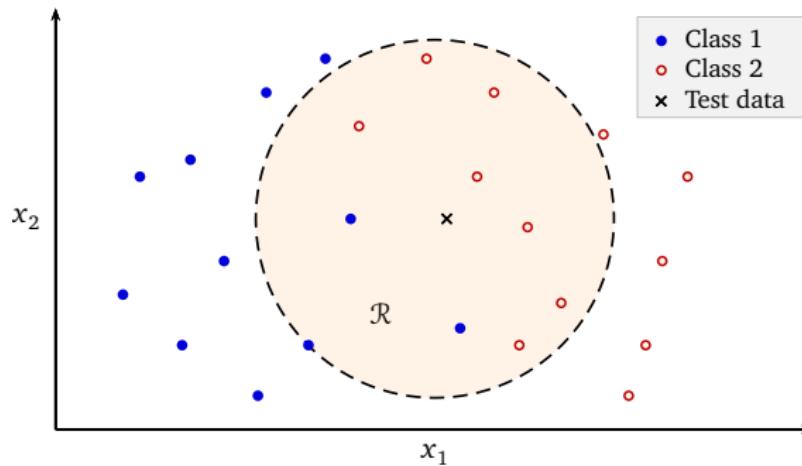
```
t = readtable('patients.xls');
x1 = t.Weight; % Entrada (feature) 1
x2 = t.Height; % Entrada (feature) 2
y = t.Gender; % Salida, respuesta
X = [x1, x2];

i = randperm(100); % Índices aleatorios para partir el dataset
i_trn = i(1:70); X_trn = X(i_trn,:); y_trn = y(i_trn); % para training
i_tst = i(71:end); X_tst = X(i_tst,:); y_tst = y(i_tst); % para testing

model = fitcknn(X_trn, y_trn, NumNeighbors=5); % Entrenar el modelo
y_hat = model.predict(X_tst); % Evaluar el modelo
disp(table(y_tst, y_hat))
C = confusionmat(y_tst, y_hat) % matriz de confusión
%
%      16      0
%       0     14
```

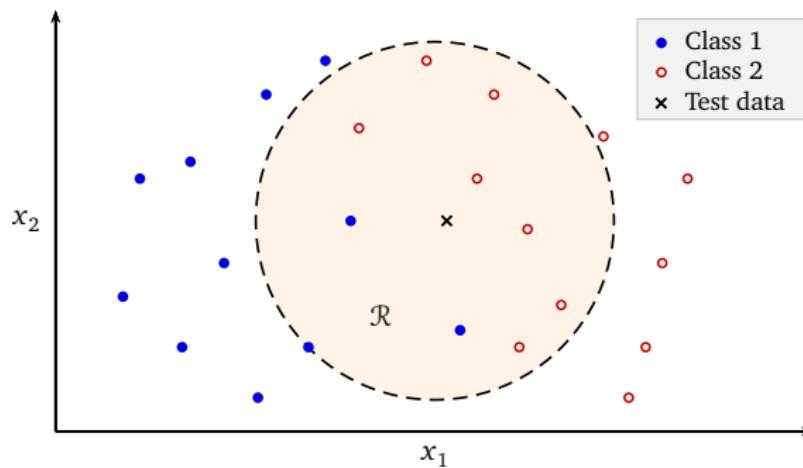
Clasificación por k -NN

k -Nearest Neighbors: El dato de prueba se asigna a la clase con la más alta frecuencia (la moda) entre los k vecinos más cercanos de los datos de entrenamiento.



Clasificación por k -NN

k -Nearest Neighbors: El dato de prueba se asigna a la clase con la más alta frecuencia (la moda) entre los k vecinos más cercanos de los datos de entrenamiento.



$$d(\mathbf{x}, \mathbf{z}) = \sqrt{(x_1 - z_1)^2 + (x_2 - z_2)^2}$$

Métricas de distancia no euclidianas

$$\mathbf{x} = [x_1, x_2, \dots, x_n]$$

$$\mathbf{z} = [z_1, z_2, \dots, z_n]$$

Nombre	Definición
Minkowski	$d(\mathbf{x}, \mathbf{z}) = \left(\sum_{i=1}^n x_i - z_i ^p \right)^{1/p}$
Manhattan	$d(\mathbf{x}, \mathbf{z}) = \sum_{i=1}^n x_i - z_i $
Chebychev	$d(\mathbf{x}, \mathbf{z}) = \max_i x_i - z_i $
Coseno	$d(\mathbf{x}, \mathbf{z}) = 1 - \frac{\mathbf{x} \cdot \mathbf{z}}{\ \mathbf{x}\ \ \mathbf{z}\ }$

Clasificación por árbol de decisión

Un **árbol de decisión** es un modelo de clasificación que divide iterativamente los datos en subconjuntos basados en características, formando un árbol de decisiones donde cada nodo representa una característica, cada rama una decisión basada en esa característica, y cada hoja una etiqueta de clase.

► MATLAB

```
t = readtable('patients.xls');
x1 = t.Systolic; % Entrada (feature) 1
x2 = t.Diastolic; % Entrada (feature) 2
x3 = t.Age; % Entrada (feature) 3
y = t.Smoker; % Salida, respuesta
X = [x1, x2, x3];

i = randperm(100); % índices aleatorios para partir el dataset
i_trn = i(1:70); X_trn = X(i_trn,:); y_trn = y(i_trn); % para training
i_tst = i(71:end); X_tst = X(i_tst,:); y_tst = y(i_tst); % para testing

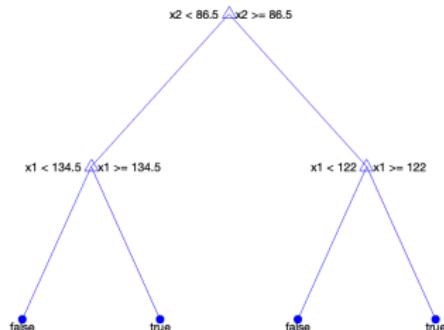
model = fitctree(X_trn, y_trn); % Entrenar el modelo
y_hat = model.predict(X_tst); % Evaluar el modelo
disp(table(y_tst, y_hat))
C = confusionmat(y_tst, y_hat) % matriz de confusión
%
%      15      0
%       3     12
```

Clasificación por árbol de decisión

MATLAB

(continuación)

```
accuracy = sum(diag(C))/sum(sum(C)) % 0.9000  
loss = 1 - accuracy % 0.1000  
view(model, "Mode", "graph")  
view(model, "Mode", "text")
```



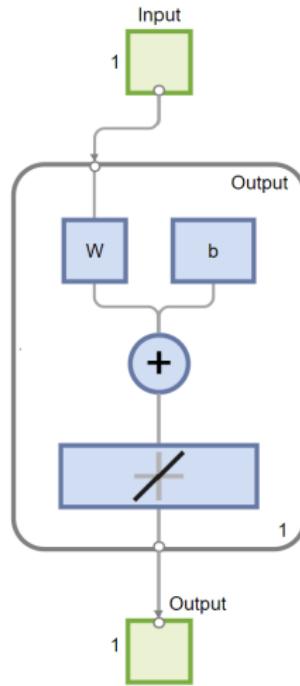
Decision tree for classification

```
1 if x2<86.5 then node 2 elseif x2>=86.5 then node 3 else false  
2 if x1<134.5 then node 4 elseif x1>=134.5 then node 5 else false  
3 if x1<122 then node 6 elseif x1>=122 then node 7 else true  
4 class = false  
5 class = true  
6 class = false  
7 class = true
```

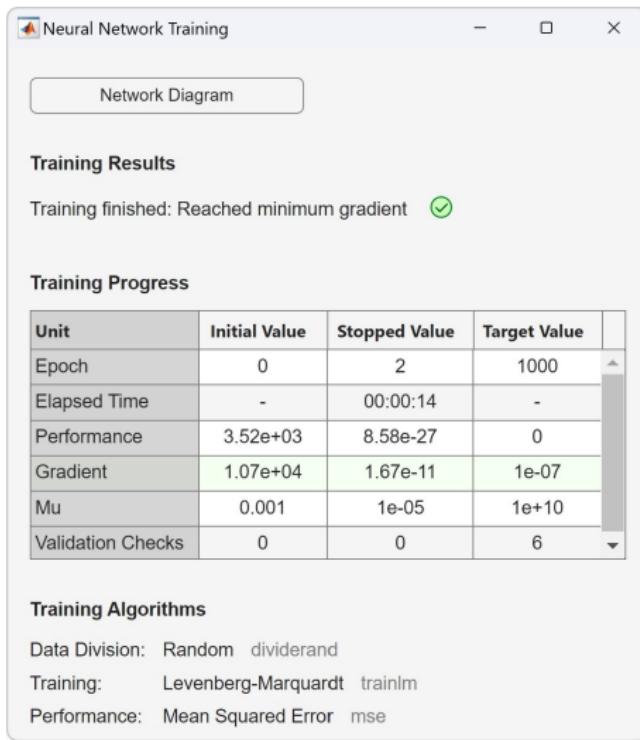
fitnet: red neuronal para regresión

MATLAB

```
c = [0,100]; % entradas conocidas
f = [32,212]; % salidas conocidas
net = fitnet([]); % sin capas ocultas
net.inputs{1}.processFcns = {};% sin preprocessing
net.outputs{1}.processFcns = {};% sin post-processing
net.divideMode = 'none';% no reservar muestras
net = train(net,c,f)
fhat = net(c) % 32.0000 212.0000
c_test = 10 % 10°C --> ?
f_pred = net(c_test) % 50.0000
net.layers{1}.transferFcn % 'purelin'
w = net.IW{1} % 1.8000
b = net.b{1} % 32.0000
%  $f = 1.8c + 32$ 
view(net)
genFunction(net, 'c2f.m')
```



Entrenamiento



patternnet: red neuronal para clasificación

MATLAB

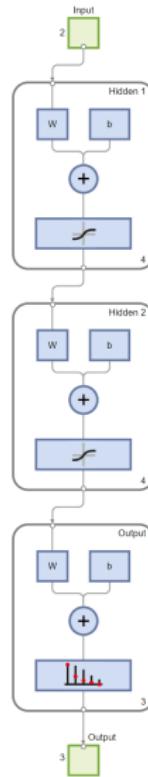
```
data = load('fisheriris.mat');
X = data.meas';
[~,~,idx] = unique(data.species);
y = full(ind2vec(idx'));

net = patternnet([4,4]);
net = train(net,X,y)

net.layers.transferFcn
    % {'tansig' }
    % {'tansig' }
    % {'softmax'}

net.IW{1}
net.b{1}
net.LW{2,1}
net.b{2}
net.LW{3,2}
net.b{3}

view(net)
```



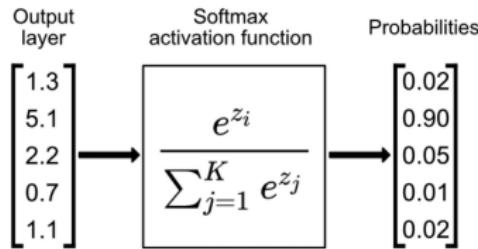
Softmax

Softmax es una función matemática usada comúnmente en la última capa de una red neuronal para problemas de clasificación multiclase. Su principal objetivo es convertir los valores de salida (también llamados **logits**) en probabilidades, lo que significa que el resultado será un vector de valores entre 0 y 1 que suman 1. Cada valor en este vector representa la probabilidad de que la entrada pertenezca a una clase en particular.

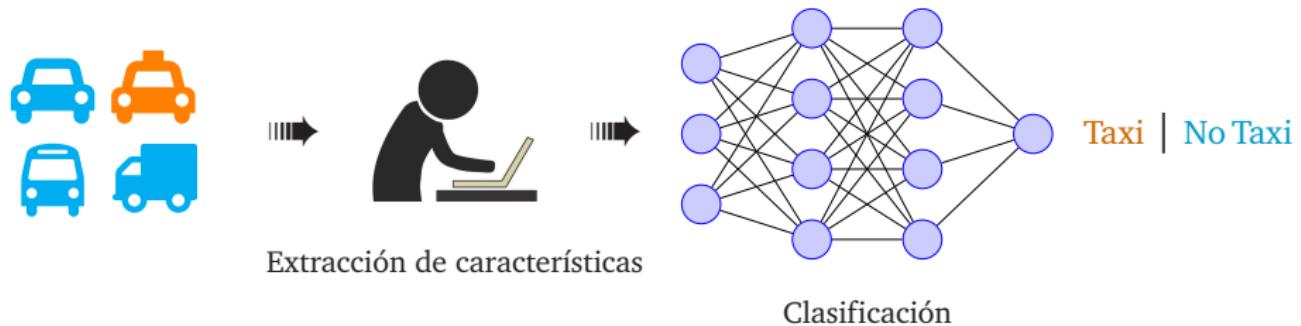
Funcionamiento de softmax

Dada una entrada $z = [z_1, z_2, \dots, z_K]$ que es un vector de logits de la capa anterior, la función softmax calcula las probabilidades p_i para cada clase i utilizando la fórmula:

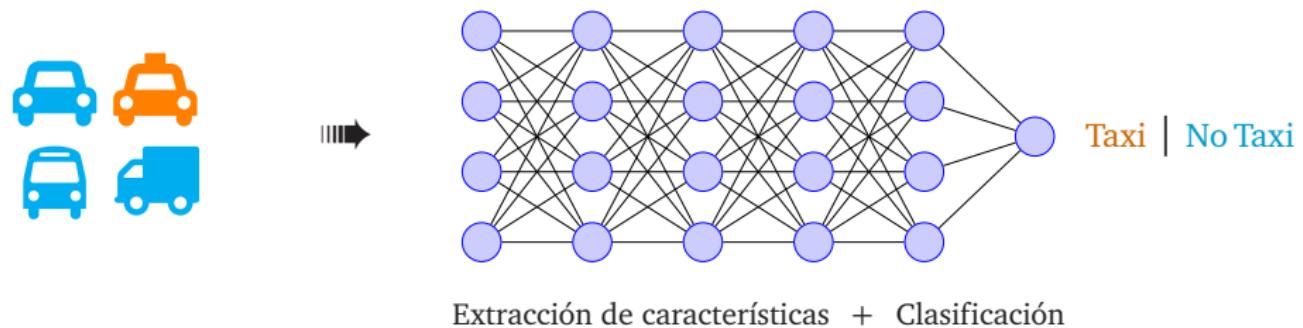
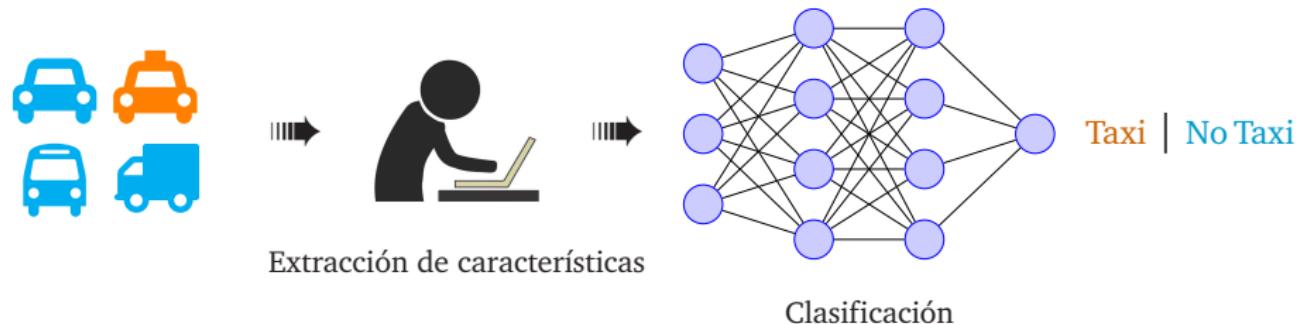
$$p_i = e^{z_i} / \sum_{j=1}^K e^{z_j}.$$



Machine learning *versus* Deep learning



Machine learning versus Deep learning



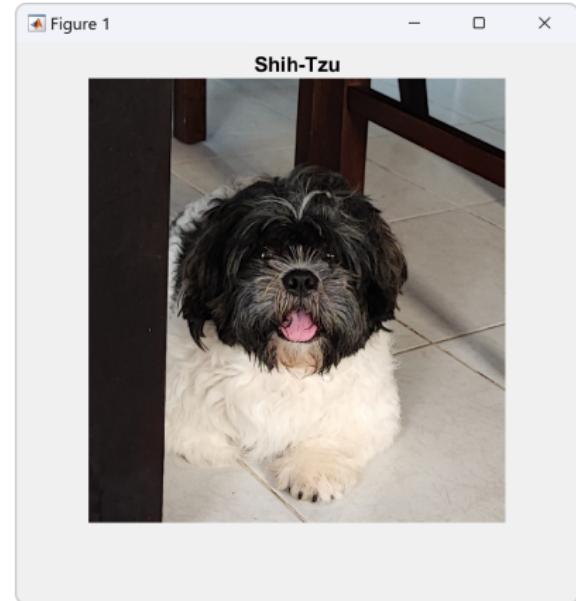
Red neuronal profunda preentrenada para imágenes

◆ MATLAB

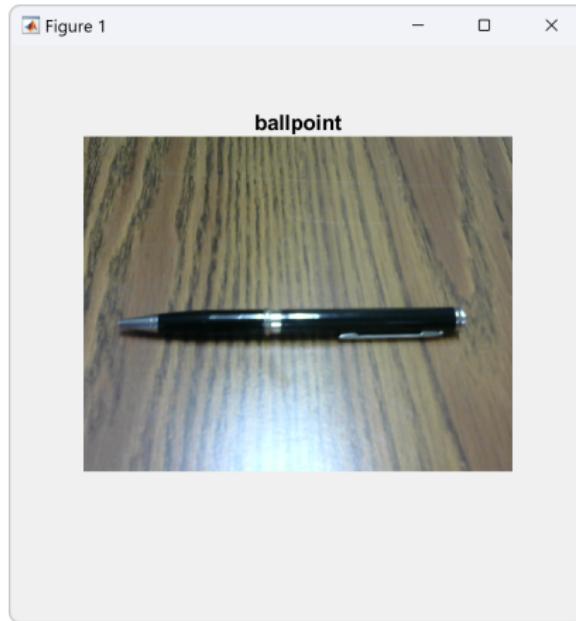
```
[net,classes] = imagePretrainedNetwork;
s = net.Layers(1).InputSize % [227,227,3]
img = imread("pedrito.jpg");
newImg = double(imresize(img,s(1:2)));
imshow(img)
scores = net.predict(newImg);
result = scores2label(scores,classes)
imshow(img)
title(result)

analyzeNetwork(net)

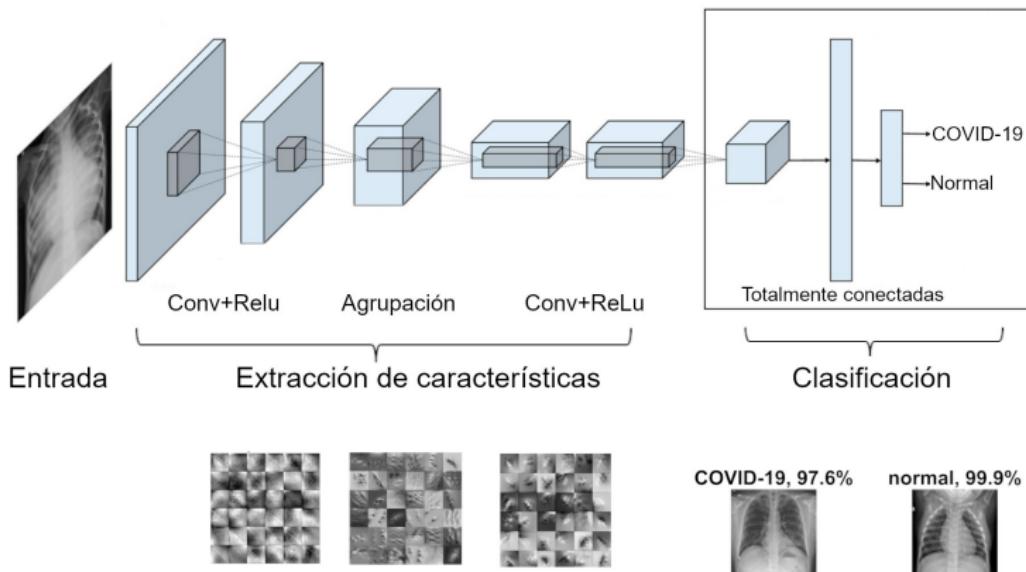
% Usando nuestras propias imágenes con
% MATLAB Support Package for USB Webcams
cam = webcam;
preview(cam)
img = snapshot(cam);
delete(cam)
```



Red neuronal profunda preentrenada para imágenes

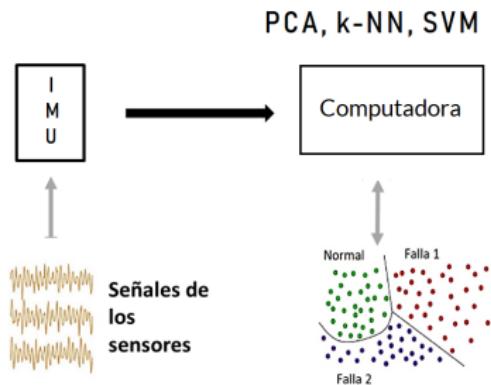


Ejemplo: Diagnóstico de COVID-19

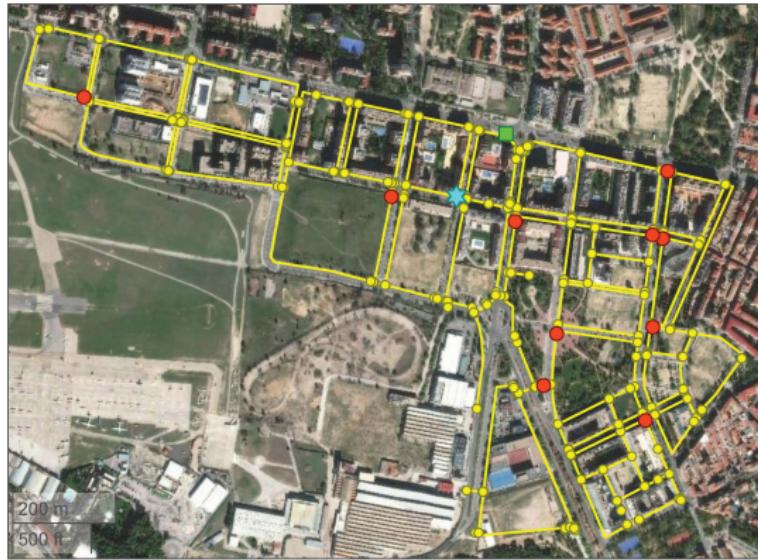


Cortés, E., & Sánchez, S. (2021). Deep Learning Transfer with AlexNet for chest X-ray COVID-19 recognition. *IEEE Latin America Transactions*, 19(6), 944–951.

Ejemplo: Diagnóstico de fallas en cuadricópteros



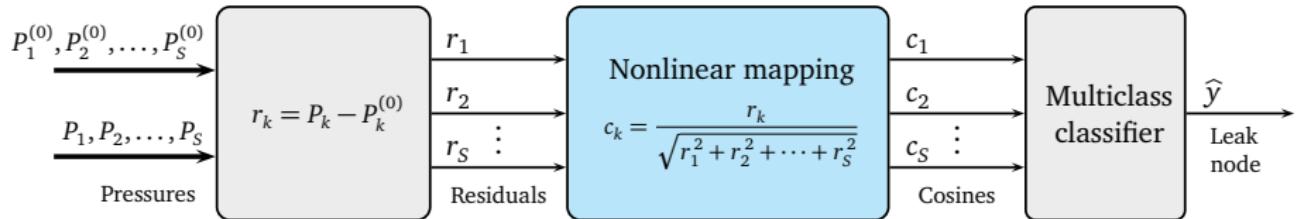
Ejemplo: Localización de fugas de agua



La estrella azul indica una fuga, los puntos rojos indican sensores de presión:

$$\mathbf{x} = [P_1, P_2, \dots, P_{10}]$$

Ejemplo: Localización de fugas de agua



Método: k -NN
Distancia: coseno
Número de vecinos: 4

“Lo que los humanos pueden hacer ahora, las máquinas lo podrán hacer mañana.”

— Geoffrey Hinton

A large, colorful word cloud centered around the words "thank you". The word "thank" is in red, "you" is in green, and "you" is in blue. Numerous other words in different languages are scattered around, such as "danke" in German, "merci" in French, "gracias" in Spanish, "mochchakkeram" in Korean, and many others. The background is white.