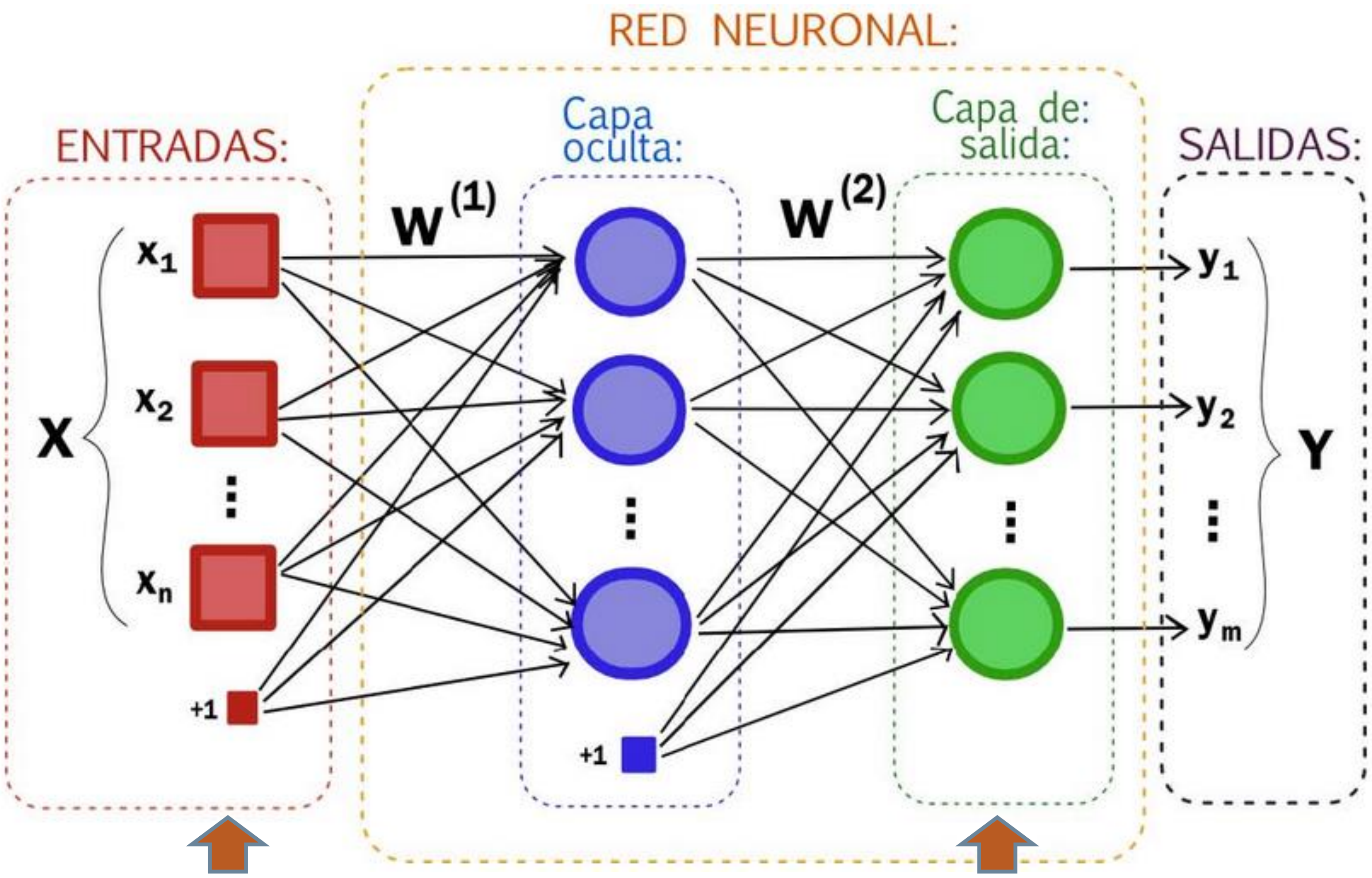
Resumen

Resolución de una tarea de clasificación

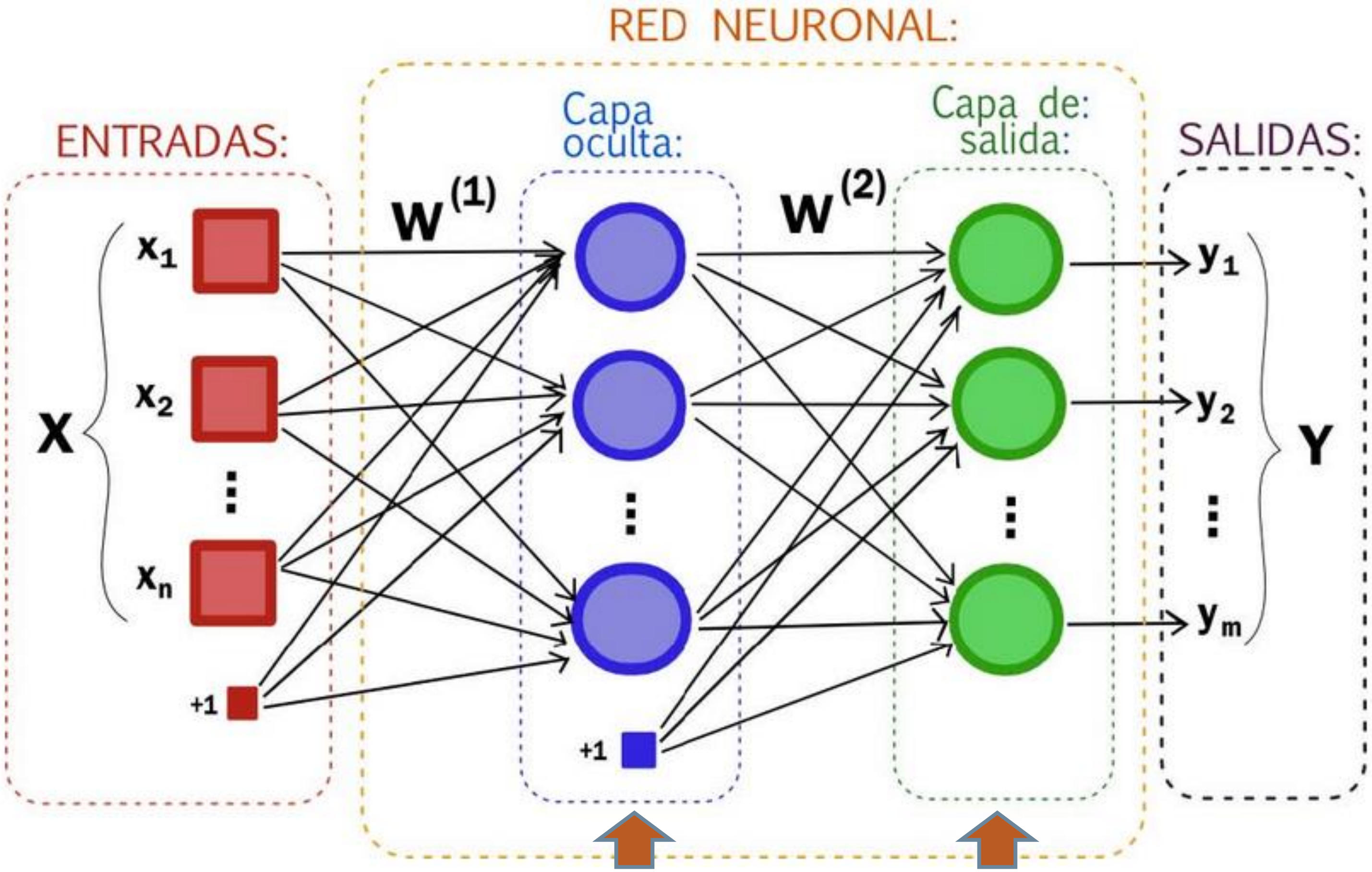
- Conjunto de datos etiquetados (aprendizaje supervisado)
- Definición de la arquitectura de la red
  - Número de capas y tamaño de cada una
  - Función de activación a usar en cada capa
- Entrenamiento
  - Función de costo
  - Técnica de optimización para reducir el error
- Evaluar el modelo

Arquitectura de la red



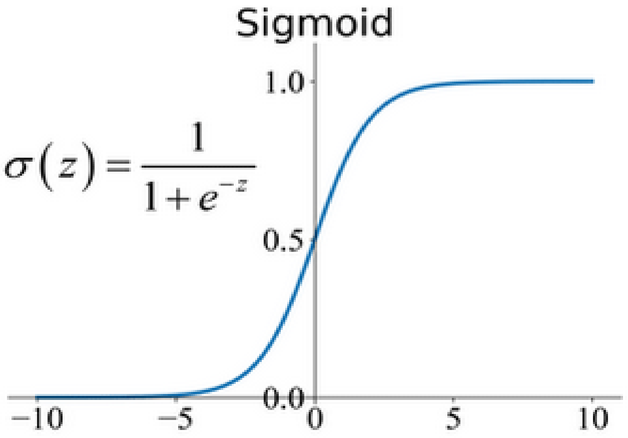
Las dimensiones de las capas de entrada y salida las define el problema

Arquitectura de la red

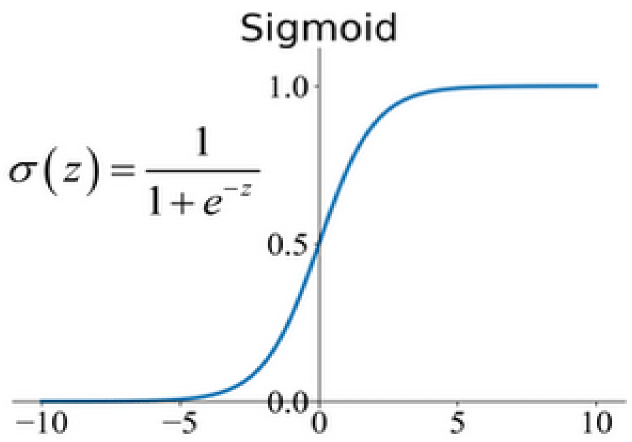


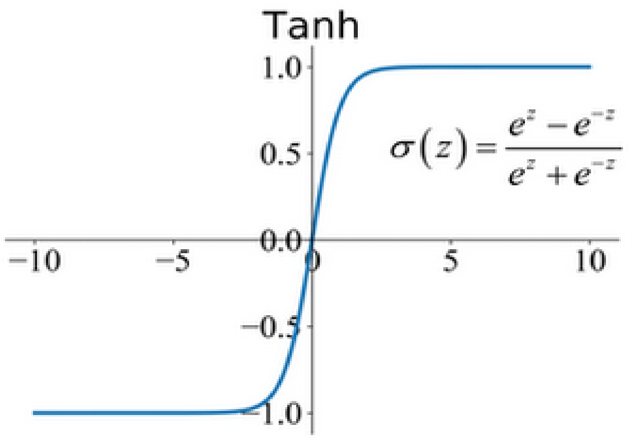
Su respuesta depende de la **Función de activación** elegida

Funciones de activación

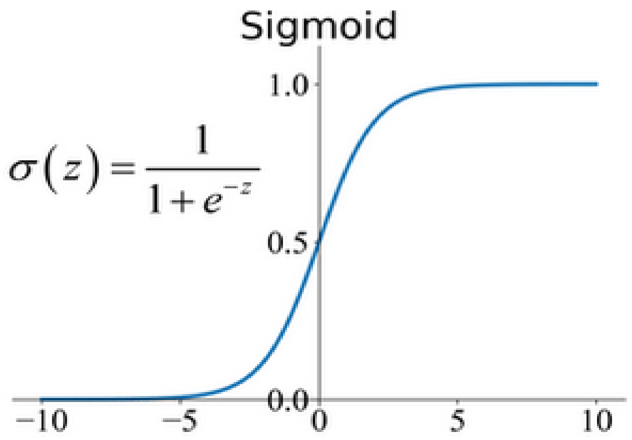


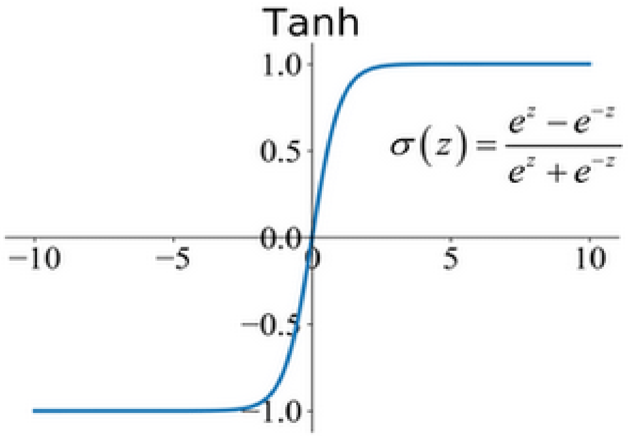
Funciones de activación

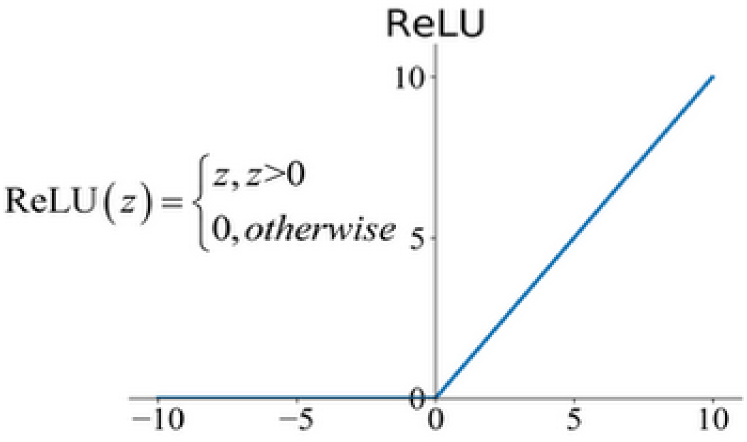




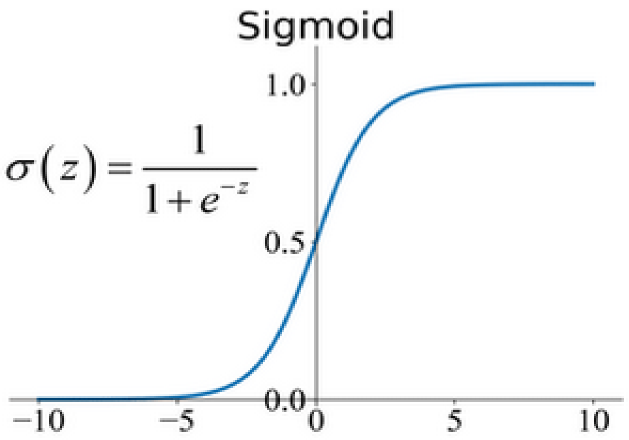
Funciones de activación

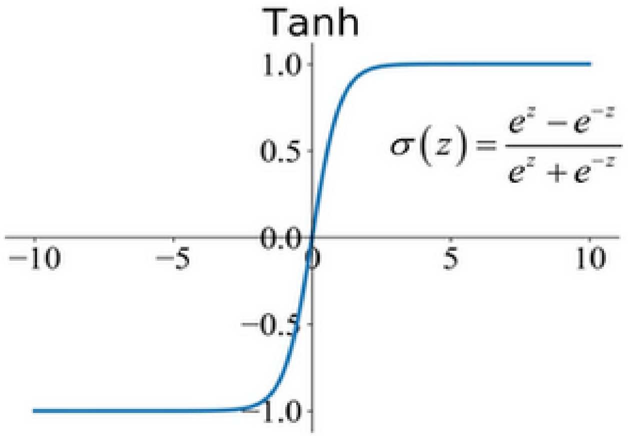


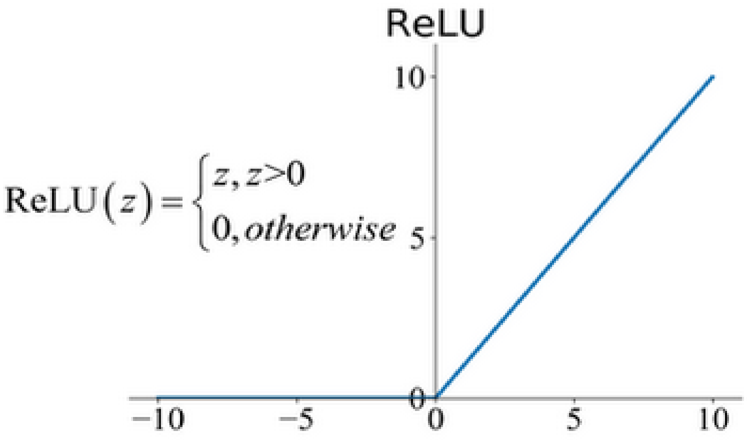


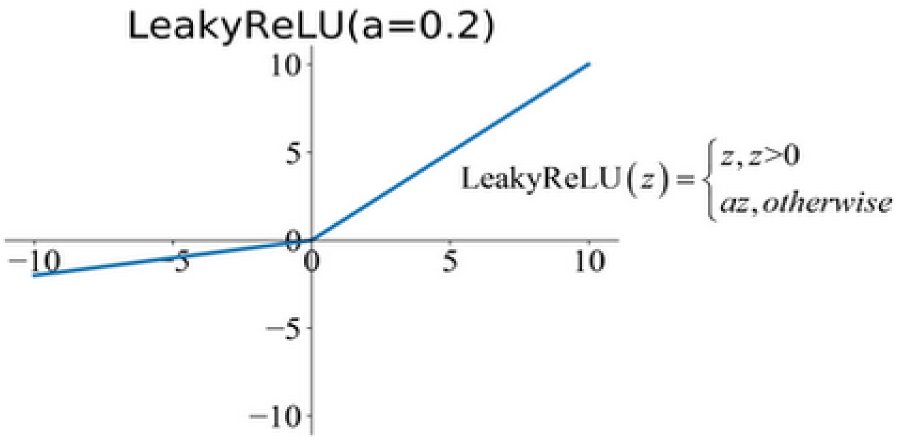


Funciones de activación

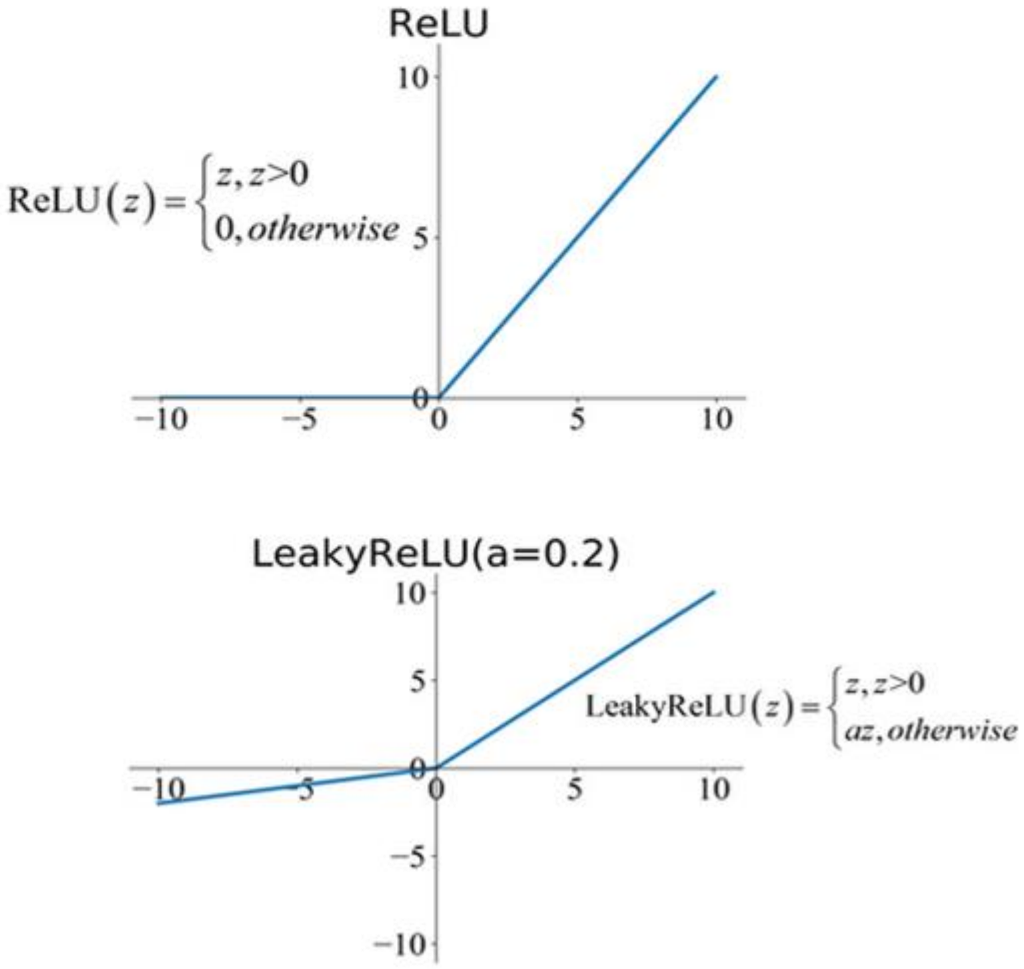








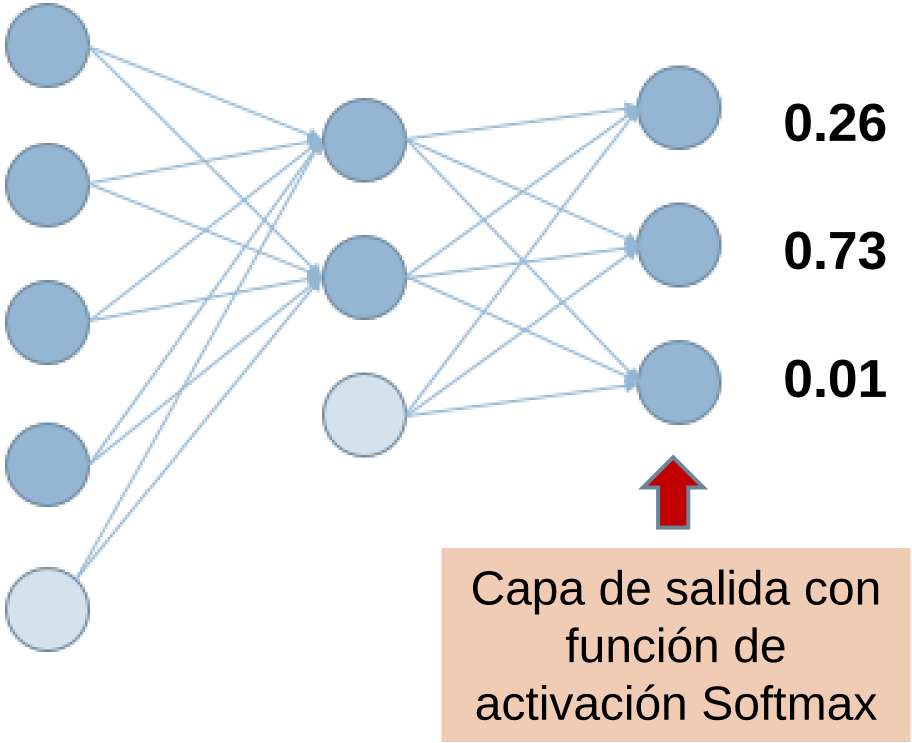
ReLU (Unidad Lineal Rectificada)



- Velocidad de aprendizaje (derivada)
- Velocidad de cómputo (fácil de calcular)
- Activa sólo algunas neuronas

Función Softmax

- Se utiliza como función de activación en la última capa para normalizar la salida de la red de manera que los valores sumen 1.



$$neta_j = \sum_i w_{ji}\, x_i + b_j$$

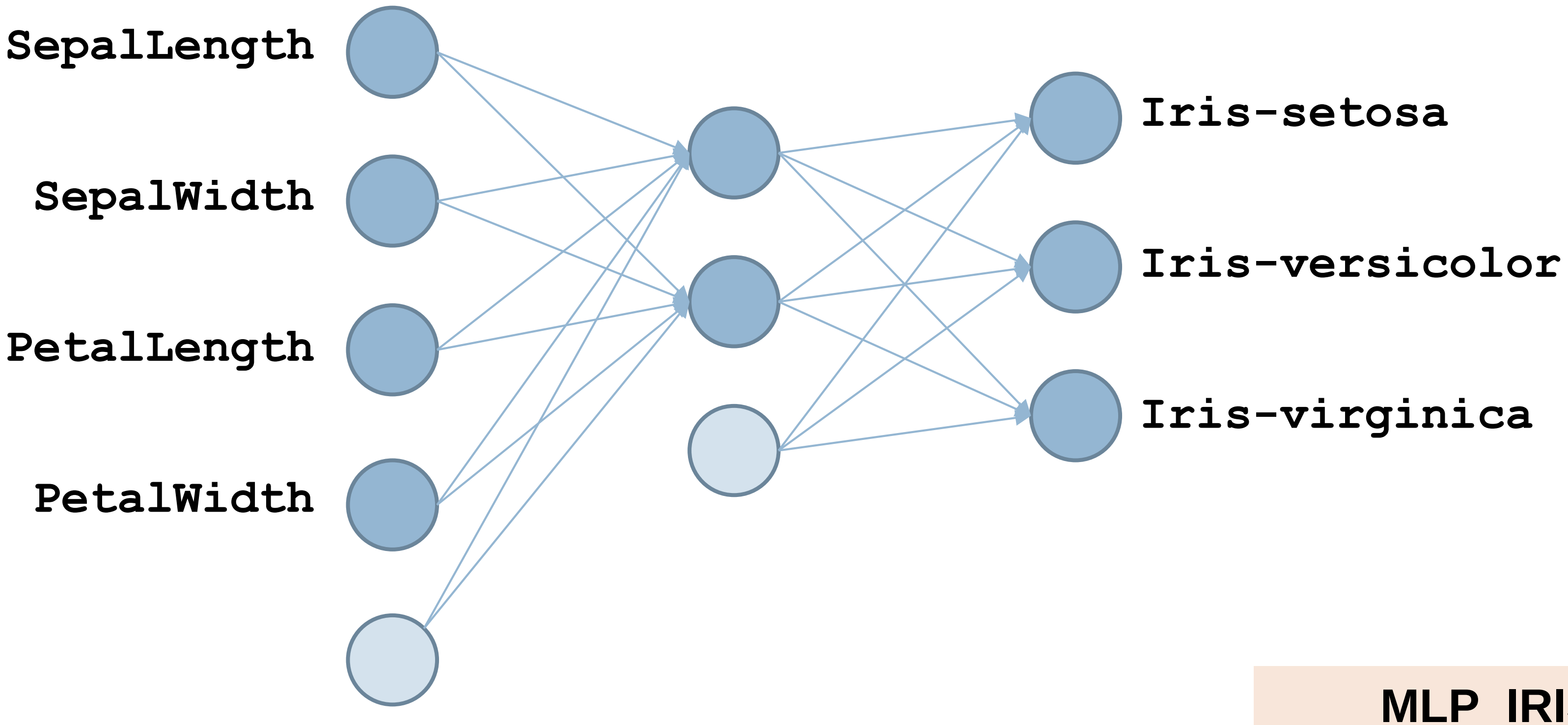
$$\hat{y}_j = \frac{e^{neta_j}}{\sum_k e^{neta_k}}$$

Ejemplo: Clasificación de flores de Iris

Id	sepallength	sepalwidth	petallength	petalwidth	class
1	5,1	3,5	1,4	0,2	Iris-setosa
2	4,9	3,0	1,4	0,2	Iris-setosa
…	…	…	…	…	…
95	5,6	2,7	4,2	1,3	Iris-versicolor
96	5,7	3,0	4,2	1,2	Iris-versicolor
97	5,7	2,9	4,2	1,3	Iris-versicolor
…	…	…	…	…	…
149	6,2	3,4	5,4	2,3	Iris-virginica
150	5,9	3,0	5,1	1,8	Iris-virginica

https://archive.ics.uci.edu/ml/datasets/Iris

Ejemplo: Clasificación de flores de Iris



MLP_IRIS.ipynb

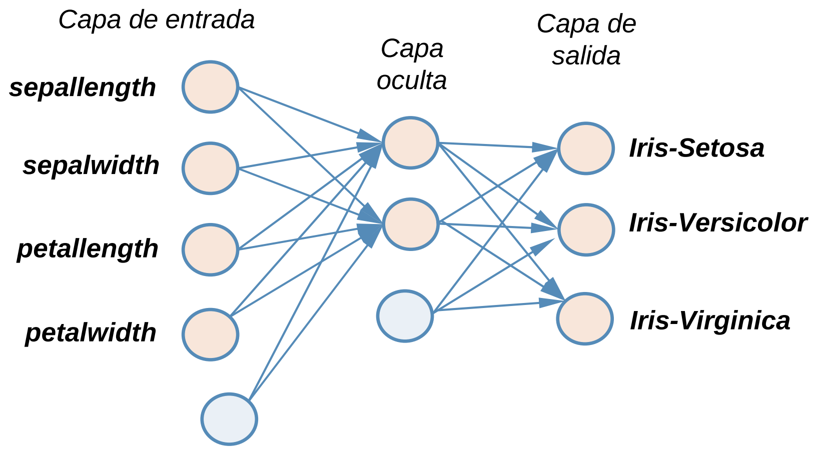
Clasificación de flores de Iris

X

```
[[-1.73,-0.05,-1.38,-1.31],
 [-0.37,-1.62, 0.22, 0.18],
 [ 1.11,-0.05, 0.93, 1.54],
 [-0.99, 0.39,-1.44,-1.31],
 [ 1.73, 1.29, 1.46, 1.81]]
```

Y

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```



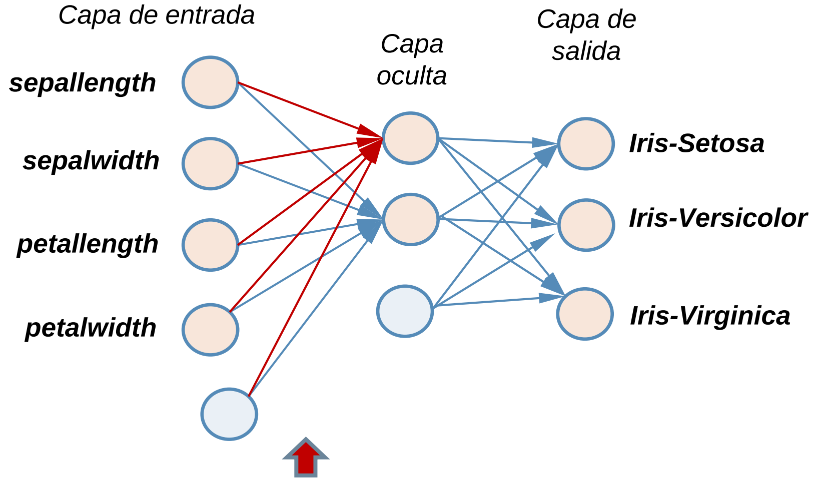
Clasificación de flores de Iris

X

```
[[-1.73,-0.05,-1.38,-1.31],
 [-0.37,-1.62, 0.22, 0.18],
 [ 1.11,-0.05, 0.93, 1.54],
 [-0.99, 0.39,-1.44,-1.31],
 [ 1.73, 1.29, 1.46, 1.81]]
```

Y

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```




```
[[ 0.15, -0.13,  0.23, -0.45],
 [-0.29, -0.41, -0.19,  0.37]]
```

W1

```
[[-0.45],
 [-0.10]]
```

b1

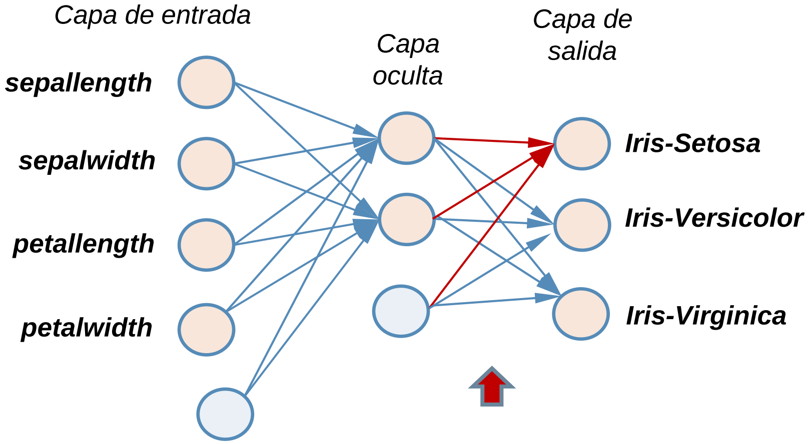
Clasificación de flores de Iris

X

```
[[-1.73,-0.05,-1.38,-1.31],
 [-0.37,-1.62, 0.22, 0.18],
 [ 1.11,-0.05, 0.93, 1.54],
 [-0.99, 0.39,-1.44,-1.31],
 [ 1.73, 1.29, 1.46, 1.81]]
```

Y

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```




```
[[-4.79, -0.99],
 [ 0.38,  0.09],
 [ 2.40, -0.38]]
```

$\underbrace{}_{W2}$

```
[[-0.29],
 [-0.17],
 [-0.27]]
```

$\underbrace{}_{b2}$

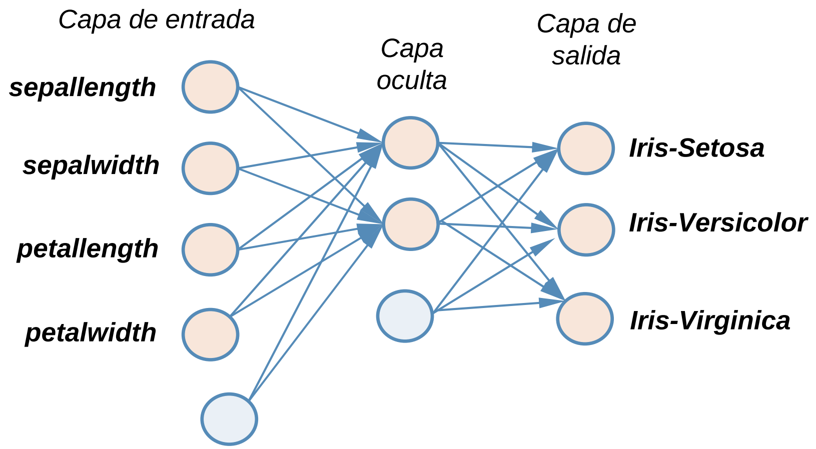
Clasificación de flores de Iris

X

```
[[-1.73,-0.05,-1.38,-1.31],
 [-0.37,-1.62, 0.22, 0.18],
 [ 1.11,-0.05, 0.93, 1.54],
 [-0.99, 0.39,-1.44,-1.31],
 [ 1.73, 1.29, 1.46, 1.81]]
```

Y

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```




```
FunH='relu' ; FunO='sigmoid'
```

Ingresar el primer ejemplo a la red y calcular su salida

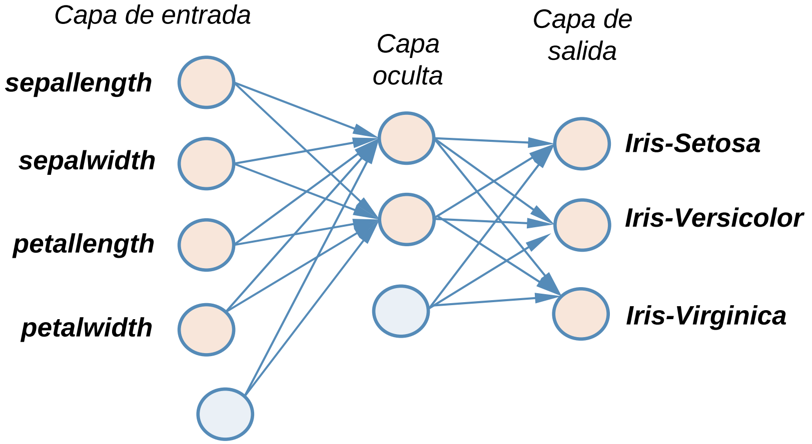
Calculando la salida de la capa oculta

X

```
[[-1.73,-0.05,-1.38,-1.31],
 [-0.37,-1.62, 0.22, 0.18],
 [ 1.11,-0.05, 0.93, 1.54],
 [-0.99, 0.39,-1.44,-1.31],
 [ 1.73, 1.29, 1.46, 1.81]]
```

Y

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```



- Salida de la capa oculta

```
netasH = W1 * x.T + b1
```

```
FunH='relu' ; FunO='sigmoid'
```

$\mathbf{x}^{\mathrm{T}}$

```
[[ 0.15, -0.13,  0.23, -0.45],        [[-1.73],      [[-0.45],       [[-0.4309]
 [-0.29, -0.41, -0.19,  0.37]]    *    [-0.05],   +   [-0.10]]   =    [ 0.1997]]
                                       [-1.38],
                                       [-1.31]]
```

W1 **b1**

$$neta_{pj}^{h} = \sum_{i=1}^{n} w_{ji}^{h} x_{pi} + \theta_{j}^{h}$$

$$i_{pj} = f_{j}^{h}(neta_{pj}^{h})$$

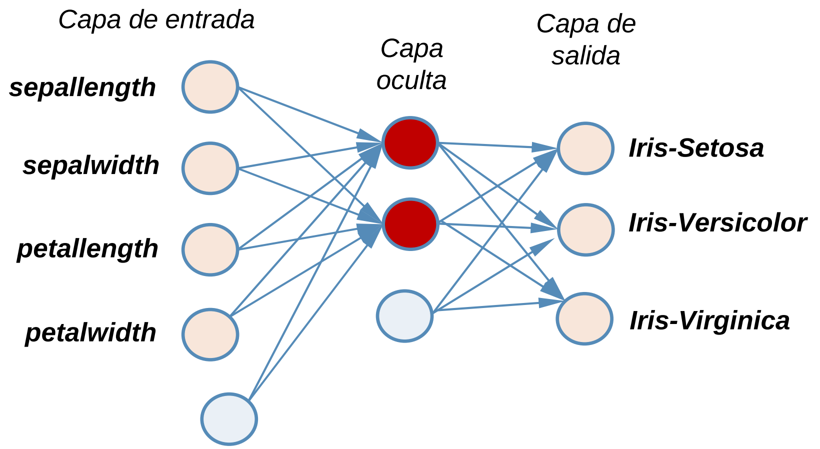
Calculando la salida de la capa oculta

X

```
[[-1.73,-0.05,-1.38,-1.31],
 [-0.37,-1.62, 0.22, 0.18],
 [ 1.11,-0.05, 0.93, 1.54],
 [-0.99, 0.39,-1.44,-1.31],
 [ 1.73, 1.29, 1.46, 1.81]]
```

Y

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```



- Salida de la capa oculta

```
FunH='relu' ; FunO='sigmoid'
```

```
netasH = W1 * x.T + b1
```

$\mathbf{x}^{T}$

```
[[ 0.15, -0.13,  0.23, -0.45],       [[-1.73],     [[-0.45],     [[-0.4309]
 [-0.29, -0.41, -0.19,  0.37]]   *    [-0.05],  +   [-0.10]]  =   [ 0.1997]]
                                      [-1.38],
                                      [-1.31]]
```

W1 **b1**

$$neta^{h}_{pj} = \sum_{i=1}^{n} w^{h}_{ji} x_{pi} + \theta^{h}_{j}$$

```
salidasH = netasH*(netasH>0)    =   [[ 0      ]
                                     [ 0.1997]]
```

$$i_{pj} = f^{h}_{j}(neta^{h}_{pj})$$

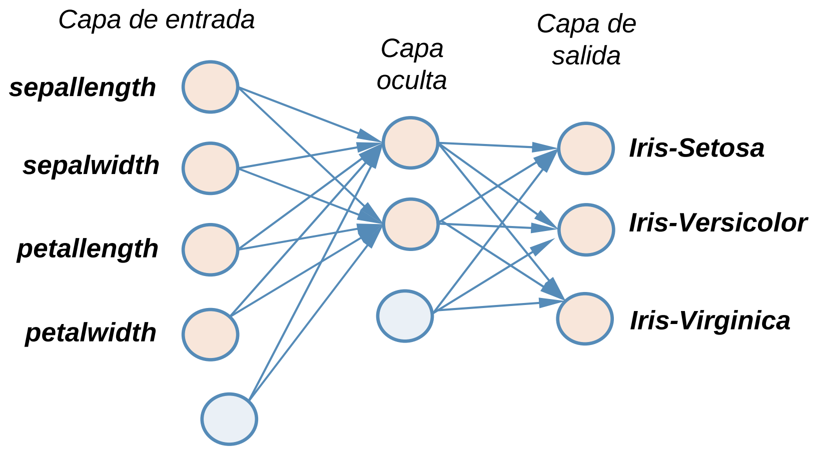
Calculando la salida de la red (capa de salida)

X

```
[[-1.73,-0.05,-1.38,-1.31],
 [-0.37,-1.62, 0.22, 0.18],
 [ 1.11,-0.05, 0.93, 1.54],
 [-0.99, 0.39,-1.44,-1.31],
 [ 1.73, 1.29, 1.46, 1.81]]
```

Y

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```



- Salida de red

```
netasO = W2 * salidasH + b2
```

```
FunH='relu' ; FunO='sigmoid'
```

W2		salidasH		b2		netasO
[[-4.79, -0.99], [0.38, 0.09], [2.40, -0.38]]	*	[[0] [0.1997]]	+	[[-0.29], [-0.17], [-0.27]]	=	[[-0.487703] [-0.152027] [-0.345886]]

$$neta_{pk}^{o} = \sum_{j=1}^{L} w_{kj}^{o} i_{pj} + \theta_{k}^{o}$$

$$o_{pk} = f_{k}^{o}(neta_{pk}^{o})$$

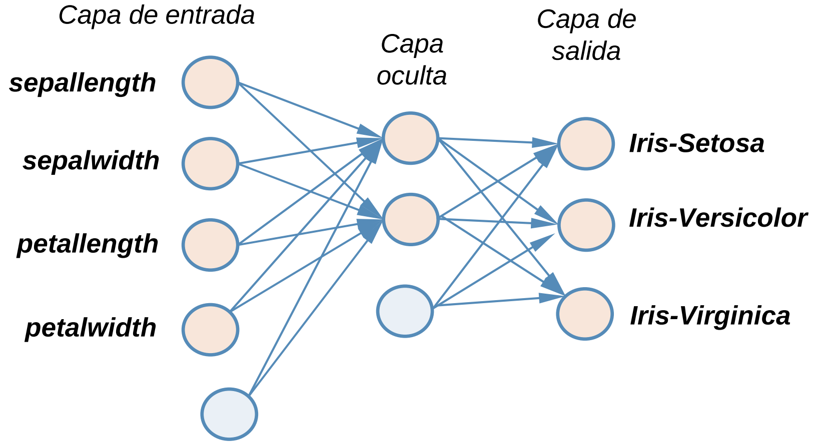
Calculando la salida de la red (capa de salida)

X

```
[[-1.73,-0.05,-1.38,-1.31],
 [-0.37,-1.62, 0.22, 0.18],
 [ 1.11,-0.05, 0.93, 1.54],
 [-0.99, 0.39,-1.44,-1.31],
 [ 1.73, 1.29, 1.46, 1.81]]
```

Y

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```



- Salida de red

```
netasO = W2 * salidasH + b2
```

```
FunH='relu' ; FunO='sigmoid'
```

```
                               salidasH                    netasO
[[-4.79, -0.99],              [[ 0      ]     [[-0.29],     [[-0.487703]
 [ 0.38,  0.09],      *        [ 0.1997]]  +   [-0.17],  =   [-0.152027]
 [ 2.40, -0.38]]                               [-0.27]]      [-0.345886]]
       W2                                        b2
```

$$neta_{pk}^{o} = \sum_{j=1}^{L} w_{kj}^{o} i_{pj} + \theta_{k}^{o}$$

$$o_{pk} = f_{k}^{o}(neta_{pk}^{o})$$

```
salidasO = 1 / (1+np.exp(-netasO)) = [[0.38043483]
                                      [0.46206628]
                                      [0.41438041]]
```

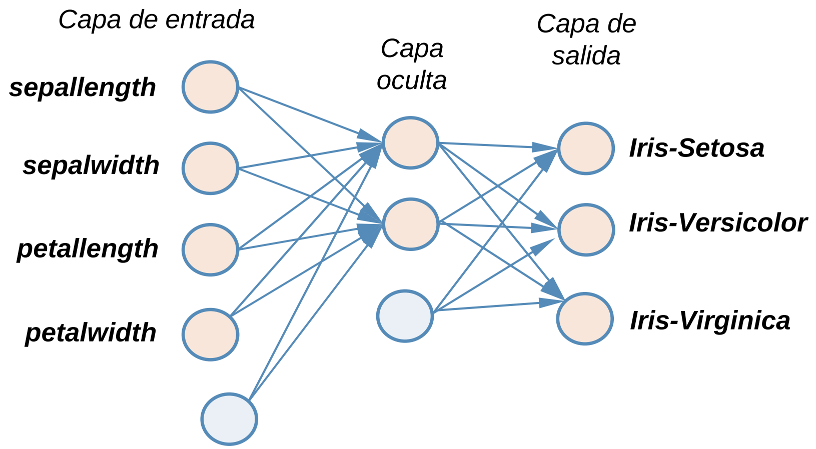
Error de la capa de salida

X

```
[[-1.73,-0.05,-1.38,-1.31],
 [-0.37,-1.62, 0.22, 0.18],
 [ 1.11,-0.05, 0.93, 1.54],
 [-0.99, 0.39,-1.44,-1.31],
 [ 1.73, 1.29, 1.46, 1.81]]
```

Y

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```




```
FunH='relu' ; FunO='sigmoid'
```

- Error en la respuesta de la red para este ejemplo

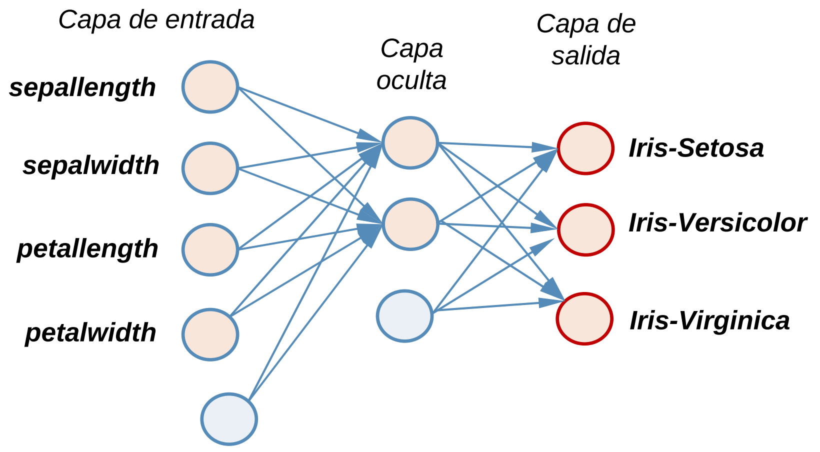
```
ErrorSalida =  y.T - salidasO
```

```
                 [[1.0]      [[0.38043483]       [[ 0.61956517]
ErrorSalida =     [0.0]  -    [0.46206628]    =   [-0.46206628]
                  [0.0]]      [0.41438041]]       [-0.41438041]]
```

salidasO

Factores de corrección de los pesos

X	Y
[[-1.73,-0.05,-1.38,-1.31],	[[1,0,0],
[-0.37,-1.62, 0.22, 0.18],	[0,1,0],
[1.11,-0.05, 0.93, 1.54],	[0,0,1],
[-0.99, 0.39,-1.44,-1.31],	[1,0,0],
[1.73, 1.29, 1.46, 1.81]]	[0,0,1]]



FunH='relu' ; FunO='sigmoid'

- Factores para corregir W2 y b2

salidasO*(1-salidasO)

```
deltaO =  ErrorSalida  .*  derivada_FunO

          [[ 0.61956517]      [[0.23570417]      [[ 0.14603409]
deltaO =   [-0.46206628]  .*   [0.24856103]  =    [-0.11485167]
           [-0.41438041]]      [0.24266929]]      [-0.1005574 ]]
```

$$\delta_{pk}^{o} = (y_{pk} - o_{pk}) f_k^{o}{}'(neta_{pk}^{o})$$

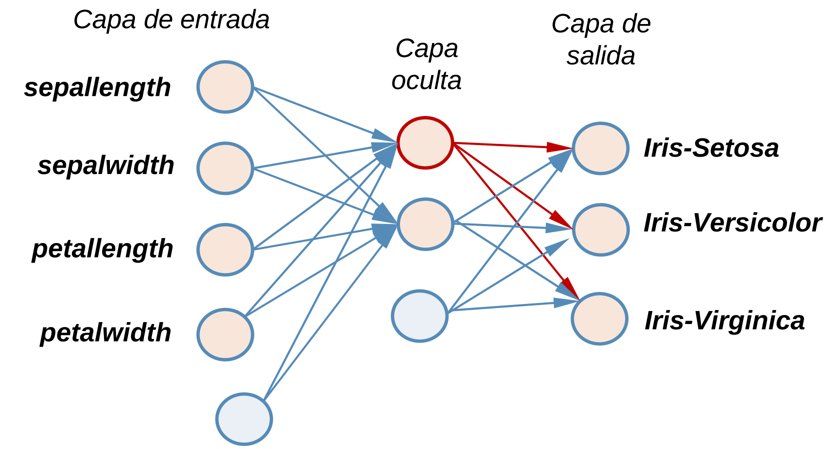
Factores de corrección de los pesos

X

```
[[-1.73,-0.05,-1.38,-1.31],
 [-0.37,-1.62, 0.22, 0.18],
 [ 1.11,-0.05, 0.93, 1.54],
 [-0.99, 0.39,-1.44,-1.31],
 [ 1.73, 1.29, 1.46, 1.81]]
```

Y

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```



- **Factores para corregir W1 y b1**

`FunH='relu' ; FunO='sigmoid'`

`(salidasH > 0)*1`

```
deltaH = deriv_FunH .* (W2.T @ deltaO)

deltaH =   [[ 0 ]     .*   ( [[-4.79  0.38  2.4 ]   @  [[ 0.14603409]
            [ 1 ]]            [-0.99  0.09 -0.38]]      [-0.11485167]
                                                        [-0.1005574 ]] )

deltaH =   [[ 0          ]
            [-0.11669859]]
```

$$\delta_{pj}^h = f_j^{h}{}'(neta_{pj}^h)\sum_k \delta_{pk}^o w_{kj}^o$$

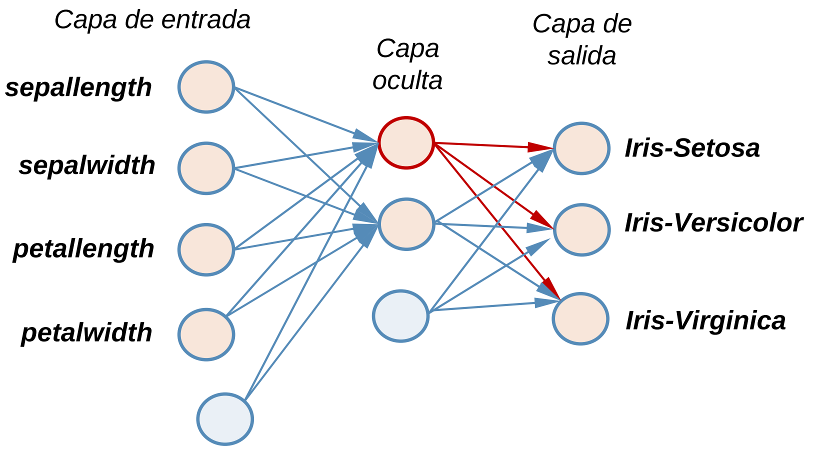
Corrigiendo de los pesos

X

```
[[-1.73,-0.05,-1.38,-1.31],
 [-0.37,-1.62, 0.22, 0.18],
 [ 1.11,-0.05, 0.93, 1.54],
 [-0.99, 0.39,-1.44,-1.31],
 [ 1.73, 1.29, 1.46, 1.81]]
```

Y

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```



- **Modificación de W2 y b2**

```
FunH='relu' ; FunO='sigmoid'
```

```
W2 = W2 + alfa * deltaO @ salidasH.T
```

```
W2 = [[-4.79, -0.99],                 [[ 0.14603409]                                     [[-4.79 -0.987]
      [ 0.38,  0.09], + alfa *  (      [-0.11485167] @  [[ 0 ], [ 0.1997]]  )    =       [ 0.38   0.088]
      [ 2.40, -0.38]]                  [-0.1005574 ]]                                     [ 2.4  -0.382]]
```

```
b2 = b2 + alfa * deltaO =  [[-0.29],          [[ 0.14603409]         [[-0.28],
                            [-0.17], + alfa *  [-0.11485167]    =     [-0.18],
                            [-0.27]]           [-0.1005574 ]]         [-0.28]]
```

Corrigiendo de los pesos

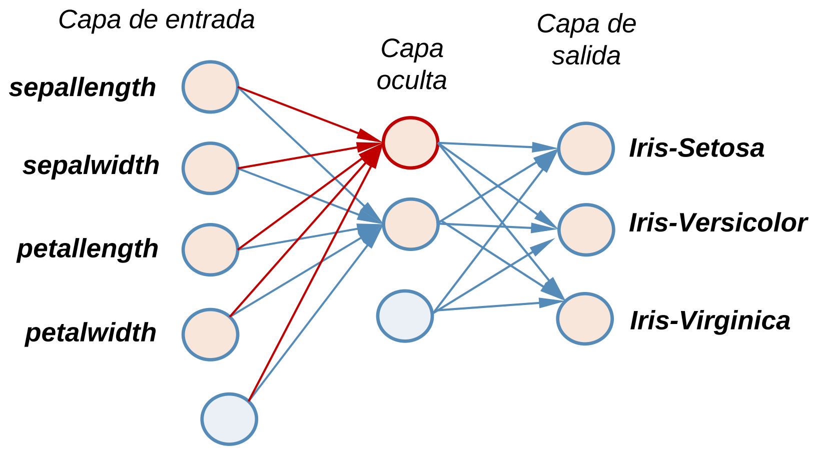
MLP_IRIS_algBPN_RELU.ipynb

X

```
[[-1.73,-0.05,-1.38,-1.31],
 [-0.37,-1.62, 0.22, 0.18],
 [ 1.11,-0.05, 0.93, 1.54],
 [-0.99, 0.39,-1.44,-1.31],
 [ 1.73, 1.29, 1.46, 1.81]]
```

Y

```
[[1,0,0],
 [0,1,0],
 [0,0,1],
 [1,0,0],
 [0,0,1]]
```



- **Modificación de W1 y b1**

```
W1 = W1 + alfa * deltaH @ x
```

```
FunH='relu' ; FunO='sigmoid'
```

```
W1 = [[ 0.15,-0.13,0.23,-0.45],
      [-0.29,-0.41,-0.19,0.37]] + alfa * ( [[ 0.          ]   @  [[-1.73,-0.05,-1.38,-1.31]] )
                                           [-0.11669859]]

W1 = [[ 0.15 -0.13  0.23 -0.45]
      [-0.27 -0.41 -0.17  0.39]]

b1 = b1 + alfa * deltaH = [[-0.45],  + alfa * [[ 0.          ]  = [[-0.45],
                          [-0.10]]             [-0.11669859]]     [-0.11]]
```

Si se ingresa el mismo ejemplo luego de modificar los pesos de la red ...

```
netasH = W1 @ xi.T + b1
salidasH = netasH*(netasH>0)
netasO =  W2 @ salidasH + b2
salidasO = 1.0/(1+np.exp(-netasO))
ErrorSalidaNew = yi.T-salidasO
print("ErrorSalida = \n", ErrorSalidaNew)
```

```
ErrorSalida =
 [[ 0.59502755]
 [-0.45716396]
 [-0.4200931 ]]
```

```
print("Error inicial = ", np.sum(ErrorSalida**2))
print("Error luego de la correccion = ", np.sum(ErrorSalidaNew**2))
```

```
Error inicial =  0.7690773719494183
Error luego de la correccion =  0.7395348839038576
```

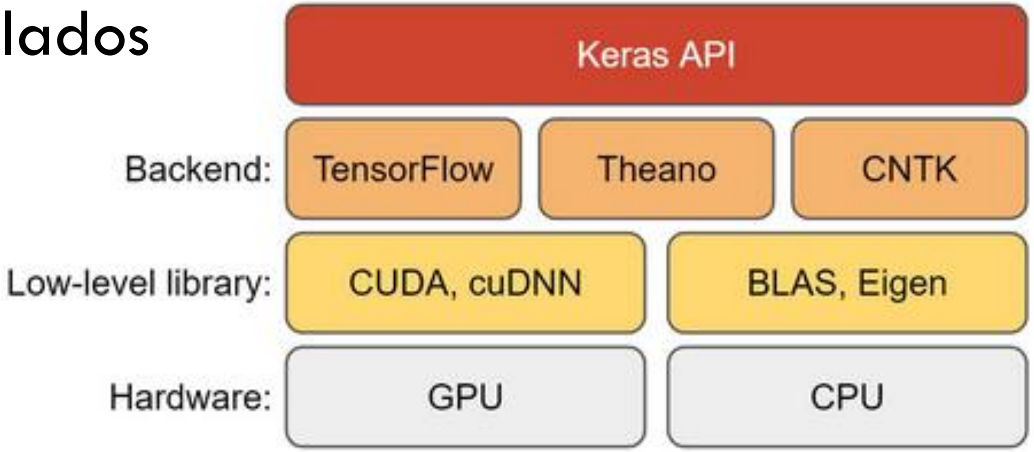
Antes de modificar los pesos de la red

```
salidaO =
 [[0.38043483]
 [0.46206628]
 [0.41438041]]
ErrorSalida =
 [[ 0.61956517]
 [-0.46206628]
 [-0.41438041]]
```

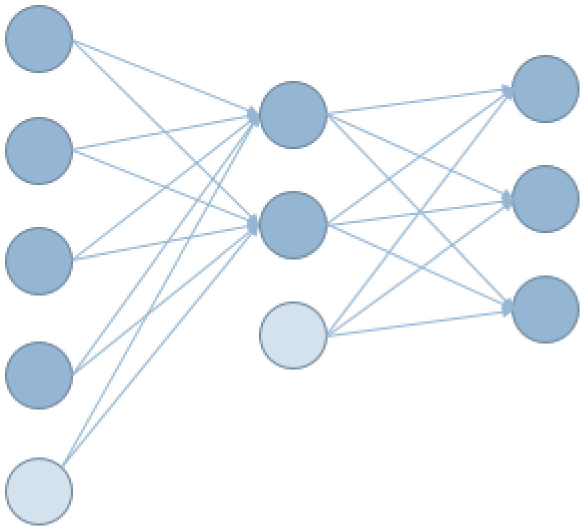
MLP_IRIS_algBPN_RELU.ipynb

Keras

- Keras es una biblioteca de código abierto escrita en Python que facilita la creación de modelos complejos de aprendizaje profundo.
- Características
  - Prototipado rápido del modelo.
  - De alto nivel (programación a nivel de capa)
  - Usa las librerías de los frameworks vinculados
    - TensorFlow
    - Theano
    - Microsoft Cognitive Toolkit (CNTK)



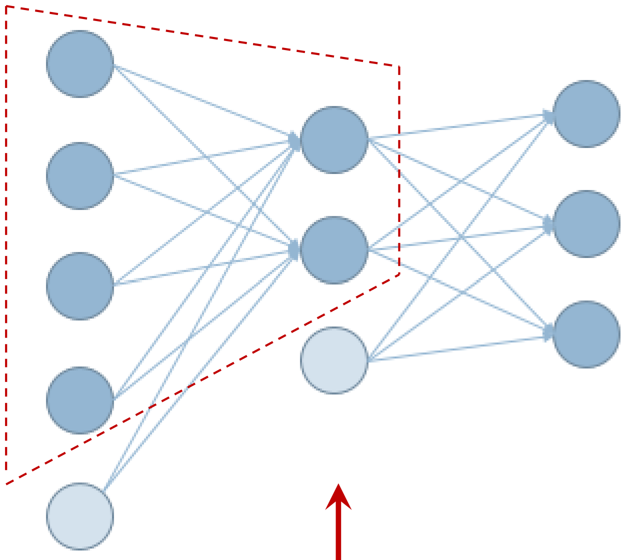
Construcción del modelo



```
from keras.models import Sequential
from keras.layers import Dense

# Crear un modelo de capas secuenciales
model=Sequential()
```

las capas densas son las que estan completamente conectadas

Construcción del modelo



```
from keras.models import Sequential
from keras.layers import Dense
```

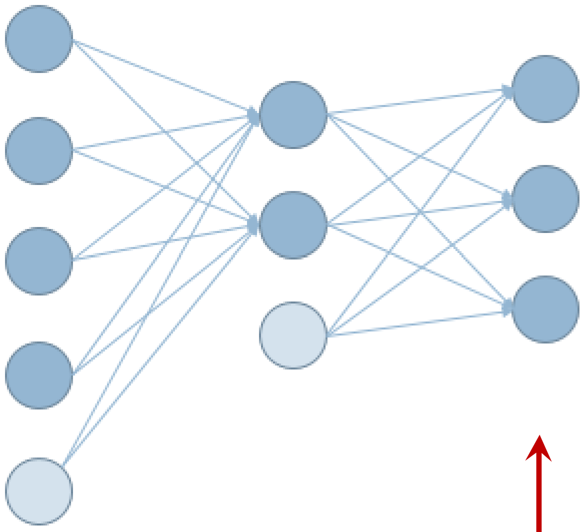
 # Crear un modelo de capas secuenciales

```
model=Sequential()
```

 # Agregar las capas al modelo

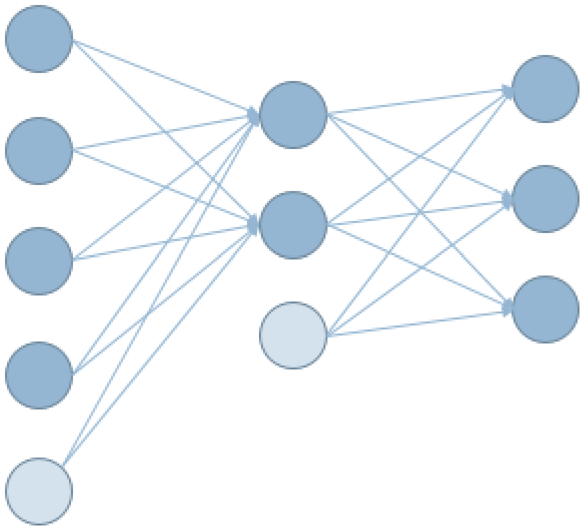
```
model.add(Dense(2, input_shape=[4], activation='tanh'))
```

Construcción del modelo



```
from keras.models import Sequential
from keras.layers import Dense

# Crear un modelo de capas secuenciales
model=Sequential()

# Agregar las capas al modelo
model.add(Dense(2, input_shape=[4], activation='tanh'))
model.add(Dense(3, activation='sigmoid'))
```

Construcción del modelo



```
from keras.models import Sequential
from keras.layers import Dense
```

 # Crear un modelo de capas secuenciales

```
model=Sequential()
```

 # Agregar las capas al modelo

```
model.add(Dense(2, input_shape=[4], activation='tanh'))
model.add(Dense(3, activation='sigmoid'))
```

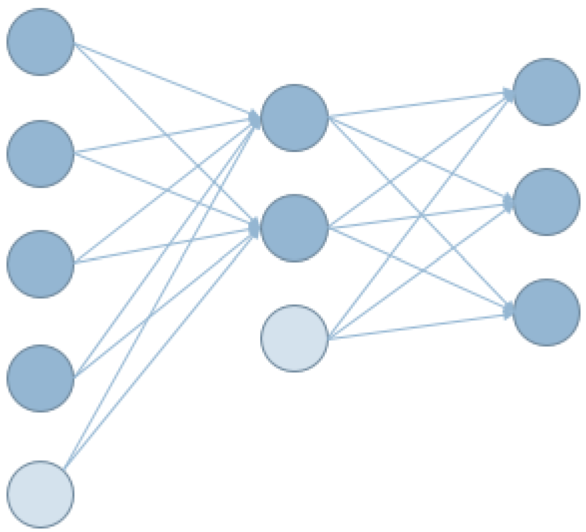
 # Imprimir un resumen del modelo

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 2)                 10
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 9
=================================================================
Total params: 19
Trainable params: 19
Non-trainable params: 0
```

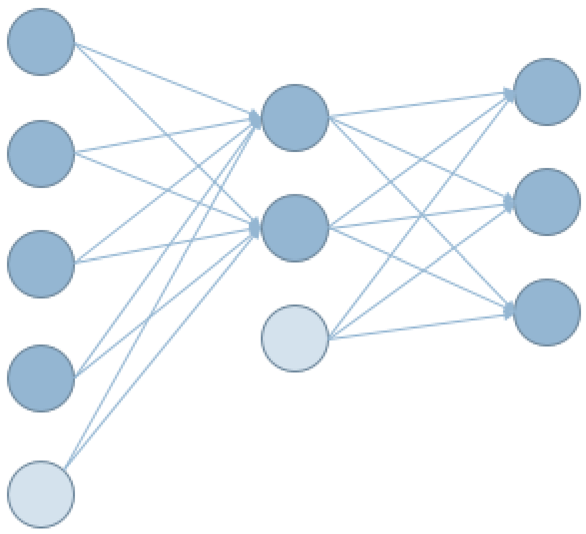
Indicando la función de activación por separado

```
from keras.models import Sequential
from keras.layers import Dense, Activation

model=Sequential()

model.add(Dense(2, input_dim=4))
model.add(Activation('tanh'))

model.add(Dense(3))
model.add(Activation('sigmoid'))

model.summary()
```



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 2)                 10

 activation (Activation)     (None, 2)                 0

 dense_1 (Dense)             (None, 3)                 9

 activation_1 (Activation)   (None, 3)                 0

=================================================================
Total params: 19
Trainable params: 19
Non-trainable params: 0
```

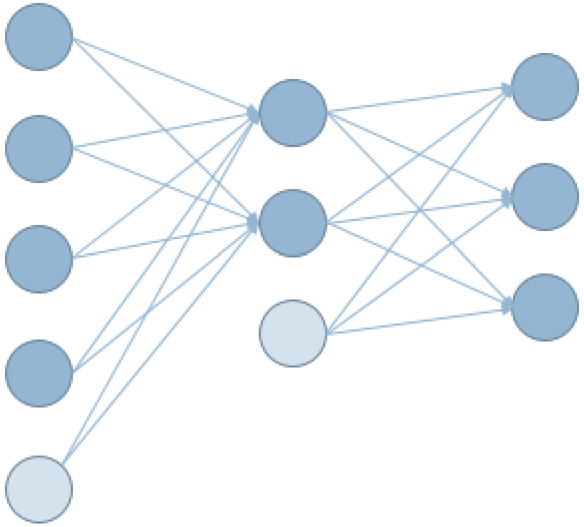
Usando una lista



```
from keras.models import Sequential
from keras.layers import Dense

model=Sequential([
        Dense(2, input_shape=[4], activation='tanh', name='Oculta'),
        Dense(3, activation='sigmoid’, name='salida')])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Oculta (Dense)              (None, 2)                 10

 salida (Dense)              (None, 3)                 9

=================================================================
Total params: 19
Trainable params: 19
Non-trainable params: 0
```

Usando una lista

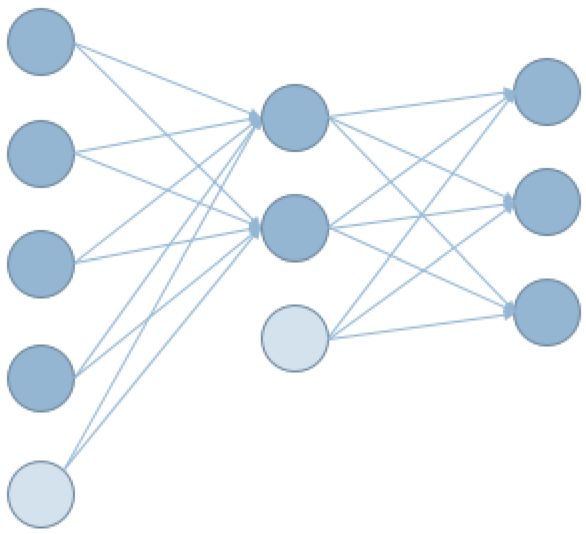


```
from keras.models import Sequential
from keras.layers import Dense, Activation

model=Sequential([
        Dense(2, input_dim=4, name='Oculta'),
        Activation('tanh', name='FunH'),
        Dense(3, name='salida'),
        Activation('sigmoid', name='FunO')])

model.summary()
```

Layer (type)	Output Shape	Param #
Oculta (Dense)	(None, 2)	10
FunH (Activation)	(None, 2)	0
salida (Dense)	(None, 3)	9
FunO (Activation)	(None, 3)	0

Total params: 19

Funcional



```
from keras.models import Model
from keras.layers import Dense, Input

I = Input(shape=(4,))

L = Dense(units=2, activation='tanh', name='Oculta')(I)

salida=Dense(units=3, activation='sigmoid', name='salida')(L)

model = Model(inputs=I, outputs = salida)

model.summary()
```

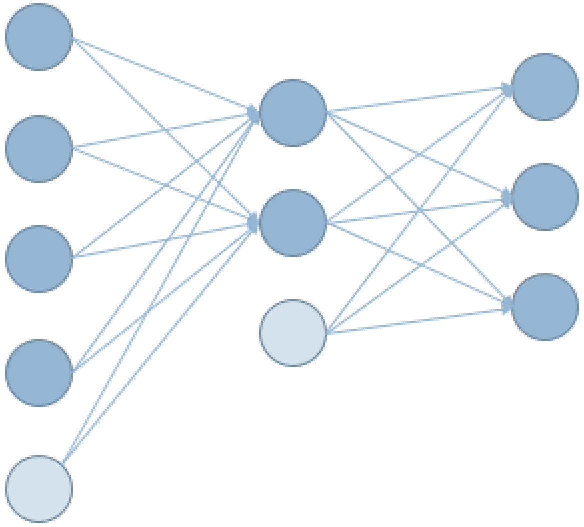
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 4)]               0

 Oculta (Dense)              (None, 2)                 10

 salida (Dense)              (None, 3)                 9

=================================================================
Total params: 19
```

```python
from tensorflow.keras.models import Se
from tensorflow.keras.layers import De
```

```
Model: "model"

 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
 input_1 (InputLayer)           [(None, 32)]         0           []
 input_2 (InputLayer)           [(None, 64)]         0           []
 dense (Dense)                  (None, 16)           528         ['input_1[0][0]']
 dense_1 (Dense)                (None, 32)           2080        ['input_2[0][0]']
 concatenate (Concatenate)      (None, 48)           0           ['dense[0][0]',
                                                                  'dense_1[0][0]']
 dense_2 (Dense)                (None, 1)            49          ['concatenate[0][0]']
==================================================================================
Total params: 2,657
```

```python
# Definir entradas múltiples
entrada_1 = Input(shape=(32,))
entrada_2 = Input(shape=(64,))

# Procesar las entradas de forma independiente
x1 = Dense(16, activation='relu')(entrada_1)
x2 = Dense(32, activation='relu')(entrada_2)

# Combinar las salidas de las dos ramas
fusionado = concatenate([x1, x2])

# Capa de salida
salida = Dense(1, activation='sigmoid')(fusionado)

# Crear el modelo
modelo_complejo = Model(inputs=[entrada_1, entrada_2], outputs=salida)

modelo_complejo.summary()
```

Definiendo capas



```
from keras.models import Model
from keras.layers import Dense, Input

I = Input(shape=(4,), name='entrada')

oculta = Dense(units=2, activation='tanh',name='Oculta')

salida = Dense(units=3, activation='sigmoid', name='salida')

red = salida(oculta(I))

model = Model(inputs=I, outputs = red)

model.summary()
```

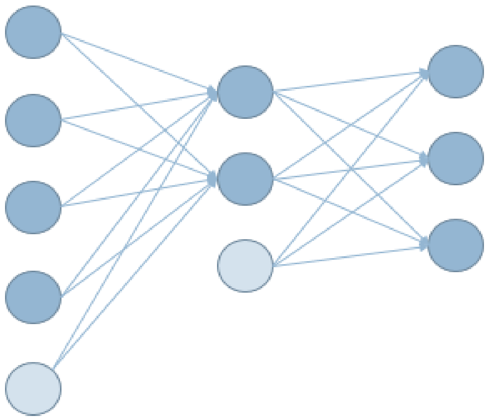
```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 entrada (InputLayer)        [(None, 4)]               0

 Oculta (Dense)              (None, 2)                 10

 salida (Dense)              (None, 3)                 9

=================================================================
Total params: 19
```

Resumen

Resolución de una tarea de clasificación

- Conjunto de datos etiquetados (aprendizaje supervisado)
- Definición de la arquitectura de la red
  - Número de capas y tamaño de cada una
  - Función de activación a usar en cada capa
- Entrenamiento
  - Función de costo
  - Técnica de optimización para reducir el error
- Evaluar el modelo

Configuración para entrenamiento



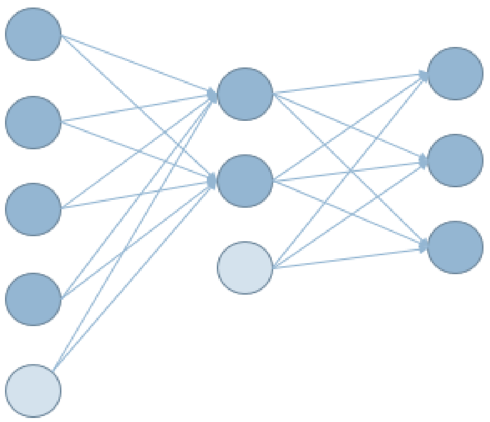
Keras_IRIS.ipynb

```
from keras.models import Sequential
from keras.layers import Dense

model=Sequential()
model.add(Dense(2, input_shape=[4], activation='tanh'))
model.add(Dense(3, activation='sigmoid'))

# Configuración para entrenamiento
model.compile(optimizer='sgd', loss='mse', metrics='accuracy')
```

Descenso de gradiente estocástico

Error Cuadrático Medio

Configuración para entrenamiento

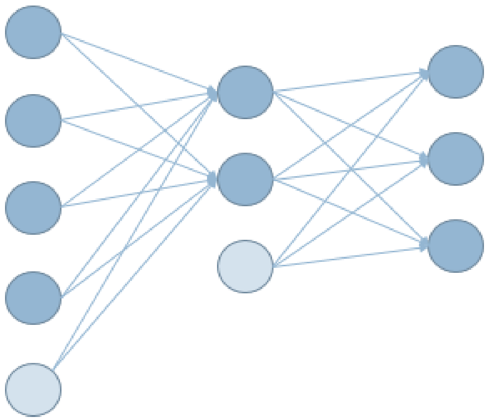


Keras_IRIS_SGD.ipynb

```
from keras.optimizers import SGD
from keras.models import Sequential
from keras.layers import Dense

model=Sequential()
model.add(Dense(2, input_shape=[4], activation='tanh'))
model.add(Dense(3, activation='sigmoid'))

# Configuración para entrenamiento
model.compile(optimizer=SGD(learning_rate=0.1), loss='mse', metrics='accuracy')
```

Tasa de aprendizaje

Configuración para entrenamiento



```
from keras.models import Sequential
from keras.layers import Dense

model=Sequential()
model.add(Dense(2, input_shape=[4], activation='tanh'))
model.add(Dense(3, activation='softmax'))

# Configuración para entrenamiento
model.compile( loss='categorical_crossentropy',  optimizer='sgd',
               metrics ='accuracy')
```

Keras_Iris_Softmax.ipynb

Carga de datos

X,T = cargar_datos()

Y = keras.utils.to_categorical(T)

***T** debe ser un vector numérico. Puede usar lo siguiente para convertirlo de ser necesario:*

```
from sklearn import preprocessing
encoder = preprocessing.LabelEncoder()
T = encoder.fit_transform(T)
```

X → *Conjunto de ejemplos de entrada*

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5	3.6	1.4	0.2
5	5.4	3.9	1.7	0.4

Y → *Rtas esperadas para cada neurona de la capa de salida*

	0	1	2
0	1	0	0
1	0	1	0
2	0	0	1
3	0	0	1
4	0	1	0
5	0	0	1

X e Y son matrices de numpy

Entrenamiento del modelo

```
X,Y = cargar_datos()   # X e Y son matrices de numpy
# Entrenar el modelo
model.fit(X,Y, epochs=100, batch_size=20)
```

Predicción del modelo

X,Y = cargar_datos() ***# X e Y son matrices de numpy***

 # Entrenar el modelo

model.fit(X,Y, epochs=100, batch_size=20)

 # predecir la salida del modelo

Y_pred = model.predict(X)

	0	1	2
0	0.967722	0.189344	0.00421873
1	0.0372113	0.510963	0.346058
2	0.00325751	0.261545	0.917956
3	0.00823823	0.319694	0.795647
4	0.0717264	0.611822	0.171516
5	0.0134856	0.482814	0.59486

Y_pred *tiene las mismas dimensiones que* **Y**

Error del modelo

```
X,Y = cargar_datos()   # X e Y son matrices de numpy
# Entrenar el modelo
model.fit(X,Y, epochs=100, batch_size=20)
# predecir la salida del modelo
Y_pred = model.predict(X)

# Calcular el error del modelo
score = model.evaluate(X_train, Y_trainB)
print('Error :', score[0])
print('Accuracy:', score[1])
```

Muestra el valor de la función de Costo y la precisión del modelo al finalizar el entrenamiento

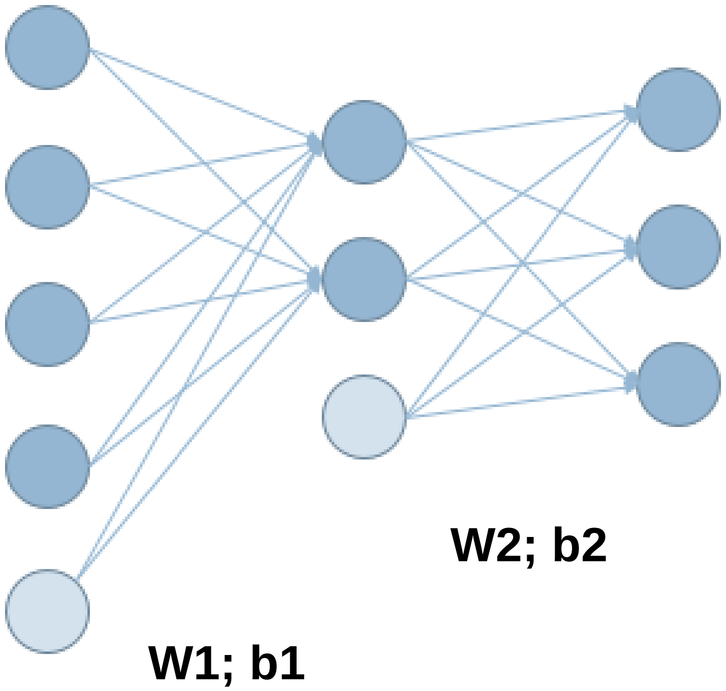
Métricas

```
# Entrenar el modelo
model.fit ( X, Y, epochs=100, batch_size=20)
# Predicciones del modelo
Y_pred = model.predict (X)

Y_pred_nro = np.argmax (Y_pred, axis=1) # conversión a entero

Y_true = np.argmax(Y, axis=1)

print("%% aciertos %.3f" % metrics.accuracy_score(Y_true, Y_pred_nro))
```

ver
Keras_IRIS.ipynb

Pesos de la red

```
model.fit(…)
…
capaOculta = model.layers[0]
W1, b1 = capaOculta.get_weights()

capaSalida = model.layers[1]
W2,b2 = capaSalida.get_weights()
```



Salvar el modelo

- Una vez entrenado el modelo, si los resultados han sido buenos lo guardamos para su uso posterior

OPCION 1
Guardamos todo el modelo

model = …
model.fit(…)
…
model.save("miModelo.h5")

OPCION 2
Guardamos sólo los pesos

model = …
model.fit(…)
…
model.save_weights("pesos_de_miModelo.h5")

Requiere definir el modelo antes de cargar

https://filext.com/es/online-file-viewer.html

Cargar el modelo

OPCION 1 – Carga el modelo completo

```
from keras.models import load_model
model = load_model(“miModelo.h5”)
```

OPCION 2 – Cargar sólo los pesos

```
model = … (definir el modelo)
…
model.load_weights(“pesos_de_miModelo.h5”)
```

Técnicas de optimización

- Descenso de gradiente estocástico (SGD) y el uso de mini-lotes
- Capacidad de generalización de la red - Sobreajuste
- Mejoras introducidas
  - Momento: utiliza información de los gradientes anteriores
  - RMSProp: considera distintas magnitudes de cambio para reducir oscilaciones
  - Adam: combina los dos anteriores. Es el más usado.

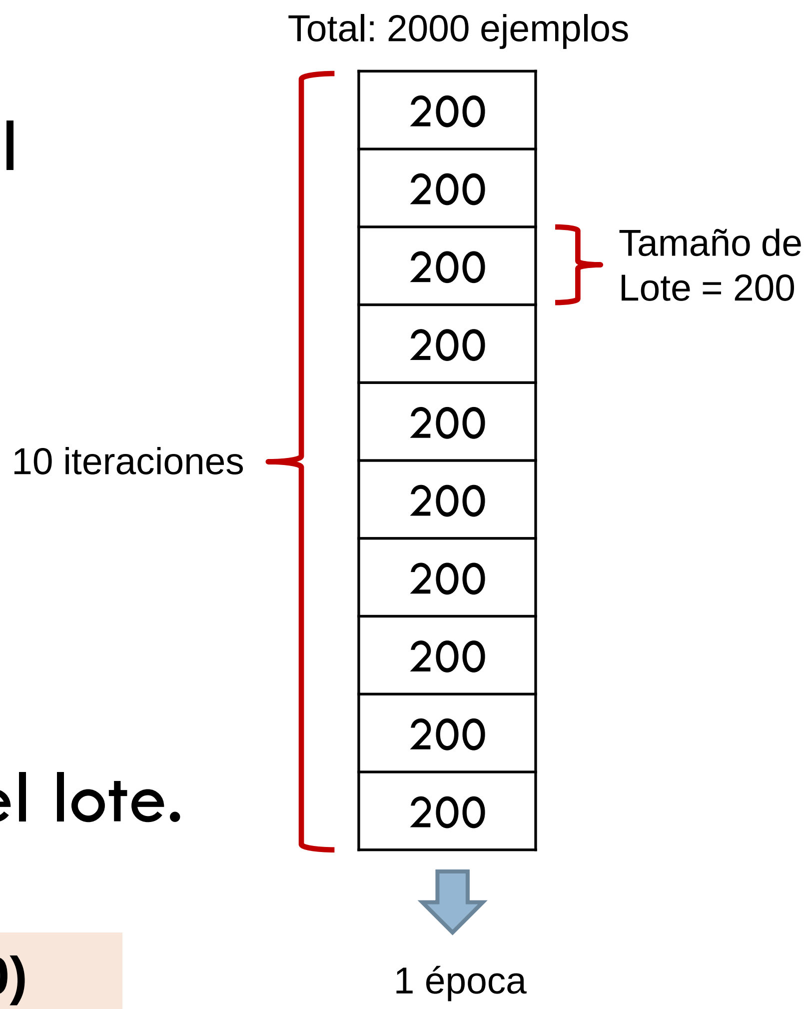
Descenso de gradiente en mini-lotes

- En lugar de ingresar los ejemplos de a uno, ingresamos N a la red y buscamos minimizar el error cuadrático promedio del lote.

- La función de costo será

$$C = \frac{1}{N}\sum_{i=1}^{N}(d_i - f(neta_i))^2$$

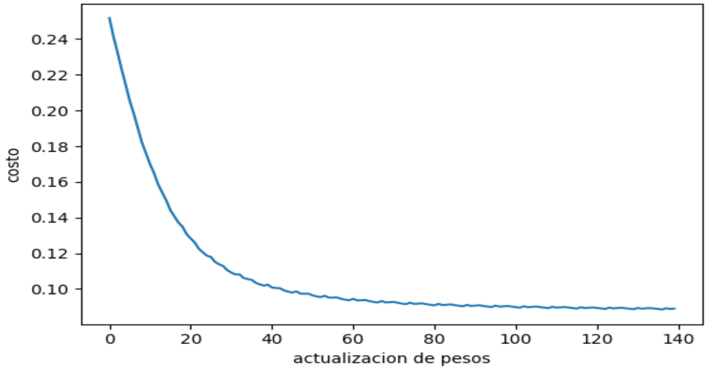
- N es la cantidad de ejemplos que conforman el lote.

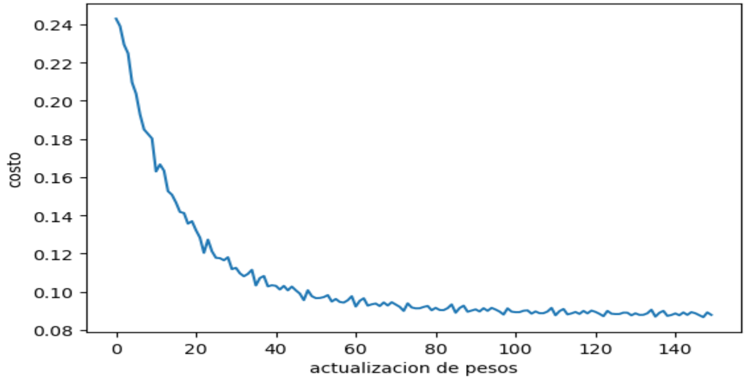


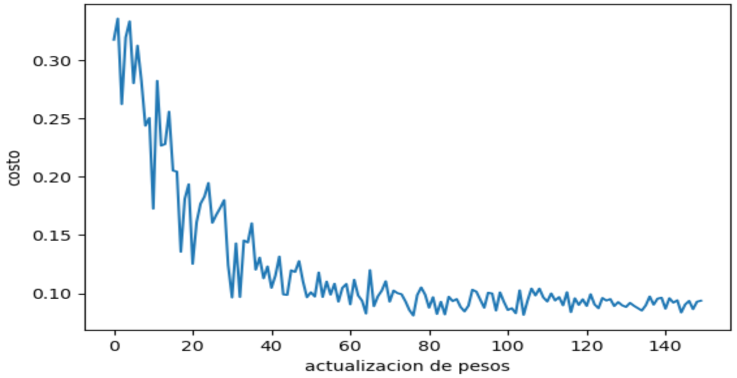

```
model.fit(X, Y, epochs=2000, batch_size=200)
```

Descenso de gradiente

Batch	Mini-batch	Stochastic
Ingresa TODOS los ejemplos y luego actualiza los pesos.	Ingresa un LOTE de N ejemplos y luego actualiza los pesos	Ingresa UN ejemplo y luego actualiza los pesos
$C = \frac{1}{M}\sum_{i=1}^{M}(d_i - f(neta_i))^2$	$C = \frac{1}{N}\sum_{i=1}^{N}(d_i - f(neta_i))^2 \quad N \ll M$	$C = (d - f(neta))^2$





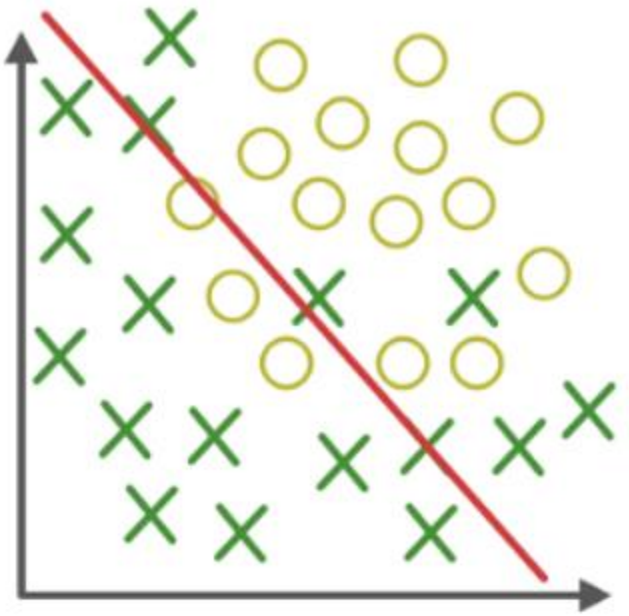


Ver **MLP_MNIST_8x8.ipynb** y **MLP_MNIST_8x8_miniLote.ipynb**

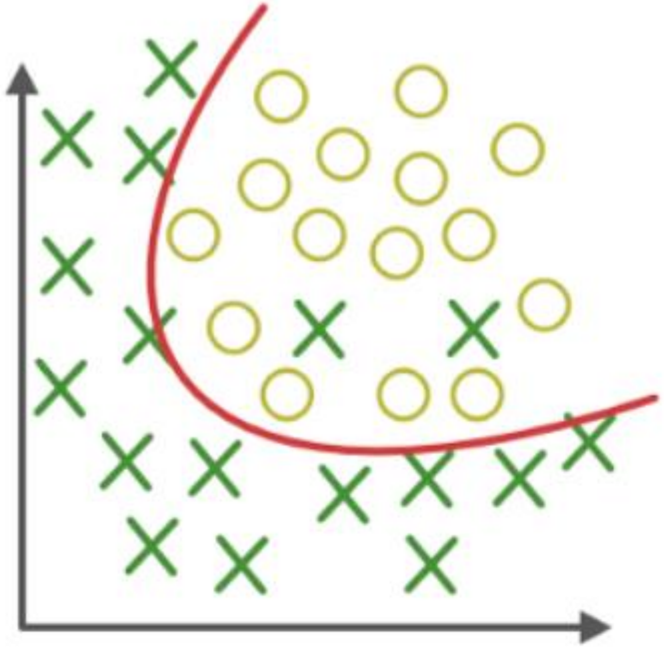
Técnicas de optimización

- Descenso de gradiente estocástico (SGD) y el uso de mini-lotes
- Capacidad de generalización de la red - Sobreajuste
- Mejoras introducidas
  - Momento: utiliza información de los gradientes anteriores
  - RMSProp: considera distintas magnitudes de cambio para reducir oscilaciones
  - Adam: combina los dos anteriores. Es el más usado.

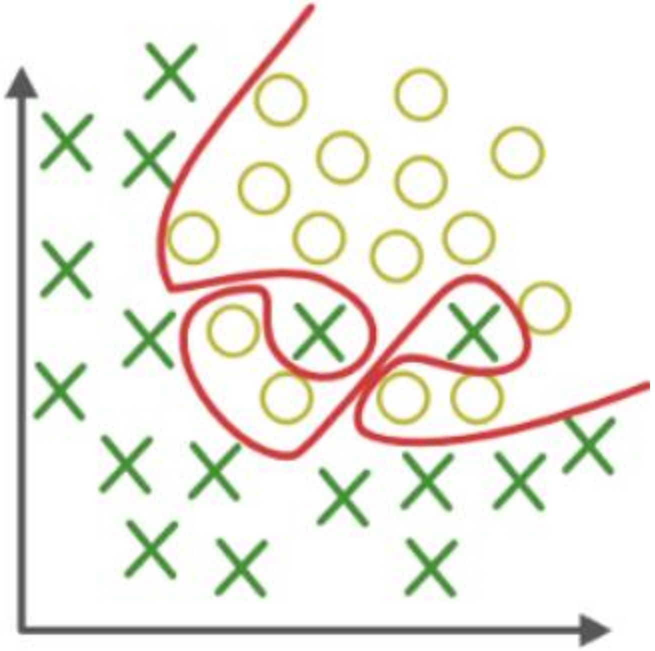
Capacidad de generalización de la red



Underfitting
(demasiado simple)



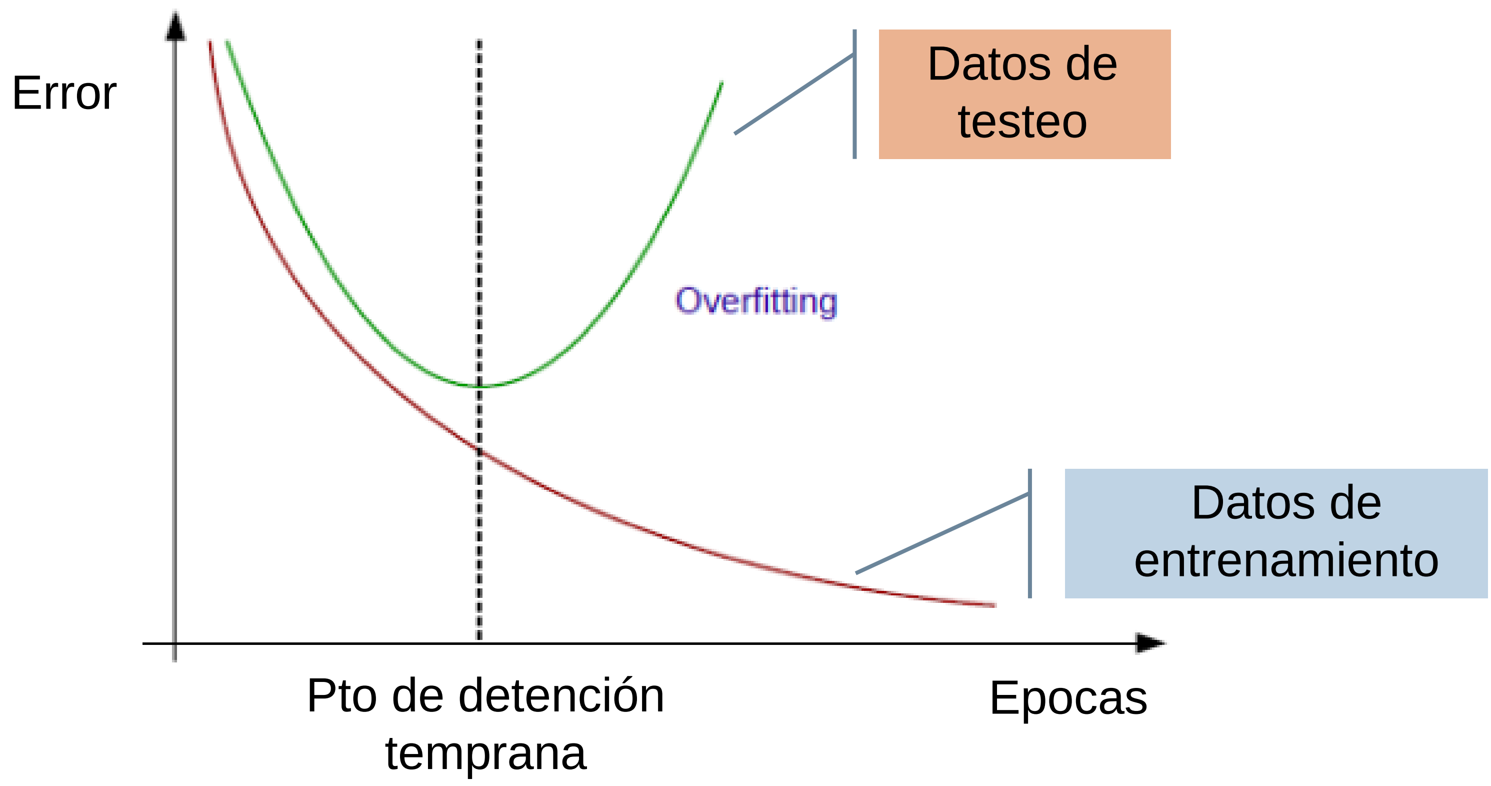
Generalización
correcta

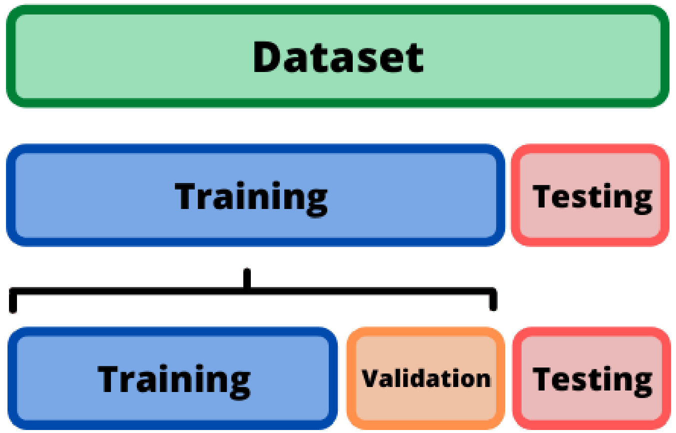


Overfitting
(demasiados
parámetros)

Sobreajuste

- Parada temprana (early-stopping)





Parada temprana

```
from keras.callbacks import EarlyStopping
model = …
model.compile( … )

es = EarlyStopping(monitor='val_accuracy', patience=30, min_delta=0.0001)
H = model.fit(x = X_train, y = Y_train, epochs=4000, batch_size = 20,
              validation_data = (X_test, Y_test),  callbacks=[es])

print("Epocas = %d" % es.stopped_epoch)
```

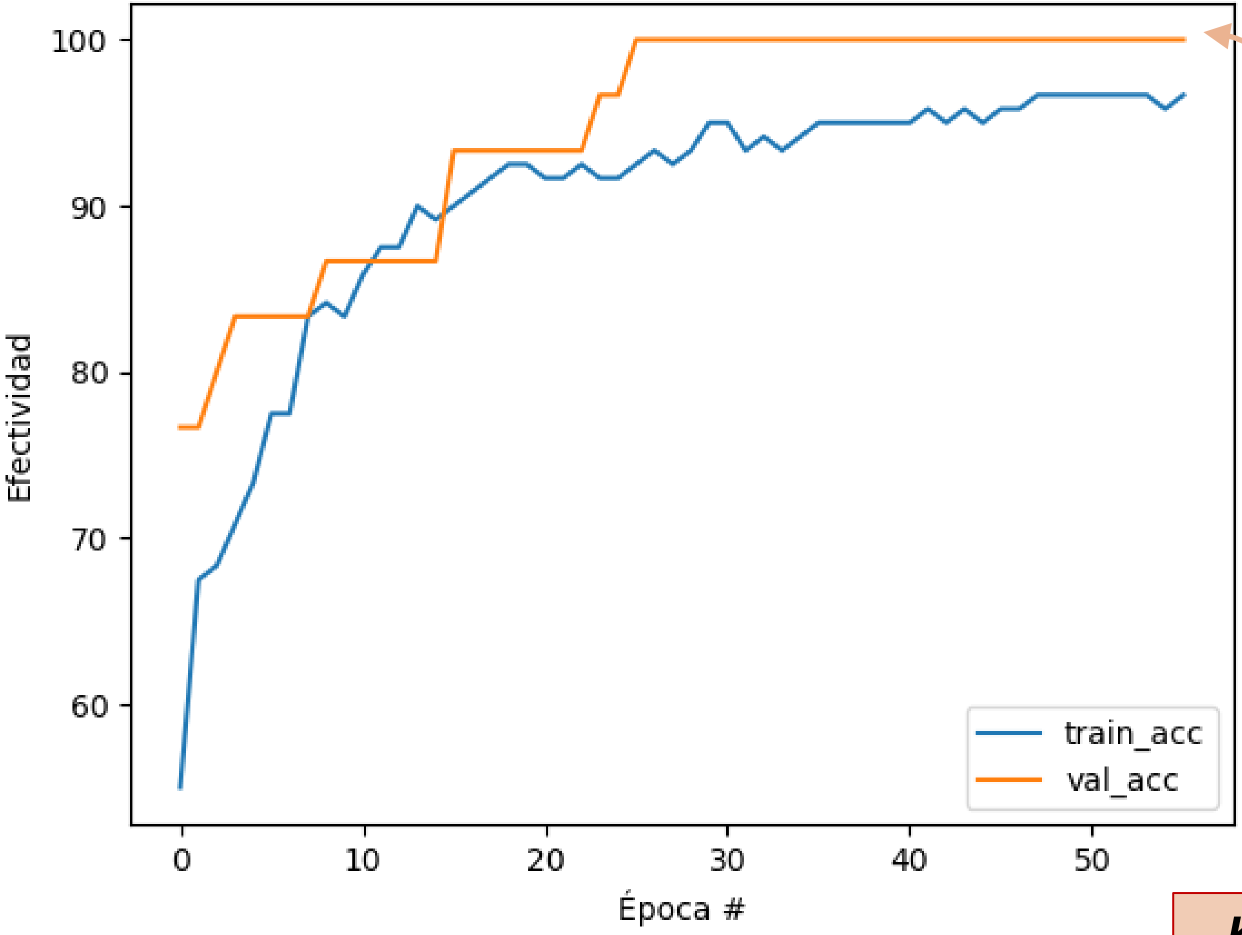
Keras_Iris_softmax_earlyStop.ipynb ; Keras_IRIS_Softmax_earlyStop_Valida.ipynb

EarlyStopping

- Detiene el entrenamiento cuando una métrica ha dejado de mejorar.
- Parámetros principales
  - **monitor**: valor a monitorear
  - **min_delta**: un cambio absoluto en el valor monitoreado inferior a min_delta, se considerará como que no hubo mejora.
  - **patience:** Número de épocas sin mejora tras las cuales se detendrá el entrenamiento.
  - **modo:** Uno de {"auto", "min", "max"}. En el modo "min", el entrenamiento se detendrá cuando el valor monitoreado haya dejado de disminuir; en el modo "max" se detendrá cuando el valor monitoreado haya dejado de aumentar; en el modo "auto", la dirección se infiere automáticamente del nombre del valor monitoreado.
  - **restore_best_weights**: Si se restauran los pesos del modelo de la época con el mejor resultado del valor monitoreado.

https://keras.io/api/callbacks/early_stopping/

Evolución del entrenamiento



_monitor='val_accuracy‘_

patience=30

_min_delta=0.0001_

_Keras_IRIS_Softmax_earlyStop.ipynb_

Reducción del sobreajuste

- Si lo que se busca es reducir el sobreajuste puede probar
  - Incrementar la cantidad de ejemplos de entrenamiento.
  - Reducir la complejidad del modelo, es decir usar menos pesos (menos capas o menos neuronas por capa).
  - Aplicar una técnica de regularización
    - Regularización L2
    - Regularización L1
    - Dropout

Tienen por objetivo que los pesos de la red se mantengan pequeños

Sobreajuste - Regularización L2

- También conocida como técnica de decaimiento de pesos

$$C = C_o + \frac{\lambda}{2}\sum_k w_k^2$$

donde C_o es la función de costo original sin regularizar

- La derivada de la función de costo regularizada será

$$\frac{\partial C}{\partial w_k} = \frac{\partial C_0}{\partial w_k} + \lambda\, w_k$$

Sobreajuste - Regularización L2

Función de costo regularizada	Derivada
$C = C_o + \frac{\lambda}{2}\sum_k w_k^2$	$\frac{\partial c}{\partial w_k} = \frac{\partial c_0}{\partial w_k} + \lambda\, w_k$

- Actualización de los pesos

$$w_k = w_k - \alpha \frac{\partial C_0}{\partial w_k} - \lambda\, w_k$$

$$\boxed{w_k = (1-\lambda)\, w_k - \alpha \frac{\partial C_0}{\partial w_k}}$$

Sobreajuste - Regularización L1

Función de costo regularizada	Derivada
$C = C_o + \lambda \sum_k \lvert w_k \rvert$	$\frac{\partial C}{\partial w_k} = \frac{\partial C_0}{\partial w_k} + \lambda \, sign(w_k)$

- Actualización de los pesos

$$w_k = w_k - \alpha \frac{\partial C_0}{\partial w_k} - \lambda \, sign(w_k)$$

L1 vs L2

Regularización L1	Regularización L2
• **Lleva los pesos a 0:** útil para que el modelo ignore características irrelevantes.	• **Mantiene todos los pesos pequeños:** pero no los hace exactamente 0.
• **Selección automática de características:** ideal para datos con muchas variables donde solo unas pocas son relevantes.	• **Mejor generalización:** útil cuando se espera que todas las características sean relevantes.
• **Aplicaciones:** Modelos de alta dimensionalidad (por ejemplo, compresión de modelos, selección de características).	• **Aplicaciones:** Modelos profundos, evitar sobreajuste en redes neuronales complejas.

Keras.regularizers

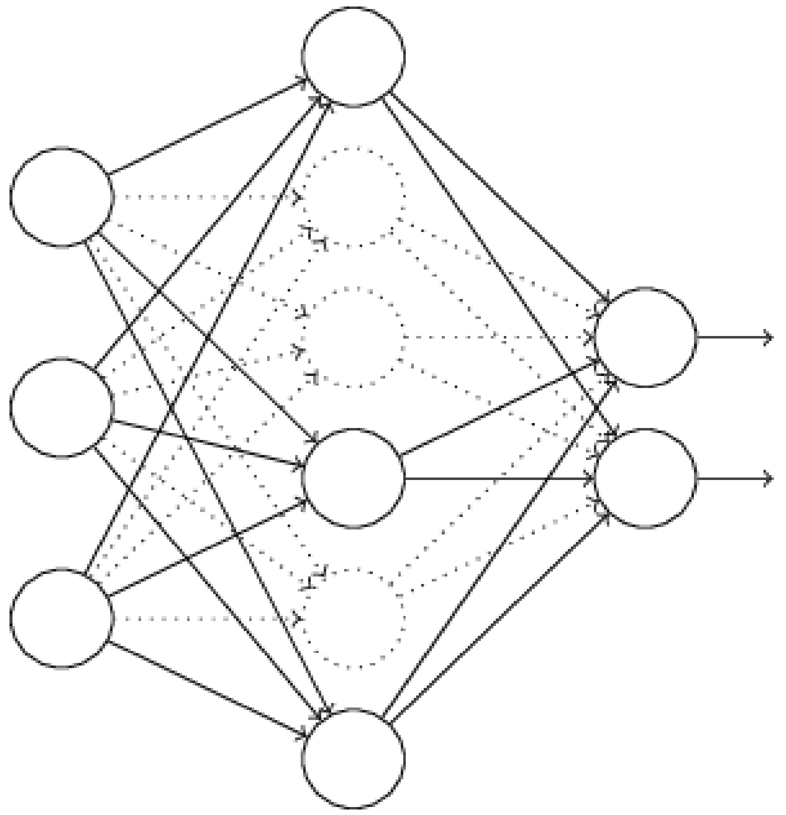
```
from keras.layers import Dense
from keras.regularizers import l2, l1, l1_l2
...
model.add(Dense(32, kernel_regularizer=l2(0.01),
                    bias_regularizer=l2(0.01)))
```

- Se pueden aplicar ambos

```
model.add(Dense(32,kernel_regularizer=l1_l2(l1=0.01,l2=0.01),
                    bias_regularizer=l1_l2(l1=0.01,l2=0.01)))
```

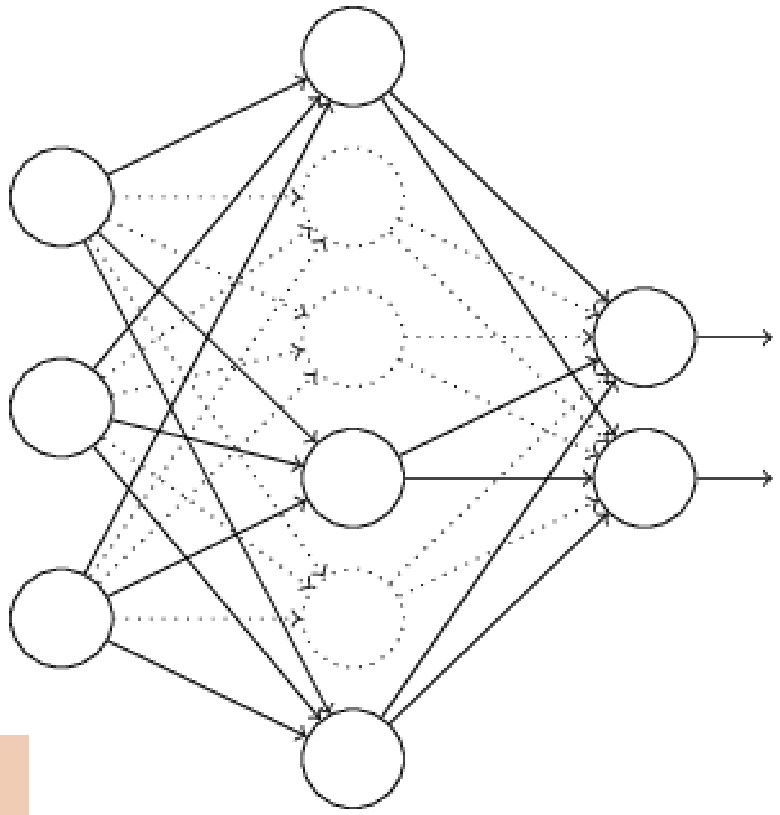
Sobreajuste - Dropout

- No modifica la función de costo sino la arquitectura de la de la red.
- Proceso
  - Selecciona aleatoriamente las neuronas que no participarán en la próxima iteración y las "borra" temporalmente.
  - Actualiza los pesos (del mini lote si corresponde).
  - Restaura las neuronas "borradas".
  - Repite hasta que se estabilice.



Keras dropout

```
from keras.layers import Dense
from keras.layers import Dropout
…
model.add(Dense(6, input_shape=[3]))
model.add(Dropout(0.5))
model.add(Dense(2))
```



Probabilidad de anular cada entrada de la capa anterior
En este caso el 50% de las entradas serán anuladas

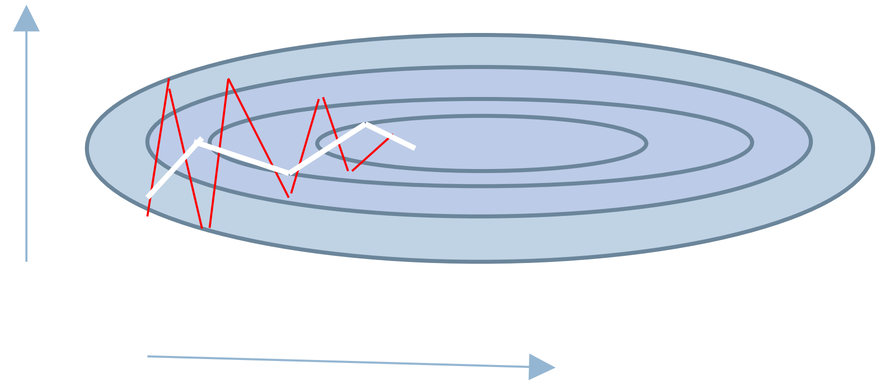
Técnicas de optimización

- Descenso de gradiente estocástico (SGD)
- Capacidad de generalización de la red - Sobreajuste
- Mejoras introducidas
  - <u>Momento</u>: utiliza información de los gradientes anteriores
  - <u>RMSProp</u>: considera distintas magnitudes de cambio para reducir oscilaciones
  - <u>Adam</u>: combina los dos anteriores. Es el más usado.

SGD con momento

$$v_t = \quad \beta v_{t-1} + (1-\beta)(\nabla C)_t$$

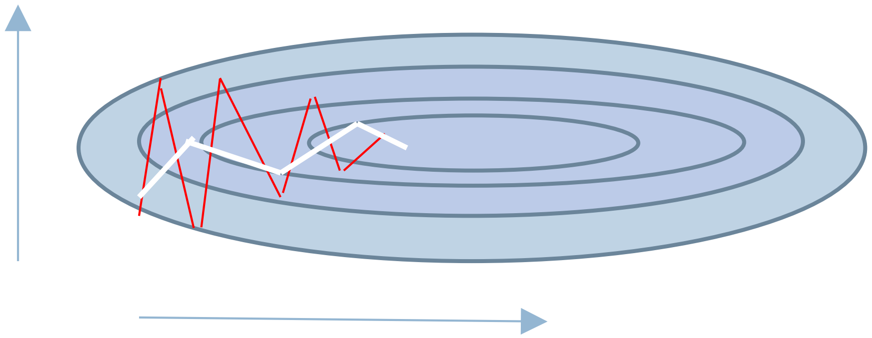
$$w_t = w_t - \alpha\, v_t$$



- Las modificaciones sobre W tienen en cuenta el promedio de los gradientes anteriores.
- La cantidad de gradientes anteriores a considerar son aprox. $\frac{1}{1-\beta}$
- Esto reduce las oscilaciones.

SGD con momento

$$v_t = \quad \beta v_{t-1} + (1-\beta)(\nabla C)_t$$

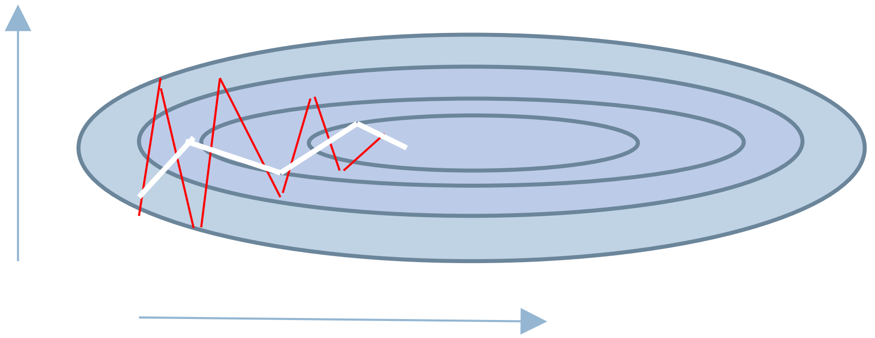


- Usemos $\beta = 0.9$ en la iteración $t = 10$

$$v_{10} = \quad 0.9 * v_9 + (1-0.9)(\nabla C)_{10}$$

SGD con momento

$$v_t = \quad \beta v_{t-1} + (1-\beta)(\nabla C)_t$$

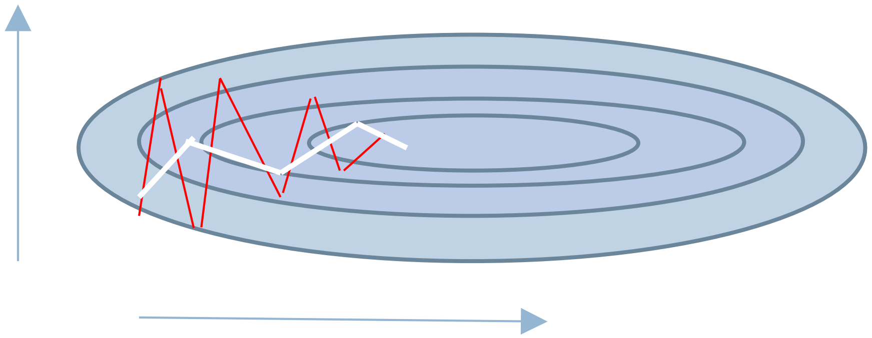


- Usemos $\beta = 0.9$ en la iteración $t = 10$

$$v_{10} = \quad 0.9 * v_9 + (1-0.9)(\nabla C)_{10} = 0.1\, \nabla C_{10} + 0.9\, v_9$$

SGD con momento

$$v_t = \quad \beta v_{t-1} + (1-\beta)(\nabla C)_t$$



- Usemos $\beta = 0.9$ en la iteración $t = 10$

$$v_{10} = \quad 0.9 * v_9 + (1-0.9)(\nabla C)_{10} = 0.1\, \nabla C_{10} + 0.9\, v_9$$

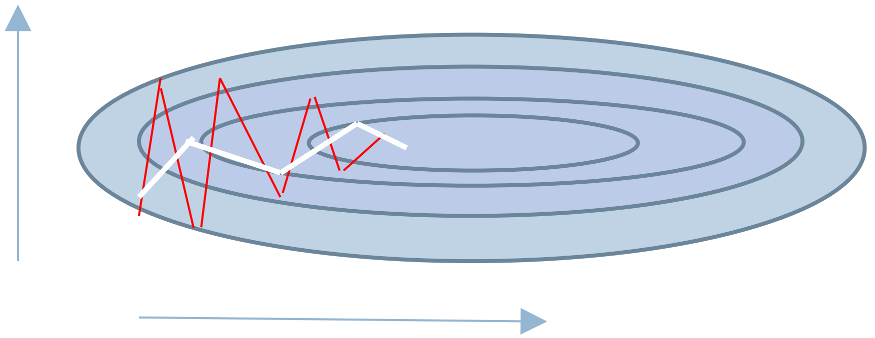
$$v_{10} = \quad 0.1\, \nabla C_{10} + 0.9\, (0.1\, \nabla C_9 + 0.9\, v_8)$$

$$v_{10} = \quad 0.1\, \nabla C_{10} + 0.1 * 0.9\ \ \nabla C_9 + 0.9^2 v_8$$

$$v_{10} = \quad 0.1\, \nabla C_{10} + 0.1 * 0.9\ \ \nabla C_9 + 0.9^2 (0.9\, v_7 + 0.1\, \nabla C_8)$$

SGD con momento

$$v_t = \quad \beta v_{t-1} + (1-\beta)(\nabla C)_t$$



- Usemos $\beta = 0.9$ en la iteración $t = 10$

$$v_{10} = \quad 0.9 * v_9 + (1-0.9)(\nabla C)_{10} = 0.1\, \nabla C_{10} + 0.9\, v_9$$

$$v_{10} = \quad 0.1\, \nabla C_{10} + 0.9\, (0.1\, \nabla C_9 + 0.9\, v_8)$$

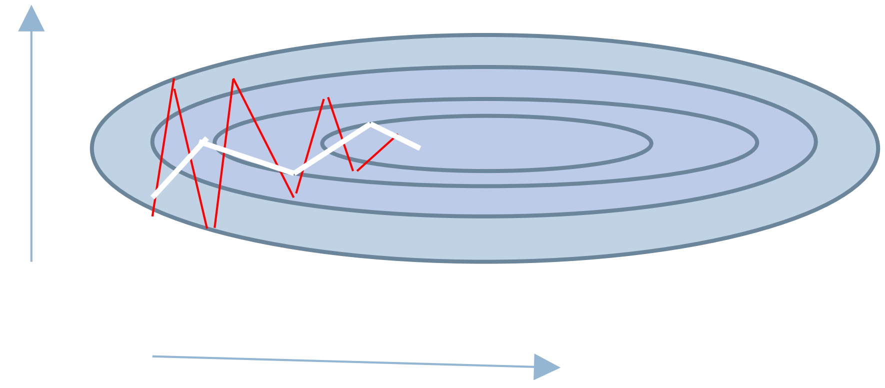
$$v_{10} = \quad 0.1\, \nabla C_{10} + 0.1 * 0.9 \;\; \nabla C_9 + 0.9^2 v_8$$

$$v_{10} = \quad 0.1\, \nabla C_{10} + 0.1 * 0.9 \;\; \nabla C_9 + 0.1 * 0.9^2 \;\; \nabla C_8 + 0.9^3 v_7 + \cdots$$

La cantidad de gradientes anteriores a considerar son aprox. $\frac{1}{1-\beta}$ $\therefore$ si β=0.9 serán aprox. 10

SGD con momento

```
Vw = 0
Vb = 0
for t in range(iteraciones):
    Calcular gradientes ∇w y ∇b
    Vw = beta * Vw + (1-beta)* ∇w
    Vb = beta * Vb + (1-beta)* ∇b
    W = W - alfa * Vw
    b = b - alfa * Vb
```

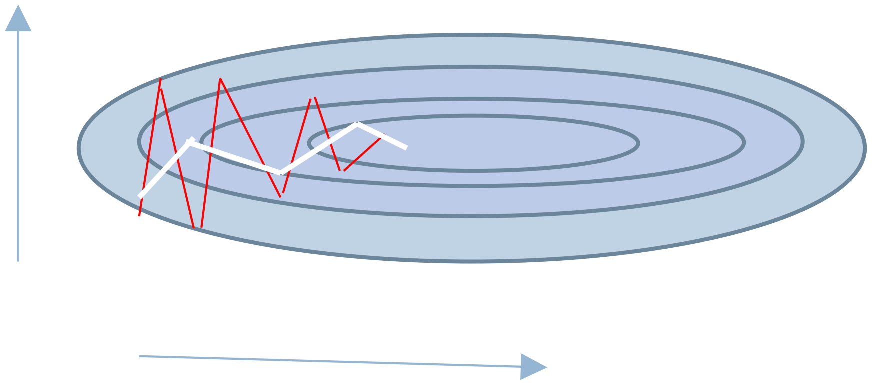


keras.optimizers.SGD(learning_rate=0.01, momentum=0.9)

RMSprop

$$s = \beta\, s + (1-\beta)\,(\nabla C)^2$$

$$w = w - \alpha \frac{\nabla C}{\sqrt{s+\varepsilon}}$$



- Las modificaciones sobre W tienen en cuenta el promedio de los gradientes anteriores.
- Las modificaciones más grandes serán divididas por coeficientes más grandes; por lo tanto se reducen.
- Las modificaciones más chicas se incrementan.

Es más eficiente que SGD+Momento

RMSprop

```
from keras.optimizers import RMSprop

X,Y = cargar_datos()

model = Sequential()
model.add(…)

model.compile(
            loss='categorical_crossentropy',
            optimizer = RMSprop(learning_rate=0.001),
            metrics=['accuracy'])

model.fit(X,Y, epochs=10, batch_size=32)
```

ADAM

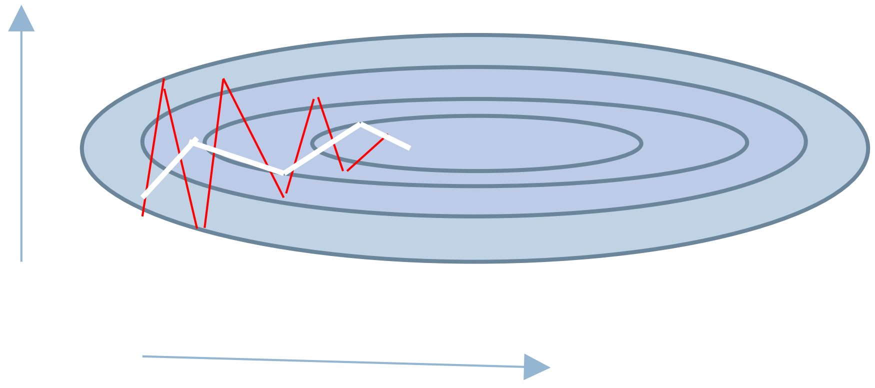
https://keras.io/api/optimizers/adam/

- Combina momento y RMSprop

$$v = \beta_1 v + (1 - \beta_1) \nabla C$$

$$s = \beta_2 s + (1 - \beta_2) (\nabla C)^2$$

$$w = w - \alpha \frac{v}{\sqrt{s + \varepsilon}}$$



- Los valores recomendados son $\beta_1 = 0.9$ y $\beta_2 = 0.999$

```
model.compile(optimizer='adam', loss='mse')
```

Resumen

Resolución de una tarea de clasificación

- Conjunto de datos etiquetados (aprendizaje supervisado)
- Definición de la arquitectura de la red
  - Número de capas y tamaño de cada una
  - Función de activación a usar en cada capa
- Entrenamiento
  - Función de costo
  - Técnica de optimización para reducir el error
- Evaluar el modelo

Evaluación del modelo

- Matriz de confusión
- Métricas
  - Accuracy
  - Precisión
  - Recall
  - F1-score
  - AUC, Curva ROC

Clasificación binaria

- Los resultados se etiquetan como positivos (P) o negativos (N)
- Luego, la matriz de confusión tendrá la siguiente forma:

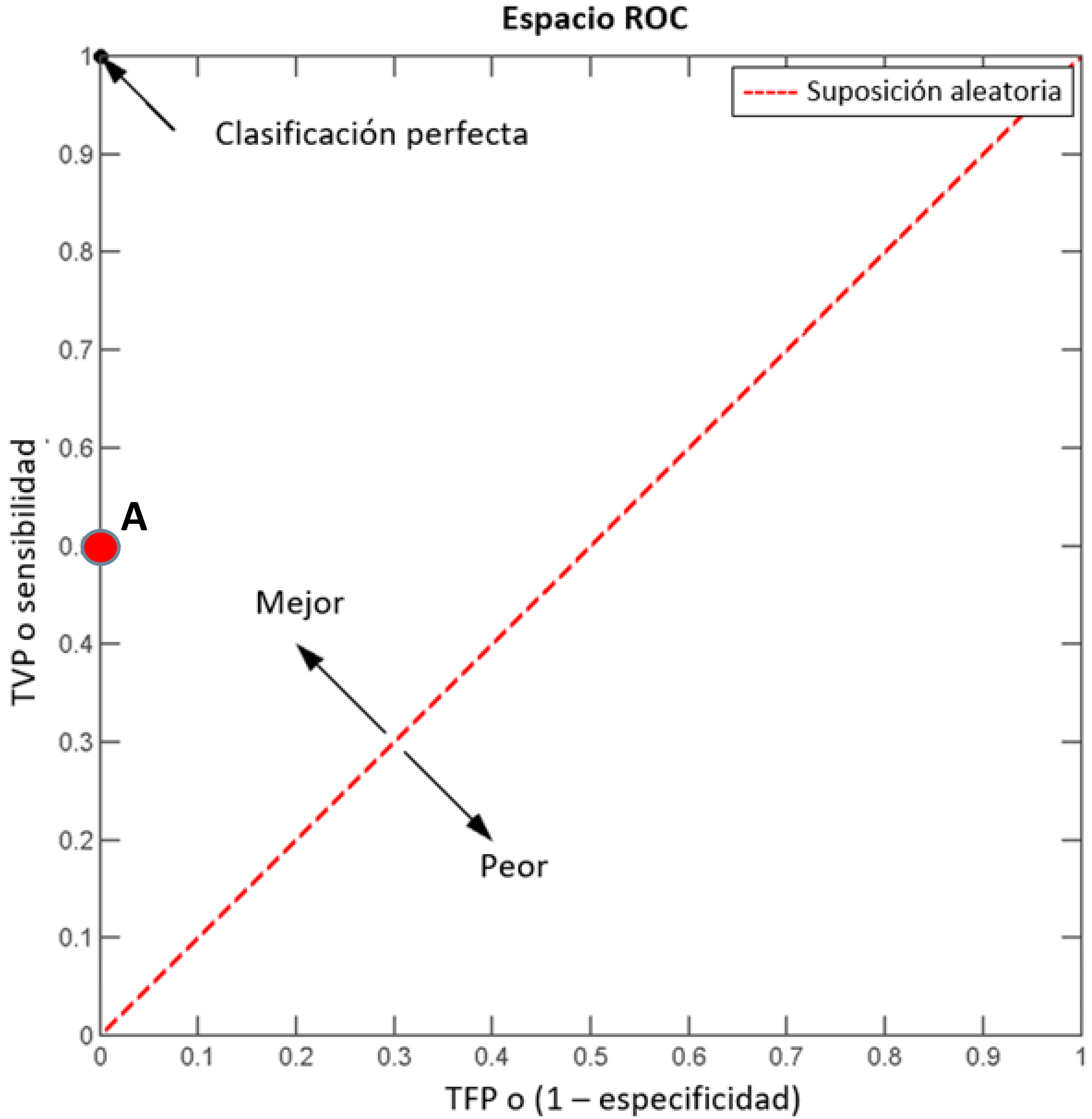
	Predice P	Predice N	
Clase P	VP	FN	P = VP + FN
Clase N	FP	VN	N = FP+VN

  - Tasa de verdaderos positivos → TVP = VP / P (Sensibilidad)
  - Tasa de Falsos Positivos → TFP = FP / N (Falsas alarmas)
  - Tasa de verdaderos negativos → TVN = VN / N (Especificidad)

A

VP=6	FN=6	12
FP=0	VN=8	8
6	14	20

TVP = 6/12=0.5
TFP = 0/8 = 0



A

VP=6	FN=6	12
FP=0	VN=8	8
6	14	20

TVP = 6/12=0.5
TFP = 0/8 = 0

B

VP=9	FN=3	12
FP=6	VN=2	8
15	5	20

TVP = 9/12 = 0.75
TFP = 6/8 = 0.75

C

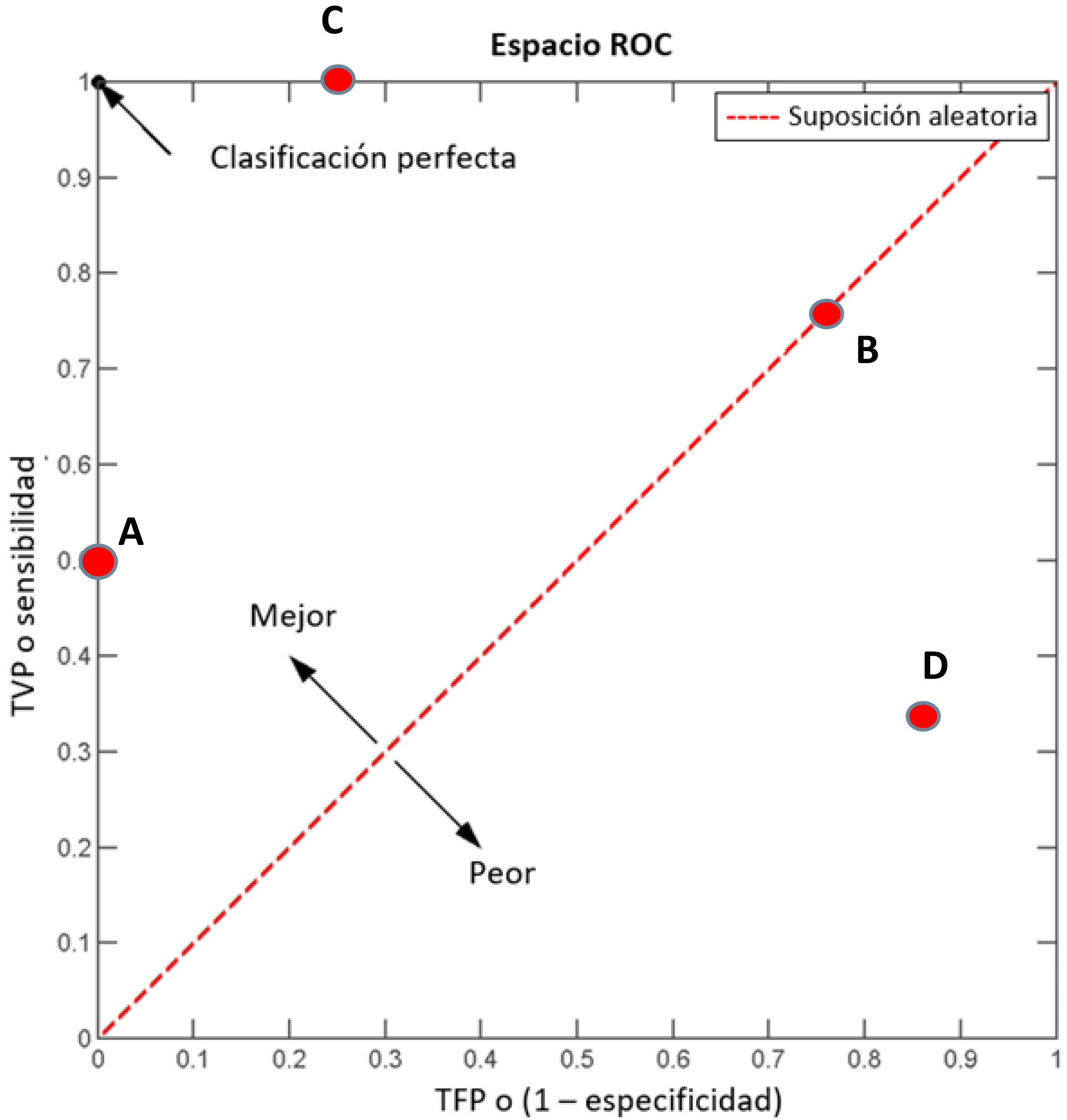
VP=12	FN=0	12
FP=2	VN=6	8
14	6	20

TVP = 12/12 = 1
TFP = 2/8 = 0.25

D

VP=4	FN=8	12
FP=7	VN=1	8
11	9	20

TVP = 4/12 = 0.33
TFP = 7/8 = 0.875



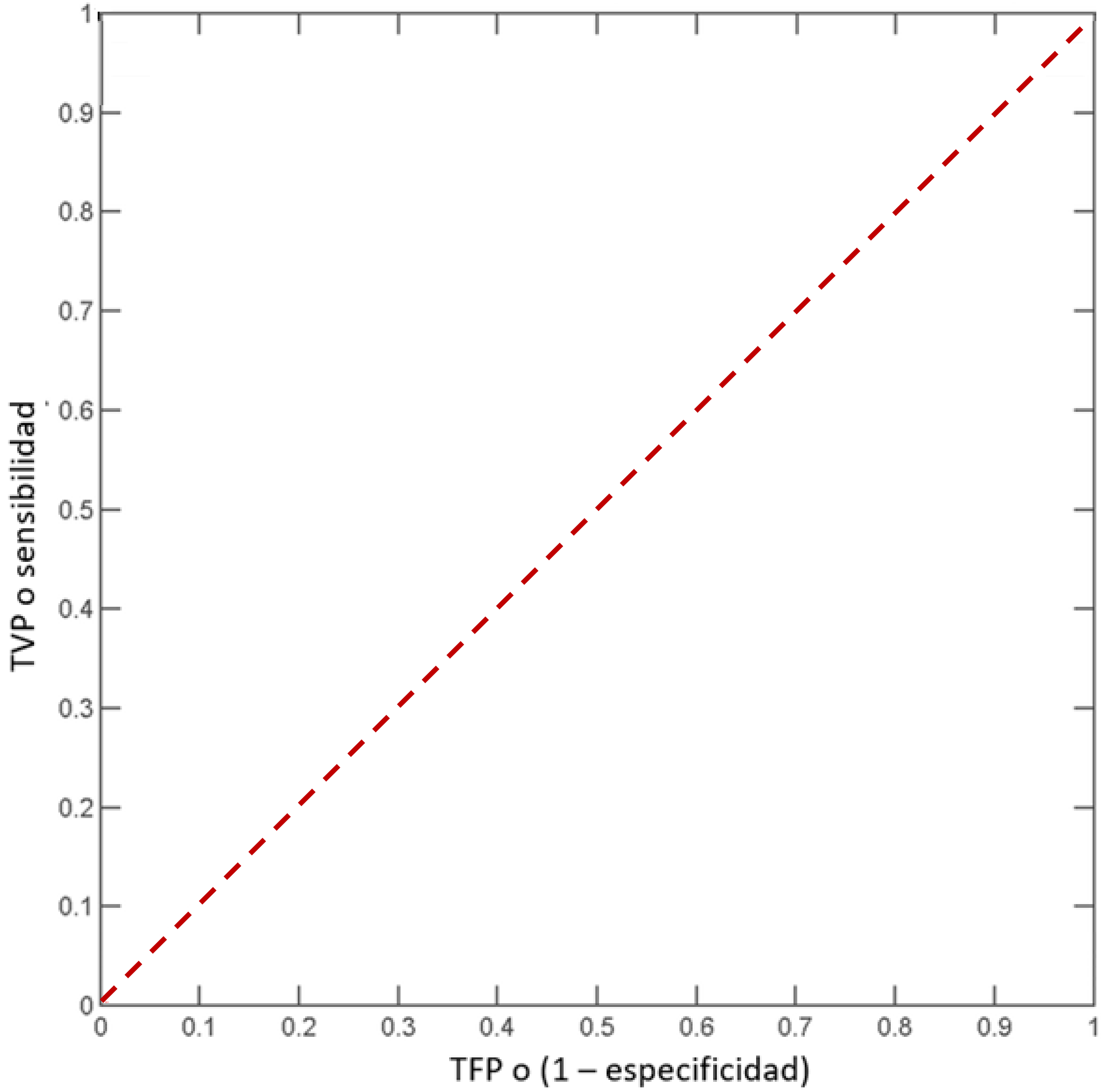
Roca o Mina

- A partir de los datos del archivo “Sonar.csv” se desea construir una red neurona multiperceptrón para discriminar entre señales de sonar rebotadas en un cilindro de metal (“Mine”) y aquellas rebotadas en una roca más o menos cilíndrica (“Rock”).
- Probar con distintas configuraciones
- Indicar cuál recomendaría a la hora de predecir si es una mina o no utilizando: accuracy, f1-score y AUC.

ID	Clase	Confianza	Predice
5	Mina	0.99	
7	Mina	0.99	
9	Mina	0.99	
1	Mina	0.9	
10	Mina	0.9	
20	Mina	0.9	
8	Roca	0.8	
14	Mina	0.8	
15	Mina	0.8	
18	Roca	0.8	
19	Mina	0.8	
3	Mina	0.7	
6	Mina	0.7	
12	Mina	0.65	
4	Roca	0.6	
16	Roca	0.6	
11	Roca	0.5	
2	Roca	0.4	
13	Roca	0.3	
17	Roca	0.1	

12 minas y **8 rocas**

CURVA ROC

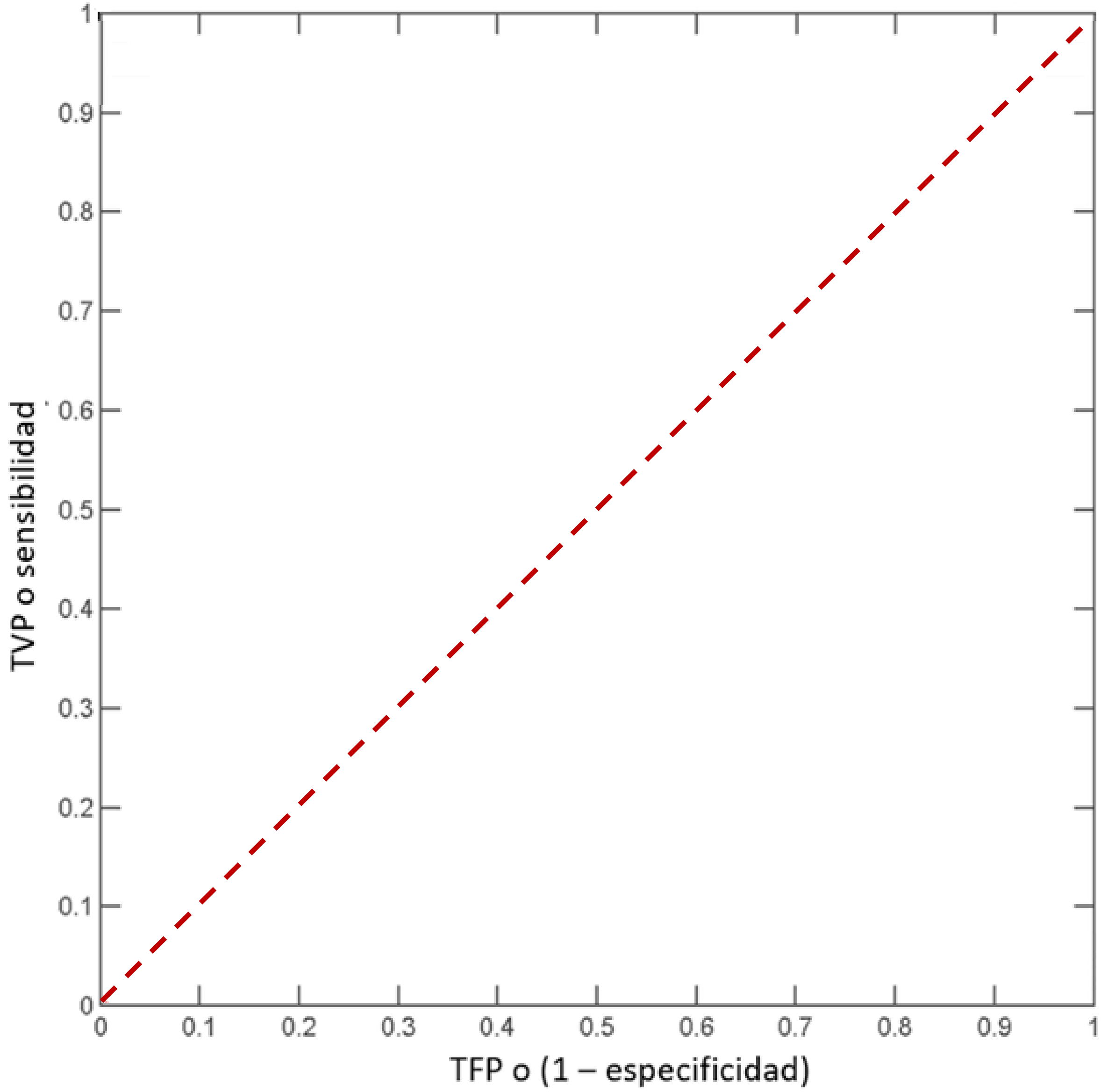


ID	Clase	Confianza	Predice
5	Mina	0.99	
7	Mina	0.99	
9	Mina	0.99	
1	Mina	0.9	
10	Mina	0.9	
20	Mina	0.9	
8	Roca	0.8	
14	Mina	0.8	
15	Mina	0.8	
18	Roca	0.8	
19	Mina	0.8	
3	Mina	0.7	
6	Mina	0.7	
12	Mina	0.65	
4	Roca	0.6	
16	Roca	0.6	
11	Roca	0.5	
2	Roca	0.4	
13	Roca	0.3	
17	Roca	0.1	

12 minas y **8 rocas**

Umbral = 0.99

CURVA ROC



ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Roca
10	Mina	0.9	Roca
20	Mina	0.9	Roca
8	Roca	0.8	Roca
14	Mina	0.8	Roca
15	Mina	0.8	Roca
18	Roca	0.8	Roca
19	Mina	0.8	Roca
3	Mina	0.7	Roca
6	Mina	0.7	Roca
12	Mina	0.65	Roca
4	Roca	0.6	Roca
16	Roca	0.6	Roca
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y **8 rocas**

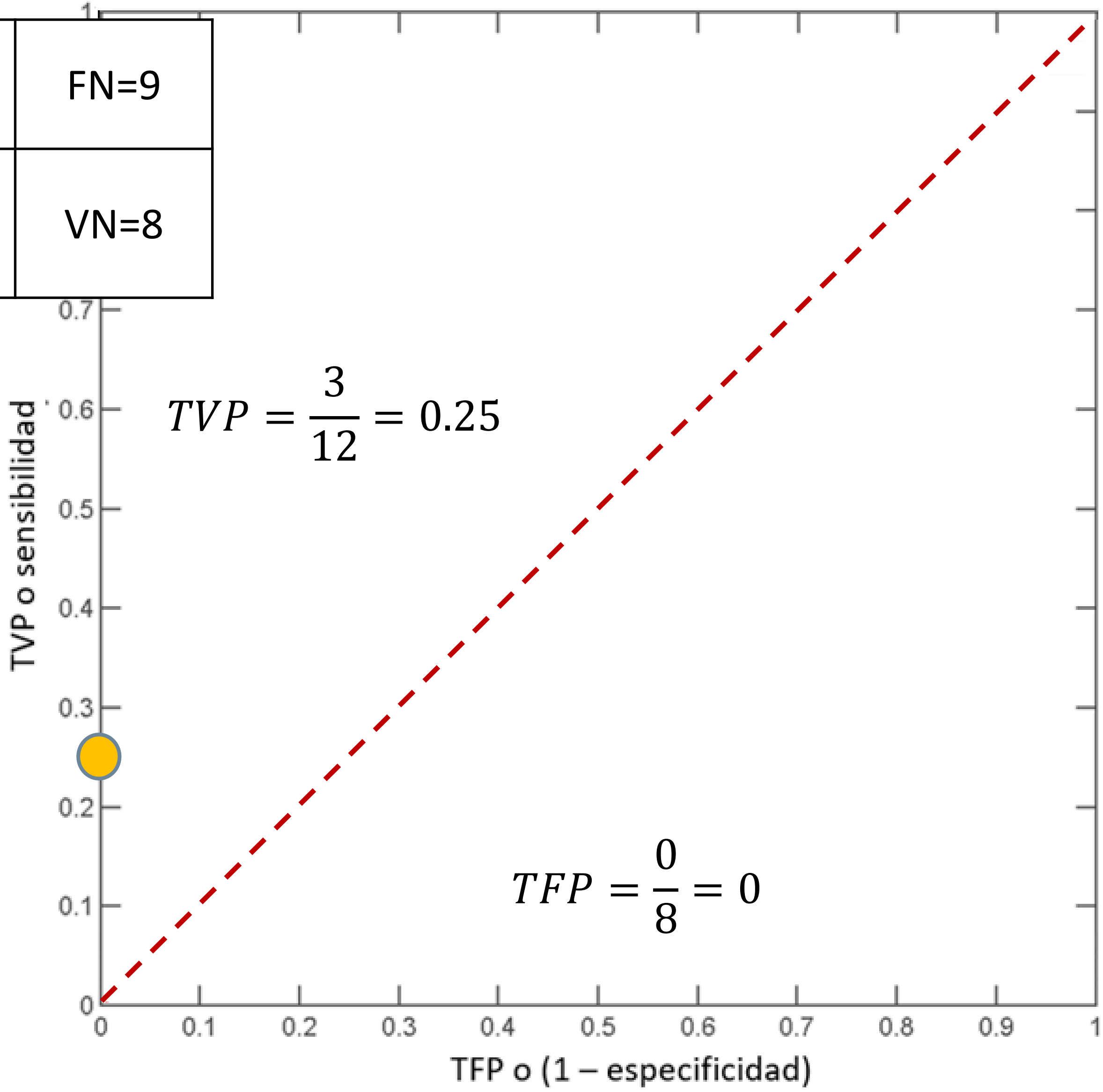
Umbral = 0.99

CURVA ROC

VP=3	FN=9
FP=0	VN=8

$$TVP = \frac{3}{12} = 0.25$$

$$TFP = \frac{0}{8} = 0$$



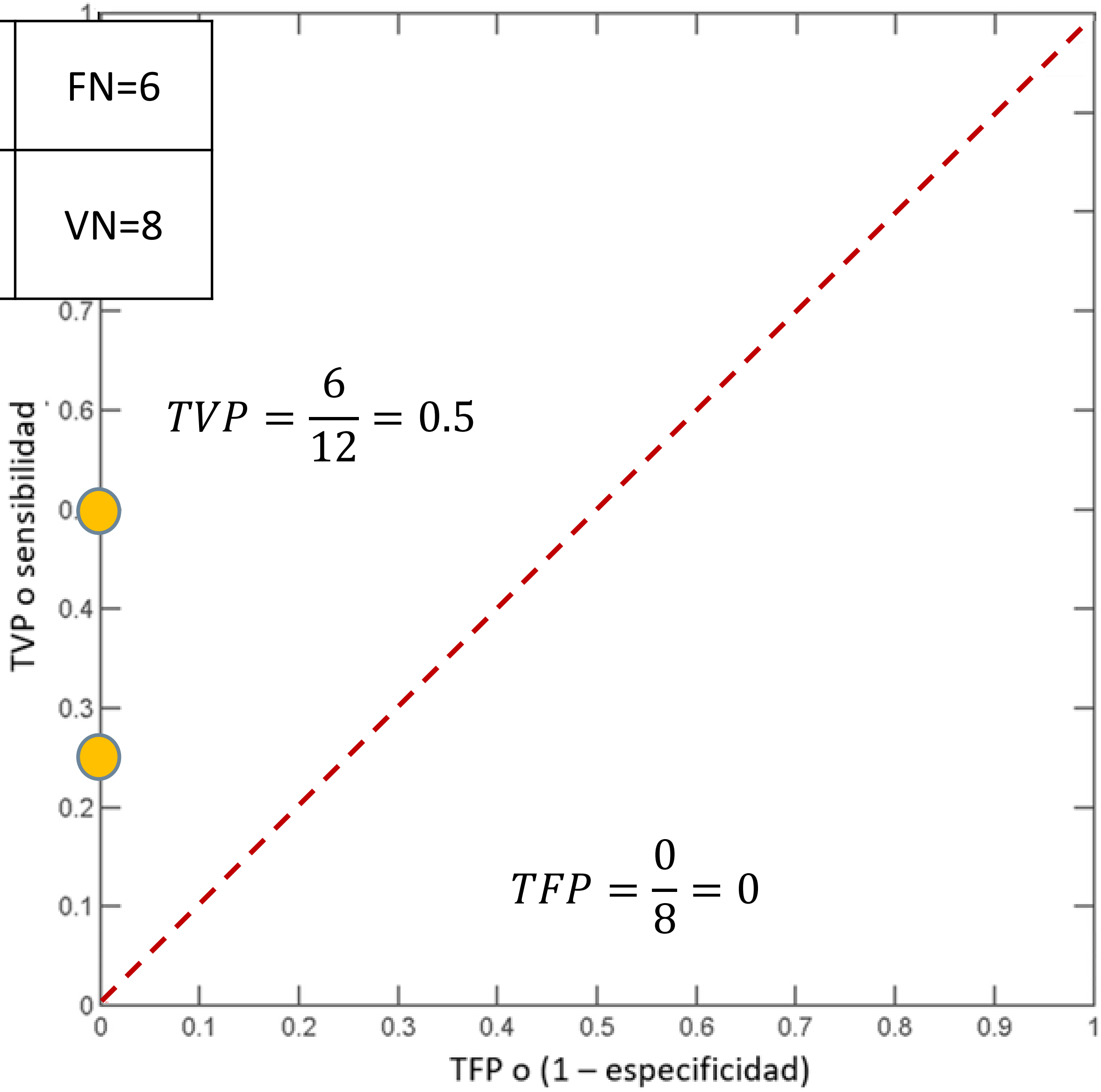
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Roca
14	Mina	0.8	Roca
15	Mina	0.8	Roca
18	Roca	0.8	Roca
19	Mina	0.8	Roca
3	Mina	0.7	Roca
6	Mina	0.7	Roca
12	Mina	0.65	Roca
4	Roca	0.6	Roca
16	Roca	0.6	Roca
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y **8 rocas**

Umbral = 0.9

VP=6	FN=6
FP=0	VN=8

CURVA ROC



$$TVP = \frac{6}{12} = 0.5$$

$$TFP = \frac{0}{8} = 0$$

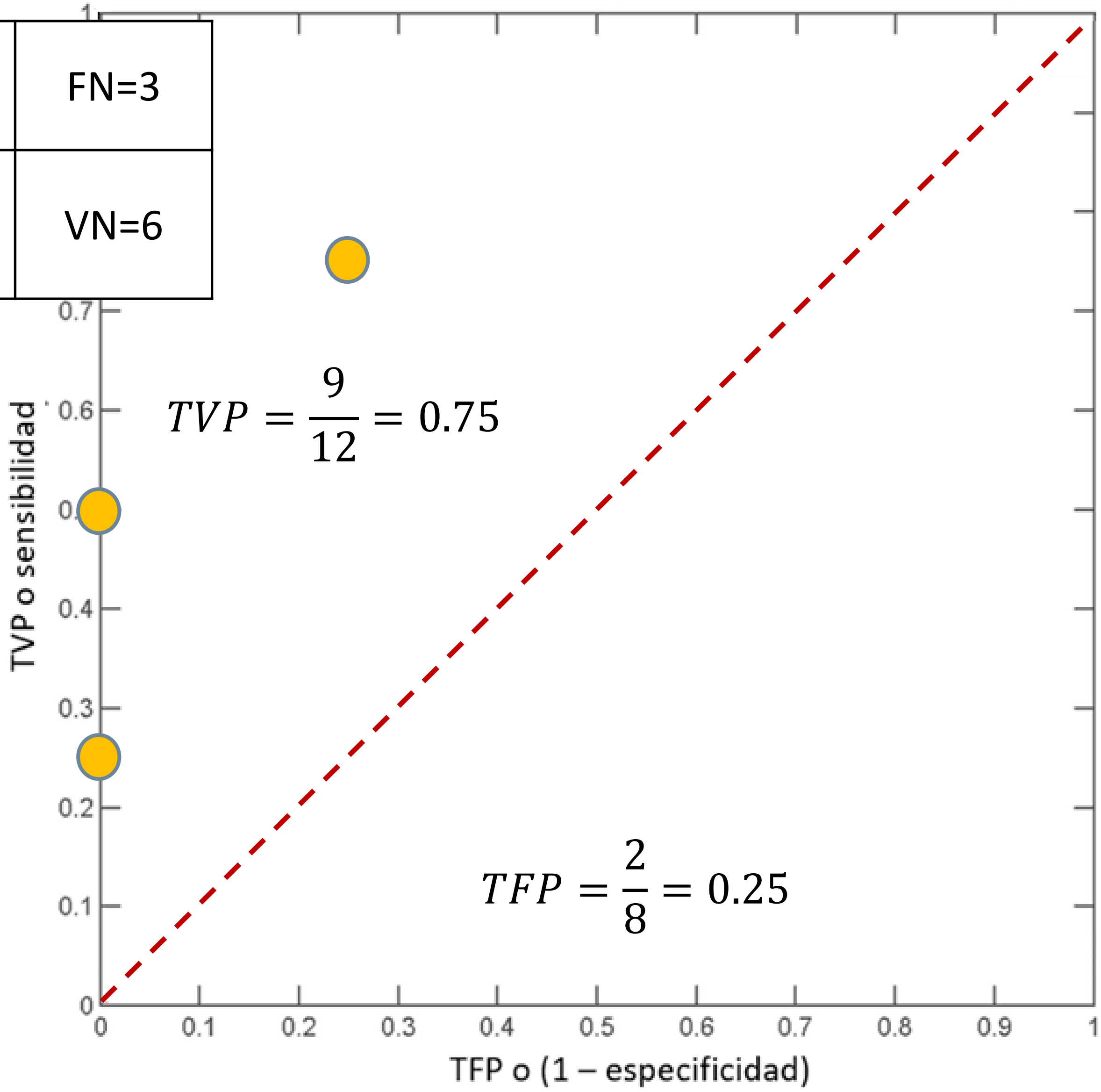
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Roca
6	Mina	0.7	Roca
12	Mina	0.65	Roca
4	Roca	0.6	Roca
16	Roca	0.6	Roca
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y **8 rocas**

Umbral = 0.8

VP=9	FN=3
FP=2	VN=6

CURVA ROC



$$TVP = \frac{9}{12} = 0.75$$

$$TFP = \frac{2}{8} = 0.25$$

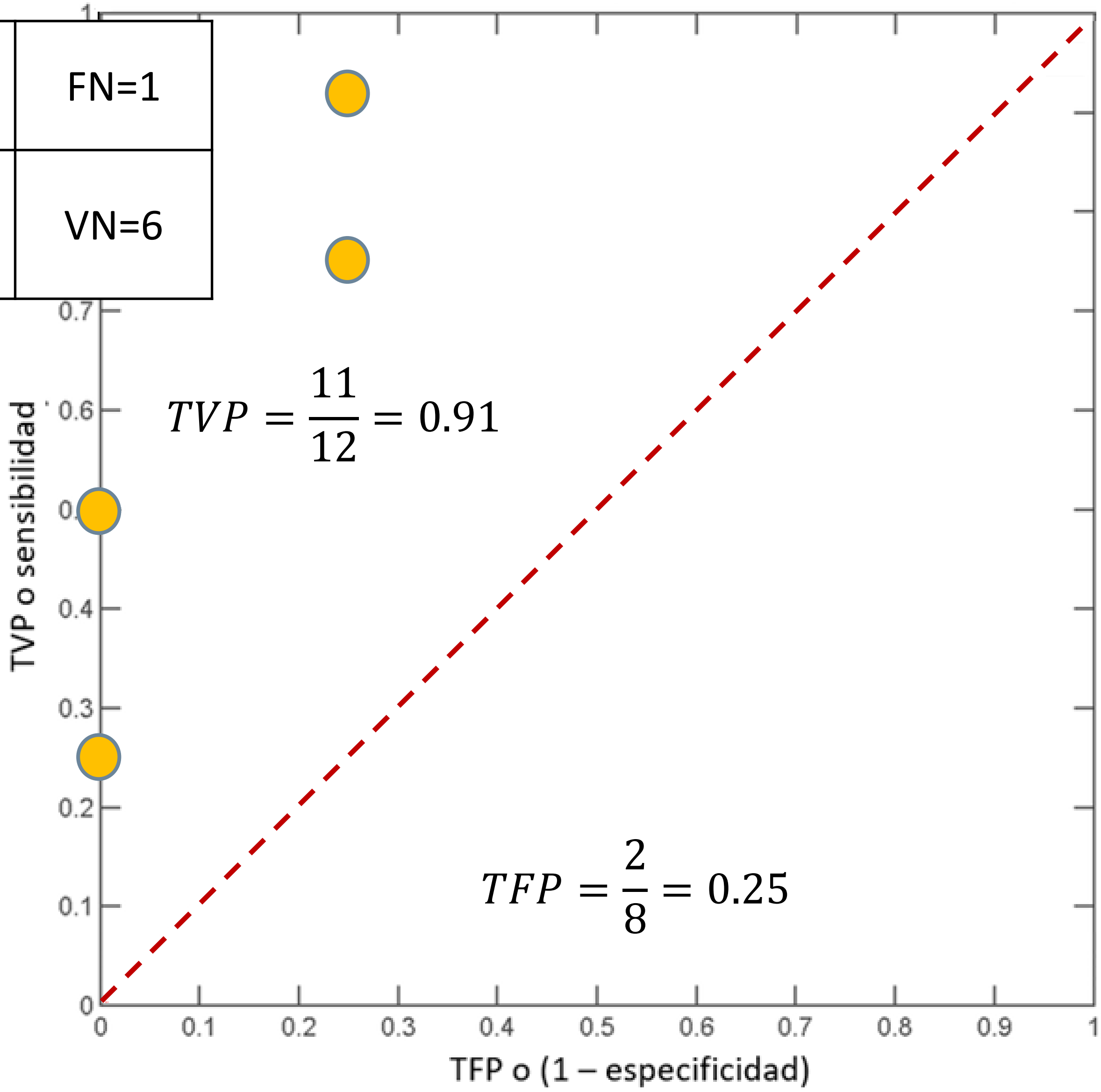
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Roca
4	Roca	0.6	Roca
16	Roca	0.6	Roca
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y **8 rocas**

Umbral = 0.7

VP=11	FN=1
FP=2	VN=6

CURVA ROC



$$TVP = \frac{11}{12} = 0.91$$

$$TFP = \frac{2}{8} = 0.25$$

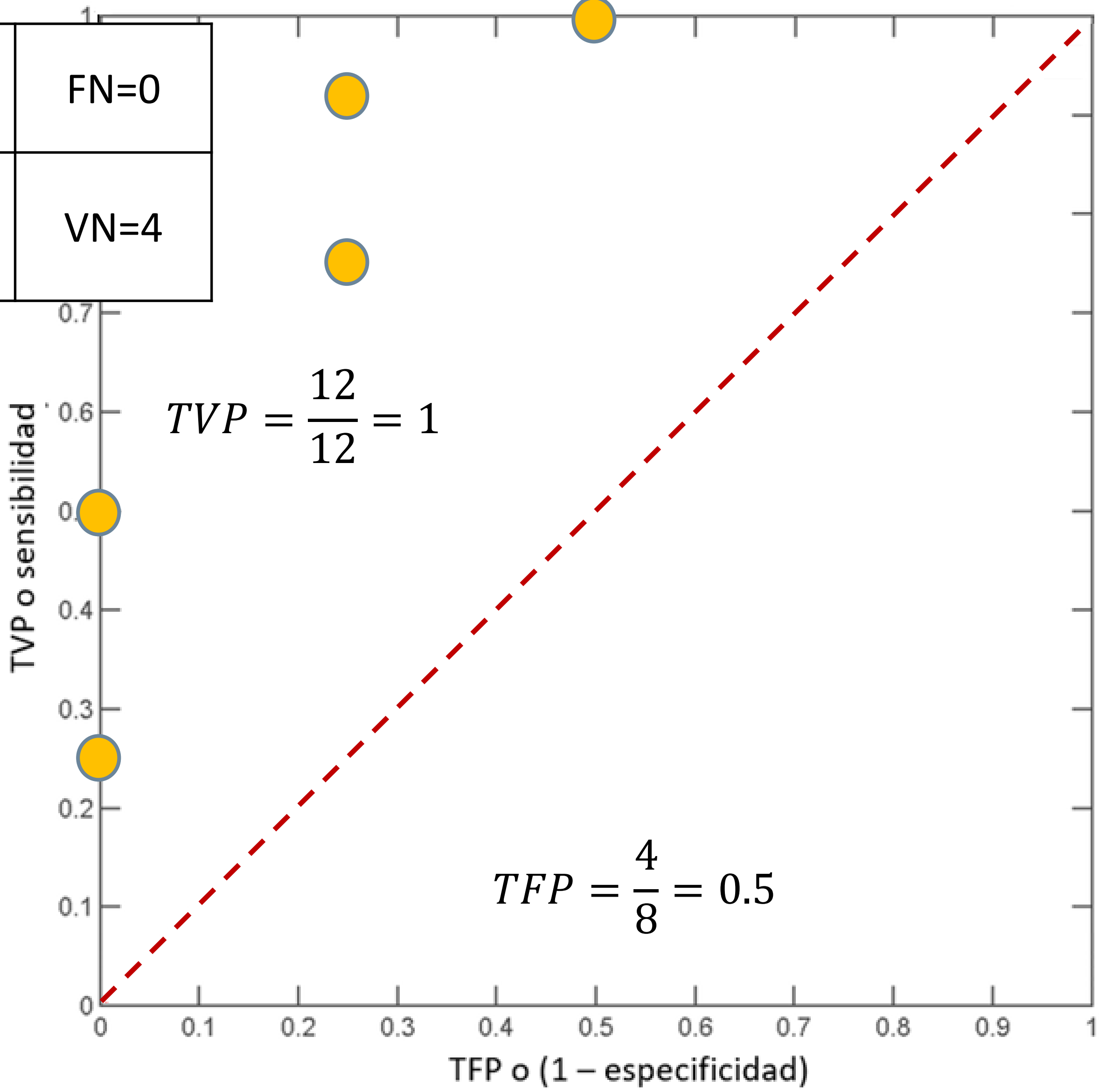
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y **8 rocas**

Umbral = 0.6

CURVA ROC

VP=12	FN=0
FP=4	VN=4



$$TVP = \frac{12}{12} = 1$$

$$TFP = \frac{4}{8} = 0.5$$

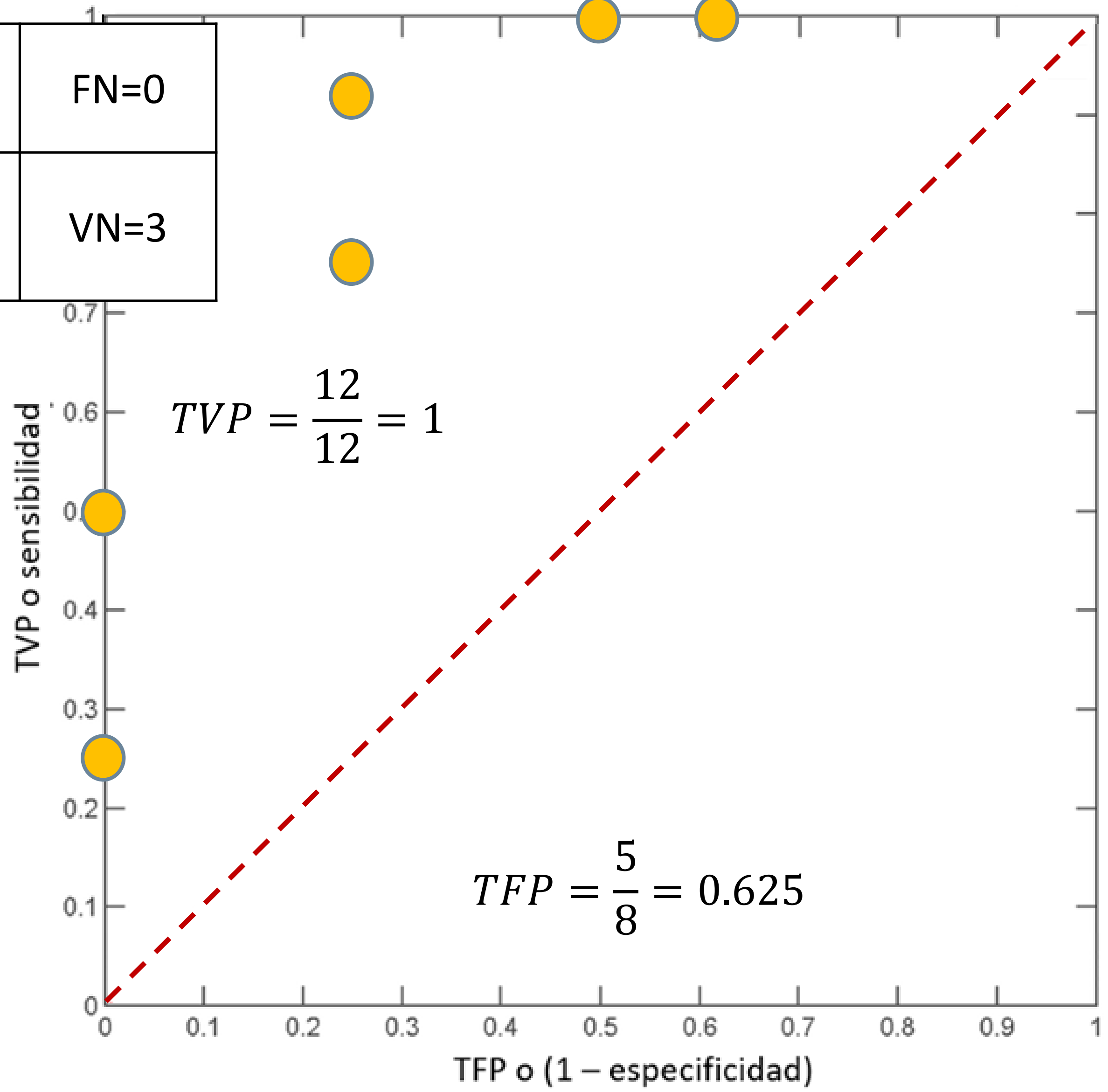
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y **8 rocas**

Umbral = 0.5

CURVA ROC

VP=12	FN=0
FP=5	VN=3



$$TVP = \frac{12}{12} = 1$$

$$TFP = \frac{5}{8} = 0.625$$

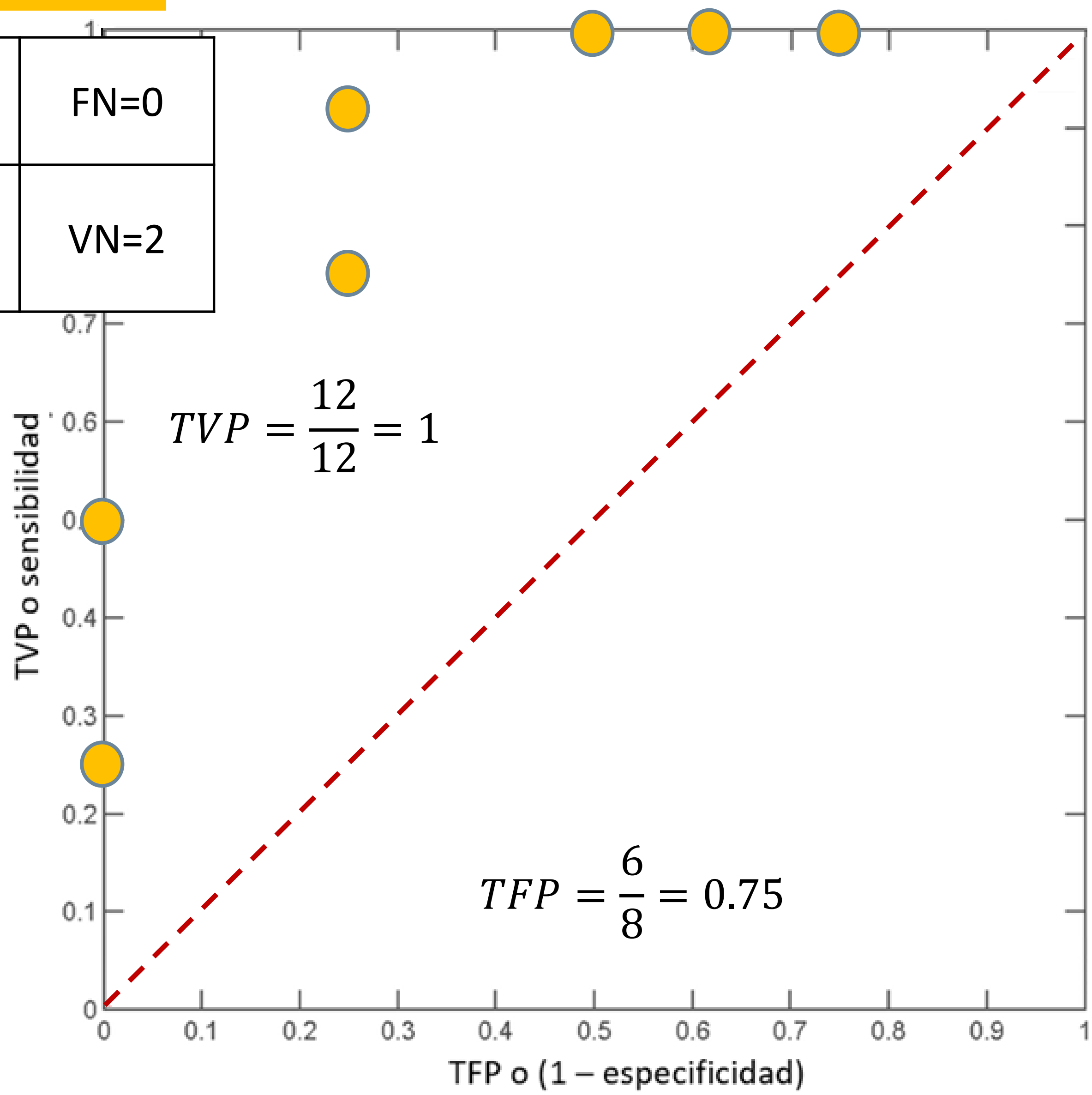
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Mina
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y 8 rocas

Umbral = 0.4

CURVA ROC

VP=12	FN=0
FP=6	VN=2



$$TVP = \frac{12}{12} = 1$$

$$TFP = \frac{6}{8} = 0.75$$

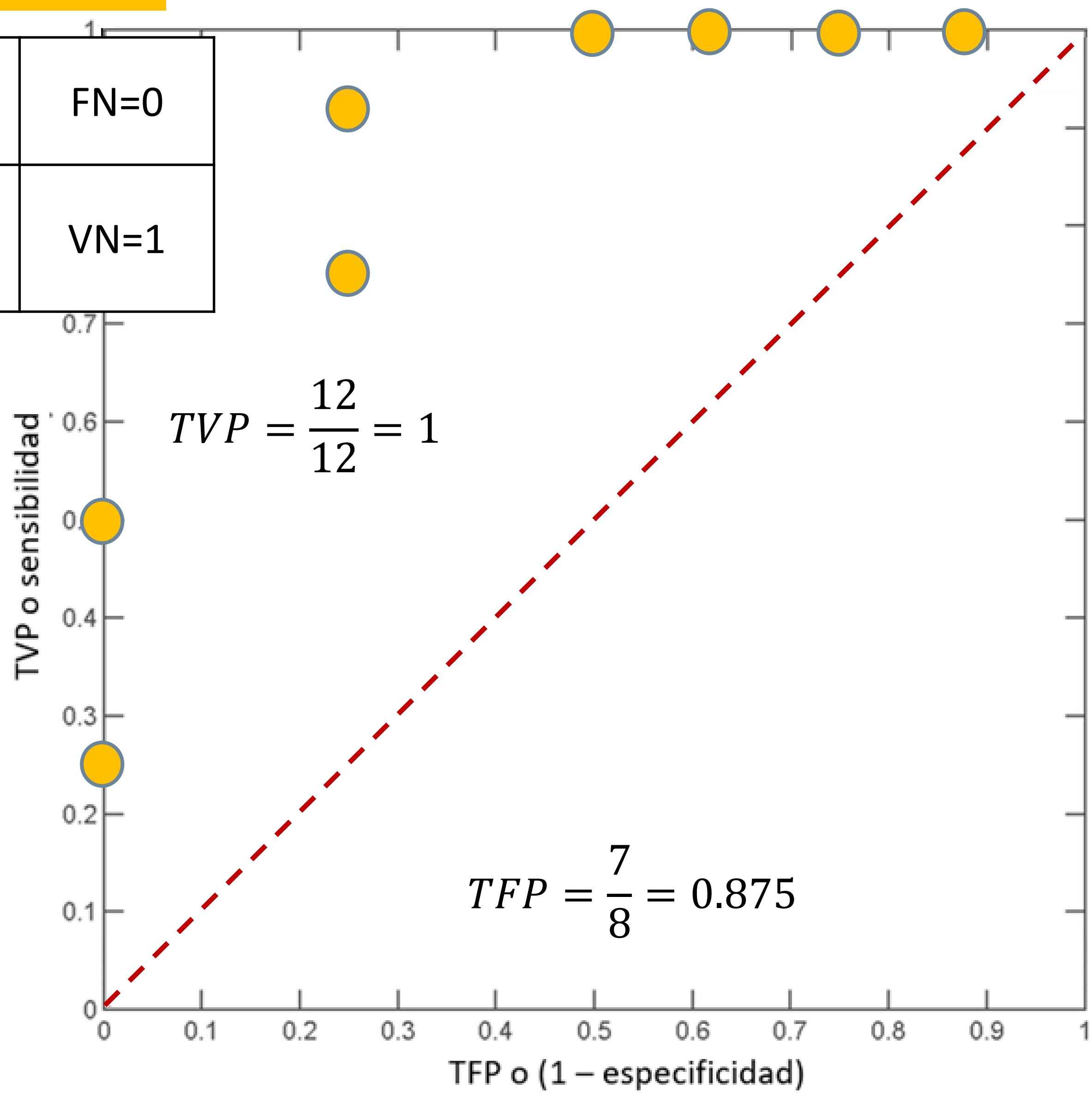
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Mina
13	Roca	0.3	Mina
17	Roca	0.1	Roca

12 minas y 8 rocas

Umbral = 0.3

CURVA ROC

VP=12	FN=0
FP=7	VN=1



$$TVP = \frac{12}{12} = 1$$

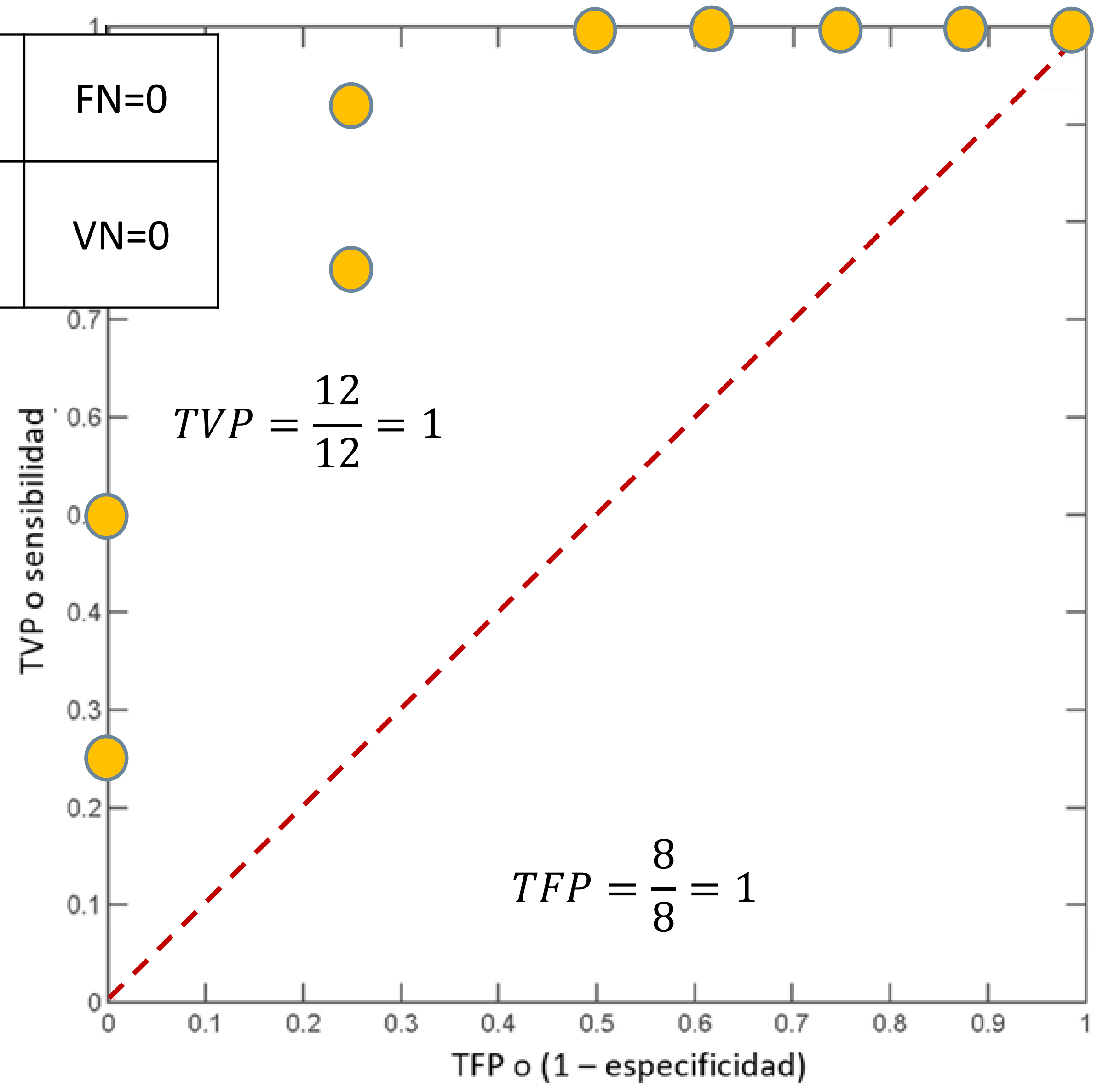
$$TFP = \frac{7}{8} = 0.875$$

ID	Clase	Confianza	Predice
5	**Mina**	0.99	**Mina**
7	**Mina**	0.99	**Mina**
9	**Mina**	0.99	**Mina**
1	**Mina**	0.9	**Mina**
10	**Mina**	0.9	**Mina**
20	**Mina**	0.9	**Mina**
8	**Roca**	0.8	**Mina**
14	**Mina**	0.8	**Mina**
15	**Mina**	0.8	**Mina**
18	**Roca**	0.8	**Mina**
19	**Mina**	0.8	**Mina**
3	**Mina**	0.7	**Mina**
6	**Mina**	0.7	**Mina**
12	**Mina**	0.65	**Mina**
4	**Roca**	0.6	**Mina**
16	**Roca**	0.6	**Mina**
11	**Roca**	0.5	**Mina**
2	**Roca**	0.4	**Mina**
13	**Roca**	0.3	**Mina**
17	**Roca**	0.1	**Mina**

12 minas y **8 rocas**

CURVA ROC

VP=12	FN=0
FP=8	VN=0

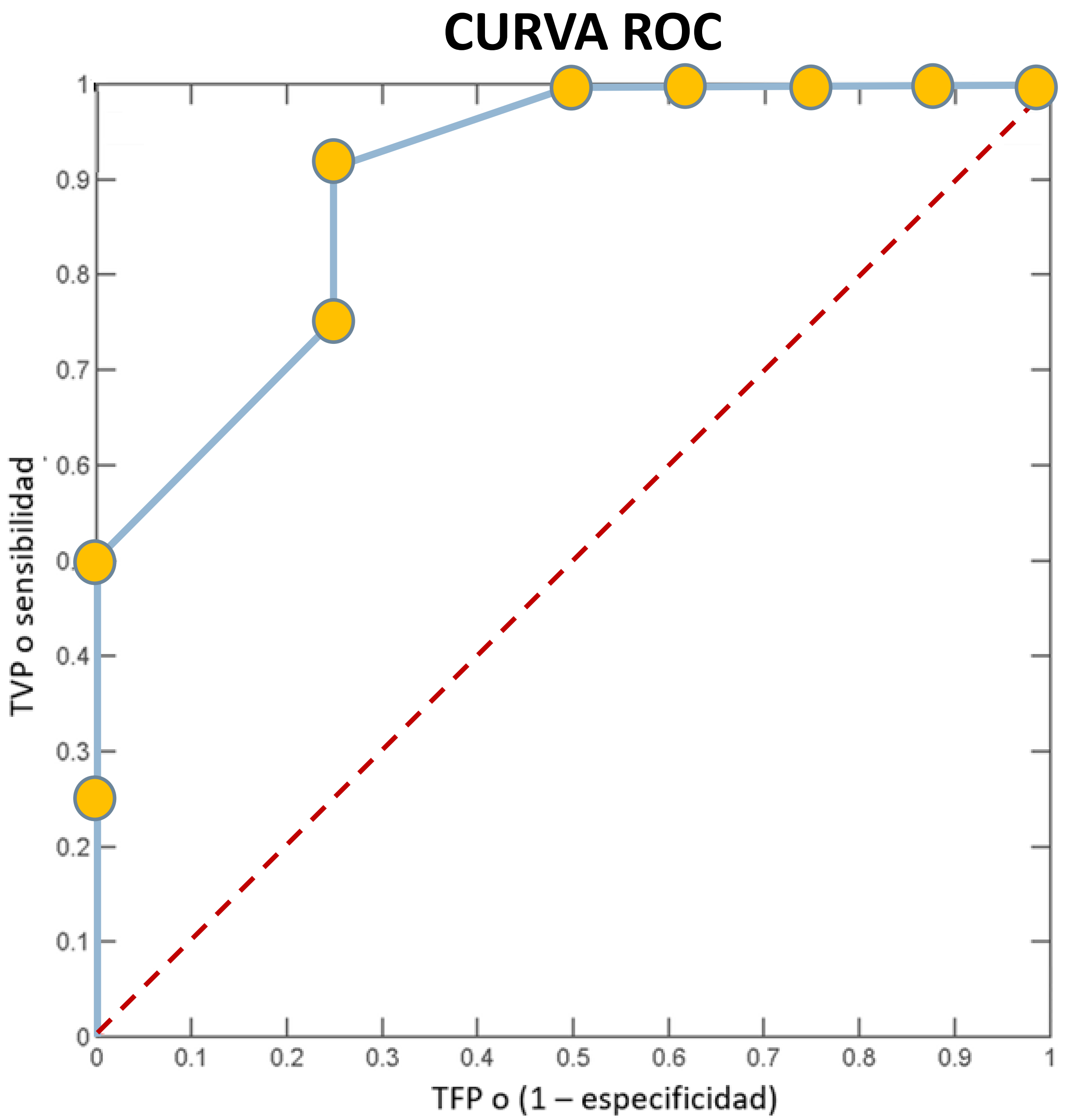


$$TVP = \frac{12}{12} = 1$$

$$TFP = \frac{8}{8} = 1$$

ID	Clase	Confianza	Predice
5	**Mina**	0.99	
7	**Mina**	0.99	
9	**Mina**	0.99	
1	**Mina**	0.9	
10	**Mina**	0.9	
20	**Mina**	0.9	
8	**Roca**	0.8	
14	**Mina**	0.8	
15	**Mina**	0.8	
18	**Roca**	0.8	
19	**Mina**	0.8	
3	**Mina**	0.7	
6	**Mina**	0.7	
12	**Mina**	0.65	
4	**Roca**	0.6	
16	**Roca**	0.6	
11	**Roca**	0.5	
2	**Roca**	0.4	
13	**Roca**	0.3	
17	**Roca**	0.1	

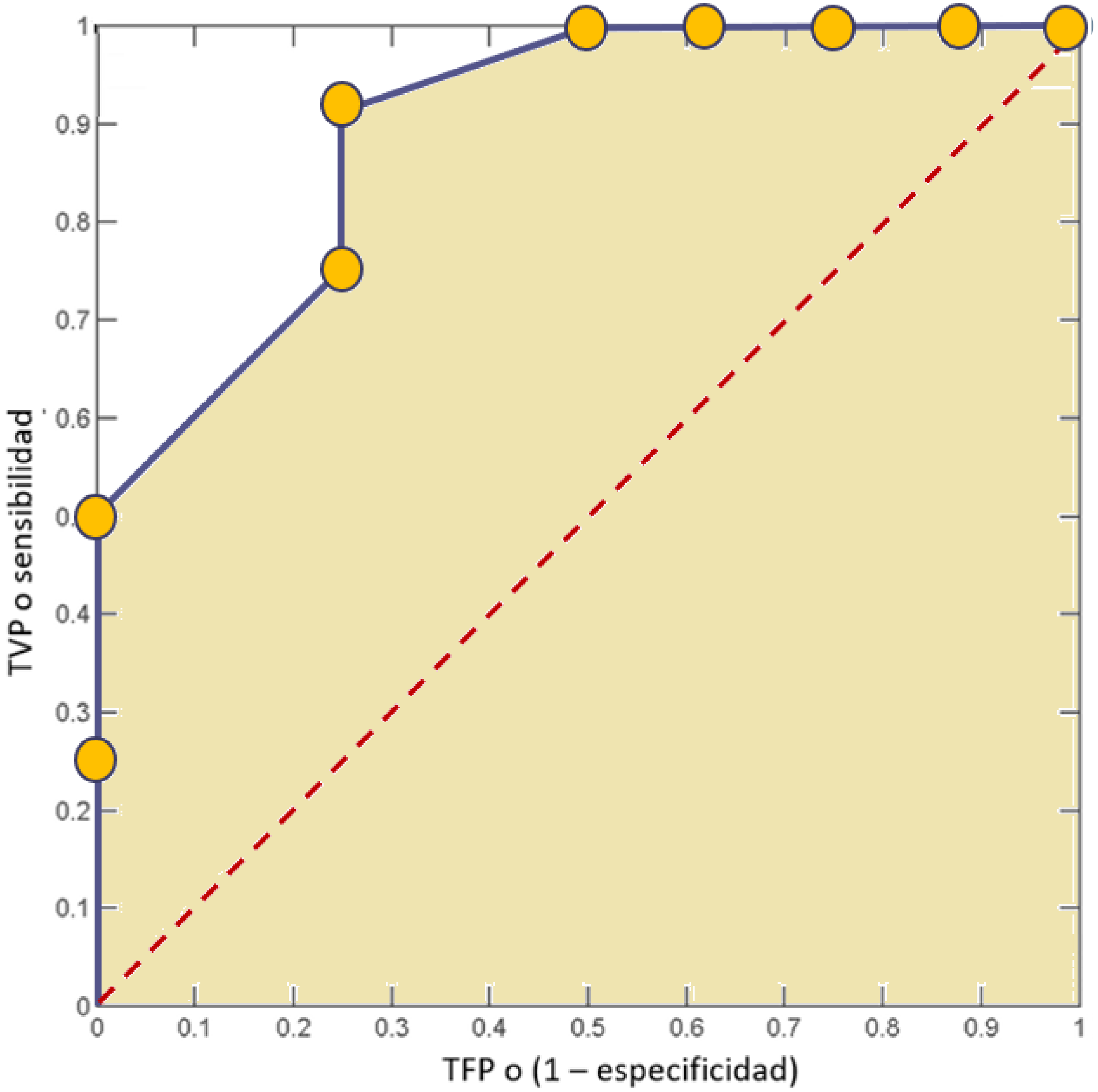
12 minas y **8 rocas**



ID	Clase	Confianza	Predice
5	Mina	0.99	
7	Mina	0.99	
9	Mina	0.99	
1	Mina	0.9	
10	Mina	0.9	
20	Mina	0.9	
8	Roca	0.8	
14	Mina	0.8	
15	Mina	0.8	
18	Roca	0.8	
19	Mina	0.8	
3	Mina	0.7	
6	Mina	0.7	
12	Mina	0.65	
4	Roca	0.6	
16	Roca	0.6	
11	Roca	0.5	
2	Roca	0.4	
13	Roca	0.3	
17	Roca	0.1	

12 minas y **8 rocas**

Area bajo la curva ROC (AUC)



AUC ROC

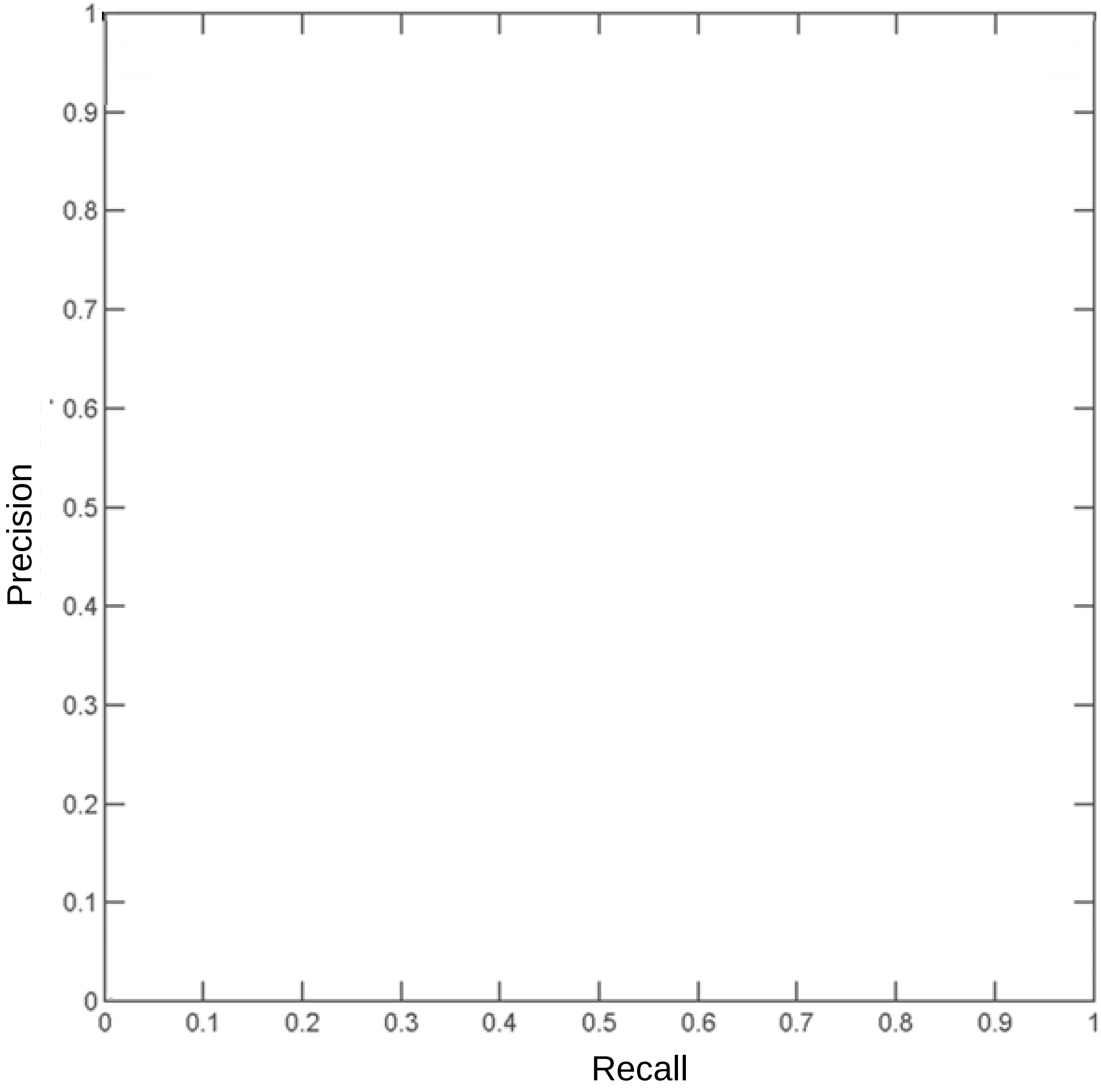
- **AUC (Area Under the Curve)** ROC es el área bajo la curva ROC.
  - Representa la capacidad de un modelo para diferenciar entre clases.
  - Un valor de AUC ROC cercano a 1 indica un buen rendimiento, mientras que un valor de 0.5 indica un rendimiento similar al de una clasificación aleatoria.

- **¿Cuándo se usa AUC ROC?**
  - Se utiliza cuando las clases están **más o menos balanceadas** o cuando se quiere medir qué tan bien un modelo puede **separar entre las clases positivas y negativas.**
  - Es útil para ver el **rendimiento general del modelo** a través de diferentes umbrales, especialmente en casos donde se puede ajustar el umbral de clasificación.

ID	Clase	Confianza	Predice
5	Mina	0.99	
7	Mina	0.99	
9	Mina	0.99	
1	Mina	0.9	
10	Mina	0.9	
20	Mina	0.9	
8	Roca	0.8	
14	Mina	0.8	
15	Mina	0.8	
18	Roca	0.8	
19	Mina	0.8	
3	Mina	0.7	
6	Mina	0.7	
12	Mina	0.65	
4	Roca	0.6	
16	Roca	0.6	
11	Roca	0.5	
2	Roca	0.4	
13	Roca	0.3	
17	Roca	0.1	

12 minas y **8 rocas**

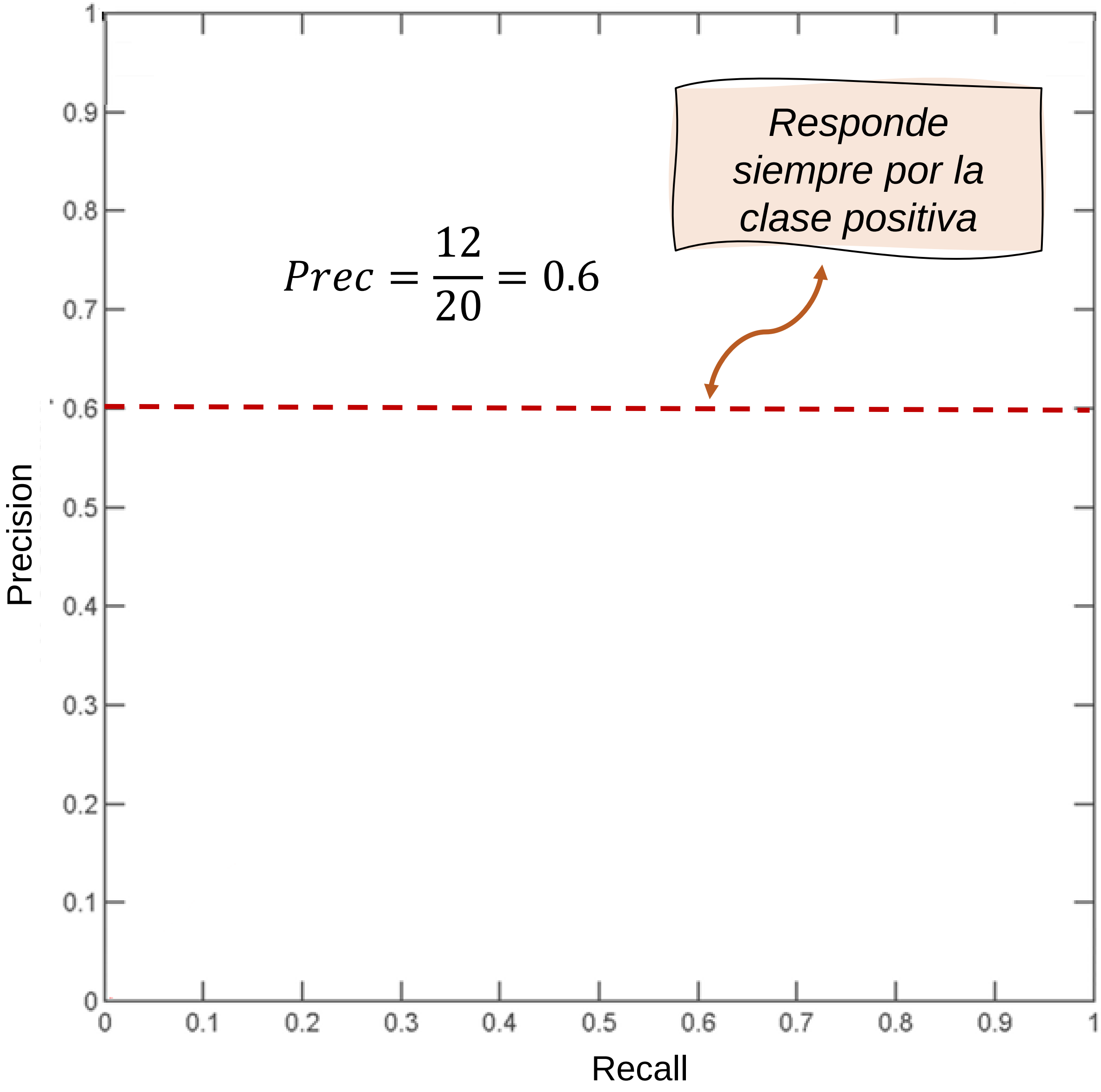
CURVA PR



ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Mina
13	Roca	0.3	Mina
17	Roca	0.1	Mina

12 minas y **8 rocas**

CURVA PR

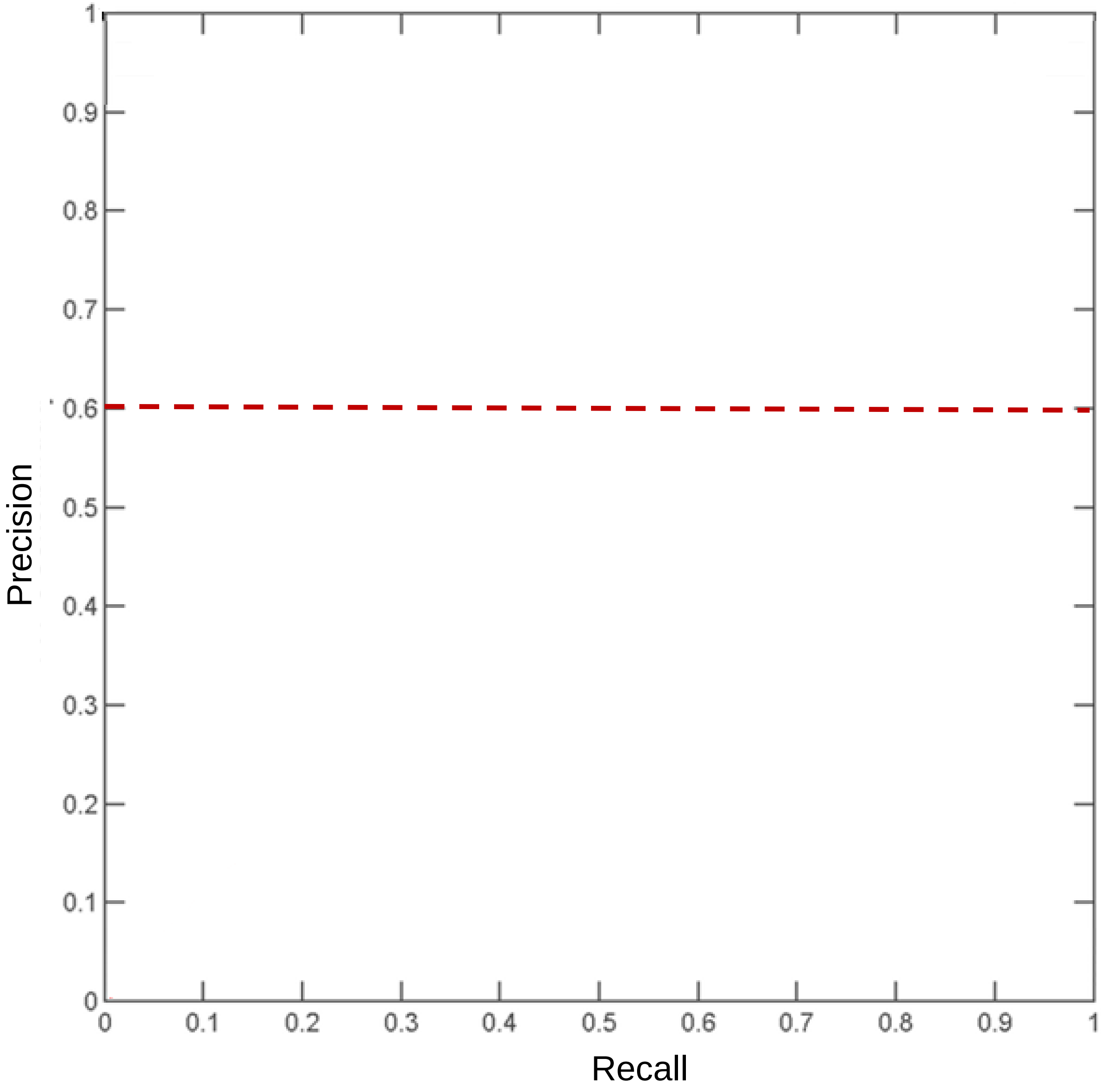


ID	Clase	Confianza	Predice
5	Mina	0.99	
7	Mina	0.99	
9	Mina	0.99	
1	Mina	0.9	
10	Mina	0.9	
20	Mina	0.9	
8	Roca	0.8	
14	Mina	0.8	
15	Mina	0.8	
18	Roca	0.8	
19	Mina	0.8	
3	Mina	0.7	
6	Mina	0.7	
12	Mina	0.65	
4	Roca	0.6	
16	Roca	0.6	
11	Roca	0.5	
2	Roca	0.4	
13	Roca	0.3	
17	Roca	0.1	

12 minas y 8 rocas

Umbral = 0.99

CURVA PR



ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Roca
10	Mina	0.9	Roca
20	Mina	0.9	Roca
8	Roca	0.8	Roca
14	Mina	0.8	Roca
15	Mina	0.8	Roca
18	Roca	0.8	Roca
19	Mina	0.8	Roca
3	Mina	0.7	Roca
6	Mina	0.7	Roca
12	Mina	0.65	Roca
4	Roca	0.6	Roca
16	Roca	0.6	Roca
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y **8 rocas**

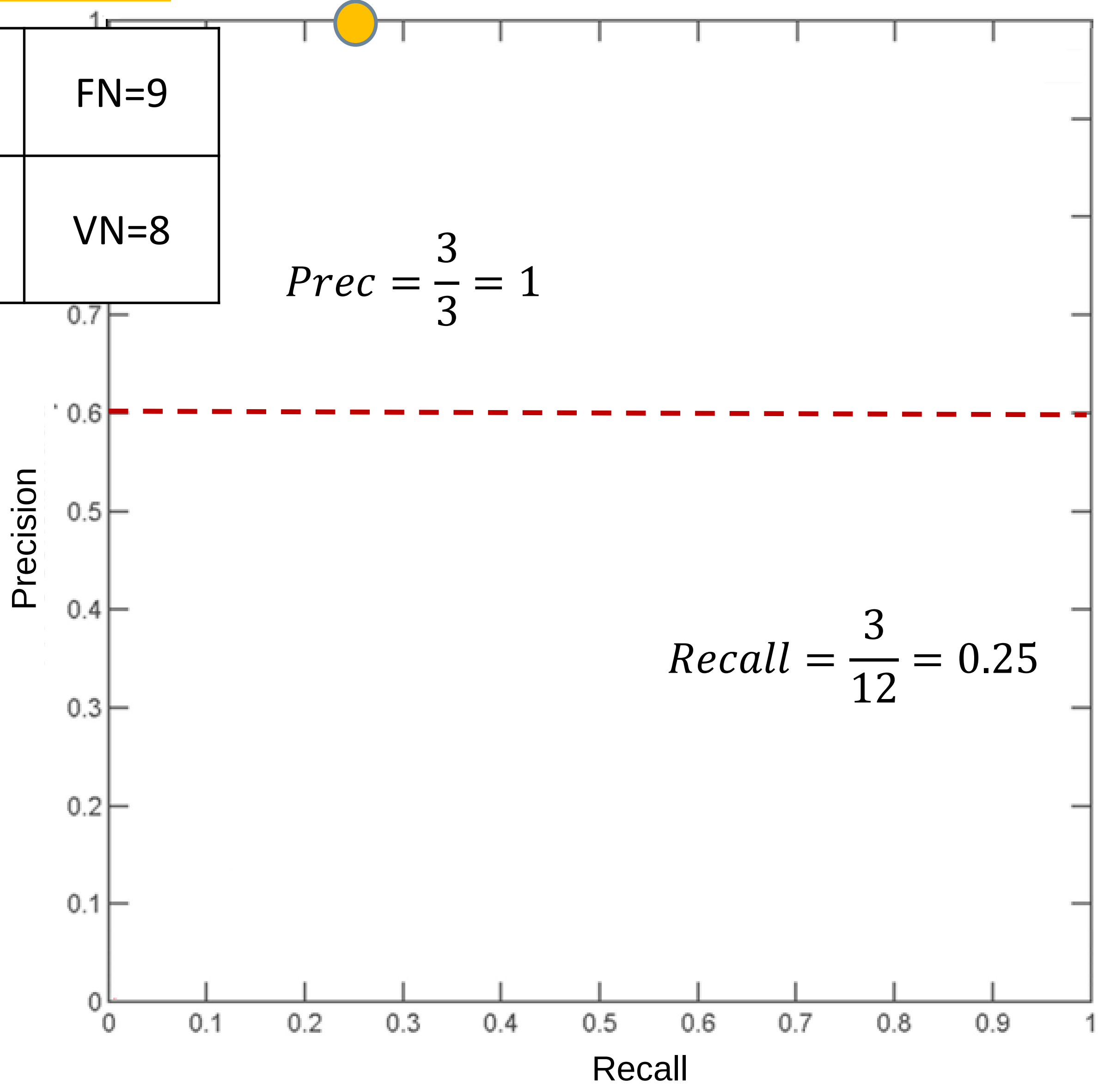
Umbral = 0.99

CURVA PR

VP=3	FN=9
FP=0	VN=8

$$Prec = \frac{3}{3} = 1$$

$$Recall = \frac{3}{12} = 0.25$$



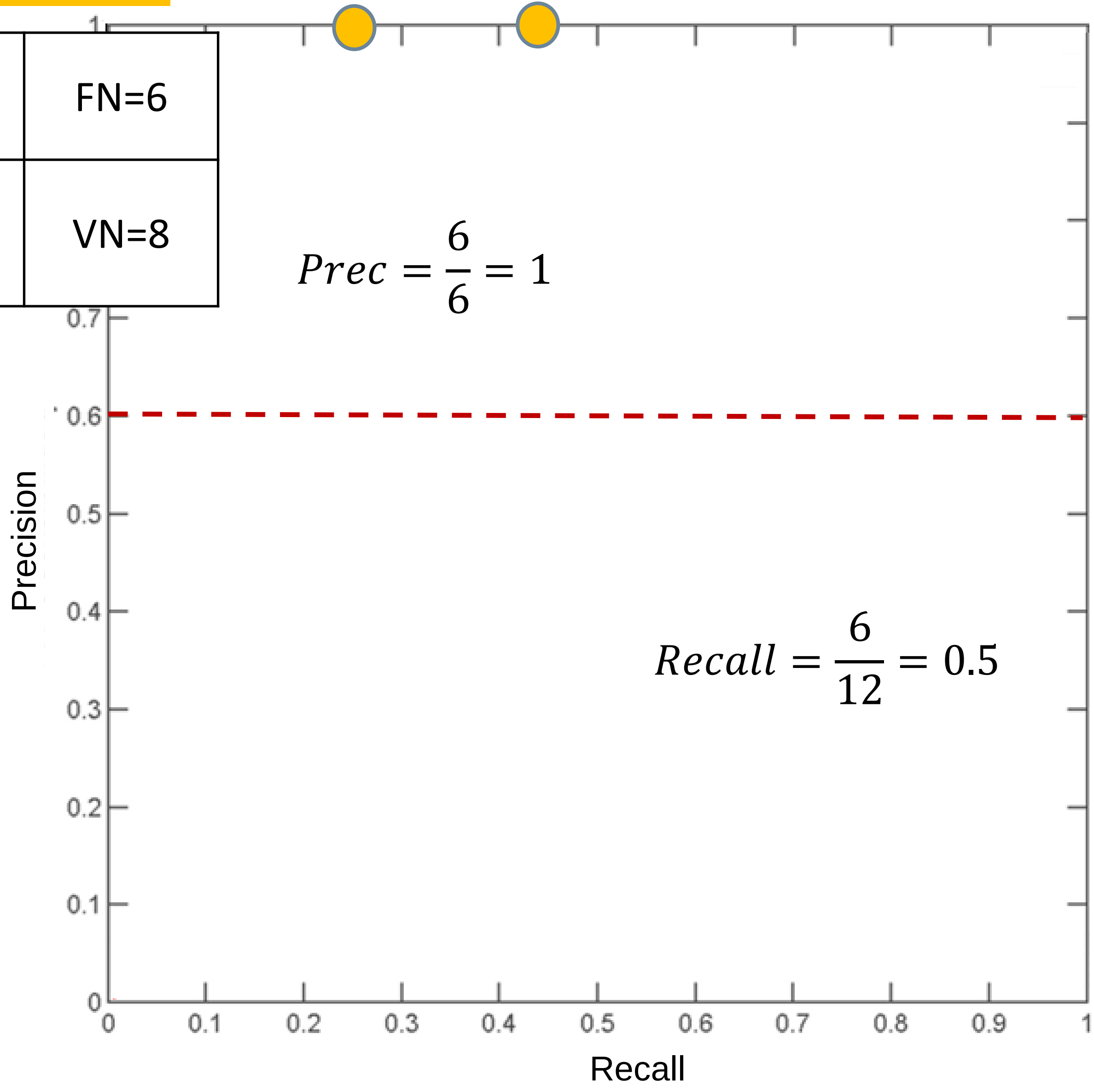
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Roca
14	Mina	0.8	Roca
15	Mina	0.8	Roca
18	Roca	0.8	Roca
19	Mina	0.8	Roca
3	Mina	0.7	Roca
6	Mina	0.7	Roca
12	Mina	0.65	Roca
4	Roca	0.6	Roca
16	Roca	0.6	Roca
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y **8 rocas**

Umbral = 0.9

CURVA PR

VP=6	FN=6
FP=0	VN=8



$$Prec = \frac{6}{6} = 1$$

$$Recall = \frac{6}{12} = 0.5$$

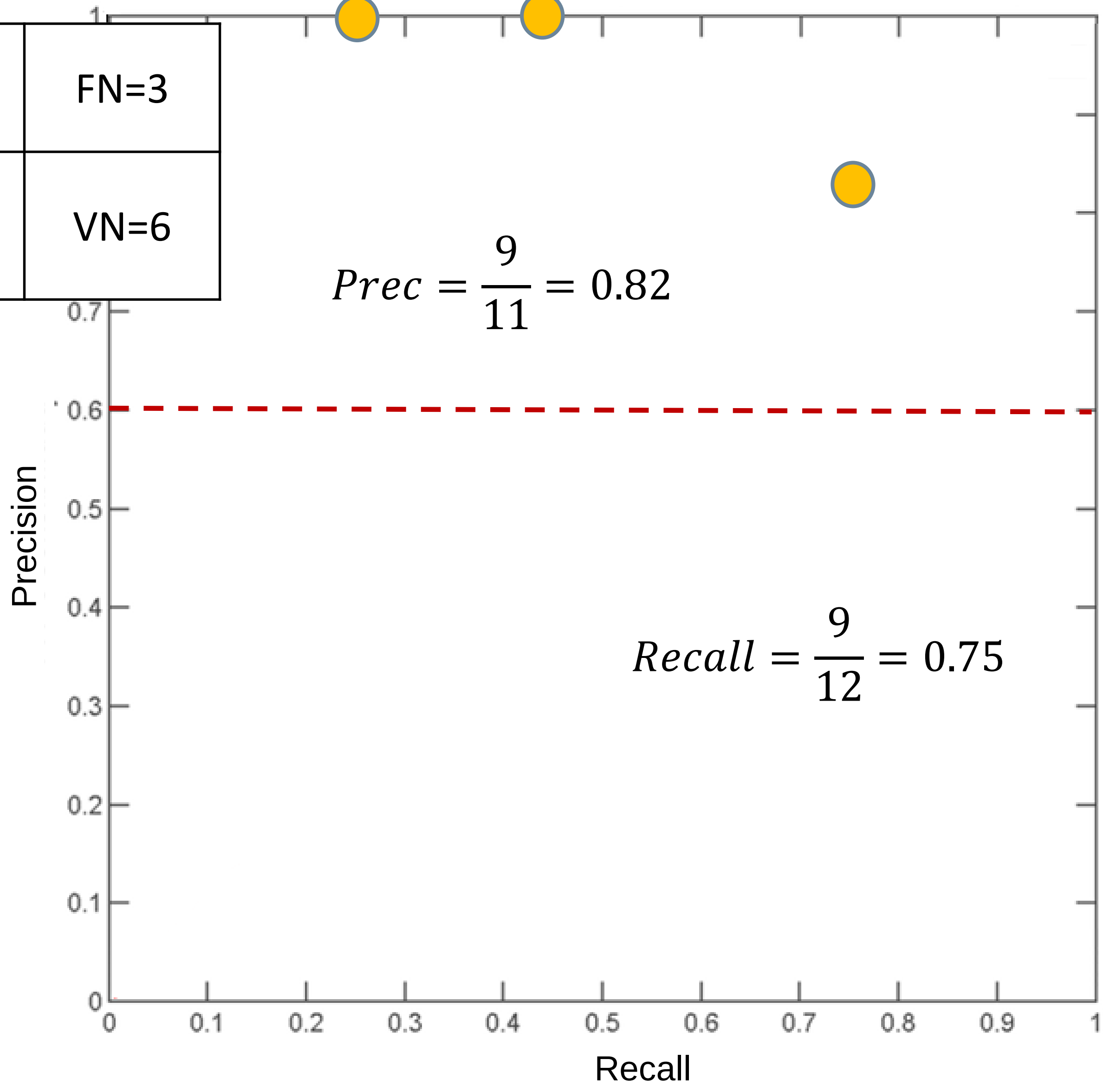
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Roca
6	Mina	0.7	Roca
12	Mina	0.65	Roca
4	Roca	0.6	Roca
16	Roca	0.6	Roca
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y **8 rocas**

Umbral = 0.8

VP=9	FN=3
FP=2	VN=6

CURVA PR



$$Prec = \frac{9}{11} = 0.82$$

$$Recall = \frac{9}{12} = 0.75$$

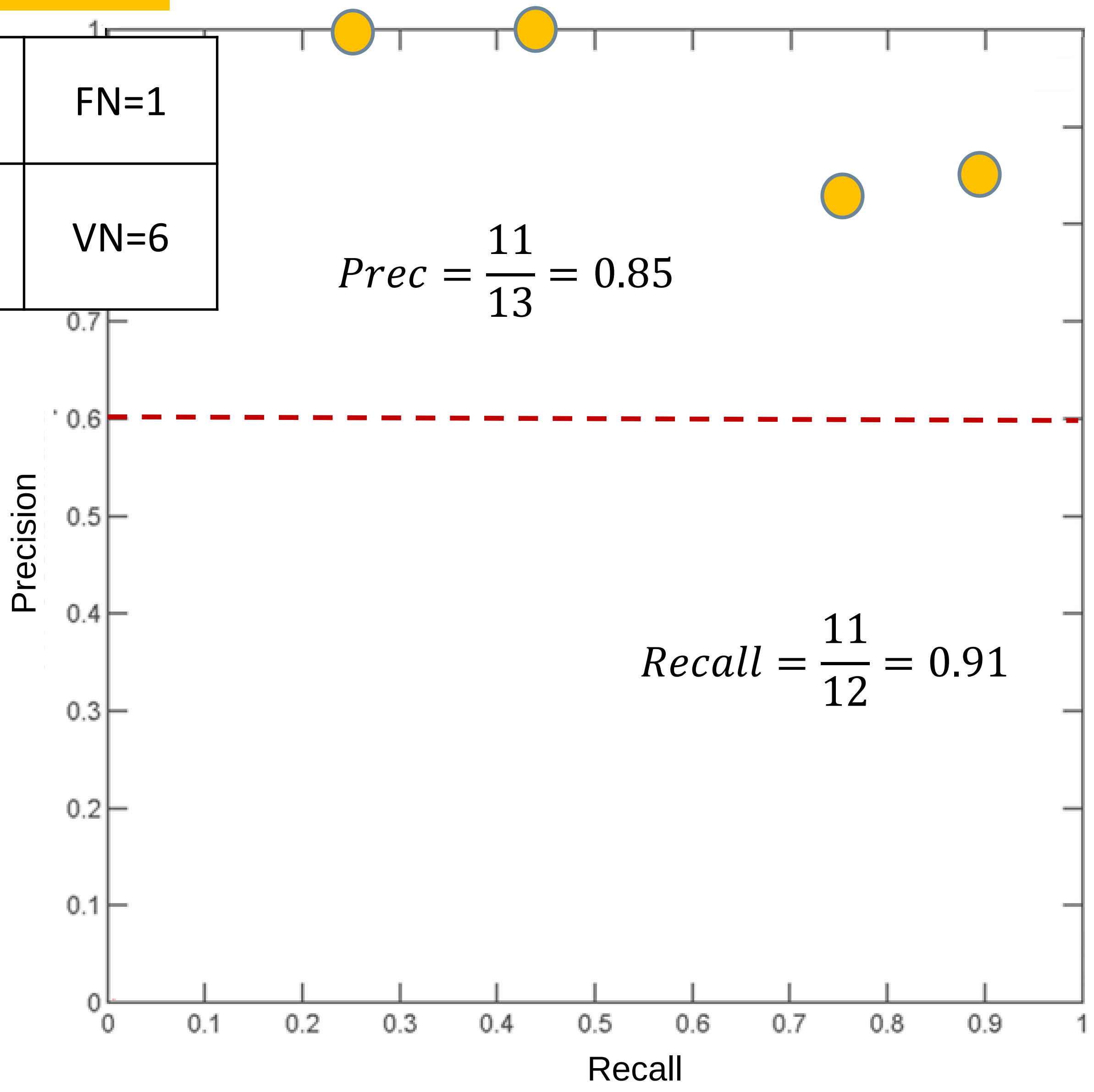
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Roca
4	Roca	0.6	Roca
16	Roca	0.6	Roca
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y **8 rocas**

Umbral = 0.7

VP=11	FN=1
FP=2	VN=6

CURVA PR



$$Prec = \frac{11}{13} = 0.85$$

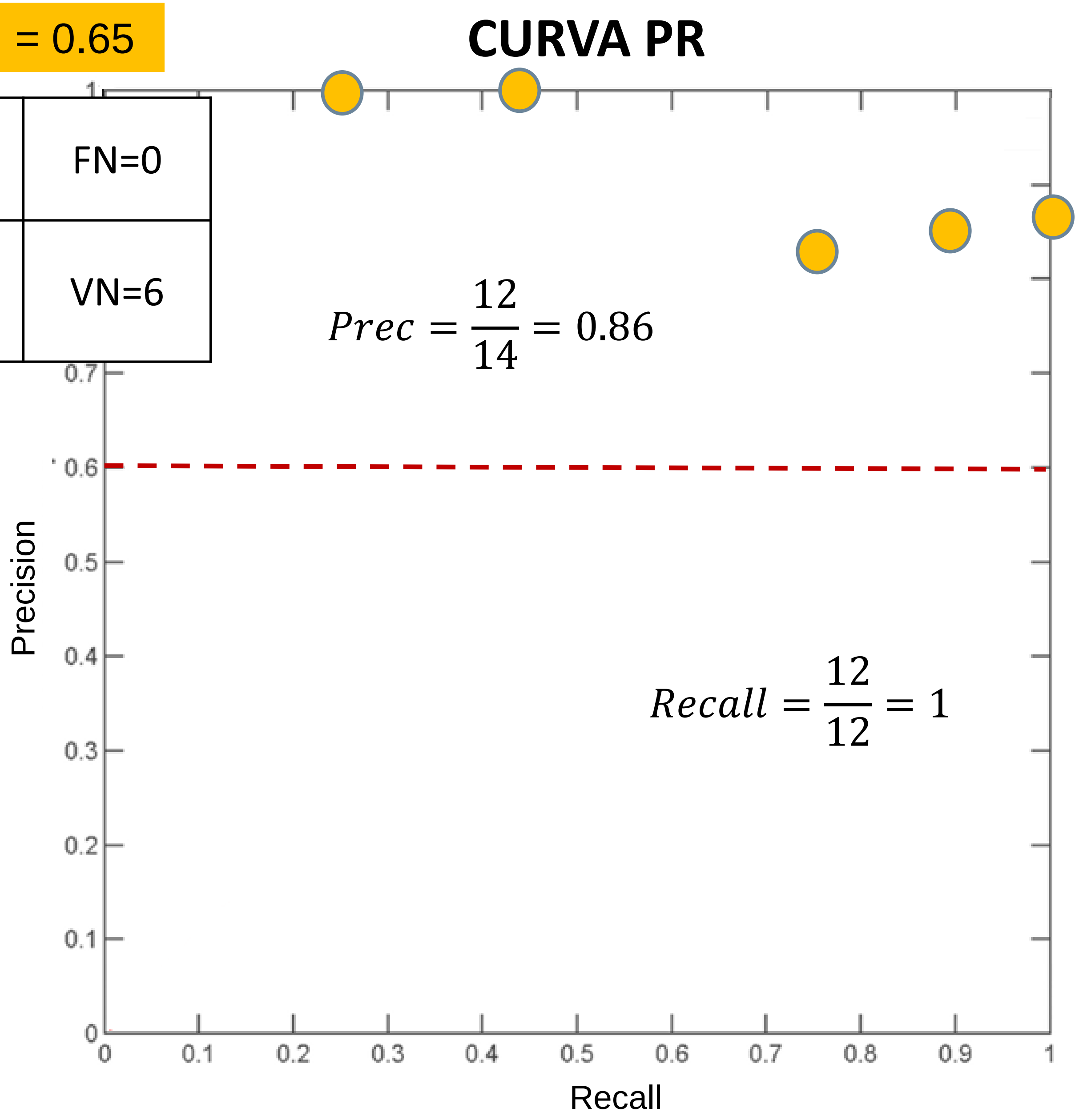
$$Recall = \frac{11}{12} = 0.91$$

ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Roca
16	Roca	0.6	Roca
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y **8 rocas**

Umbral = 0.65

VP=12	FN=0
FP=2	VN=6



$$Prec = \frac{12}{14} = 0.86$$

$$Recall = \frac{12}{12} = 1$$

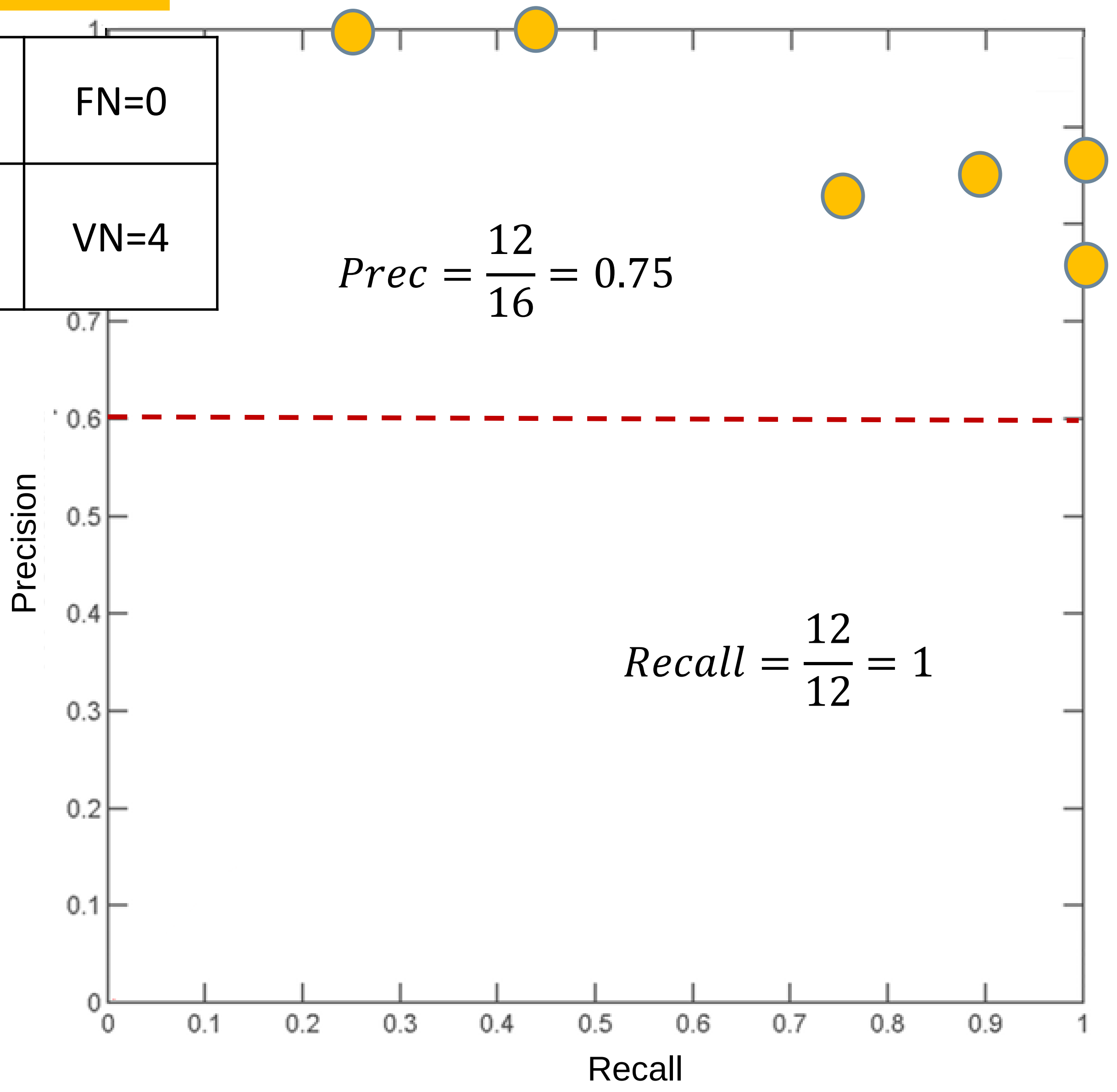
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Roca
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y **8 rocas**

Umbral = 0.6

VP=12	FN=0
FP=4	VN=4

CURVA PR



$$Prec = \frac{12}{16} = 0.75$$

$$Recall = \frac{12}{12} = 1$$

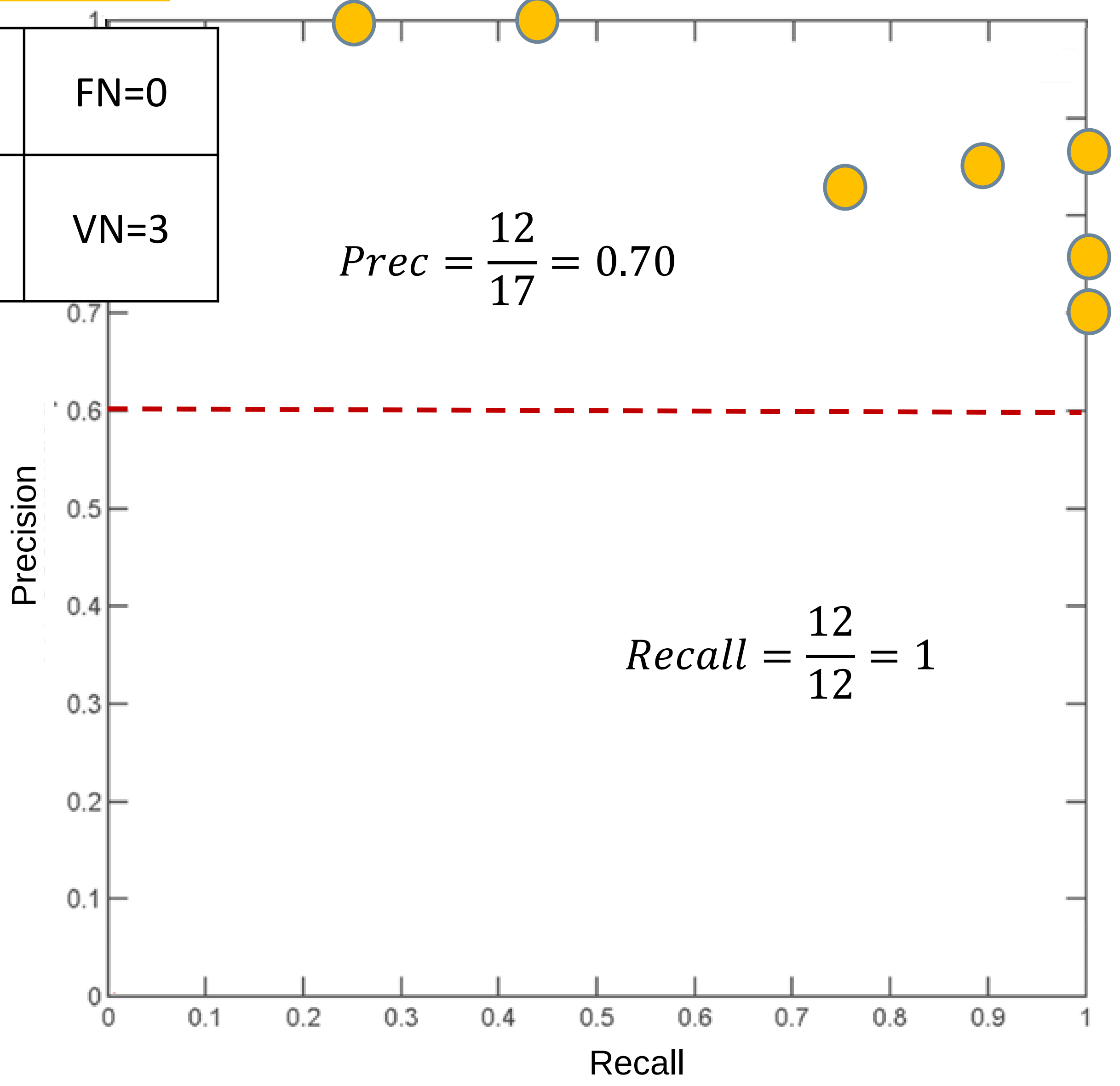
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Roca
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y 8 rocas

Umbral = 0.5

VP=12	FN=0
FP=5	VN=3

CURVA PR



$$Prec = \frac{12}{17} = 0.70$$

$$Recall = \frac{12}{12} = 1$$

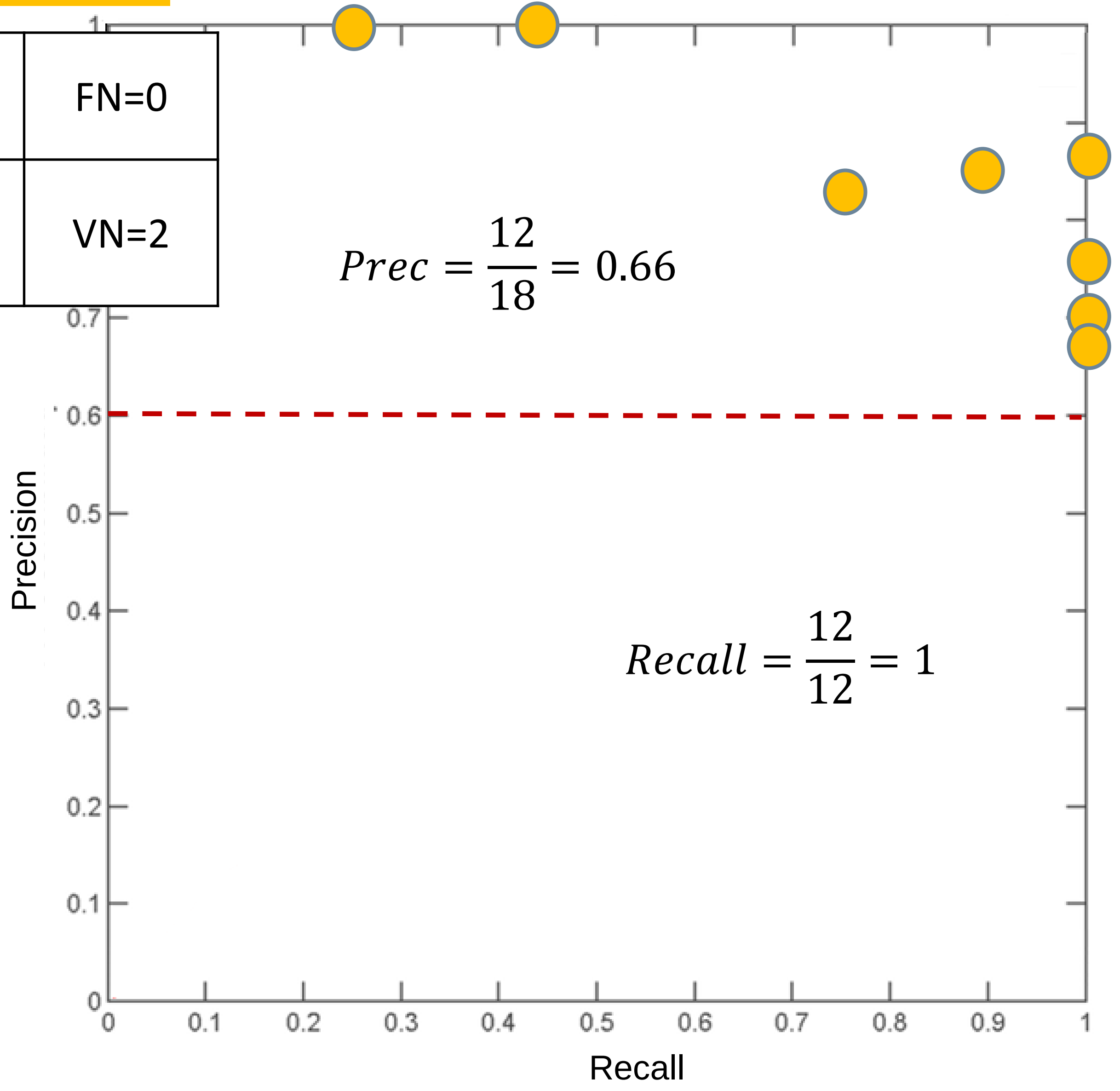
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Mina
13	Roca	0.3	Roca
17	Roca	0.1	Roca

12 minas y **8 rocas**

Umbral = 0.4

VP=12	FN=0
FP=6	VN=2

CURVA PR



$$Prec = \frac{12}{18} = 0.66$$

$$Recall = \frac{12}{12} = 1$$

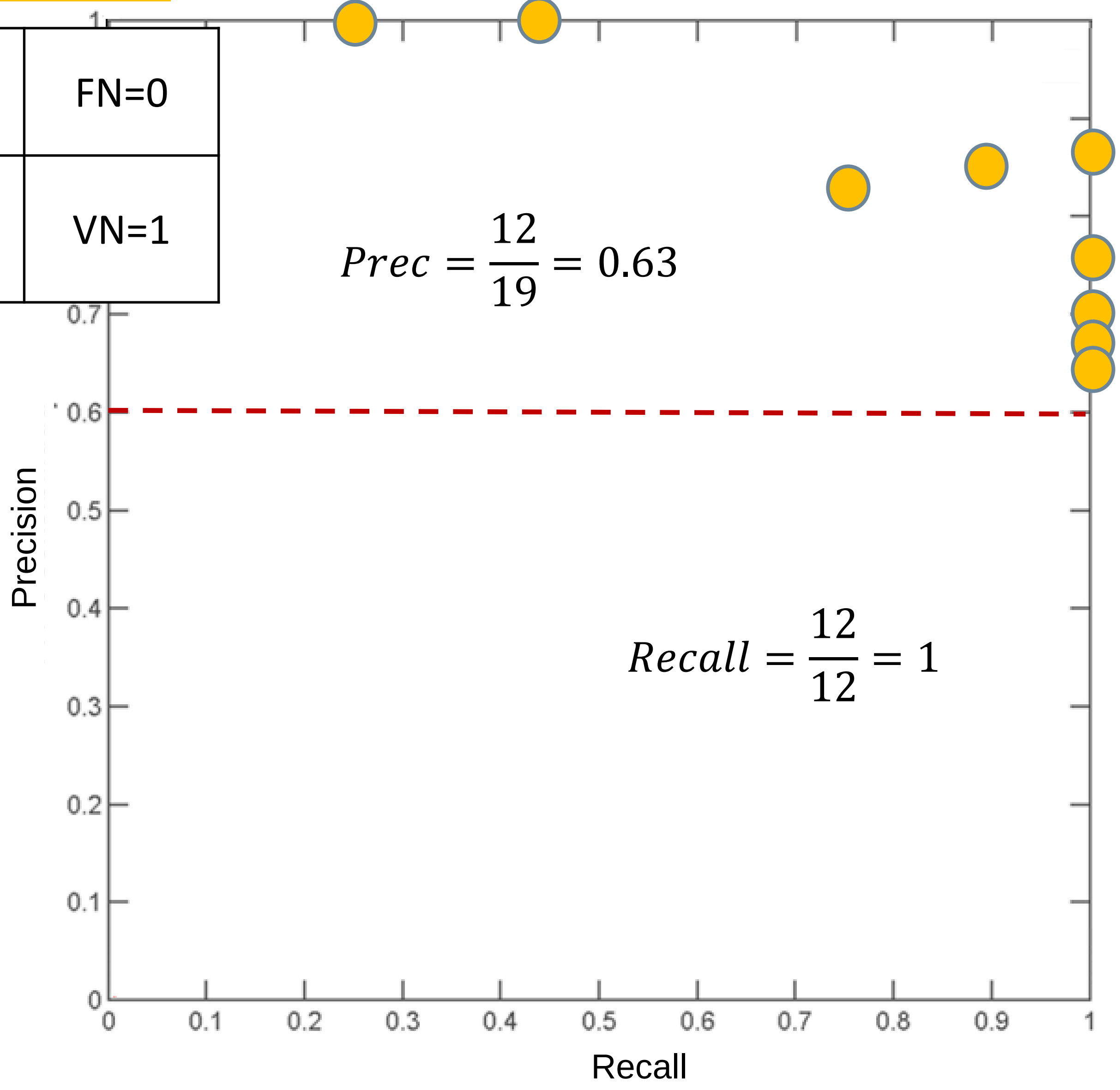
ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Mina
13	Roca	0.3	Mina
17	Roca	0.1	Roca

12 minas y **8 rocas**

Umbral = 0.3

VP=12	FN=0
FP=7	VN=1

CURVA PR



$$Prec = \frac{12}{19} = 0.63$$

$$Recall = \frac{12}{12} = 1$$

ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Mina
13	Roca	0.3	Mina
17	Roca	0.1	Mina

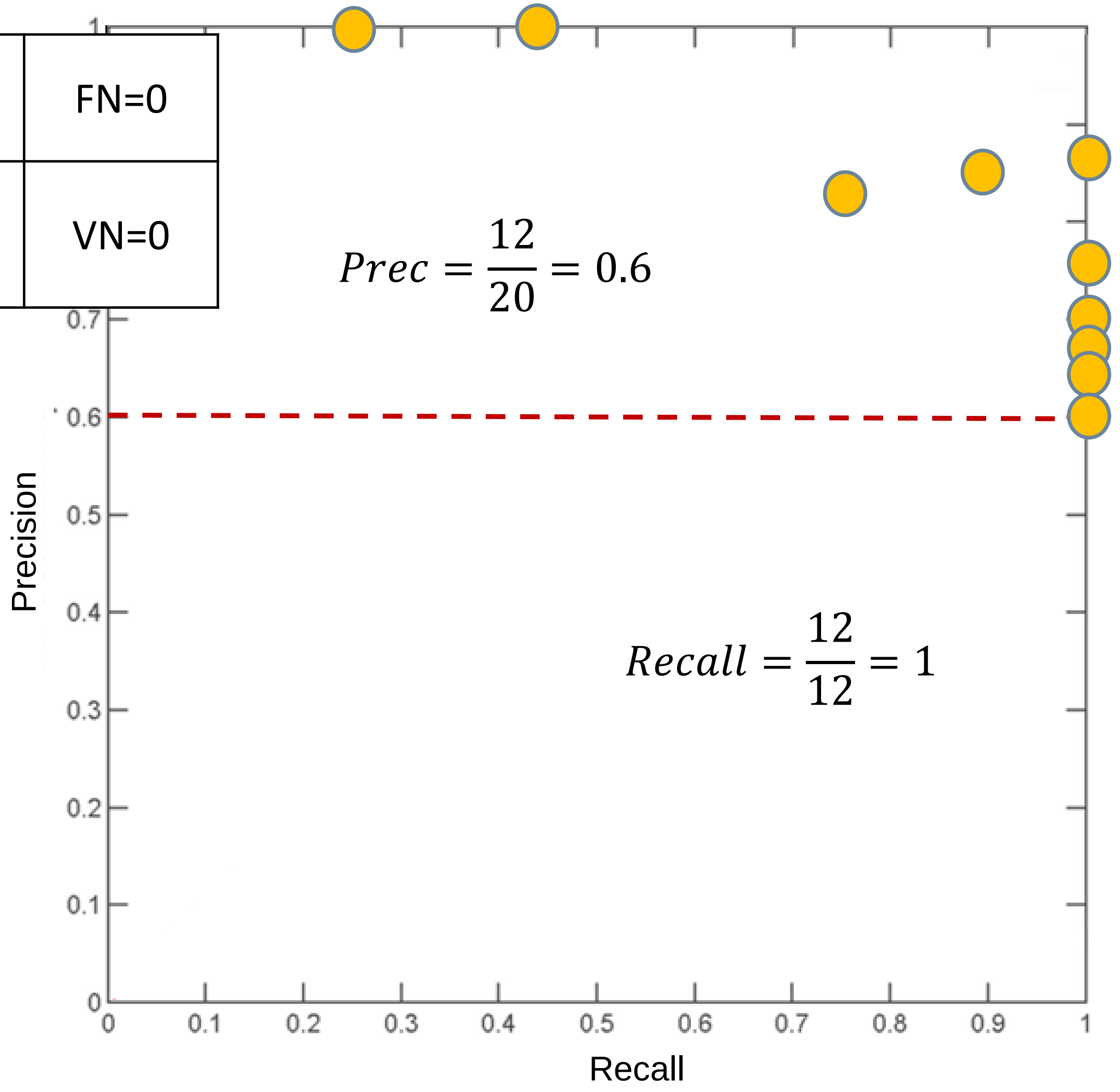
12 minas y **8 rocas**

CURVA PR

VP=12	FN=0
FP=8	VN=0

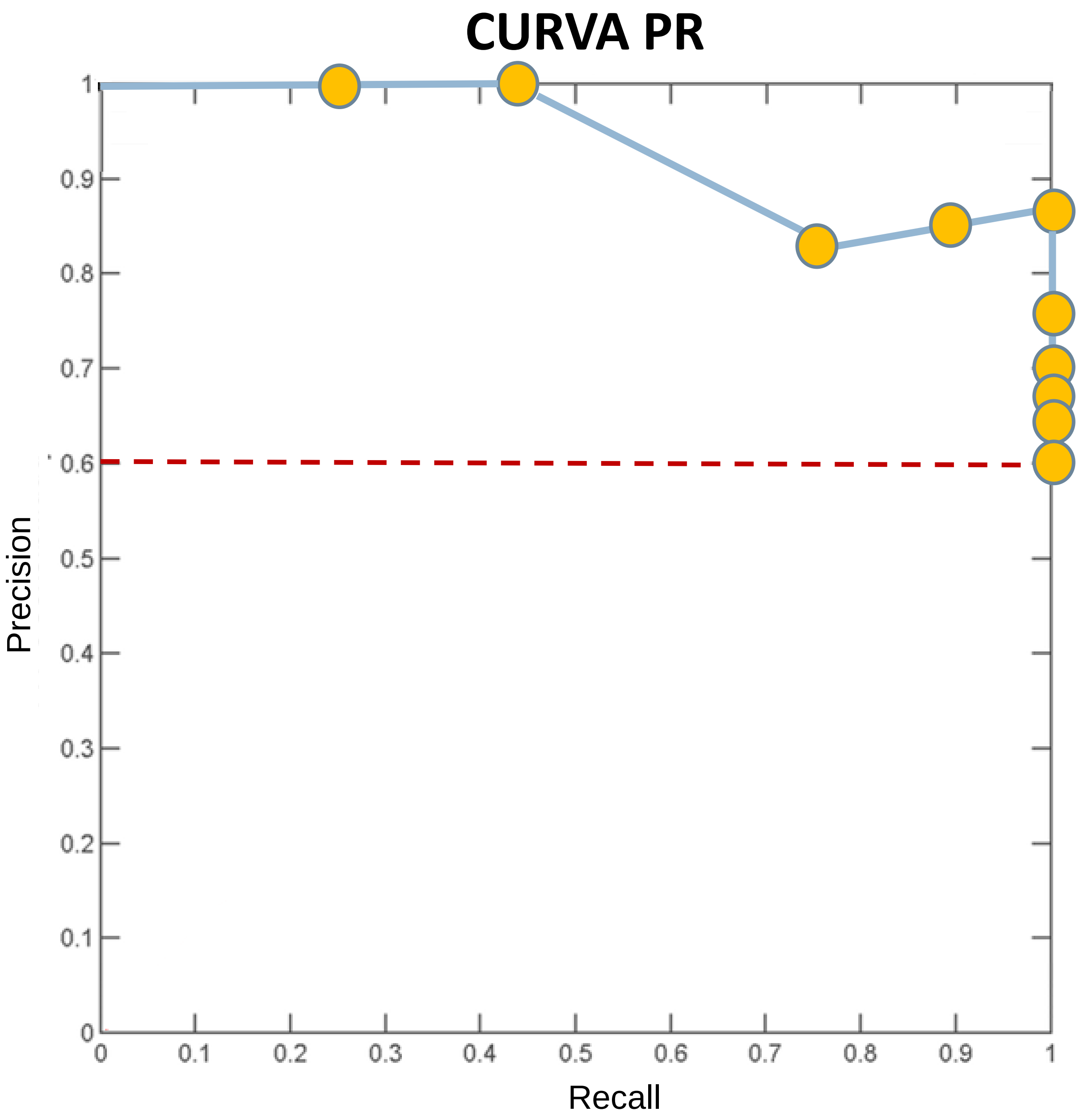
$$Prec = \frac{12}{20} = 0.6$$

$$Recall = \frac{12}{12} = 1$$



ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Mina
13	Roca	0.3	Mina
17	Roca	0.1	Mina

12 minas y **8 rocas**

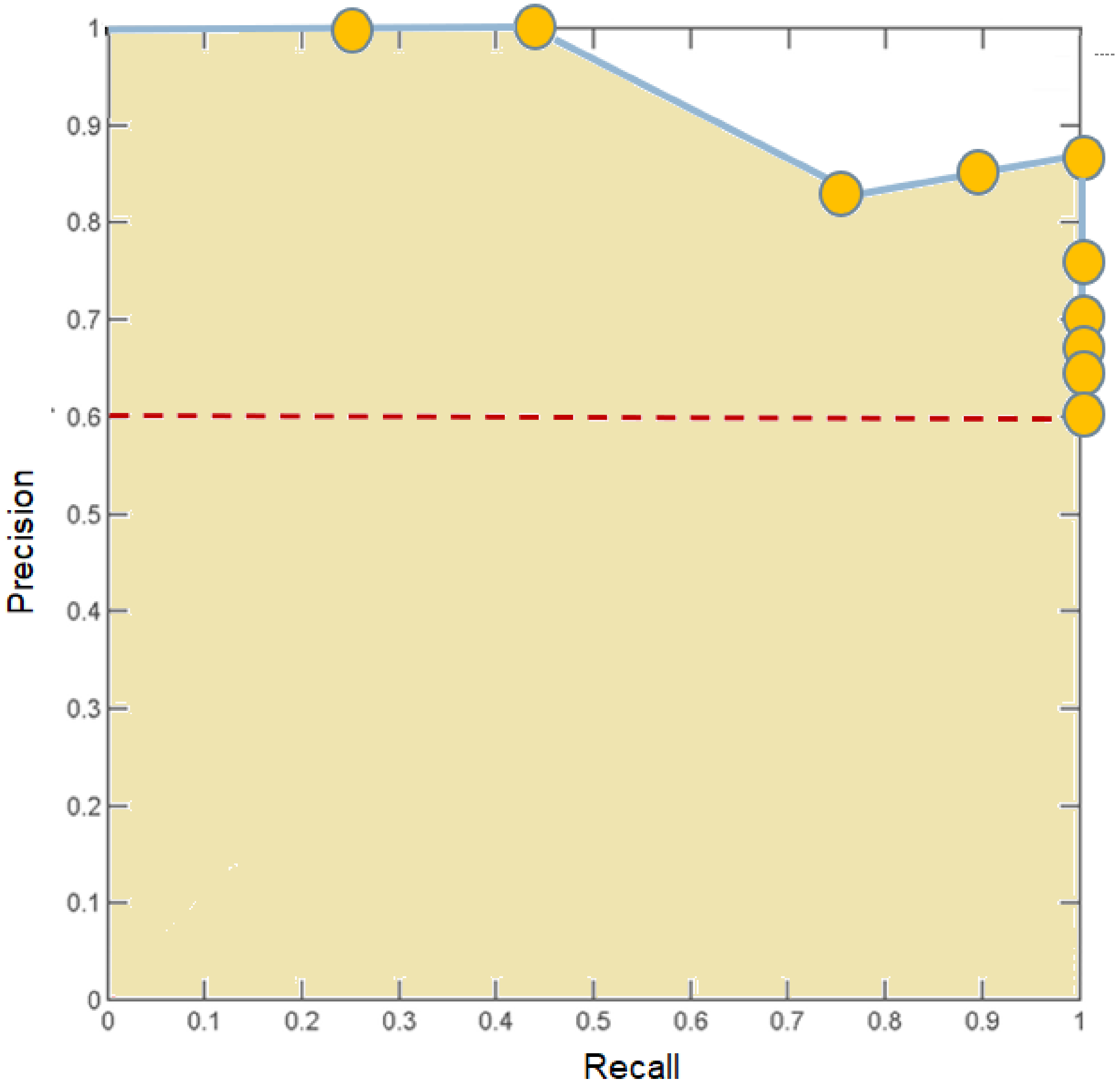


Curvas_ROC_PR.ipynb

ID	Clase	Confianza	Predice
5	Mina	0.99	Mina
7	Mina	0.99	Mina
9	Mina	0.99	Mina
1	Mina	0.9	Mina
10	Mina	0.9	Mina
20	Mina	0.9	Mina
8	Roca	0.8	Mina
14	Mina	0.8	Mina
15	Mina	0.8	Mina
18	Roca	0.8	Mina
19	Mina	0.8	Mina
3	Mina	0.7	Mina
6	Mina	0.7	Mina
12	Mina	0.65	Mina
4	Roca	0.6	Mina
16	Roca	0.6	Mina
11	Roca	0.5	Mina
2	Roca	0.4	Mina
13	Roca	0.3	Mina
17	Roca	0.1	Mina

12 minas y **8 rocas**

AREA AUC-PR



AUC PR

- **AUC PR** es el área bajo la curva Precision-Recall.
  - Esta métrica es útil cuando se mide el **balance entre la precisión y el recall**.
  - En problemas donde hay un gran desbalance de clases, AUC PR proporciona una visión más ajustada de cómo está funcionando el modelo en la clase minoritaria.

- **¿Cuándo se usa AUC PR?**
  - Se utiliza cuando las clases están **desbalanceadas**.
  - Si las clases están **desbalanceadas** (ej: se tienen muchos más negativos que positivos), el **FPR** puede ser muy bajo simplemente porque hay pocos falsos positivos en comparación con la gran cantidad de verdaderos negativos. Esto puede hacer que el AUC ROC parezca alto, incluso si el modelo no está funcionando bien en la clase minoritaria (positiva).
  - En este tipo de escenarios, el AUC ROC puede ser **engañoso**, porque un buen rendimiento en la clase mayoritaria puede ocultar el mal desempeño en la clase minoritaria.

Roca o Mina

- A partir de los datos del archivo "Sonar.csv" se desea construir una red neurona multiperceptrón para discriminar entre señales de sonar rebotadas en un cilindro de metal ("Mine") y aquellas rebotadas en una roca más o menos cilíndrica ("Rock").
- Probar con distintas configuraciones
- Indicar cuál recomendaría a la hora de predecir si es una mina o no utilizando: accuracy, f1-score y AUC.

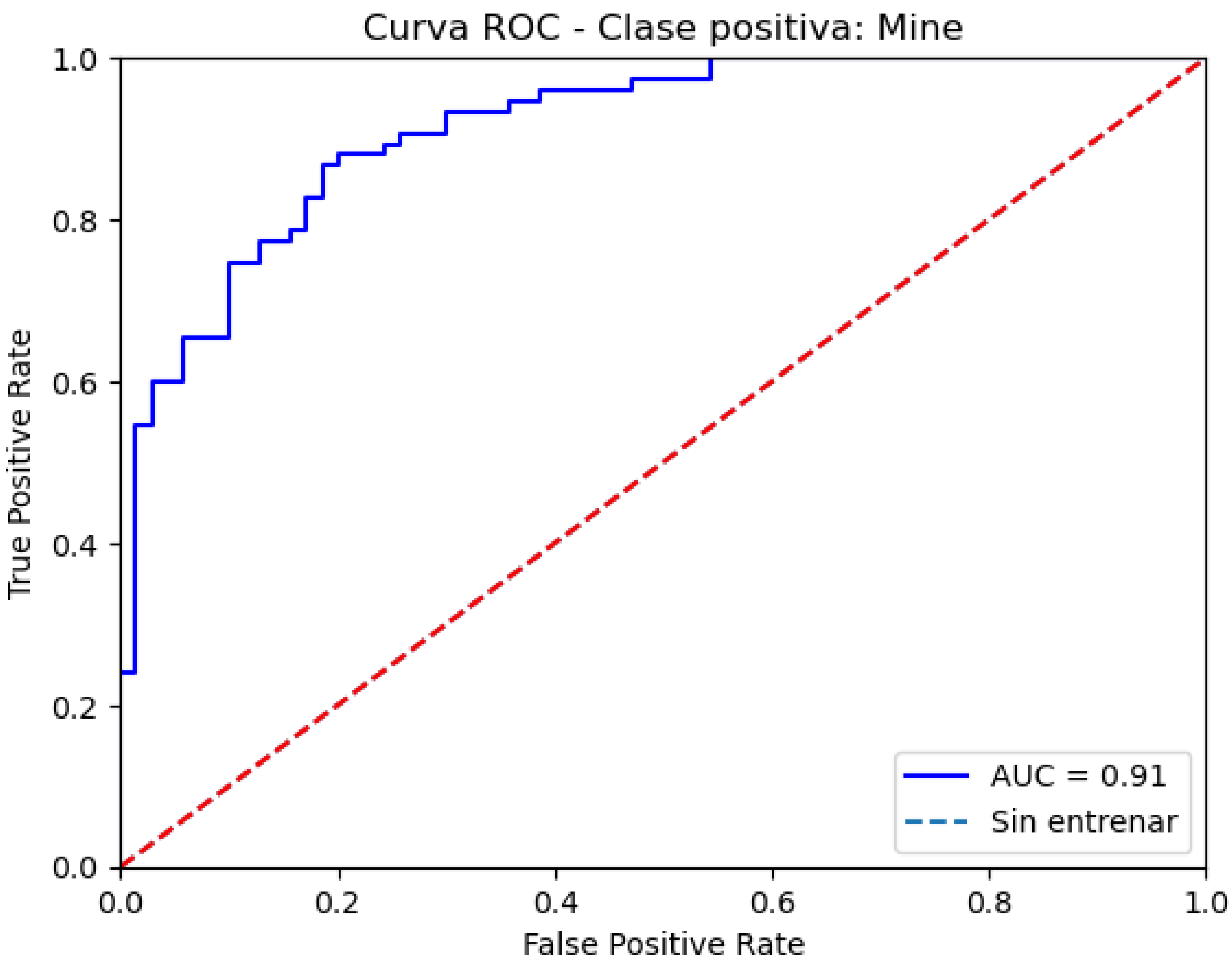
Curva ROC

```
fpr, tpr, threshold = metrics.roc_curve(Y_true,Y_prob)
roc_auc = metrics.auc(fpr, tpr)

plt.figure()
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
```

Keras_SONAR_softmax_AUC.ipynb

Curva ROC

Keras_SONAR_softmax_AUC.ipynb



Curva Precision-Recall

Keras_SONAR_softmax_AUC.ipynb