Aprendizaje automático

Redes de Neuronas Artificiales

UNED

Nieves Pedreira Souto

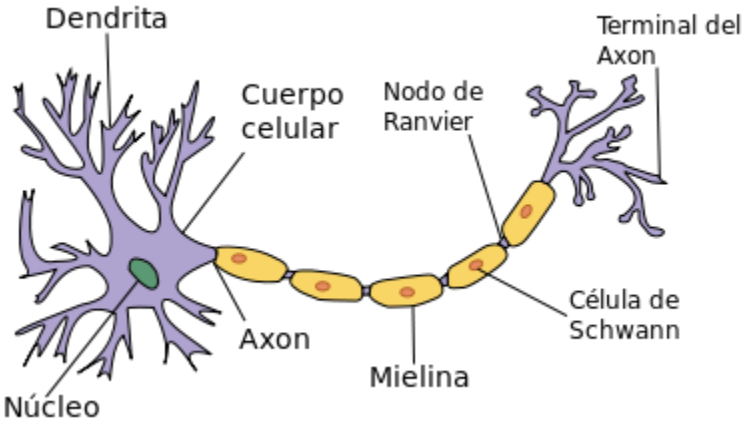
El sistema nervioso: la neurona

Las Redes de Neuronas Artificiales son un paradigma de aprendizaje a partir de ejemplos que se basa en la estructura y el funcionamiento del Sistema Nervioso.

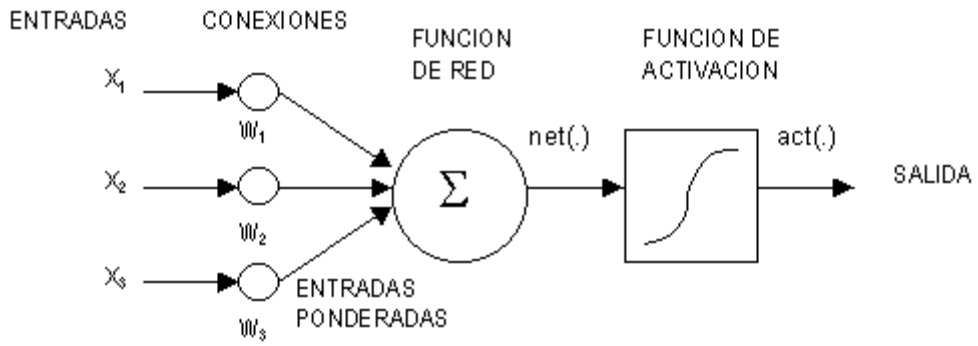
El elemento principal es la neurona.

Las neuronas pueden recibir y transmitir información (señales eléctricas y químicas) entre ellas, formando redes.

La señal generada por la neurona y transportada a lo largo del axón es un impulso eléctrico, mientras que la señal que se transmite entre los terminales de una neurona y las dendritas de otra es de origen químico.



Neurona biológica

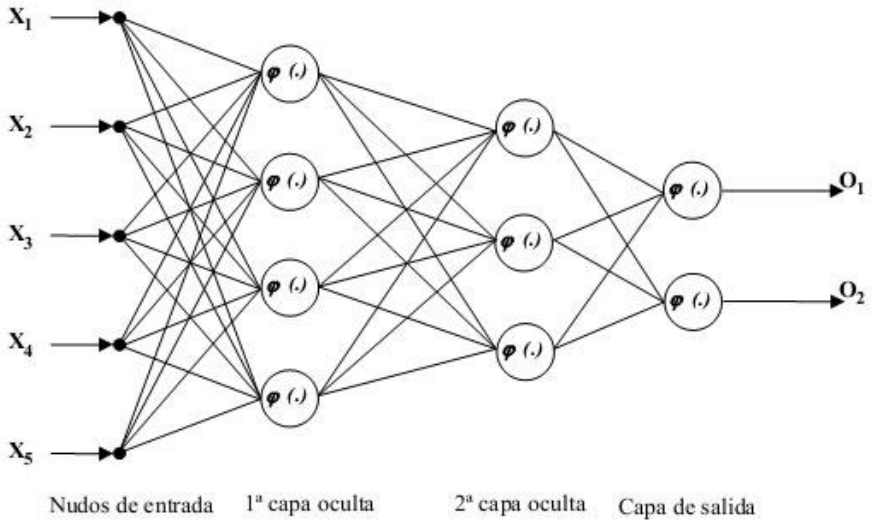


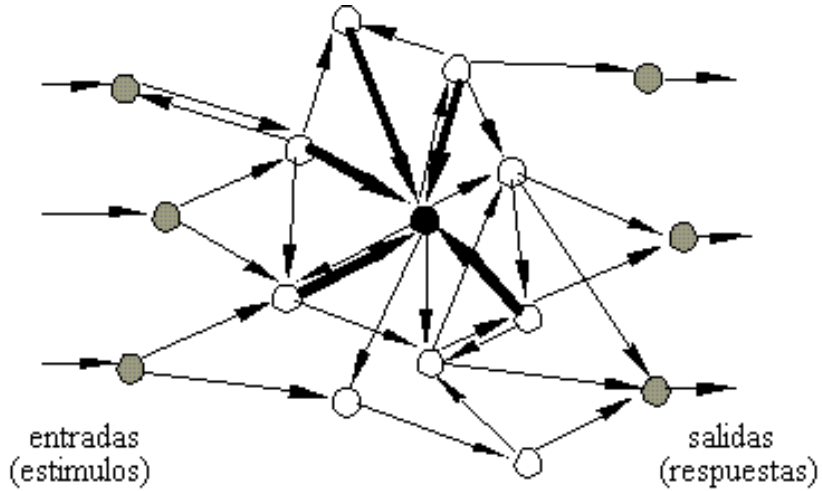
Neurona artificial (perceptrón)

Estructuras de red

El aprendizaje a través de las Redes de Neuronas Artificiales, no se realiza con la descripción de un algoritmo, sino que se procesa la información recibida en función de la estructura y funcionalidad de la red.

Son modelos que intentan reproducir el comportamiento del cerebro. Realizan una simplificación del modelo, averiguando cuáles son los elementos relevantes.

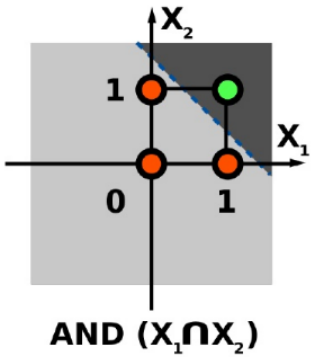




Existen muchos modelos de redes diferentes, tanto en su diseño como en sus reglas de aprendizaje y funciones de activación.

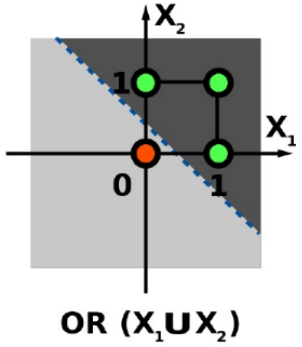
El funcionamiento de un perceptrón de manera aislada, permite resolver problemas linealmente separables en dos regiones

Ejemplo Función AND



X1	X2	AND
0	0	0
0	1	0
1	0	0
1	1	1

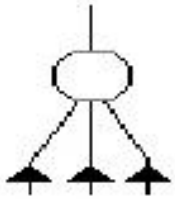
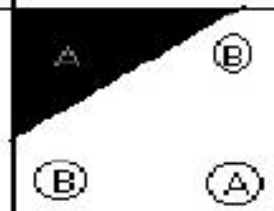
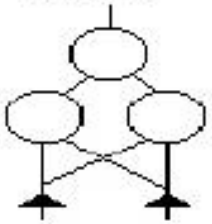
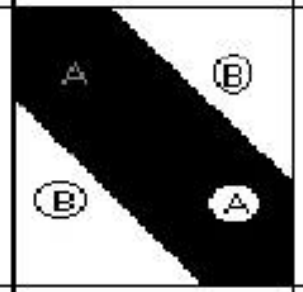
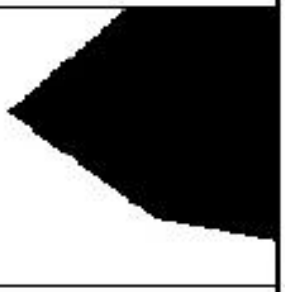
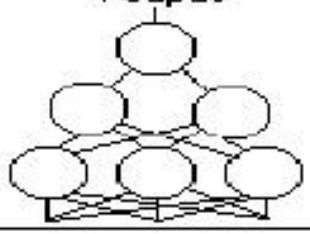
Ejemplo Función OR



X1	X2	OR
0	0	0
0	1	1
1	0	1
1	1	1

Al aumentar el número de perceptrones en capas y formar una red, es posible resolver problemas más complejos y no linealmente separables. Según la complejidad del problema debería aumentarse el número de capas de la red, pero un número elevado de capas o de neuronas haría que la red no pudiese generalizar bien.

Complejidad del problema

Estructura	Regiones de Decision	Problema de la XOR	Formas de regiones generalizadas
2 capas 	Medio plano limitado por un hiperplano		
3 capas 	Regiones cerradas o convexas		
4 capas 	arbitraria complejidad limitada por el numerode neuronas		

Elementos de las RNAs

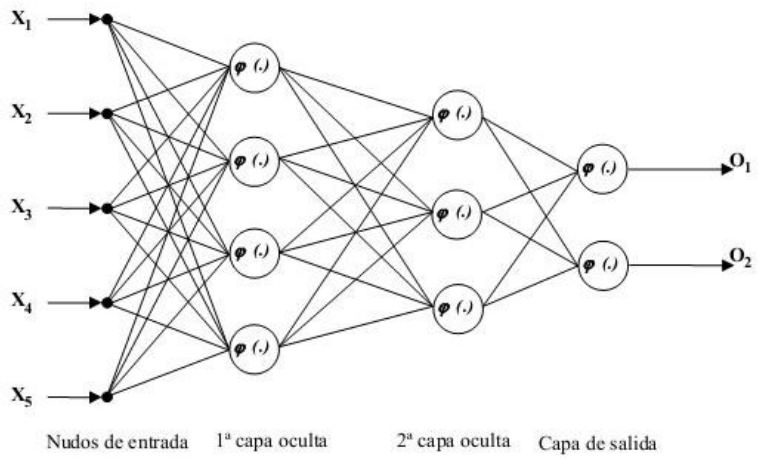
Neuronas

Conexiones

Pesos

Función de activación

Regla de aprendizaje



Arquitectura de la red: número de capas, neuronas por capa, conexiones entre neuronas.

Función de activación: cada neurona se activa o se inhibe

Regla de aprendizaje: modificación de los pesos en respuesta a una información de entrada. Controlado por la tasa de aprendizaje (variable) y el momento (constante).

Los cambios que se producen durante el proceso de aprendizaje se reducen a la destrucción, modificación y creación de conexiones entre las neuronas.

Se puede afirmar que el proceso de aprendizaje ha finalizado (la red ha aprendido) cuando los valores de los pesos permanecen estables

Aprendizaje supervisado

Se realiza mediante un entrenamiento controlado por un agente externo que determina la respuesta que debería generar la red a partir de una entrada determinada.

Se comprueba la salida de la red y en el caso de que ésta no coincida con la deseada, se procederá a modificar los pesos de las conexiones, con el fin de conseguir que la salida se aproxime a la deseada.

Se consideran tres formas de llevar a cabo este tipo de aprendizaje:

Aprendizaje por corrección de error: Consiste en ajustar los pesos en función de la diferencia entre los valores deseados y los obtenidos en la salida de la red; es decir, en función del error.

Aprendizaje por refuerzo: durante el entrenamiento se indica mediante una señal de refuerzo si la salida obtenida en la red se ajusta a la deseada (éxito=+1 o fracaso=-1), y en función de ello se ajustan los pesos basándose en un mecanismo de probabilidades.

Aprendizaje estocástico: consiste en realizar cambios aleatorios en los valores de los pesos de las conexiones de la red y evaluar su efecto a partir del objetivo deseado y de distribuciones de probabilidad.

Aprendizaje no supervisado

No se tiene en cuenta influencia externa para ajustar los pesos de las conexiones entre neuronas.

En algunos casos, la salida representa el grado de familiaridad o similitud entre la información que se le está presentando en la entrada y las informaciones que se le han mostrado en el pasado.

En otro caso podría realizar una codificación de los datos de entrada, generando a la salida una versión codificada de la entrada, con menos bits, pero manteniendo la información relevante de los datos

También se puede realizar un mapeo de características, obteniéndose en las neuronas de salida una disposición geométrica que representa un mapa topográfico de las características de los datos de entrada, de tal forma que si se presentan a la red informaciones similares, se activen las neuronas de una zona determinada.

Características de las RNAs

Pueden aprender de la experiencia

Pueden generalizar nuevos casos

Pueden abstraer características esenciales a partir de entradas a priori irrelevantes

Tolerancia a fallos:

- Pueden aprender a reconocer patrones con ruido, distorsionados, o incompletos. (La triple i: información incompleta, inexacta e inconsistente)
- Pueden seguir realizando su función (con cierta degradación) aunque se destruya parte de la red.

Aprendizaje automático

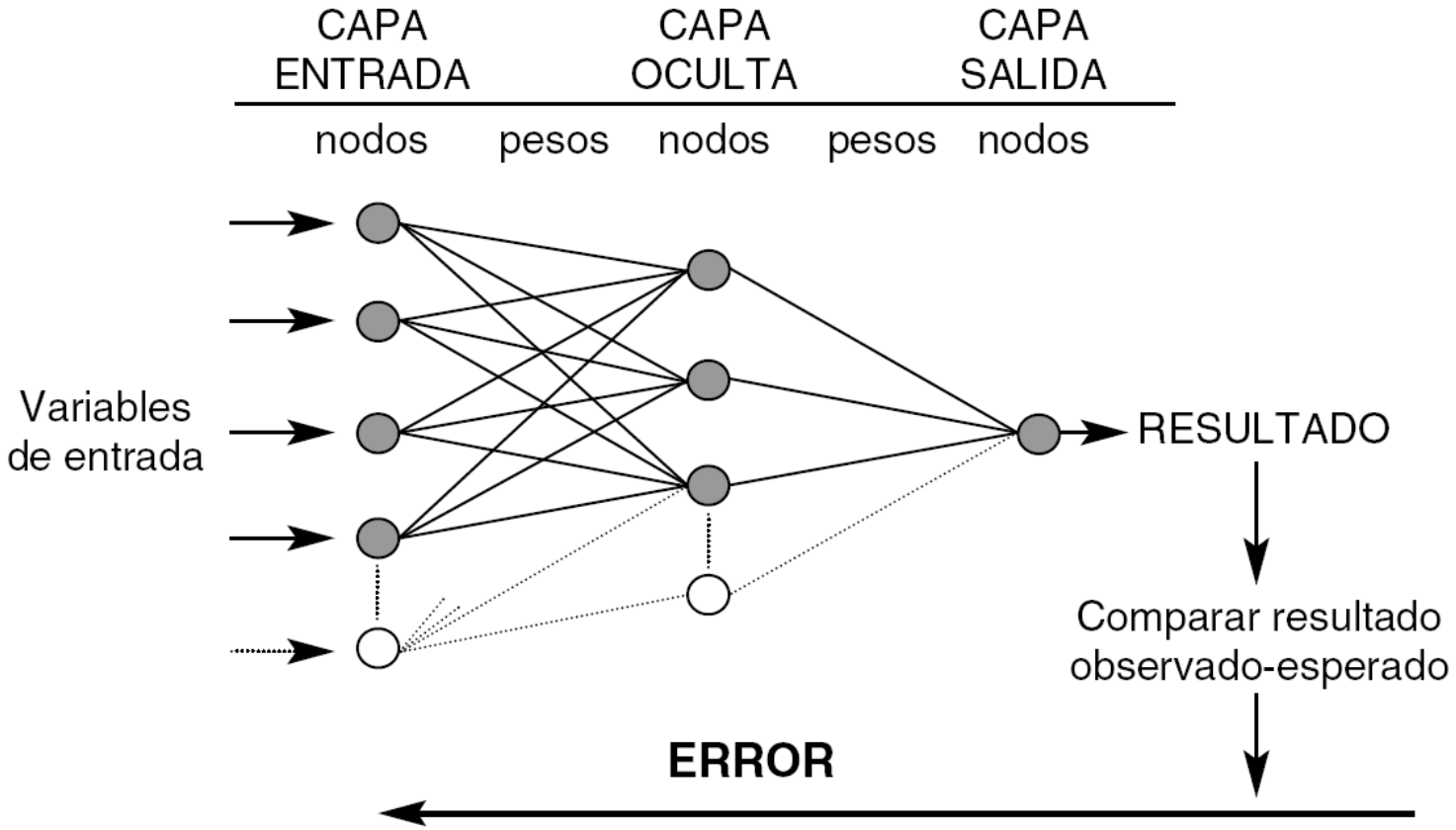
Redes de Neuronas Artificiales

Aprendizaje supervisado: Perceptrón multicapa

UNED

Nieves Pedreira Souto

Alimentación hacia delante con retropropagación del error

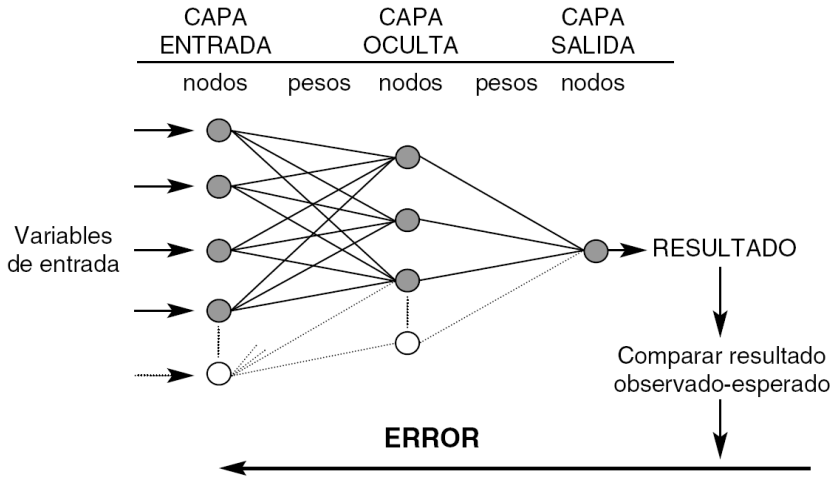


Aprendizaje supervisado: Perceptrón multicapa

Pasos para el desarrollo de la red:

1. Elegir la arquitectura
2. Elegir la función de activación. Es la función que se aplica al valor de entrada para obtener la salida
3. Podría elegirse una función de transferencia
4. Asignar valores iniciales a los pesos
5. Proceder al entrenamiento de la red y modificación de los pesos

   Se pasa el conjunto entero y se hace el cálculo del error (varias veces)
6. Si no se alcanza el error deseado, modificar la arquitectura de la red



Aprendizaje supervisado: Perceptrón multicapa

Hay que tener en cuenta que, cuanto mayor número de capas, la complejidad de la red es mayor y crece la dificultad para conseguir una configuración de pesos correcta.

Los diseños con una única capa oculta suelen ser suficientes para la resolución de la mayoría de los problemas y, sólo para problemas con una no linealidad muy acentuada, es necesario la utilización de un número de capas ocultas mayor.

El número de neuronas de las capas de entrada y salida ya viene dado por la codificación realizada de las variables de entrada y de salida de la red.

En el caso de utilizar una única capa oculta, es comúnmente aceptado el criterio de que el número necesario de neuronas para esta capa es la mitad de la suma de las neuronas utilizadas en las capas de entrada y de salida.

Este es un criterio basado en la experiencia que suele dar muy buenos resultados. Sin embargo, como criterio general que es puede no resultar adecuado para todos los problemas por lo que es necesario una experimentación en el número de neuronas a emplear en esta capa para optimizar el funcionamiento de la red.

Aprendizaje supervisado: Perceptrón multicapa

La arquitectura se elige en función de la complejidad del problema, de las entradas y las salidas

La función de activación debe ser una función no lineal. La más utilizada es la sigmoide:

$$\mathcal{F}(x) = \frac{1}{1+e^{-x}}$$

Esta función varía entre cero y uno, de manera que los valores estarían normalizados.

Se empieza el entrenamiento con valores aleatorios para los pesos (normalmente entre 0,1 y -0,1).

En el cálculo del error influye también la tasa de aprendizaje, que repercute en la velocidad de aprendizaje de la red.

Perceptrón multicapa: TASA DE APRENDIZAJE

Otro factor importante, para el proceso de entrenamiento, es la velocidad de aprendizaje de la red, que se aplica con parámetros como la "tasa de aprendizaje" o un valor de "temperatura". Estos parámetros indican el porcentaje en que se permite varíen los pesos de la red en cada época de entrenamiento. Si el valor es alto, la modificación de los pesos de una época a otra puede ser muy grande, mientras que si el valor es reducido, los pesos sólo pueden variar en pequeña proporción:

En el período de aprendizaje, valores altos de tasa de aprendizaje favorecen el acercamiento rápido a los valores óptimos de los pesos, pero no permiten el ajuste fino de estos pesos provocando un movimiento continuo alrededor del óptimo.

Valores bajos de la tasa de aprendizaje suponen un lento ajuste de los valores de los pesos pudiendo provocar la caída en mínimos locales de los que no es posible salir al tener un valor bajo de tasa de aprendizaje.

Es necesario localizar un valor de compromiso de la tasa de aprendizaje adecuado para cada problema concreto, lo suficientemente alta para acercarse al óptimo en un tiempo prudencial y evitar el riesgo de caer en mínimos locales, y suficientemente pequeño para poder tener garantías de localizar un valor cercano al óptimo.

Perceptrón multicapa. Entrenamiento

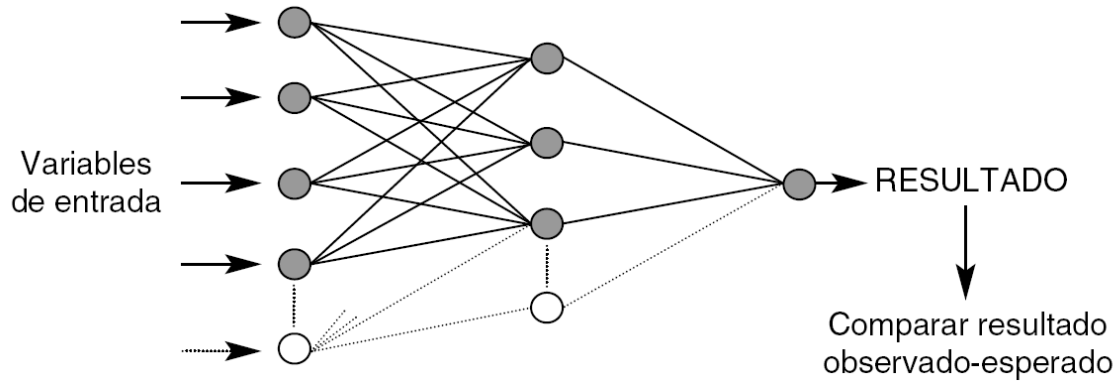
El método de aprendizaje intenta adaptar los pesos de las conexiones para minimizar el error (normalmente el error cuadrático medio) para el conjunto de ejemplos de entrenamiento

Para ello se realiza el cálculo del error de cada uno de los ejemplos, en función de la diferencia entre el valor obtenido y el deseado

El cálculo del valor de salida de cada neurona se hace aplicando la función de activación al valor de entrada

El cálculo del valor de entrada a cada neurona es el siguiente:

$$E_i^k = \sum_{j=1}^{n(c_{k-1})} s_j^{k-1} W_{ji}^k$$



El error para cada ejemplo se calcula como la diferencia entre el valor obtenido y el deseado (para cada salida de la red).

En función del error, se van recalculando los pesos hacia atrás.

$$w_{ij}^k(t+1) = w_{ij}^k(t) + \mu\delta_j^k s_i^k$$

La derivada del error indica la distancia del valor obtenido al deseado:

Valor grande: mucha diferencia Valor pequeño: poca diferencia

Valor positivo: error por exceso Valor negativo: error por defecto

El cálculo del error no es trivial:

- En la capa de salida es la diferencia entre el valor obtenido y el deseado.
- En las capas intermedias no hay salida que se pueda comparar con la obtenida, por lo que hay que hacer una estimación en función de los errores de la capa siguiente.

$$w_{ij}^k(t+1) = w_{ij}^k(t) + \mu\delta_j^k s_i^k$$

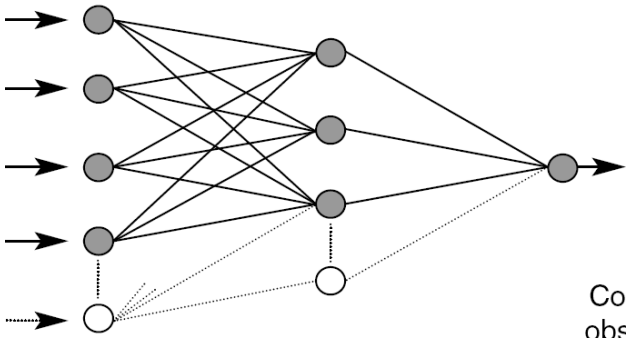
μ es la tasa de aprendizaje o ganancia (suele disminuir con el tiempo)

Puede aparecer un término de momento que provoca esa disminución.

δ se calcula de la siguiente manera:

Si la neurona j pertenece a la capa de salida:

$$\delta_j^k = f'(S_j)(\tilde{S}_j - S_j) \quad \delta_j^k = S_j(1-S_j)(\tilde{S}_j - S_j)$$



Si la neurona j pertenece a una capa oculta:

$$\delta_j^k = s_j^k(1-s_j^k)\sum_{l=0}^{L}\delta_l^{k+1}w_{jl}^{k+1}$$

Esto se realiza para cada uno de los ejemplos, y al final se calcula el error medio de la red.

El entrenamiento termina cuando el error de la red es aceptable, después de un número fijado de iteraciones o épocas, o cuando no converge.

Aprendizaje automático

Redes de Neuronas Artificiales

Aprendizaje no supervisado: Mapas autoorganizados de Kohonen

UNED

Nieves Pedreira Souto

Aprendizaje no supervisado: Mapas autoorganizados de Kohonen

Los mapas autoorganizados están compuestos por un conjunto de neuronas (o nodos) definidas por un vector de pesos, y una topología que indica la vecindad de las neuronas entre sí.

Todas las neuronas reciben el mismo vector de entradas, y su salida es la distancia euclídea entre el vector de entradas y el de pesos.

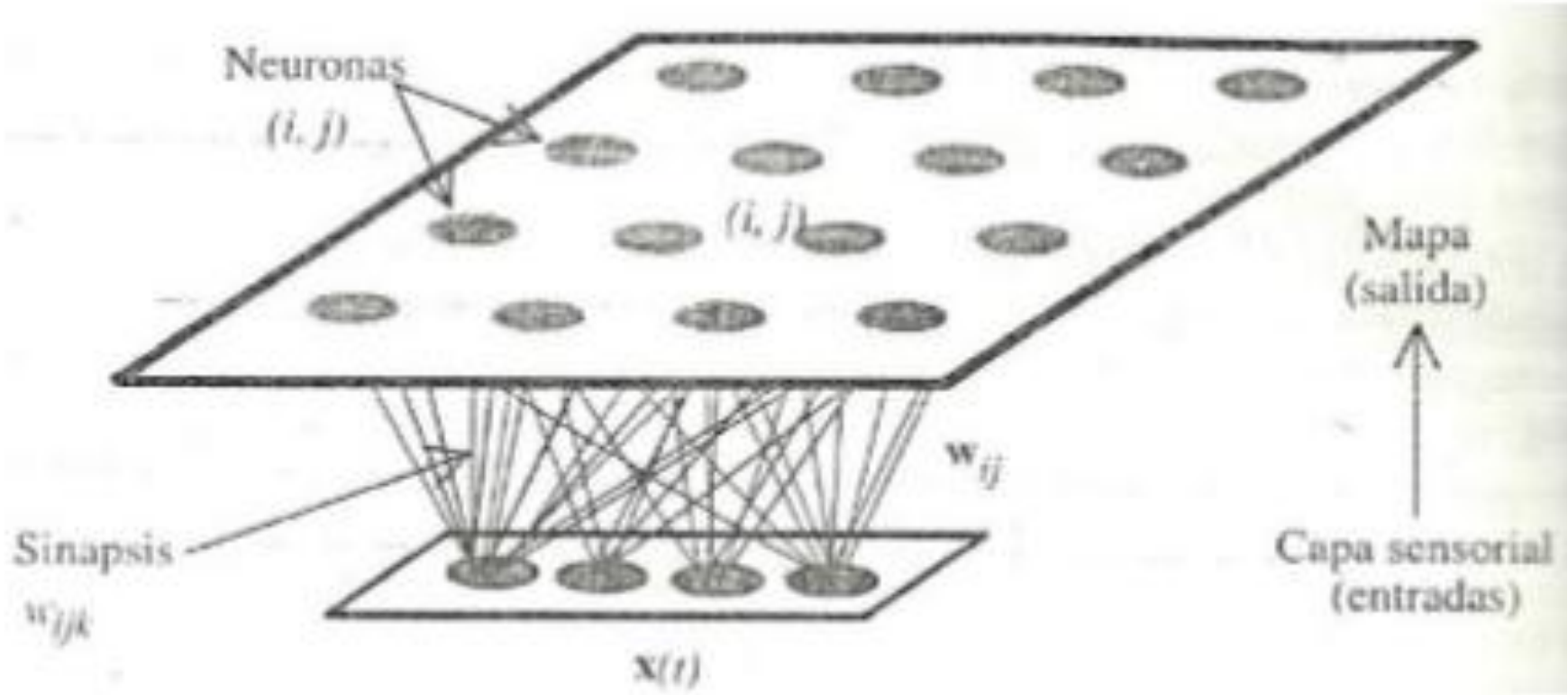
Esta red utiliza un aprendizaje no supervisado de tipo competitivo:

De todos los nodos que forman el mapa, solo uno será el responsable de generar la salida, y será aquel cuyo vector de pesos sea el más parecido a la entrada actual. Los pesos de las conexiones se ajustan en función de la neurona que haya resultado vencedora.

Los pesos de la neurona ganadora y las cercanas a ella se ajustan hacia el vector de entrada. La magnitud de los cambios se decrementan con el tiempo y con la distancia a la ganadora.

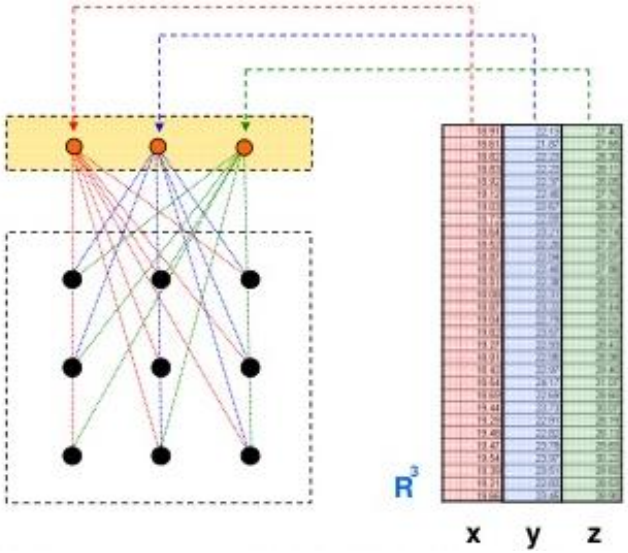
Esto se representa por la tasa de aprendizaje, que irá decrementándose con el tiempo (restándosele una cantidad fija en cada iteración, o en función del número de iteraciones), reflejado por el momento.

Mapas autoorganizados de Kohonen: arquitectura

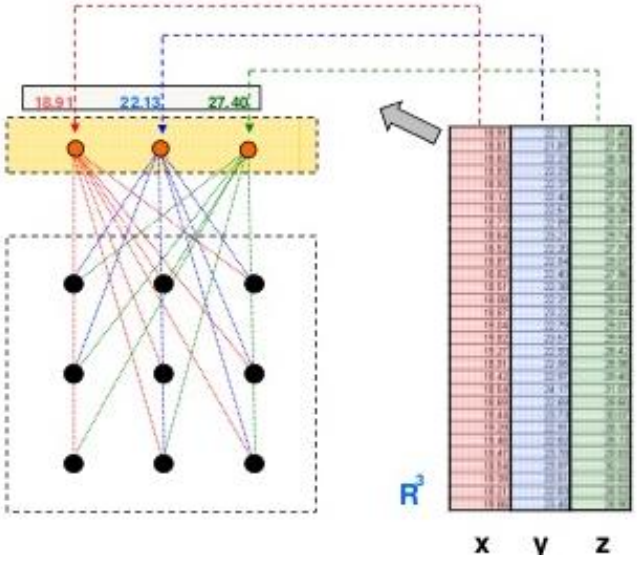


Los mapas autoorganizados son muy útiles a la hora de establecer relaciones desconocidas entre los datos de entrada, permitiendo su clasificación

Aprendizaje no supervisado: Mapas autoorganizados de Kohonen



Cada neurona de la capa de entrada esta conectada a todas las neuronas de la capa de salida.



Cada ejemplo es presentado a las neuronas de la capa de salida.

Así se buscan las neuronas mas parecidas al vector de entrada mediante un proceso de competición.

Finalmente se crean grupos con características similares.

Mapas autoorganizados de Kohonen: funcionamiento

Los pesos entre las neuronas de la capa de entrada y la capa de competición se inicializan aleatoriamente al inicio del aprendizaje, y durante el mismo serán modificados.

El vector de pesos de cada neurona de la capa de competición tiene la misma dimensión que el de entradas (todas las entradas conectadas con todas las neuronas de la capa de competición), por lo que se pueden comparar fácilmente.

La salida de cada neurona representa la distancia de su vector de pesos al vector de entrada

Diferentes funciones para este cálculo conseguirán mapas diferentes. La más habitual es la distancia euclídea.

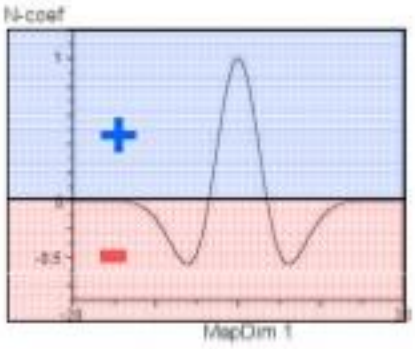
$$\tau_j = \sqrt{\sum_{i=1}^{n} (e_i - \mu_{ij})^2}$$

Mapas autoorganizados de Kohonen: entrenamiento

Para el entrenamiento, se pasa un vector de entrada y cada neurona genera su salida (la diferencia con los valores de entrada).

Gana la neurona con un valor menor. Sus pesos se verán reforzados.

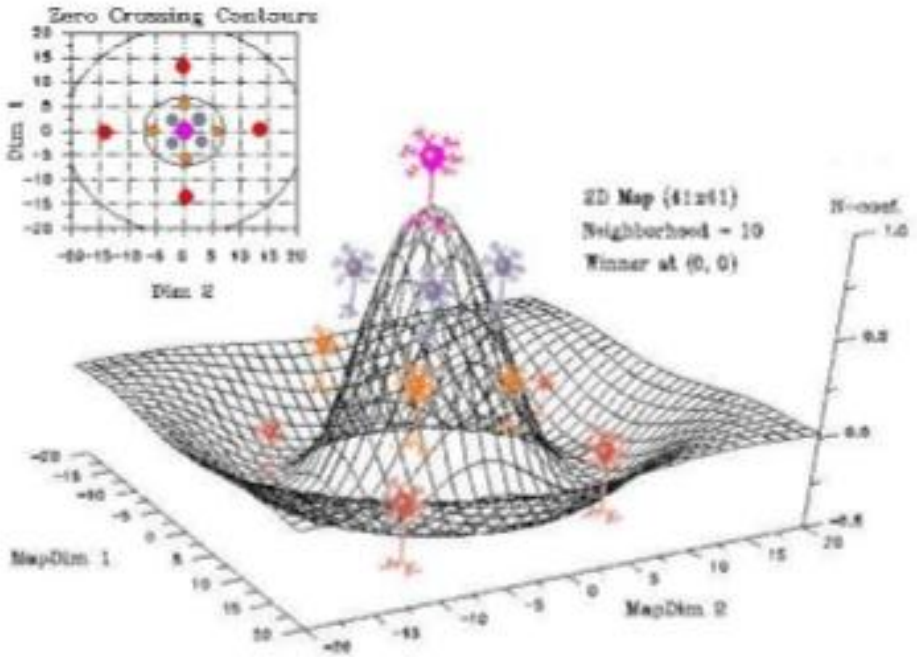
Las neuronas de la capa de competición tienen conexiones entre todas ellas. Los pesos de esas conexiones están distribuidos según una función que simula la distancia entre ellas.



Respuesta Positiva
(Excitación)

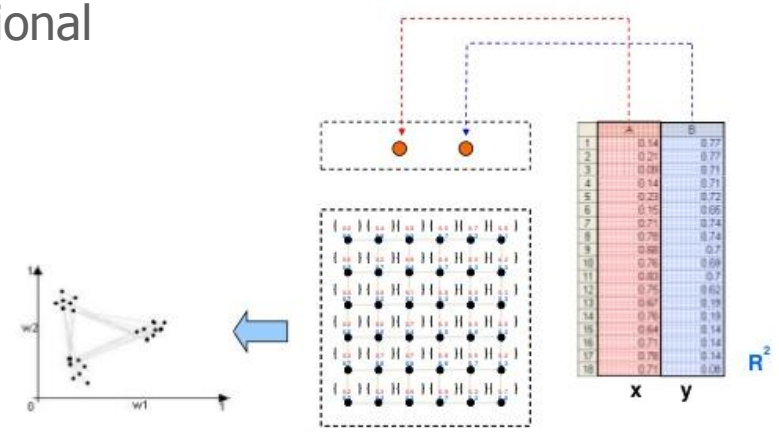
Respuesta Negativa
(Inhibición)

$$\frac{d\mu_{ij}}{dt} = \begin{cases} \frac{\alpha(t)}{d(c_i,c_j)}(\epsilon_i(t) - \mu_{ij}(t)) & si\ c_i\ ganadora \\ & y\ d(c_i,c_j) < \theta \\ 0 & en\ caso\ contrario \end{cases}$$



Aprendizaje no supervisado: entrenamiento

El hecho de que los pesos entre las neuronas de la capa de competición simulen distancias entre nodos, permite proyectar cualquier espacio multidimensional en un mapa bidimensional



Las neuronas próximas a la ganadora, también estarán más cerca del valor buscado, por lo que sus pesos también se ven reforzados (mecanismo del vecindario)

$$\frac{d\mu_{ij}}{dt} = \begin{cases} \frac{\alpha(t)}{d(c_i,c_j)}(\epsilon_i(t) - \mu_{ij}(t)) & si\ c_i\ ganadora \\ & y\ d(c_i,c_j) < \theta \\ 0 & \text{en caso contrario} \end{cases}$$

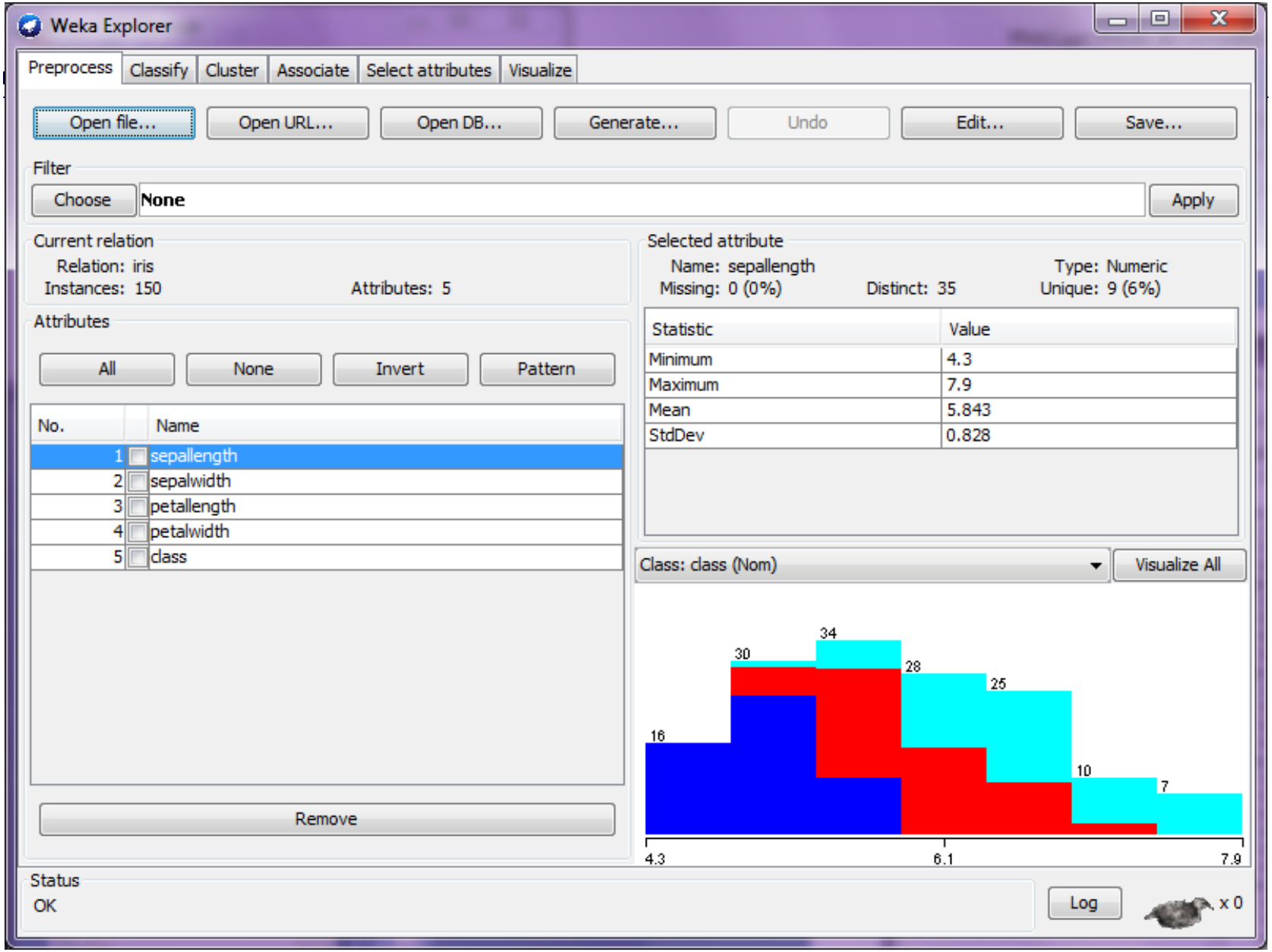
Aprendizaje automático

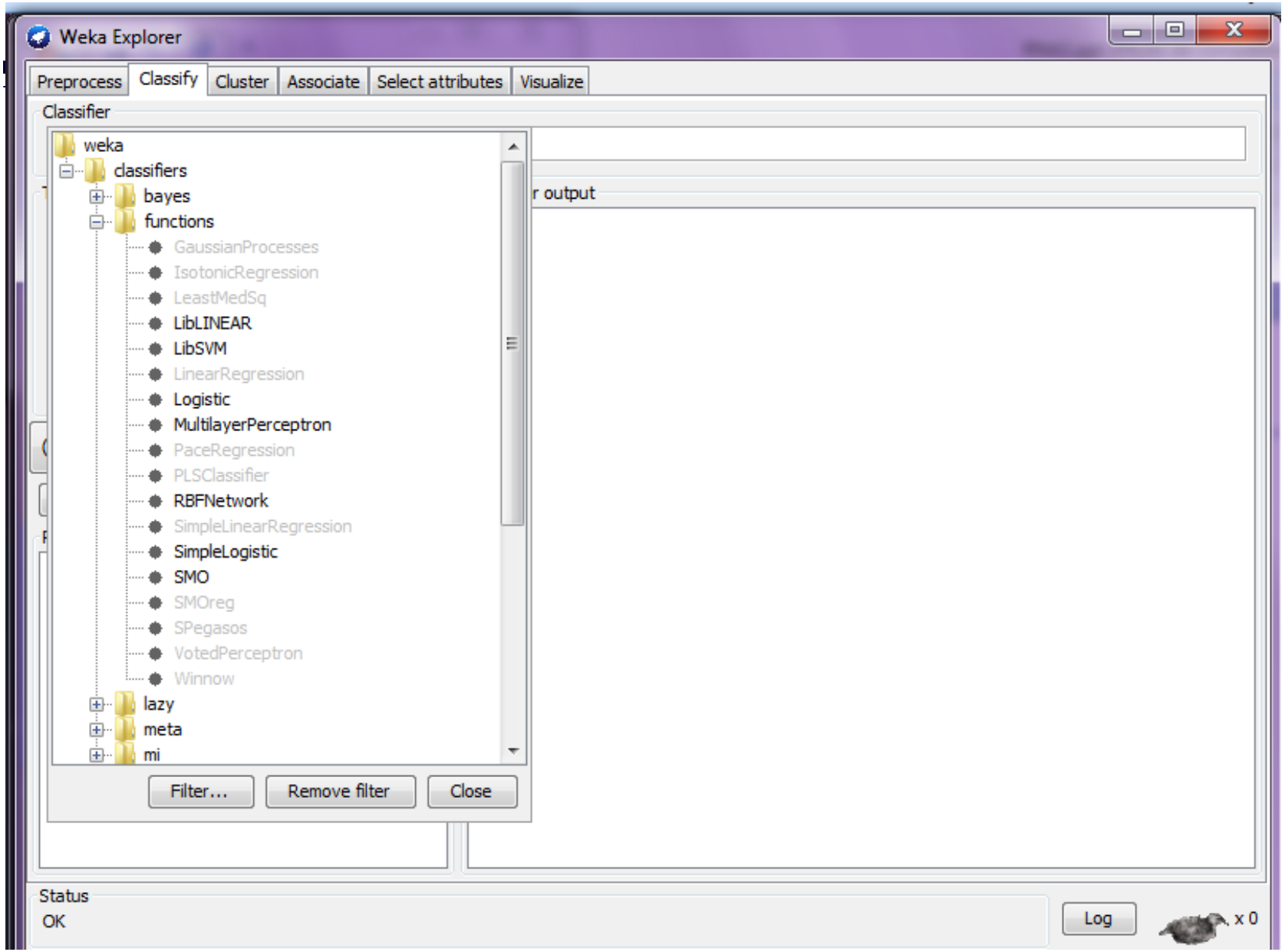
Redes de Neuronas Artificiales

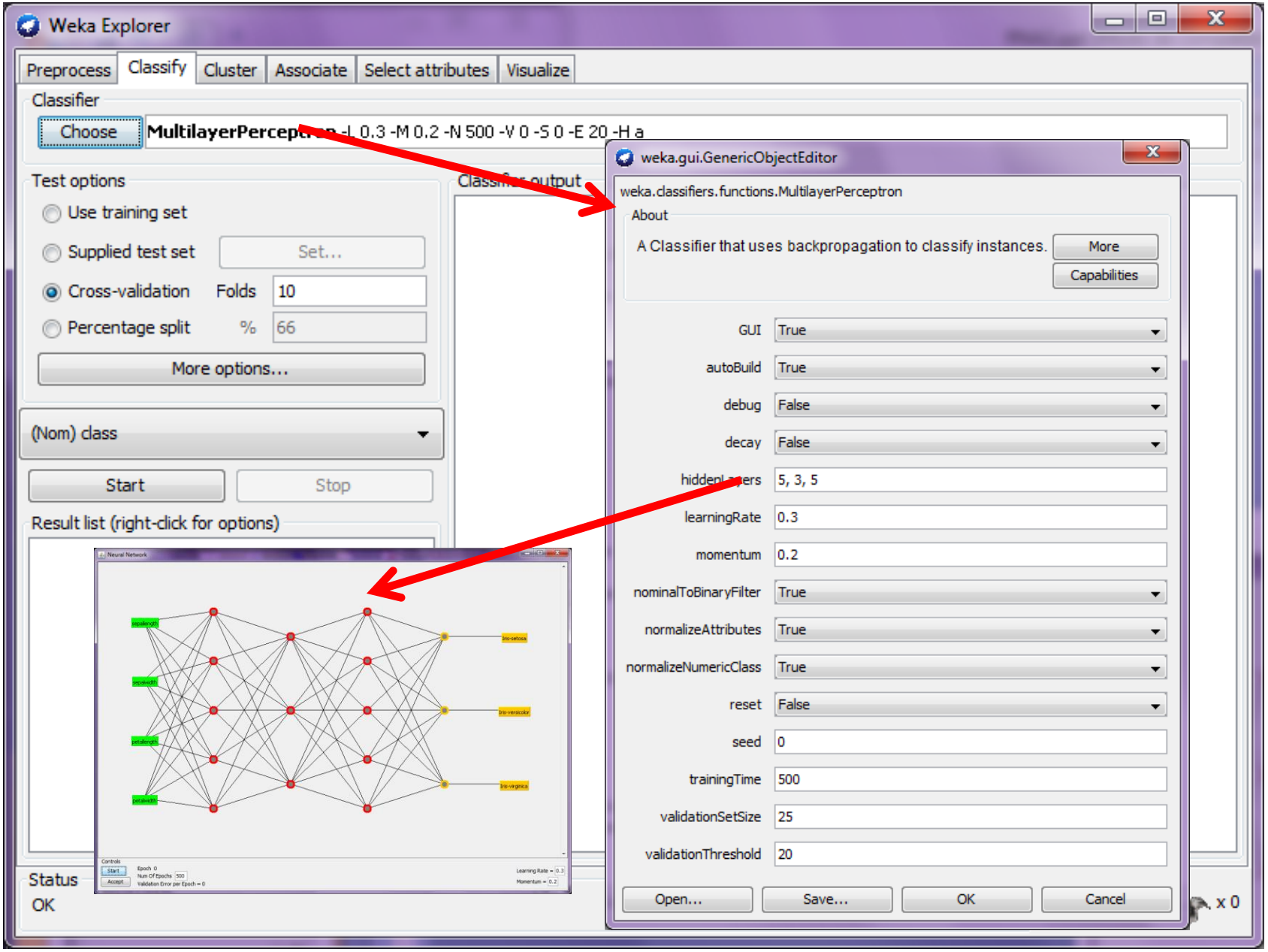
Weka: Perceptrón multicapa

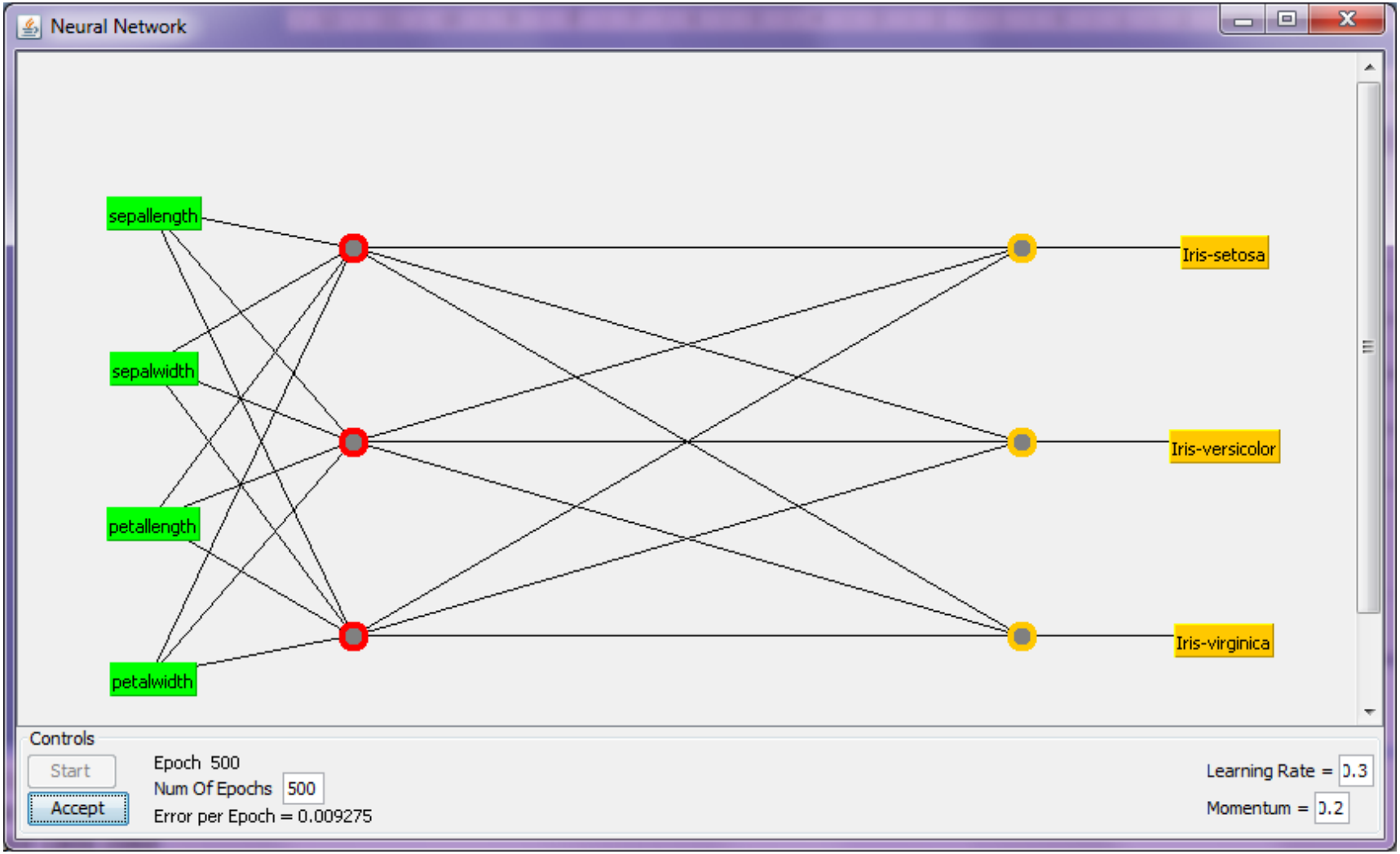
UNED

Nieves Pedreira Souto

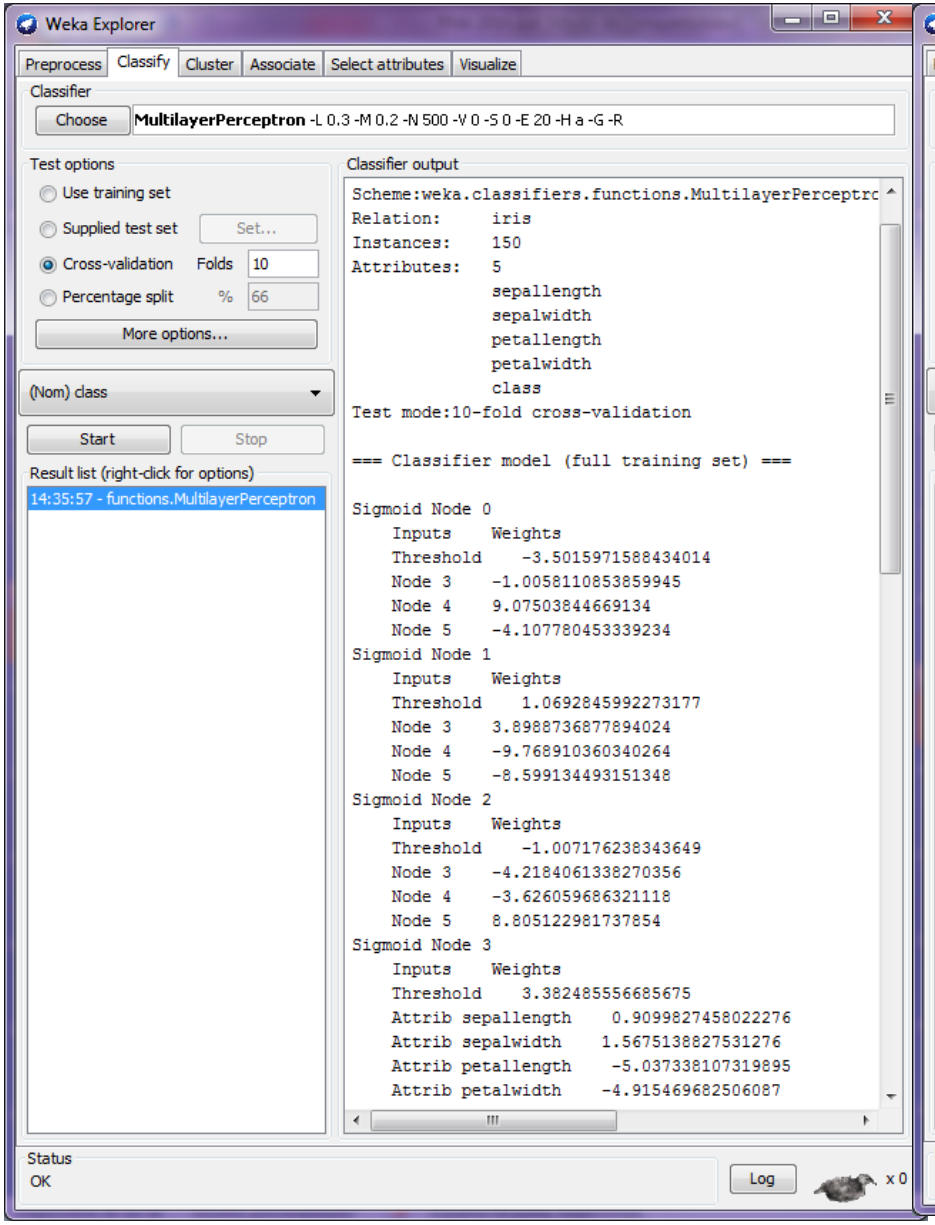

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Open file...
Open URL...
Open DB...
Generate...
Undo
Edit...
Save...
Filter
Choose
None
Apply
Current relation
Relation: iris
Instances: 150
Attributes: 5
Attributes
All
None
Invert
Pattern
No.
Name
1 sepallength
2 sepalwidth
3 petallength
4 petalwidth
5 class
Remove
Selected attribute
Name: sepallength
Type: Numeric
Missing: 0 (0%)
Distinct: 35
Unique: 9 (6%)
Statistic
Value
Minimum 4.3
Maximum 7.9
Mean 5.843
StdDev 0.828
Class: class (Nom)
Visualize All
34
30
28
25
16
10
7
4.3
6.1
7.9
Status
OK
Log
x 0

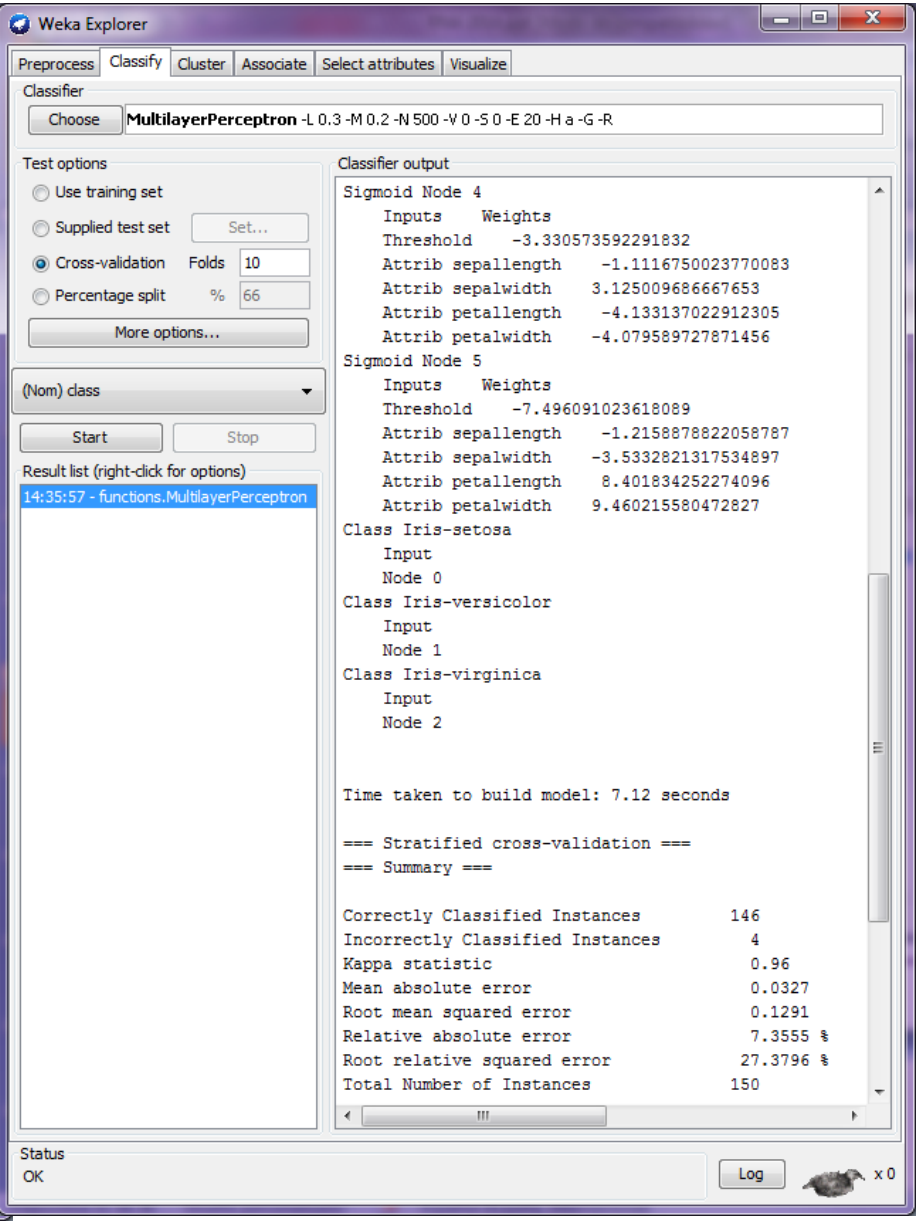

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
weka
classifiers
bayes
functions
GaussianProcesses
IsotonicRegression
LeastMedSq
LibLINEAR
LibSVM
LinearRegression
Logistic
MultilayerPerceptron
PaceRegression
PLSClassifier
RBFNetwork
SimpleLinearRegression
SimpleLogistic
SMO
SMOreg
SPegasos
VotedPerceptron
Winnow
lazy
meta
mi
r output
Filter...
Remove filter
Close
Status
OK
Log
x 0

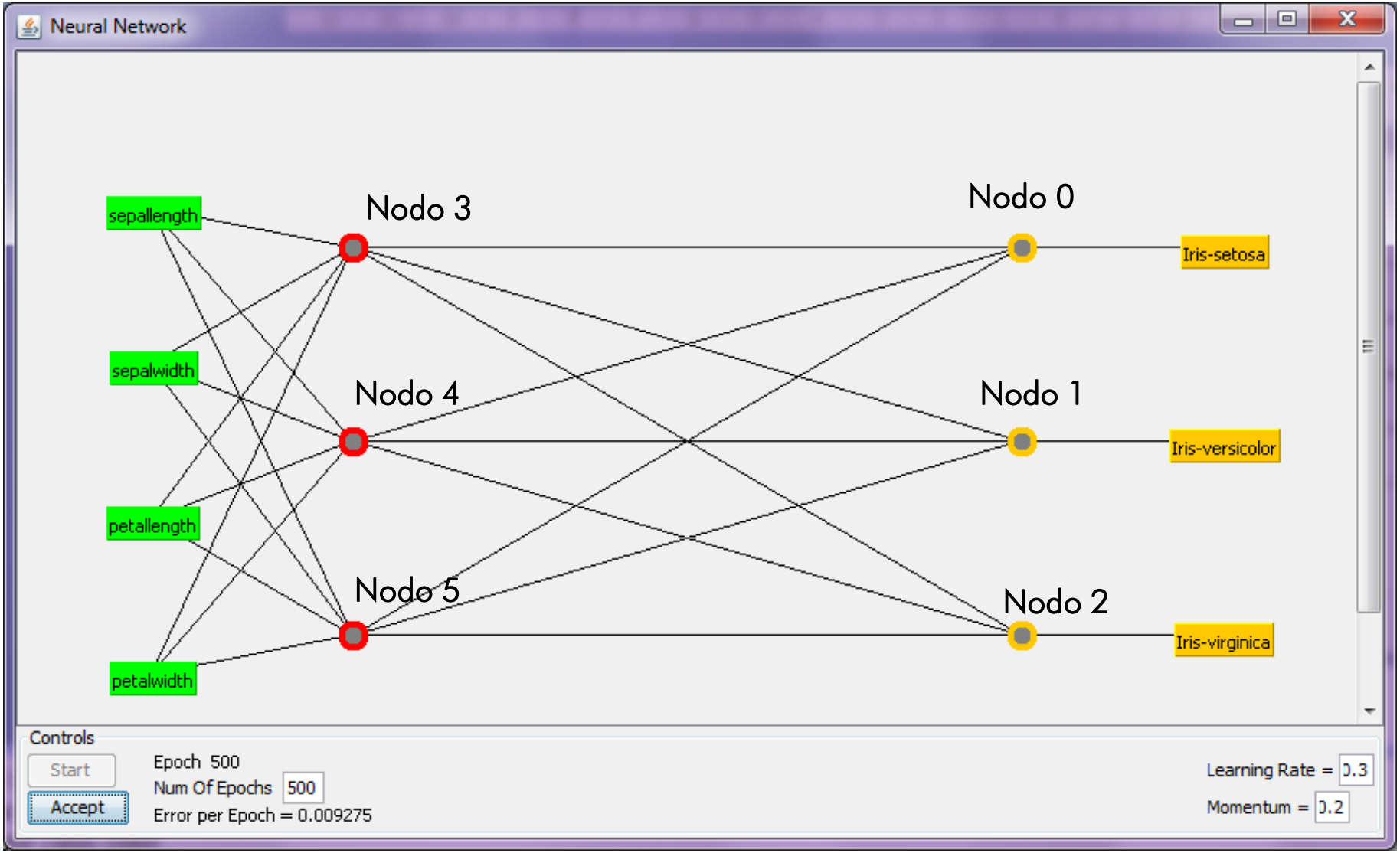

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
Status
OK
weka.gui.GenericObjectEditor
weka.classifiers.functions.MultilayerPerceptron
About
A Classifier that uses backpropagation to classify instances.
More
Capabilities
GUI
True
autoBuild
True
debug
False
decay
False
hiddenLayers
5, 3, 5
learningRate
0.3
momentum
0.2
nominalToBinaryFilter
True
normalizeAttributes
True
normalizeNumericClass
True
reset
False
seed
0
trainingTime
500
validationSetSize
25
validationThreshold
20
Open...
Save...
OK
Cancel
x 0

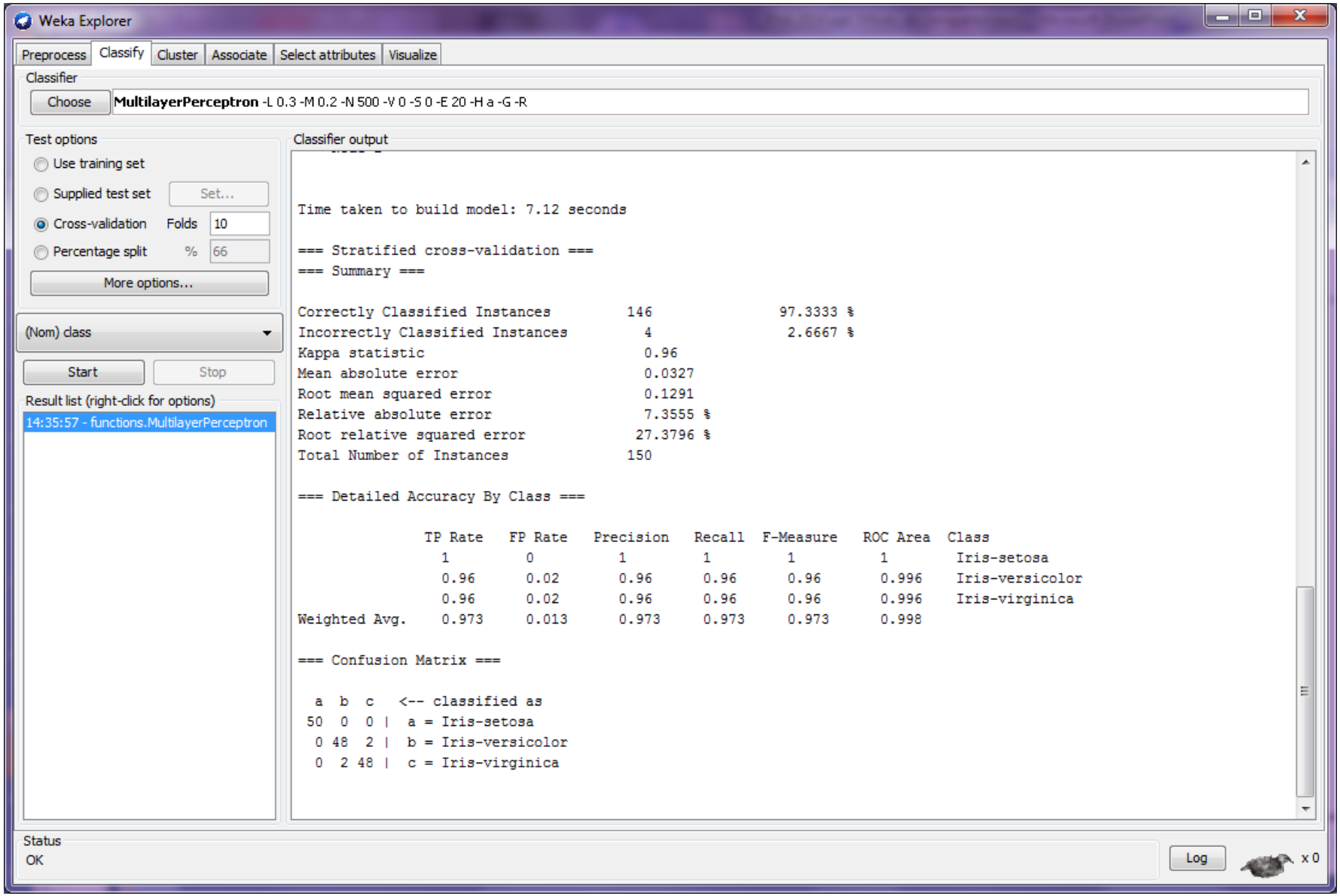


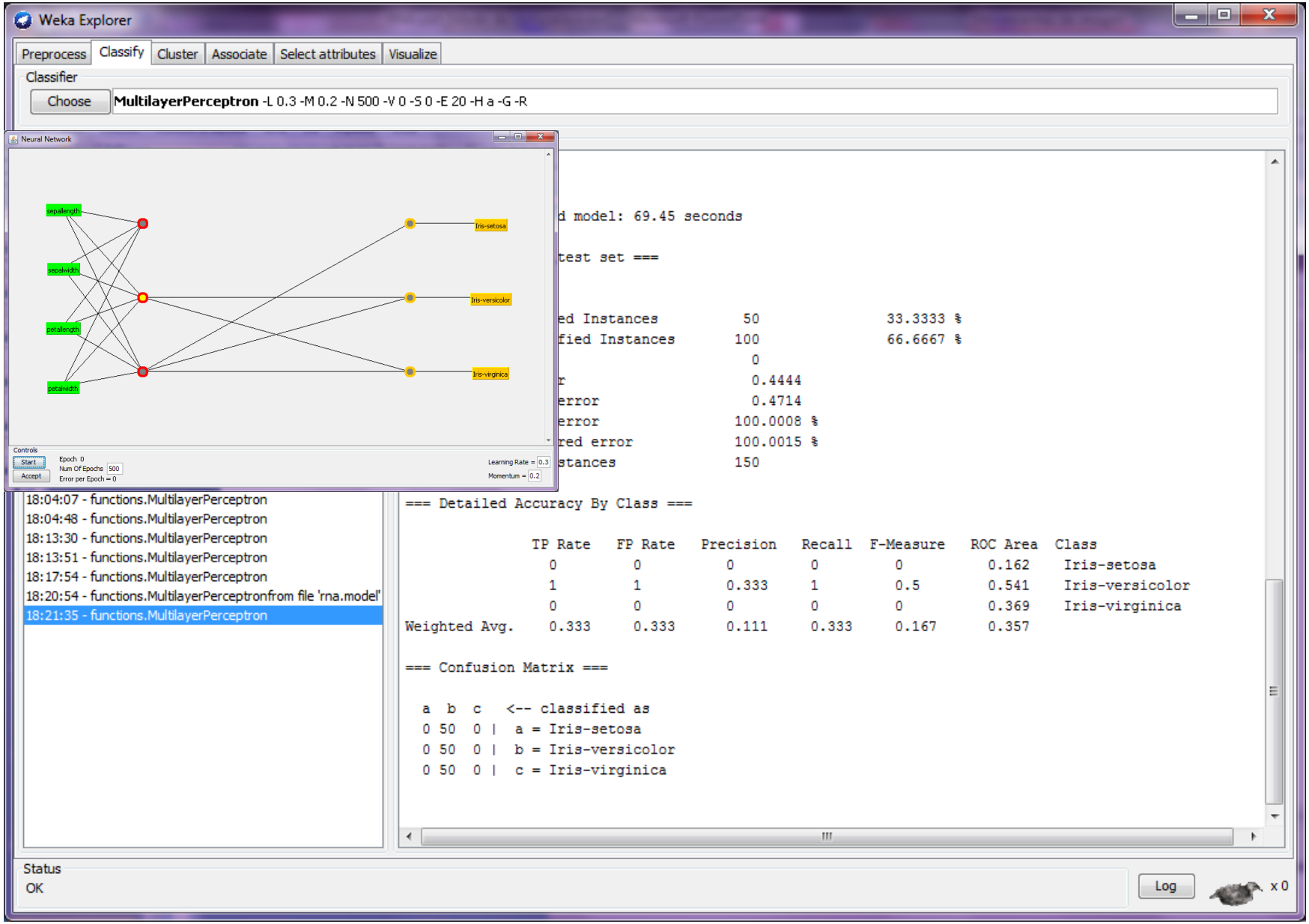
Con los valores automáticos se consigue esta red


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a -G -R
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
14:35:57 - functions.MultilayerPerceptron
Classifier output
Scheme:weka.classifiers.functions.MultilayerPerceptrc
Relation: iris
Instances: 150
Attributes: 5
sepallength
sepalwidth
petallength
petalwidth
class
Test mode:10-fold cross-validation
=== Classifier model (full training set) ===
Sigmoid Node 0
Inputs Weights
Threshold -3.5015971588434014
Node 3 -1.0058110853859945
Node 4 9.07503844669134
Node 5 -4.107780453339234
Sigmoid Node 1
Inputs Weights
Threshold 1.0692845992273177
Node 3 3.8988736877894024
Node 4 -9.768910360340264
Node 5 -8.599134493151348
Sigmoid Node 2
Inputs Weights
Threshold -1.007176238343649
Node 3 -4.2184061338270356
Node 4 -3.626059686321118
Node 5 8.805122981737854
Sigmoid Node 3
Inputs Weights
Threshold 3.382485556685675
Attrib sepallength 0.9099827458022276
Attrib sepalwidth 1.5675138827531276
Attrib petallength -5.037338107319895
Attrib petalwidth -4.915469682506087
Status
OK
Log
x 0


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a -G -R
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
14:35:57 - functions.MultilayerPerceptron
Classifier output
Sigmoid Node 4
Inputs Weights
Threshold -3.330573592291832
Attrib sepallength -1.1116750023770083
Attrib sepalwidth 3.125009686667653
Attrib petallength -4.133137022912305
Attrib petalwidth -4.079589727871456
Sigmoid Node 5
Inputs Weights
Threshold -7.496091023618089
Attrib sepallength -1.2158878822058787
Attrib sepalwidth -3.5332821317534897
Attrib petallength 8.401834252274096
Attrib petalwidth 9.460215580472827
Class Iris-setosa
Input
Node 0
Class Iris-versicolor
Input
Node 1
Class Iris-virginica
Input
Node 2
Time taken to build model: 7.12 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 146
Incorrectly Classified Instances 4
Kappa statistic 0.96
Mean absolute error 0.0327
Root mean squared error 0.1291
Relative absolute error 7.3555 %
Root relative squared error 27.3796 %
Total Number of Instances 150
Status
OK
Log
x 0


Neural Network
sepallength
sepalwidth
petallength
petalwidth
Nodo 3
Nodo 4
Nodo 5
Nodo 0
Nodo 1
Nodo 2
Iris-setosa
Iris-versicolor
Iris-virginica
Controls
Start
Accept
Epoch 500
Num Of Epochs 500
Error per Epoch = 0.009275
Learning Rate = 0.3
Momentum = 0.2


Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a -G -R
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Nom) class
Start
Stop
Result list (right-click for options)
14:35:57 - functions.MultilayerPerceptron
Classifier output
Time taken to build model: 7.12 seconds
=== Stratified cross-validation ===
=== Summary ===
Correctly Classified Instances 146 97.3333 %
Incorrectly Classified Instances 4 2.6667 %
Kappa statistic 0.96
Mean absolute error 0.0327
Root mean squared error 0.1291
Relative absolute error 7.3555 %
Root relative squared error 27.3796 %
Total Number of Instances 150
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure ROC Area Class
1 0 1 1 1 1 Iris-setosa
0.96 0.02 0.96 0.96 0.96 0.996 Iris-versicolor
0.96 0.02 0.96 0.96 0.96 0.996 Iris-virginica
Weighted Avg. 0.973 0.013 0.973 0.973 0.973 0.998
=== Confusion Matrix ===
a b c <-- classified as
50 0 0 | a = Iris-setosa
0 48 2 | b = Iris-versicolor
0 2 48 | c = Iris-virginica
Status
OK
Log
x 0

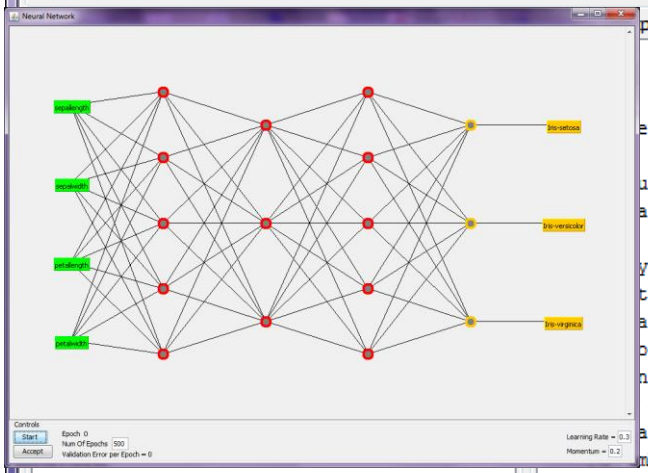

Weka Explorer
Preprocess
Classify
Cluster
Associate
Select attributes
Visualize
Classifier
Choose
MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a -G -R
Neural Network
sepallength
sepalwidth
petallength
petalwidth
Iris-setosa
Iris-versicolor
Iris-virginica
Controls
Start
Accept
Epoch 0
Num Of Epochs 500
Error per Epoch = 0
Learning Rate = 0.3
Momentum = 0.2
d model: 69.45 seconds
test set ===
ed Instances 50 33.3333 %
fied Instances 100 66.6667 %
0
r 0.4444
error 0.4714
error 100.0008 %
red error 100.0015 %
stances 150
18:04:07 - functions.MultilayerPerceptron
18:04:48 - functions.MultilayerPerceptron
18:13:30 - functions.MultilayerPerceptron
18:13:51 - functions.MultilayerPerceptron
18:17:54 - functions.MultilayerPerceptron
18:20:54 - functions.MultilayerPerceptronfrom file 'rna.model'
18:21:35 - functions.MultilayerPerceptron
=== Detailed Accuracy By Class ===
TP Rate FP Rate Precision Recall F-Measure ROC Area Class
0 0 0 0 0 0.162 Iris-setosa
1 1 0.333 1 0.5 0.541 Iris-versicolor
0 0 0 0 0 0.369 Iris-virginica
Weighted Avg. 0.333 0.333 0.111 0.333 0.167 0.357
=== Confusion Matrix ===
a b c <-- classified as
0 50 0 | a = Iris-setosa
0 50 0 | b = Iris-versicolor
0 50 0 | c = Iris-virginica
Status
OK
Log
x 0

Weka Explorer

Preprocess | Classify | Cluster | Associate | Select attributes | Visualize

Classifier

Choose **MultilayerPerceptron** -L 0.3 -M 0.2 -N 500 -V 25 -S 0 -E 20 -H "5, 3, 5" -G -R




```
17:56:43 - functions.MultilayerPerceptron
18:04:07 - functions.MultilayerPerceptron
18:04:48 - functions.MultilayerPerceptron
18:13:30 - functions.MultilayerPerceptron
18:13:51 - functions.MultilayerPerceptron
18:17:54 - functions.MultilayerPerceptron
18:20:54 - functions.MultilayerPerceptronfrom file 'rna.model'
18:21:35 - functions.MultilayerPerceptron
18:26:31 - functions.MultilayerPerceptron
18:30:55 - functions.MultilayerPerceptron
18:32:38 - functions.MultilayerPerceptron
18:33:55 - functions.MultilayerPerceptron
18:36:05 - functions.MultilayerPerceptron
```

put

2

en to build model: 4.53 seconds

uation on training set ===

ary ===

y Classified Instances	50	33.3333 %
tly Classified Instances	100	66.6667 %
atistic	0	
olute error	0.4444	
n squared error	0.4717	
absolute error	100	%
ative squared error	100.0556	%
mber of Instances	150	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0	0	0	0	0	0.826	Iris-setosa
	1	1	0.333	1	0.5	0.479	Iris-versicolor
	0	0	0	0	0	0.914	Iris-virginica
Weighted Avg.	0.333	0.333	0.111	0.333	0.167	0.74	

=== Confusion Matrix ===

a	b	c	<-- classified as
0	50	0	a = Iris-setosa
0	50	0	b = Iris-versicolor
0	50	0	c = Iris-virginica

Status

OK

Log

x 0