



Escuela Politécnica
Superior (EPS)



UNIVERSIDAD DE CÓRDOBA
Departamento de
Informática y
Análisis Numérico

Práctica 4. Árboles y Redes Neuronales

Juan Carlos Fernández Caballero (jfcaballero@uco.es)
Introducción al Aprendizaje Automático (IAA)
3º de Grado de Ingeniería Informática
Especialidad en Computación
mayo de 2020



GRUPO DE INVESTIGACIÓN AYRNA
APRENDIZAJE Y REDES NEURONALES ARTIFICIALES
uco.es/ayrna

Índice de contenidos

Árboles de decisión

Entrenamiento en árboles de decisión

Árboles de decisión en Weka

Redes Neuronales

Redes en Weka

Entregables

Bibliografía

Árboles de decisión

Entrenamiento en árboles de decisión

Árboles de decisión en Weka

Redes Neuronales

Redes en Weka

Entregables

Bibliografía

Qué son los árboles de decisión

Son modelos de **clasificación** representados mediante reglas **IF-THEN**, cuyas características son las siguientes:

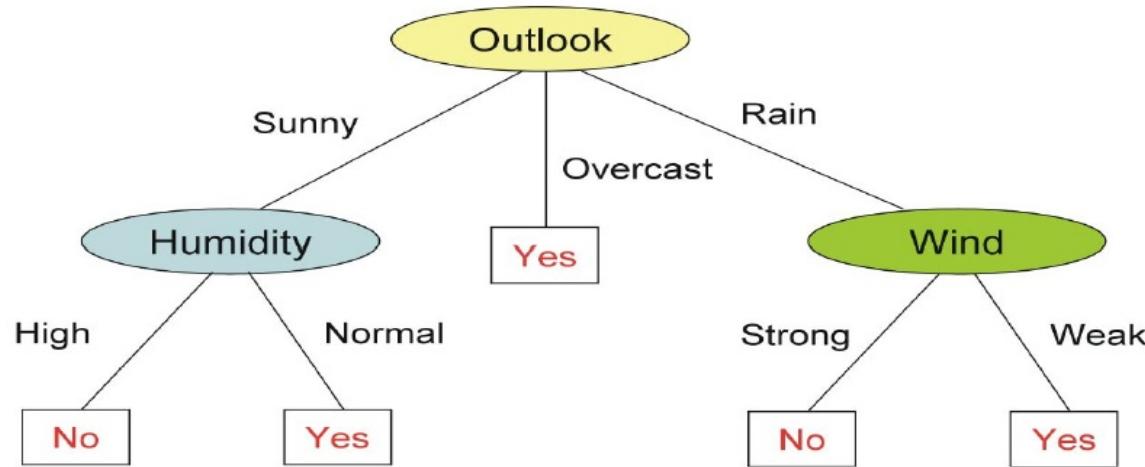
- Las **reglas** se representan mediante un **árbol**.
 - Muy **usados por la comunidad científica** por ser bien **interpretables** → Importante para los médicos en **medicina**.
 - El procedimiento de creación del arbol hace que estos modelos sean **robustos** a:
 - ▶ **Ruido** en los atributos de los patrones.
 - ▶ Patrones mal etiquetados (**outliers**).
 - Trabajan con atributos **nominales** o **discretizados**.
 - ▶ Necesidad de **preprocesamiento** en algunos casos.
 - En **Minería de Datos**, en vez de usarse para clasificar suelen usarse como *métodos descriptivos* (reglas) para la toma de decisiones.

Qué son los árboles de decisión

- Los árboles pueden ser más **apropiados** cuando **cada atributo toma un número pequeño de valores**.
 - Aunque la función objetivo debe ser discreta, existen algoritmos que permiten construir árboles de decisión cuando la **variable de salida es continua** (**árboles de regresión**).
 - Algunos árboles permiten trabajar con **valores perdidos** y con **variables independientes continuas** (C4.5).
 - Si algunas clases dominan se pueden crear árboles sesgados. A veces es necesario **equilibrar** el conjunto de datos previamente.
 - Métodos más representativos en la literatura:
 - ▶ ID3, C4.5, Random Forest, Logistic Model Tree.

Representación gráfica del árbol

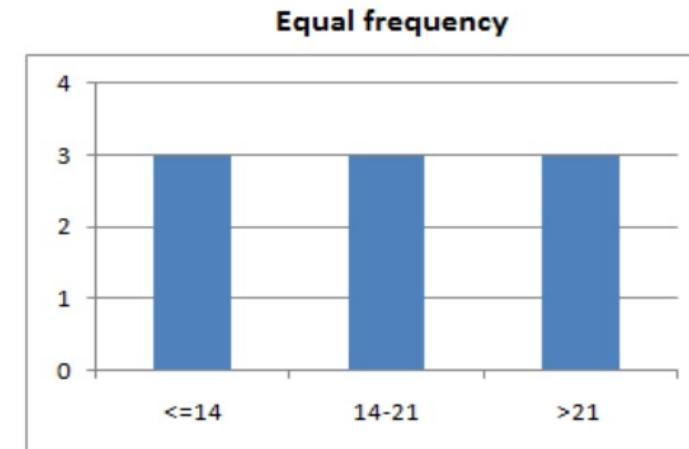
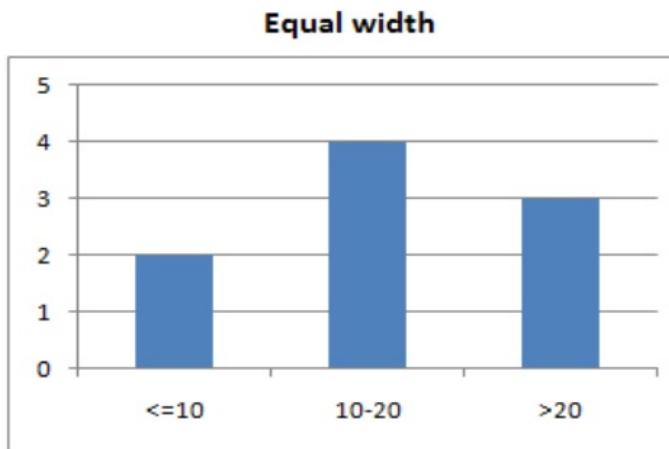
- **Nodos**: Especifican los nombres de los **atributos de entrada**.
 - **Ramas**: Indican los posibles **valores del atributo asociado al nodo**.
 - **Hojas**: También denominados *nodos terminales*, indican la **clase en la que se clasifican** las instancias.



Recordatorio: Discretización

- Igual amplitud.

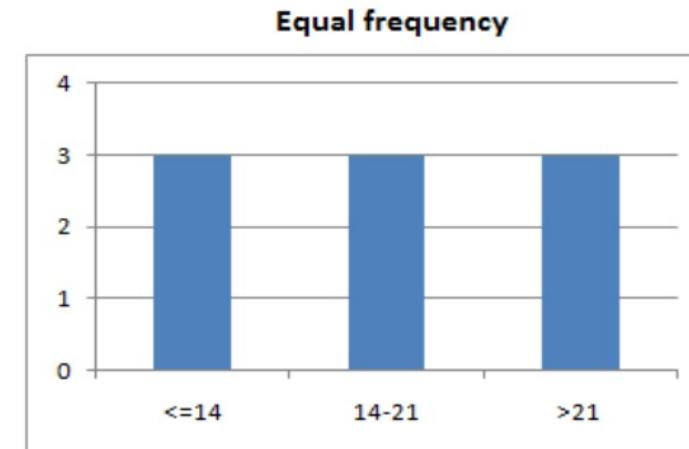
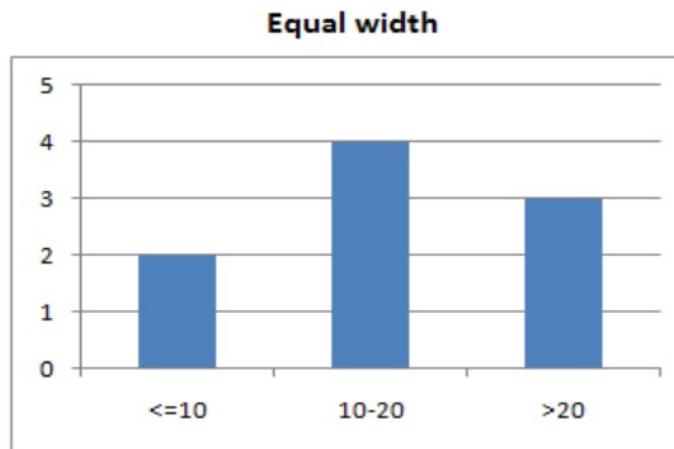
- ▶ Divide el intervalo en k **intervalos del mismo ancho**.
 - ▶ Si m es el valor mínimo y M es el valor máximo, el ancho será $W = \frac{M-m}{k}$.
 - ▶ Es la forma más simple, pero los **outliers** pueden influir en la conversión, ya que **su valor es determinante en el ancho**.
 - ▶ Además, puede generar **desbalanceo** de las categorías generadas, **pocos o muchos patrones en un intervalo**.



Recordatorio: Discretización

- **Igual frecuencia.**

- ▶ Divide el intervalo en k intervalos de distinto ancho, tratando de generar **categorías balanceadas** en cuanto al **número de patrones por categoría**.
- ▶ Es decir, se fuerza a que, tras la discretización, el número de ejemplos en cada categoría sea, aproximadamente, el mismo.
- ▶ **Problema:** Quizás haya patrones que debieran entrar en otro intervalo → Necesidad de **experto** en el problema.



Recordatorio: Discretización en Weka

Weka dispone de **dos filtros para discretización** en amplitud y frecuencia:

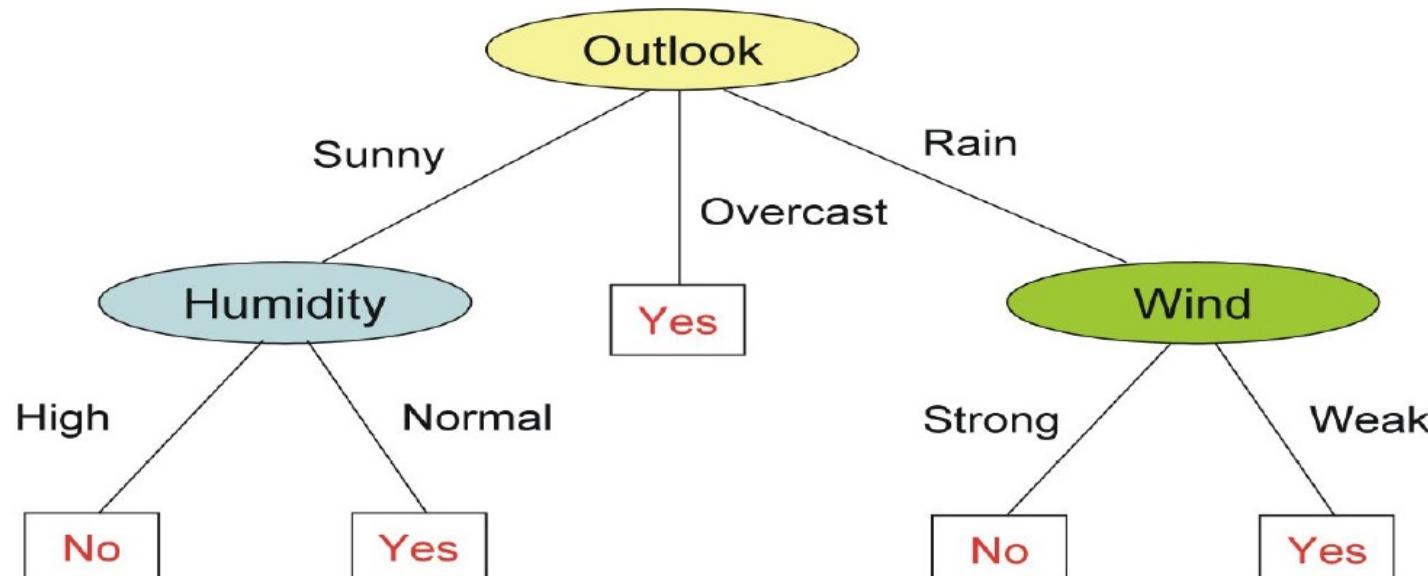
- **filters/supervised/attribute/Discretize**: Discretiza por frecuencia.
- **filters/unsupervised/attribute/Discretize**: Discretiza atributos por amplitud y frecuencia.

Como ejercicio escoja una base de datos que se pueda discretizar y compare cómo queda al usar estos filtros.

Ejemplo de árbol de decisión

Suponga un problema de **clasificación binaria**: Decidir si **jugar al tenis o no** en función de **3 variables independientes** que se relacionan con el **tiempo**.

- **Atributos de entrada:**
 - ▶ **Temperatura:** Tipo nominal {Fría, Intermedia, Alta}
 - ▶ **Humedad:** Tipo nominal binario {Alta, Normal}
 - ▶ **Viento:** Tipo nominal binario {Sí, No}
 - **Atributo de salida:**
 - ▶ La variable a predecir es la decisión binaria de **jugar o no al tenis** (biclase).



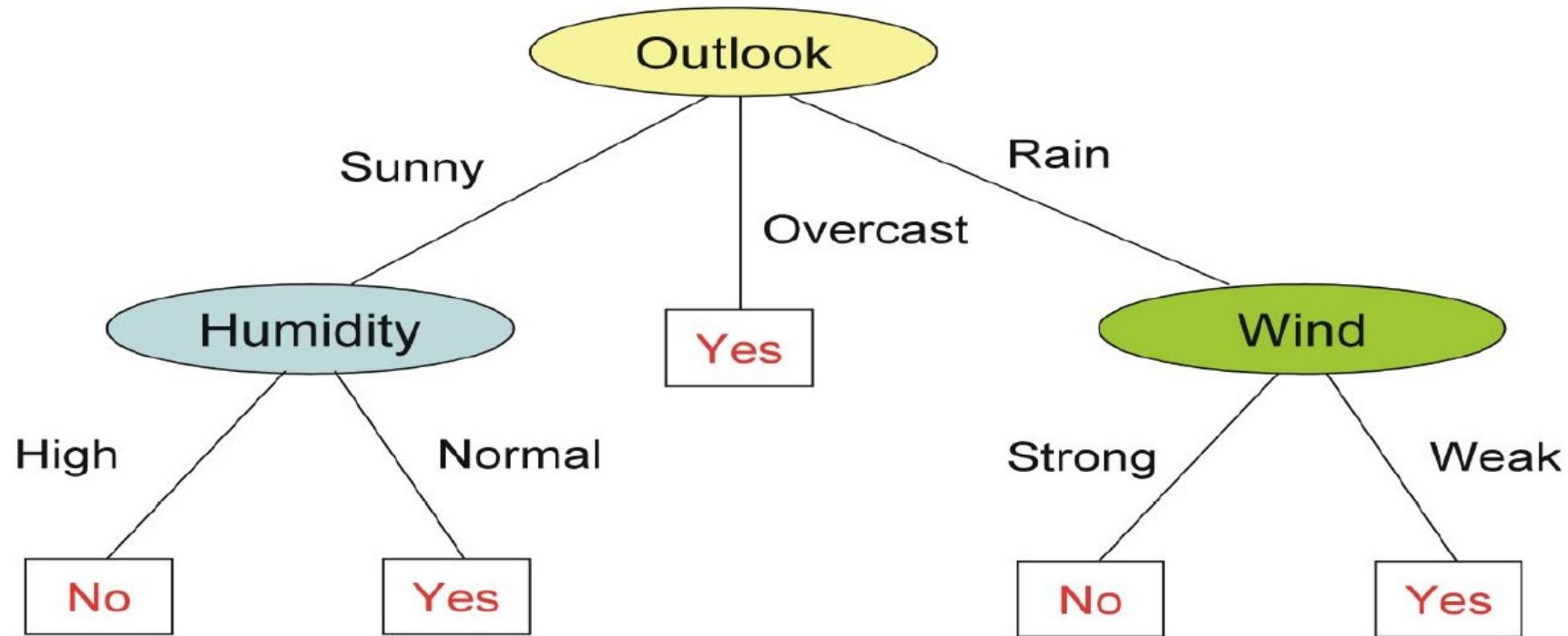
Interpretación del ejemplo

El árbol (modelo) representado en la gráfica anterior se interpreta con **5 reglas** de tipo **IF-THEN**:

1. **R1: IF** (Outlook=Sunny) **AND** (Humidity=High) **THEN**
PlayTennis=No.
2. **R2: IF** (Outlook=Sunny) **AND** (Humidity=Normal) **THEN**
PlayTennis=Yes.
3. **R3: IF** (Outlook=Overcast) **THEN** PlayTennis=Yes.
4. **R4: IF** (Outlook=Rain) **AND** (Wind=Strong) **THEN**
PlayTennis=No.
5. **R5: IF** (Outlook=Rain) **AND** (Wind=Weak) **THEN**
PlayTennis=Yes.

Interpretación de un patrón en el modelo

Supongamos un patrón de *test* con valores: ***Outlook=Sunny***, ***Temperature=Mild***, ***Humidity=Normal*** y ***Wind=Strong***.



La etiqueta real del patrón no visto (*test*) es **NO**.
¿Acierta nuestro modelo? → **NO**.

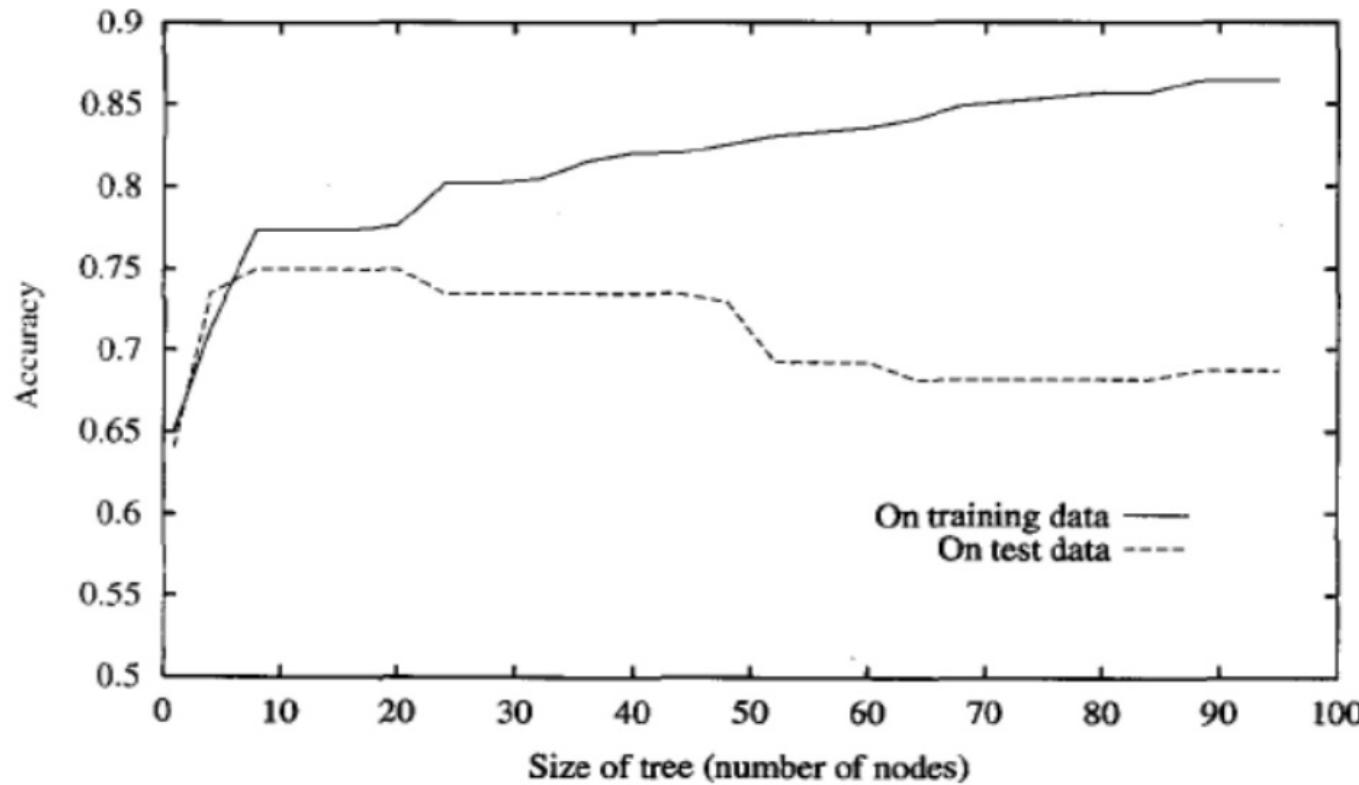
Poda en árboles de decisión

En ocasiones es necesario “**podar**” las ramas de un árbol:

- **Objetivo:** Prevenir el **sobre-entrenamiento** o una generalización incorrecta y/o un modelo **poco interpretable y complejo**. Esto se puede deber a:
 - ▶ Un **ajuste excesivo** a los datos.
 - ▶ Un **ruido excesivo** en los datos.
- Para ello se realizan **cálculos iterativos con diferentes subárboles**, de forma que sus **errores cometidos** ayuden a la decisión de podar.
- Tanto la **construcción** de un árbol como su **poda** se hace siempre con los **patrones del conjunto de entrenamiento**.

Poda en árboles de decisión

Sobreentrenamiento de un árbol al aumentar el número de nodos y ajustarse demasiado a los datos:



Poda en árboles de decisión

De manera general, existen dos **estrategias** para realizar la poda:

- **Pre-poda:** Se detiene el entrenamiento y crecimiento del árbol antes de que llegue a adaptarse perfectamente al conjunto de *training*.
Se utilizan para ello **técnicas iterativas durante el entrenamiento** que comprueban cuando la **ganancia de información no se considera significante** al podar.
- **Post-poda:** Permiten que el árbol se sobreajuste al conjunto de *training* y luego se efectúa sobre él una poda.

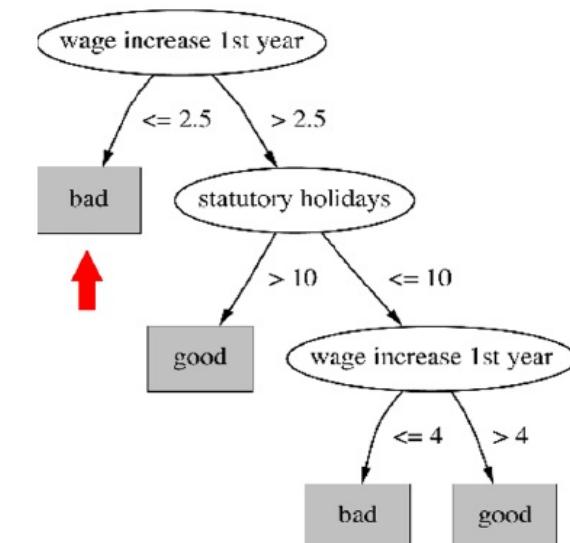
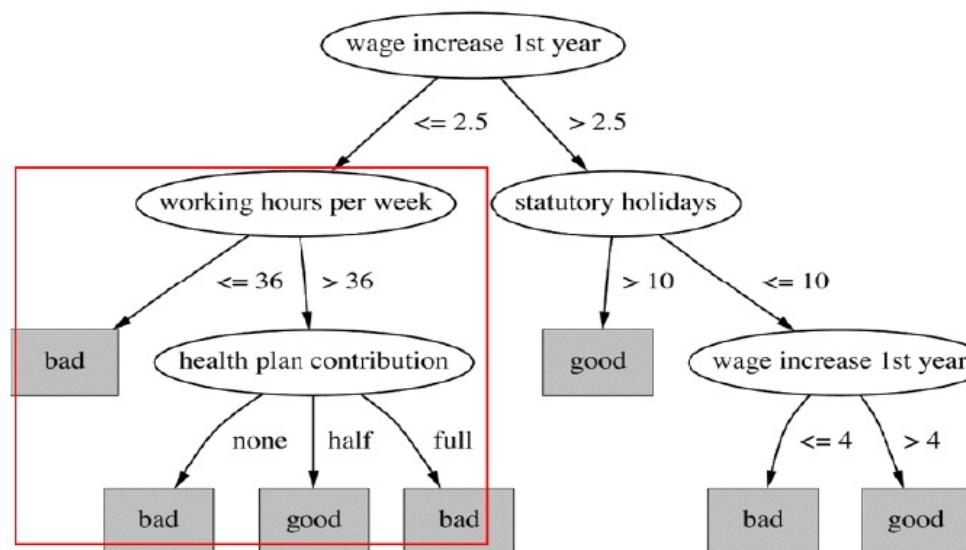
Para ello se usa un **árbol de decisión completo** y posteriormente se comprueba el **rendimiento de subárboles**. Si un subárbol iguala o mejora al árbol original se lleva a cabo la poda.

Existen varias técnicas, distinguiéndose: **1) Las de reemplazo de subárbol y 2) Las de elevación de subárbol.**

Normalmente se prefiere “**post-poda**”, ya que “**pre-poda**” podría parar el crecimiento del árbol **prematuramente**.

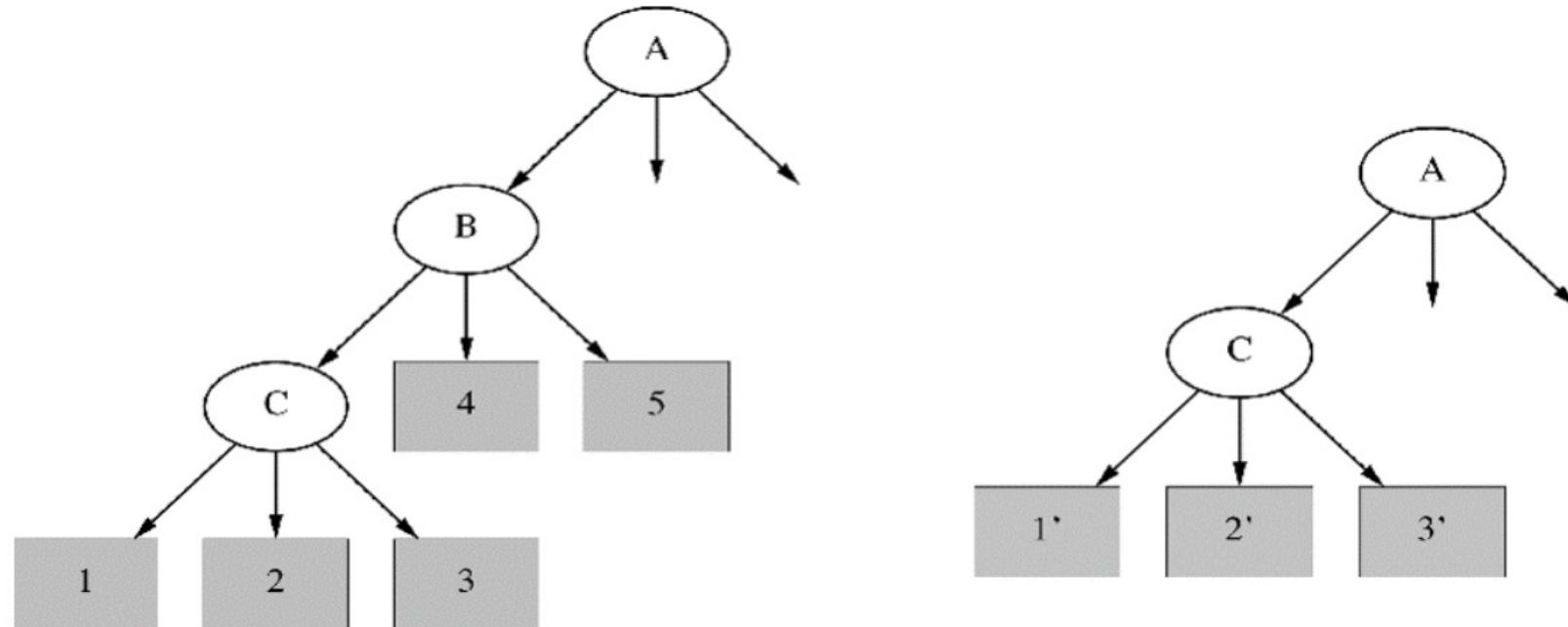
Ejemplo de reemplazo del subárbol con post-poda

- Toda la parte izquierda del árbol se sustituye en este caso por una etiqueta u hoja final.
- Los patrones son clasificados en la clase mayoritaria (*bad* en este caso).



Ejemplo de elevación de subárbol con post-poda

- En este ejemplo el nodo C es elevado, **reemplazando** al nodo B, que desaparece, disminuyendo por tanto la **profundidad** del árbol.
 - Los **patrones** de las hojas 4 y 5 tienen que ser reclasificados en las hojas 1', 2' y 3'.



Árboles de decisión

Entrenamiento en árboles de decisión

Árboles de decisión en Weka

Redes Neuronales

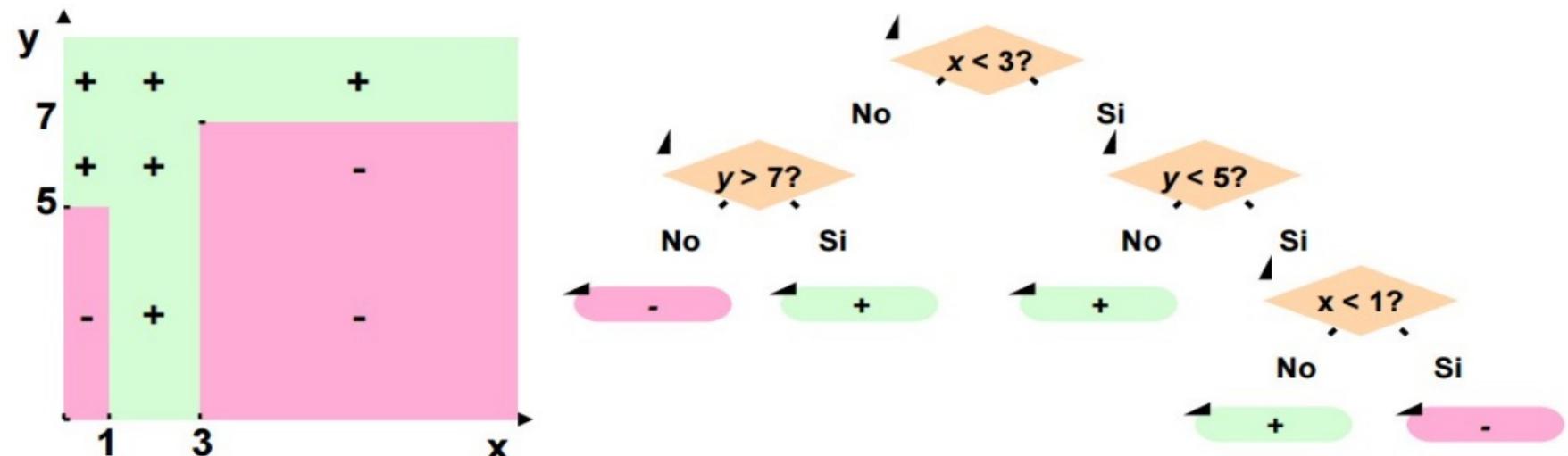
Redes en Weka

Entregables

Bibliografía

Entrenamiento de los árboles de decisión

El entrenamiento consiste en **dividir el espacio de ejemplos en rectángulos** paralelos a los ejes (variables independientes) y asignar una clase a cada uno de ellos.



Entrenamiento de los árboles de decisión

- En la creación y entrenamiento de un árbol, para la división del espacio en rectángulos, se usa el conjunto de datos de entrenamiento (*training*) etiquetados.
- Se necesitan **procesos iterativos** sobre los datos, que van obteniendo ramas y subconjuntos de datos homogéneos respecto a la clase.
 - ▶ **Greedy o Voraz** ([Wikipedia](#)).
 - ▶ **Divide y Vencerás** ([Wikipedia](#)).
- Los procesos iterativos se basan en la **Ganancia de Información**, que depende de la **Entropía**.

Entrenamiento de los árboles de decisión

Entropía:

- Medida de incertidumbre, relacionada con:
 - ▶ **1) Pureza:** Cómo de cerca está un conjunto de pertenecer a una misma clase.
 - ▶ **2) Impureza (desorden):** Cómo de cerca está un conjunto de la incertidumbre total.
- La Entropía es una medida:
 - ▶ **1) Directamente proporcional** a la impureza, incertidumbre, irregularidad.
 - ▶ **2) Inversamente proporcional** a la pureza, certidumbre, redundancia.
- **Objetivo:** Minimizar la Entropía (desorden o error).

Entrenamiento de los árboles de decisión

A partir del conjunto de **entrenamiento** y de forma general, los pasos son los siguientes:

1. Elección del **nodo raíz**: Se escogerá aquél que provoque una mejor separación de las clases.

Se hace uso de la **ganancia de información**, que a su vez usa la **entropía de la clase** y la **entropía de la clase condicionada a un valor**.

- ▶ Cantidad de información mutua o **ganancia de información**:

$$I(C, X_i) = H(C) - H(C|X_i)$$

- ▶ **Entropía de una variable**:

$$H(C) = - \sum_{c=1}^n p(c) \log_2 p(c)$$

Entrenamiento de los árboles de decisión

- ### ► Entropía de una variable condicionada a un valor:

$$H(C|X) = - \sum_{i=1}^n p(x|c) \log_2 p(c|x) =$$

$$- \sum_c \sum_x p(x|c) \log_2 p(c|x)$$

- Se elige como primer nodo aquél que maximiza la expresión: $I(C, X_i)$, es decir, **minimiza la Entropía (error o desorden)**.

2. Cuando se dividen las ramas, se crean **nuevos subconjuntos de training** para cada una de ellas.
 3. Se repite el proceso (puntos 1 y 2) para cada rama generada.

Ejemplo de árbol de decisión (hacer en casa)

En Moodle dispone de un ejemplo completo para la creación de un árbol en el problema de **jugar o no a tenis** en función de variables meteorológicas.

- El árbol de decisión se debe crear a partir de la siguiente tabla con datos de ***training***, usando las expresiones de **ganancia de información** y **entropía** anteriores.

Day	Outlook	Temperature	Humidity	Wind	Decision
1	Rain	Mild	Normal	Weak	Yes
2	Sunny	Cool	Normal	Weak	Yes
3	Sunny	Mild	High	Weak	No
4	Overcast	Cool	Normal	Strong	Yes
5	Rain	Cool	Normal	Strong	No
6	Rain	Cool	Normal	Weak	Yes
7	Rain	Mild	High	Weak	Yes
8	Overcast	Hot	High	Weak	Yes
9	Sunny	Hot	High	Strong	No
10	Sunny	Hot	High	Weak	No

Algoritmo ID3

Características del algoritmo ID3

- Arbol de decisión basado en la **ganancia de información** de los atributos, como en el ejemplo del **Tenis**.
- Usa una estrategia de búsqueda voraz (Greedy).
https://es.wikipedia.org/wiki/Algoritmo_ID3.
- El ser un algoritmo voraz (Greedy) puede suministrar un **árbol óptimo local** en lugar de óptimo global.
- **No admite valores perdidos '?'.**
- **OJO: No admite atributos numéricos**, en caso de existir han de ser **discretizados previamente**.

Algoritmo ID3

Características del algoritmo ID3

- Su **complejidad crece linealmente** con el **número de instancias de entrenamiento**, pero **exponencialmente** con el **número de atributos**.
- Tiende a **favorecer la elección** de los atributos con mayor número de valores.
- **No realiza poda**, por lo que tiende a producir **árboles que sobreajustan** las instancias de entrenamiento.

Árboles de decisión

Entrenamiento en árboles de decisión

Árboles de decisión en Weka

Redes Neuronales

Redes en Weka

Entregables

Bibliografía

Algoritmo C4.5 (ID3 mejorado)

A partir de ID3 surge un algoritmo **mejorado**, llamado **C4.5**.

Características del algoritmo C4.5

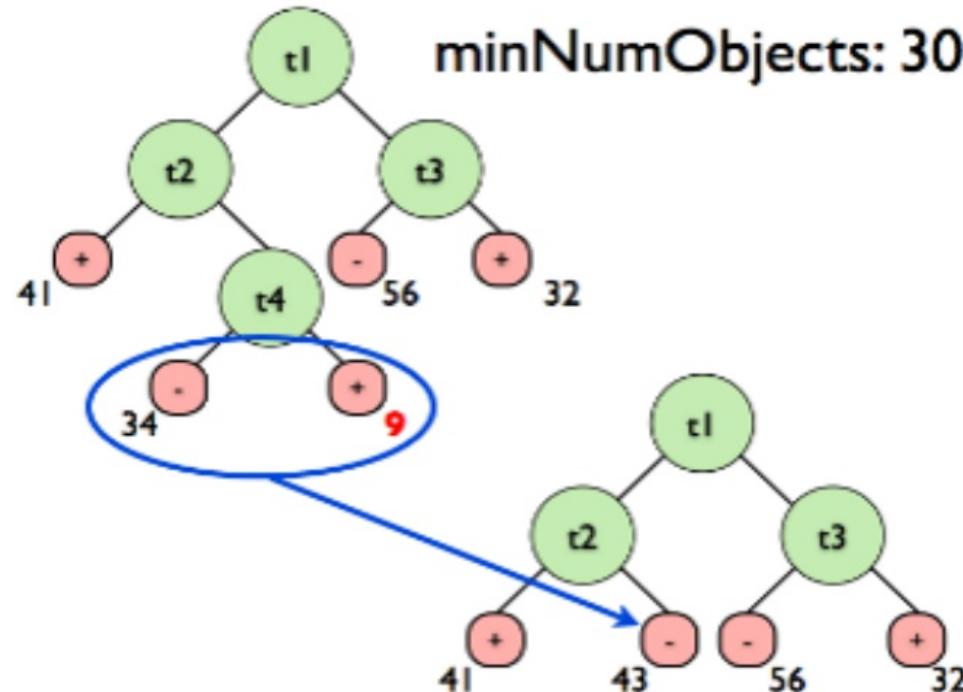
- **Nombrado como J48 en Weka → *classifiers/trees/J48*.**
- Permite atributos **continuos** (1.71, 1.719, 1.7154,...) o **discretos** (0, 1, 2, 3,...).
Los atributos continuos se **discretizan** automáticamente, considerando los umbrales que aporten la **mayor ganancia**.
- Permite trabajar con valores perdidos '?'.
Los ejemplos con valores perdidos en algún atributo **no se usan** en el **cálculo de la entropía o de la ganancia**.
- **Permite podar** después de su creación.
En el proceso de poda se van eliminando antecedentes de las reglas mientras el error siga siendo menor o igual al que se produce al no podar.

Algoritmo C4.5 (ID3 mejorado)

Parámetros relevantes de C4.5 (J48)

- **binarySplits**: Transformar valores nominales a binarios. Es más complejo trabajar con binarios pero podrían proporcionar resultados más precisos en árboles.
- **unpruned**: Determinar si se aplica poda o no.
- **subtreeRaising**: Tipo de poda a aplicar (reemplazamiento de subárbol o elevación).
- **confidenceFactor**: Factor de poda a aplicar. A menor valor mayor poda.
- **numFolds**: Número de subconjuntos de entrenamiento a usar para tomar la decisión de realizar una poda.
- **minNumObj**: Número mínimo de patrones por hoja. A menor número el árbol será mayor.

Algoritmo C4.5 (ID3 mejorado)

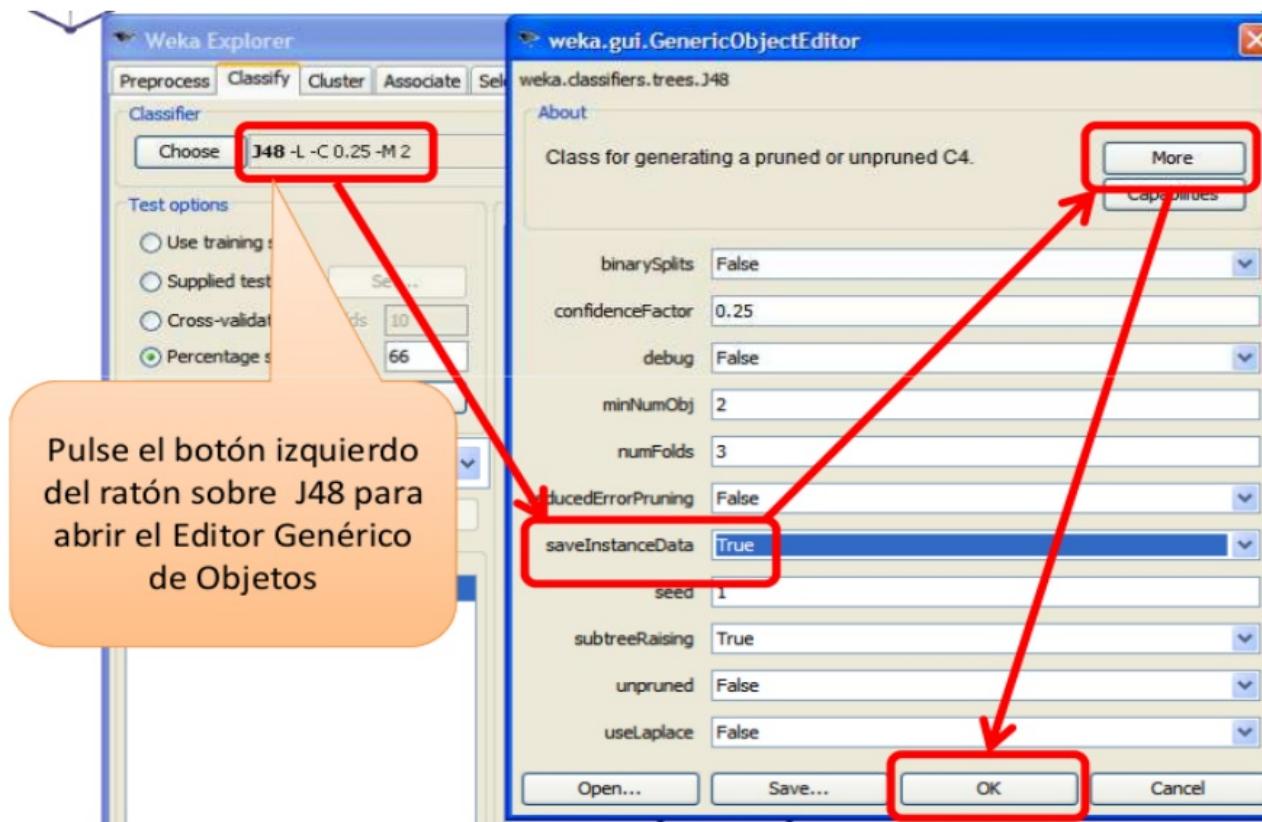


Los patrones son clasificados en la clase mayoritaria (-).

Weka con J48 e Iris

Con un **hold-out 66-44**, seleccionar pestaña **Classify** y posteriormente **classifiers.trees.J48**.

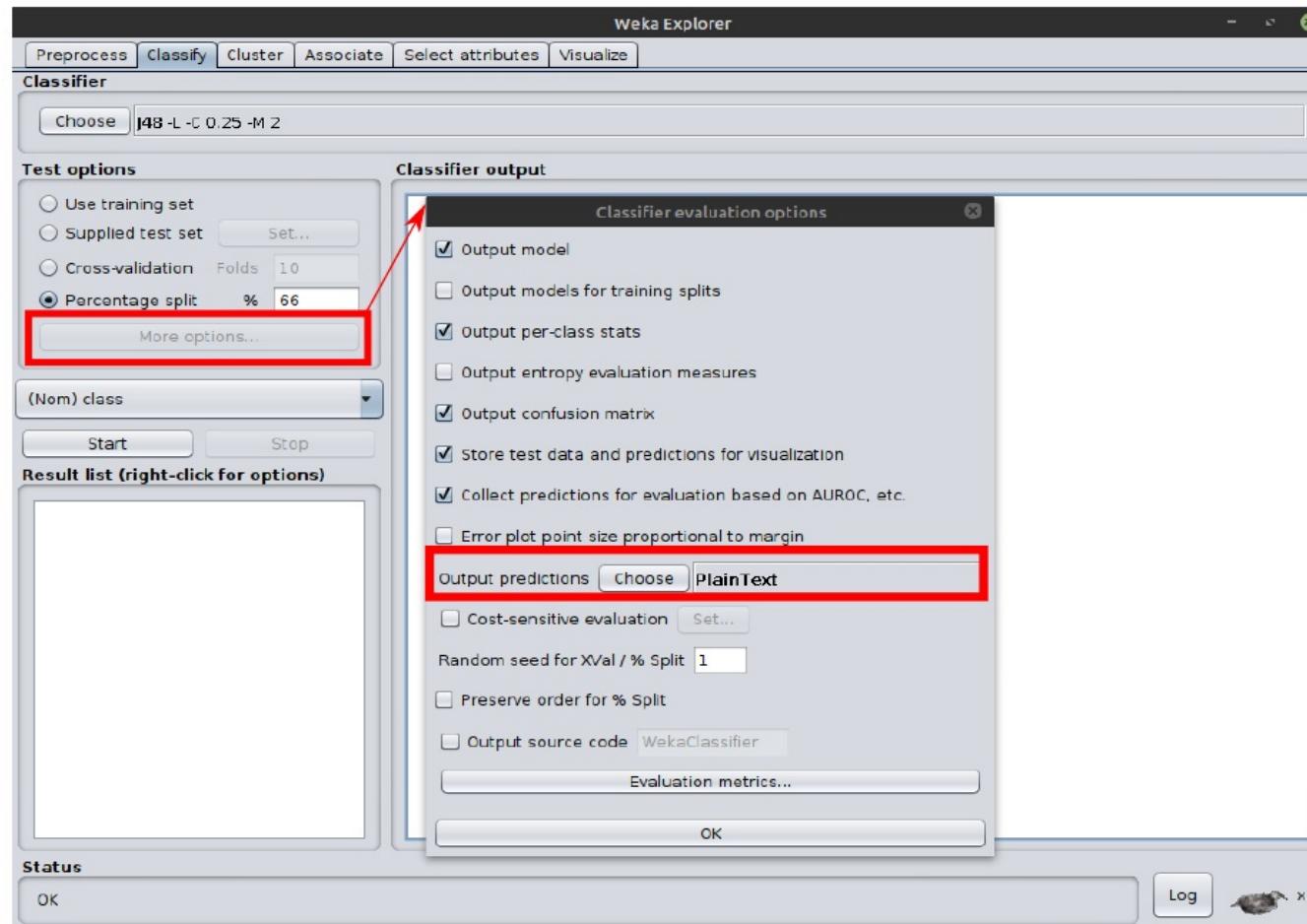
- ***saveInstanceData = True***: Permite ver las instancias de entrenamiento que se han clasificado en una hoja (click en la hoja) al usar posteriormente la opción “*Visualize Tree*”.



Weka con J48 e Iris

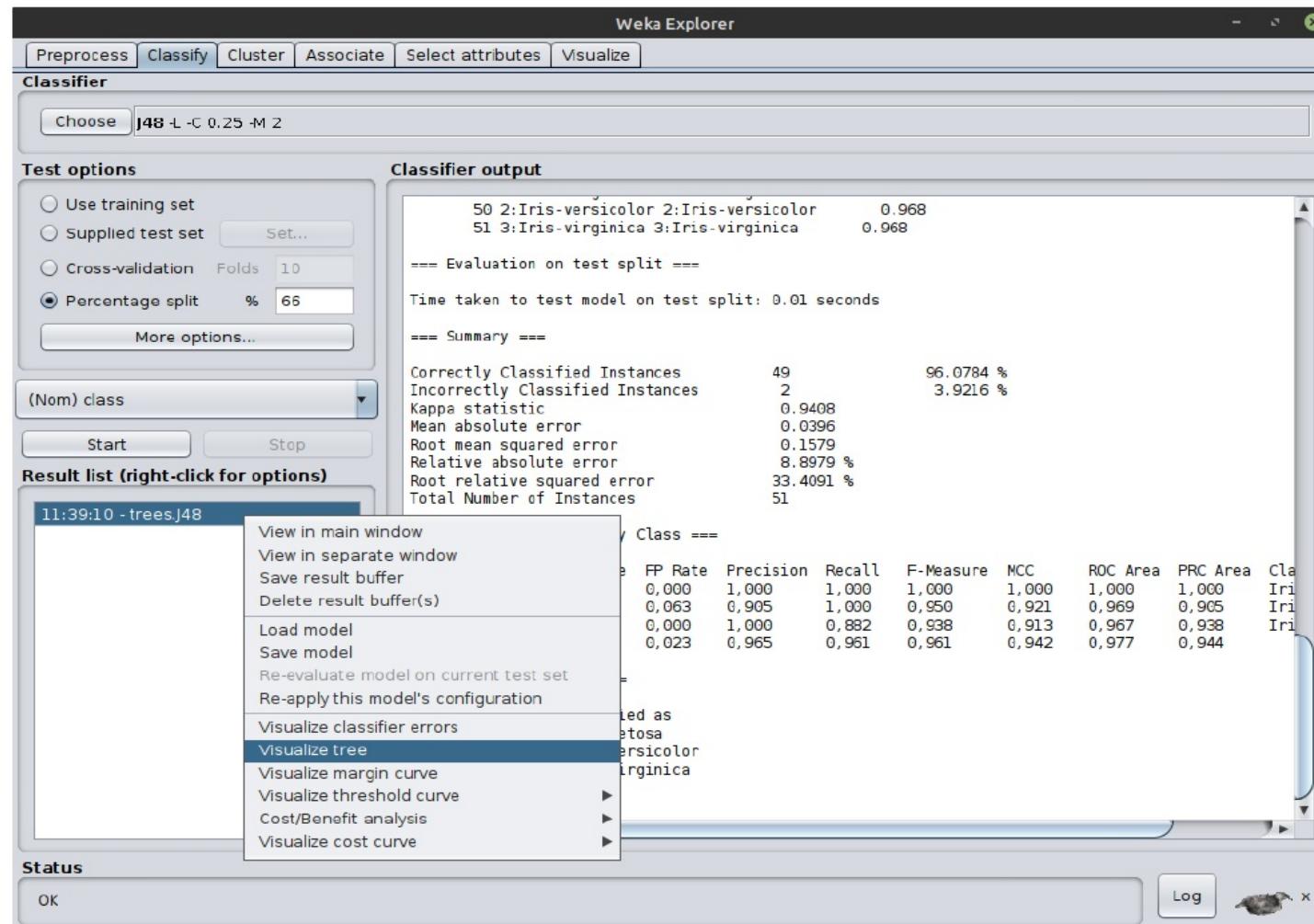
En “**More Options**”, seleccionar texto plano para “***Output Predictions***”.

- Esto permite ver en texto cuales han sido las predicciones del árbol sobre el conjunto de *test*.



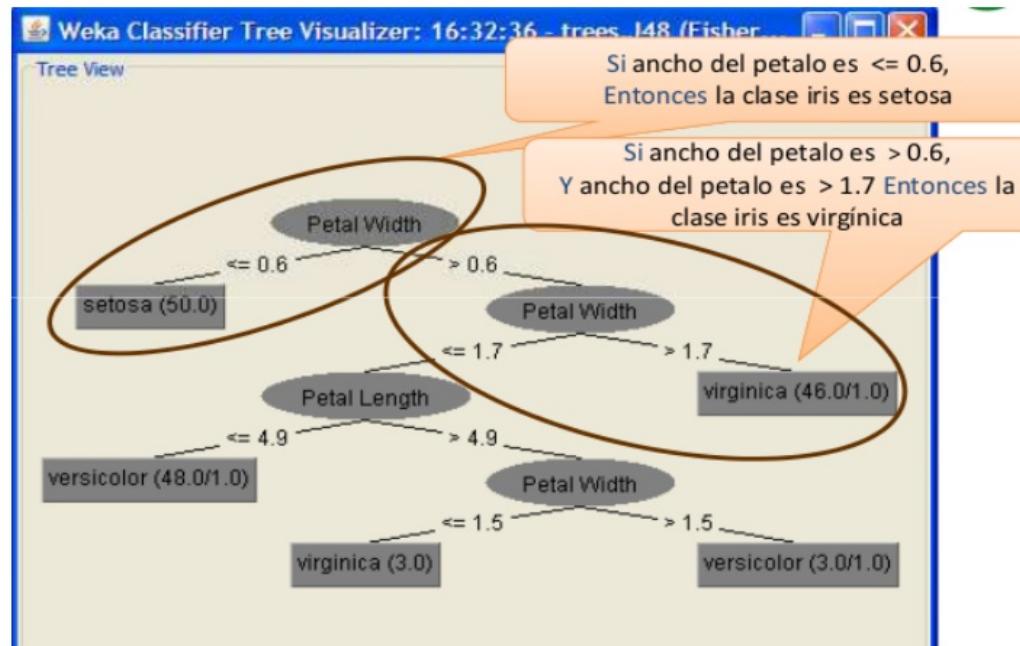
Weka con J48 e Iris

Una vez pulsado “**Start**”, en “**Result List**” seleccionar “**Visualize Tree**”.



Weka con J48 e Iris

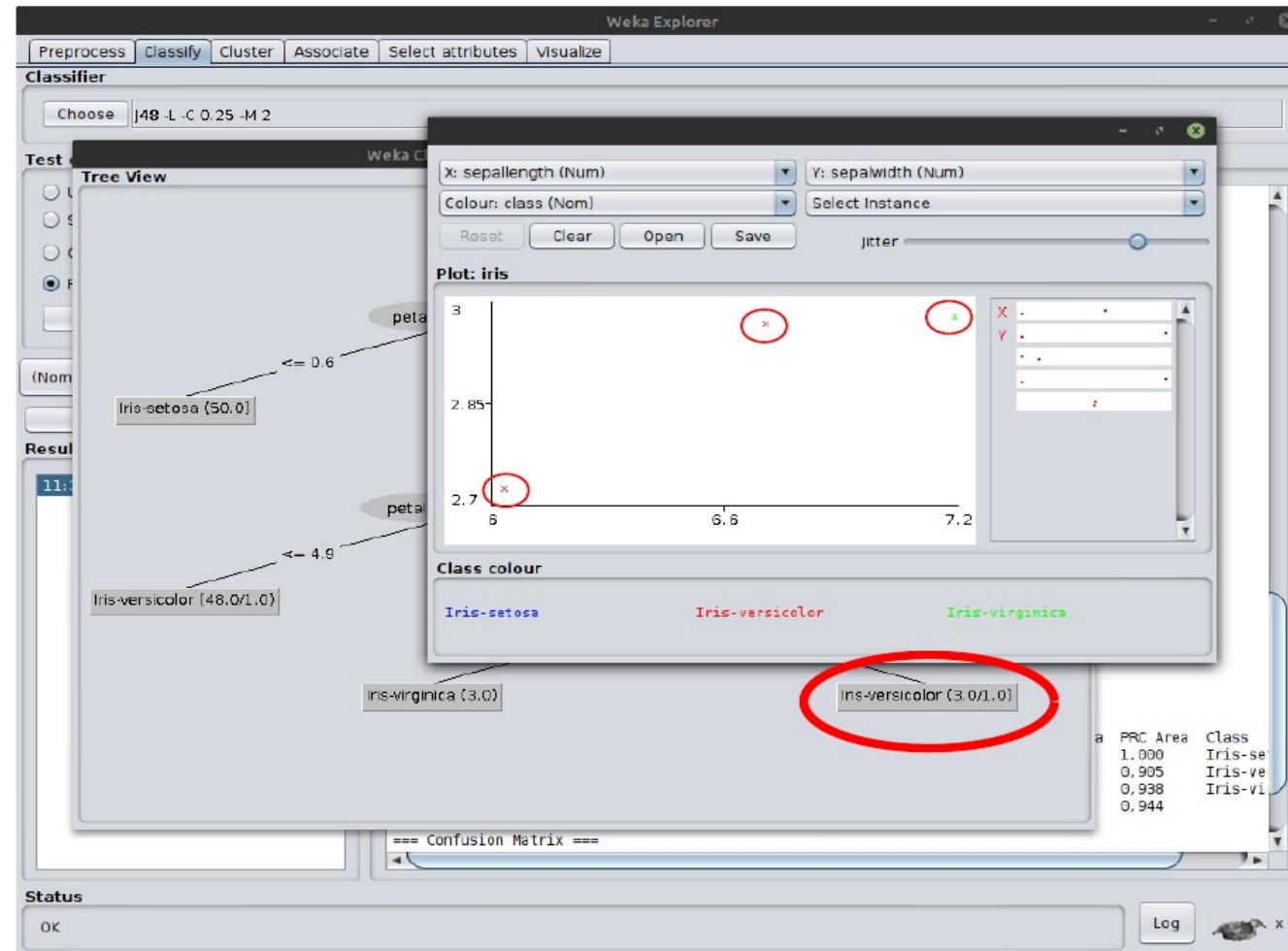
Interpretación al seleccionar “*Visualize Tree*”.



- Si “Petall Width” > 1,7 : virginica (46,0/1,0)
En esa hoja hay 46 patrones (de los 150 usados como *training*) clasificados como Iris-virginica y 1 de ellos esta mal clasificado.
Recuerde: El arbol se obtiene usando todo como *conjunto de entrenamiento*.
- ¿Aparecen los 4 atributos del problema? **No**, el árbol solo necesita dos.

Weka con J48 e Iris

Recuerde la opción `saveInstanceData = True`. Instancias de entrenamiento de los 3 patrones de la hoja *Iris – versicolor*(3,0/1,0).



Weka con J48 e Iris

Reglas obtenidas en la salida, tamaño del árbol y número de hojas.

Weka Explorer

Preprocess Classify Cluster Associate Select attributes Visualize

Classifier

Choose J48 -L -C 0.25 -M 2

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)

11:39:10 -trees.J48

Classifier output

== Classifier model (full training set) ==
J48 pruned tree

petalwidth <= 0.6: Iris-setosa (50.0)
petalwidth > 0.6
| petalwidth <= 1.7
| | petallength <= 4.9: Iris-versicolor (48.0/1.0)
| | petallength > 4.9
| | | petalwidth <= 1.5: Iris-virginica (3.0)
| | | petalwidth > 1.5: Iris-versicolor (3.0/1.0)
| petalwidth > 1.7: Iris-virginica (46.0/1.0)

Number of Leaves : 5
Size of the tree : 9

Time taken to build model: 0.01 seconds

--- Predictions on test split ---

inst#	actual	predicted	error	prediction
1	2:Iris-versicolor	2:Iris-versicolor		0.968
2	3:Iris-virginica	3:Iris-virginica		0.968
3	2:Iris-versicolor	2:Iris-versicolor		0.968
4	1:Iris-setosa	1:Iris-setosa		1
5	3:Iris-virginica	3:Iris-virginica		0.968
6	2:Iris-versicolor	2:Iris-versicolor		0.968
7	1:Iris-setosa	1:Iris-setosa		1
8	1:Iris-setosa	1:Iris-setosa		1
9	2:Iris-versicolor	2:Iris-versicolor		0.968
10	3:Iris-virginica	3:Iris-virginica		0.968
11	2:Iris-versicolor	2:Iris-versicolor		0.968
12	3:Iris-virginica	3:Iris-virginica		0.968
13	1:Iris-setosa	1:Iris-setosa		1
14	2:Iris-versicolor	2:Iris-versicolor		0.968
15	2:Iris-versicolor	2:Iris-versicolor		0.968
16	3:Iris-virginica	2:Iris-versicolor	+	0.968
17	3:Iris-virginica	3:Iris-virginica		0.968

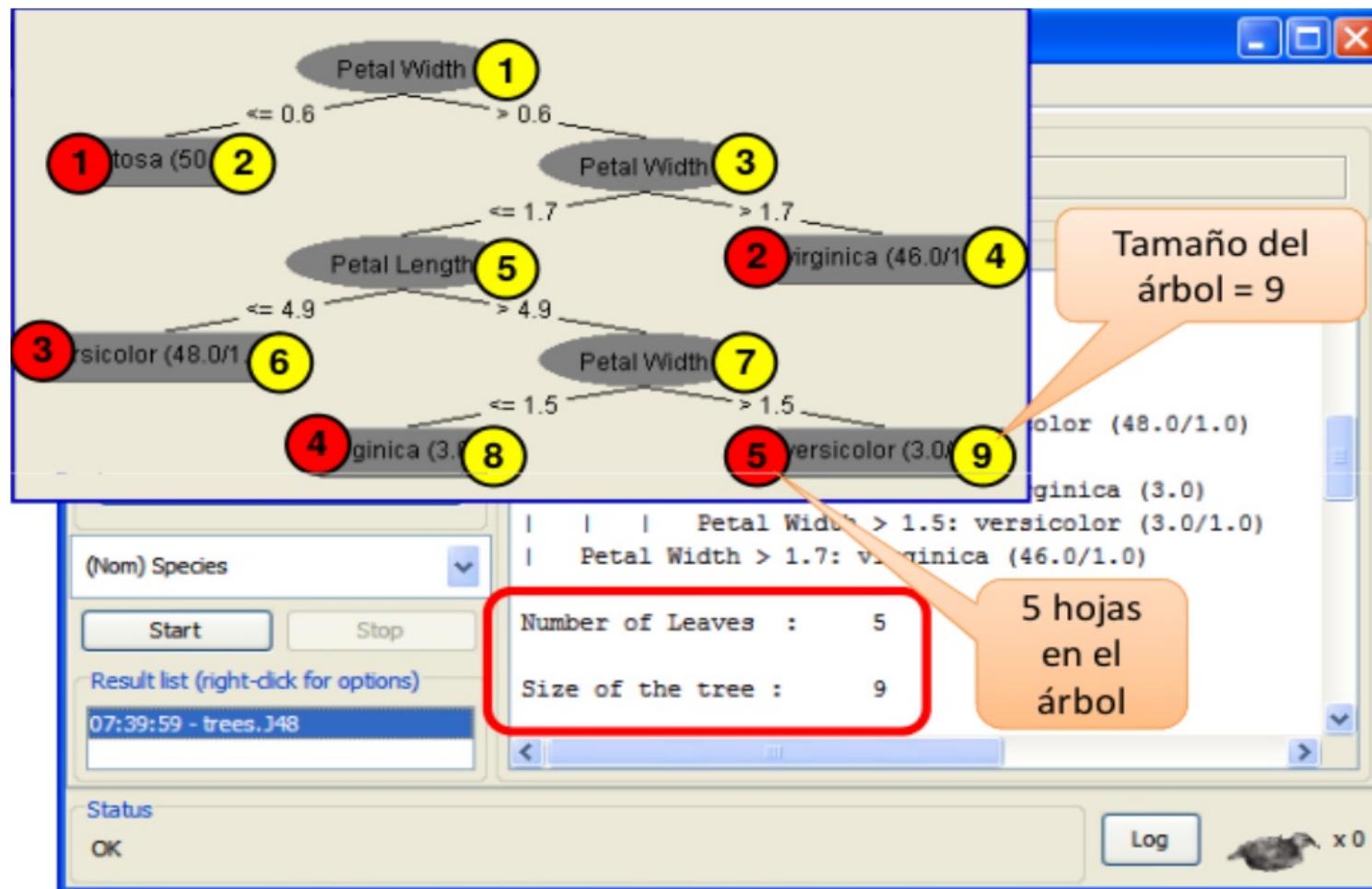
Status

OK Log

Reglas para un árbol de decisión de tamaño 9 y con 5 hojas.

Weka con J48 e Iris

Reglas obtenidas en la salida, tamaño del árbol y número de hojas.



Weka con J48 e Iris

Predicciones hechas en *test* una vez construido el árbol con todo el *train* (150 patrones). Se ha usado un 66-44, por lo que se han utilizado **51 patrones** (44 %) para obtener esas predicciones y la matriz de confusión.

The screenshot shows the Weka Explorer interface with the following details:

- Classifier chosen:** J48 -L -C 0.25 -M 2
- Test options:** Percentage split (66%) selected.
- Result list:** Shows the command "11:39:10 -trees,J48".
- Classifier output:**
 - Classifier model (full training set):**

```
petalwidth <= 0.6: Iris-setosa (50.0)
petalwidth > 0.6
|  petalwidth <= 1.7
|  |  petallength <= 4.9: Iris-versicolor (48.0/1.0)
|  |  petallength > 4.9
|  |  |  petalwidth <= 1.5: Iris-virginica (3.0)
|  |  |  |  petalwidth > 1.5: Iris-versicolor (3.0/1.0)
|  |  petalwidth > 1.7: Iris-virginica (46.0/1.0)
```
 - Number of Leaves:** 5
 - Size of the tree:** 9
 - Time taken to build model:** 0.01 seconds
 - Predictions on test split:**

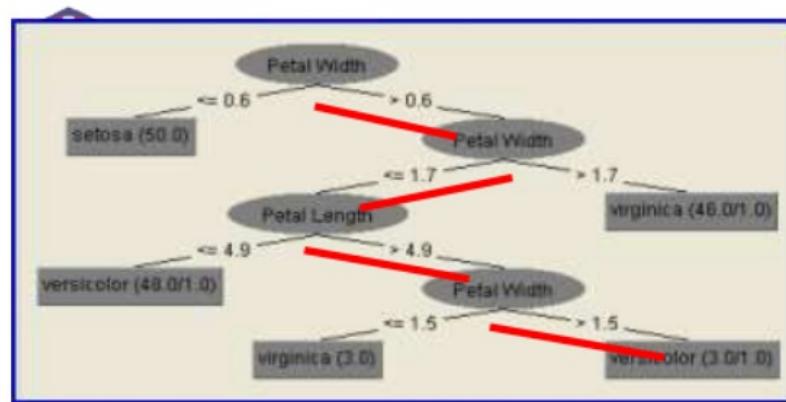
inst#	actual	predicted	error	prediction
1	2:Iris-versicolor	2:Iris-versicolor		0.968
2	3:Iris-virginica	3:Iris-virginica		0.968
3	2:Iris-versicolor	2:Iris-versicolor		0.968
4	1:Iris-setosa	1:Iris-setosa	1	
5	3:Iris-virginica	3:Iris-virginica		0.968
6	2:Iris-versicolor	2:Iris-versicolor		0.968
7	1:Iris-setosa	1:Iris-setosa	1	
8	1:Iris-setosa	1:Iris-setosa	1	
9	2:Iris-versicolor	2:Iris-versicolor		0.968
10	3:Iris-virginica	3:Iris-virginica		0.968
11	2:Iris-versicolor	2:Iris-versicolor		0.968
12	3:Iris-virginica	3:Iris-virginica		0.968
13	1:Iris-setosa	1:Iris-setosa	1	
14	2:Iris-versicolor	2:Iris-versicolor		0.968
15	2:Iris-versicolor	2:Iris-versicolor		0.968
16	3:Iris-virginica	2:Iris-versicolor	+	0.968
17	3:Iris-virginica	3:Iris-virginica		0.968

Annotations:

- A red box highlights the prediction for instance 16, which is incorrectly classified as Iris-versicolor with a probability of 0.968.
- A red arrow points from the text "Patrón incorrectamente clasificado." to this highlighted row.
- A red box highlights the entire "Predictions on test split" section.
- Text next to the highlighted section: "Instancias usadas en test. La última columna es la probabilidad obtenida para la clase asignada."
- Text next to the highlighted prediction: "Patrón incorrectamente clasificado. Probabilidad estimada de 96.8% de que pertenezca a versicolor."

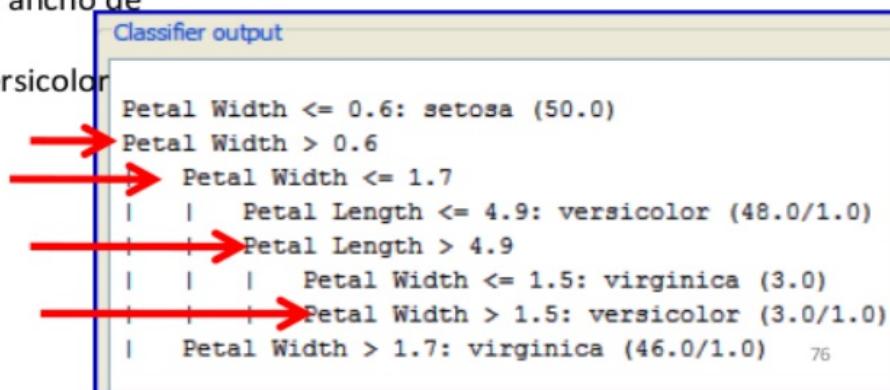
Weka con J48 e Iris

Ante un nuevo patrón no visto, usariamos el árbol (modelo) obtenido con el conjunto de train (**150 patrones**) para saber a qué clase pertenece.



Lirio de especie desconocida:
Ancho de pétalo = 1.6 cm
Longitud de pétalo = 5.0 cm

Como ancho del petalo es $> 0,6$ y
ancho del petalo es $\leq 1,7$ y
longitud del petalo $> 4,9$ y ancho de
petalo es $> 1,5$
Entonces la clase iris es versicolor



Weka con J48 e Iris

Interpretación de las métricas de salida que proporciona Weka. Se usa para ello el conjunto de **test** (51 patrones) sobre el modelo obtenido con todo como **train** (150 patrones).

Classifier output			
== Evaluation on test split ==			
--- Summary ---			
Correctly Classified Instances	49	96.0784 %	
Incorrectly Classified Instances	2	3.9216 %	
Kappa statistic	0.9408		
Mean absolute error	0.0396		
Root mean squared error	0.1579		
Relative absolute error	8.8979 %		
Root relative squared error	33.4091 %		
Total Number of Instances	51		

El árbol clasifica bien
49 de las 51
muestras del
conjunto de test.

51 muestras del
conjunto de test
(el 34%)

Weka con J48 e Iris

MATRIZ DE CONFUSIÓN

Classifier output			
==== Confusion Matrix =====			
		<-- classified as	
15	0	0	a = setosa
0	19	0	b = versicolor
0	2	15	c = virginica

Los 15 patrones de la clase setosa los clasifica correctamente

Los 19 patrones de la clase versicolor los clasifica correctamente

2 patrones de la clase virginica del conjunto de test los clasifica incorrectamente

Árboles de decisión

Entrenamiento en árboles de decisión

Árboles de decisión en Weka

Redes Neuronales

Redes en Weka

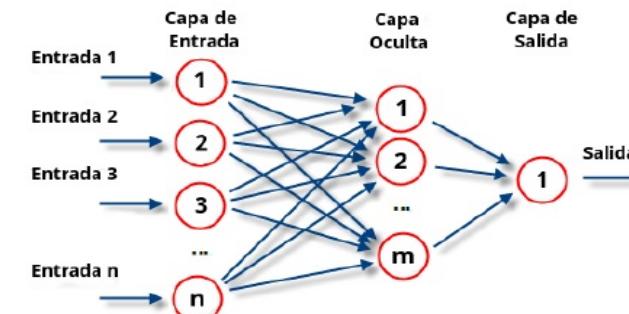
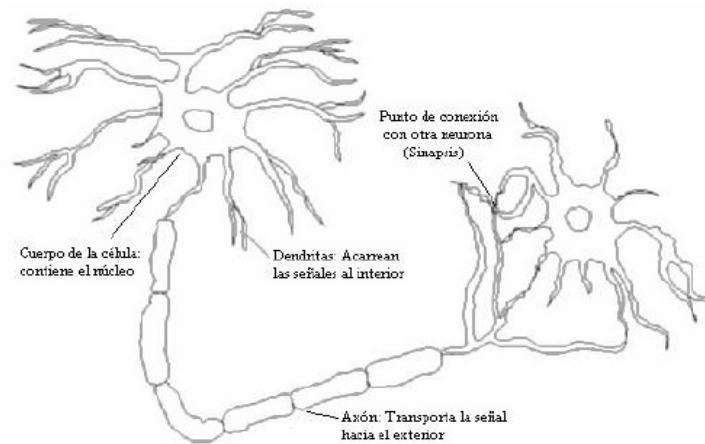
Entregables

Bibliografía

Redes Neuronales

Red Neuronal Artificial (RNA): Modelo computacional basado en un conjunto de unidades neuronales simples (modelos matemáticos) interconectadas de forma análoga a las neuronas en los cerebros biológicos mediante los axones.

- Una RNA es un **modelo no lineal**, el cual puede acumular conocimiento en sus coeficientes y estructura a través de un proceso de aprendizaje (*Ej: Algoritmo BackPropagation*).
- Después del proceso de aprendizaje, una RNA es capaz de aproximar una función continua, la cual será nuestra regla de decisión.

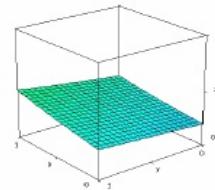


Perceptrón Multicapa (Multilayer Perceptron)

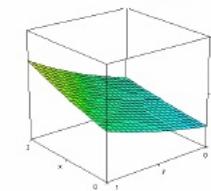
- El Perceptrón Multicapa (MultilayerPerceptron) es una RNA formada por:
 - ▶ 1 **capa de entrada** (K variables independientes).
 - ▶ Múltiples **capas ocultas** (usualmente se usa 1 capa oculta con M neuronas o funciones de activación).
 - ▶ 1 **capa de salida** con Q neuronas o funciones de activación (variables dependientes o clases).
 - La función de activación más usada es la **Unidad de base Sigmoide (SU)**:

$$f(x) = \frac{1}{1+exp^{-x}}$$

$$a) \quad f_1(x_1, x_2) = \frac{1}{1 + e^{-0.5x_1 - 0.3x_2}}$$

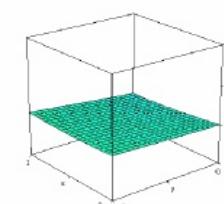


b) $f_2(x_1, x_2) = \frac{1}{1 + e^{-0.5x_1 - 0.8x_2}}$



$$c) f_3(x_1, x_2) = \frac{1}{1 + e^{-(3 \times 0.5)x_1 - (3 \times 0.3)x_2}}$$

$$\text{d)} f_4(x_1, x_2) = f_1(x_1, x_2) + f_2(x_1, x_2) - f_3(x_1, x_2)$$

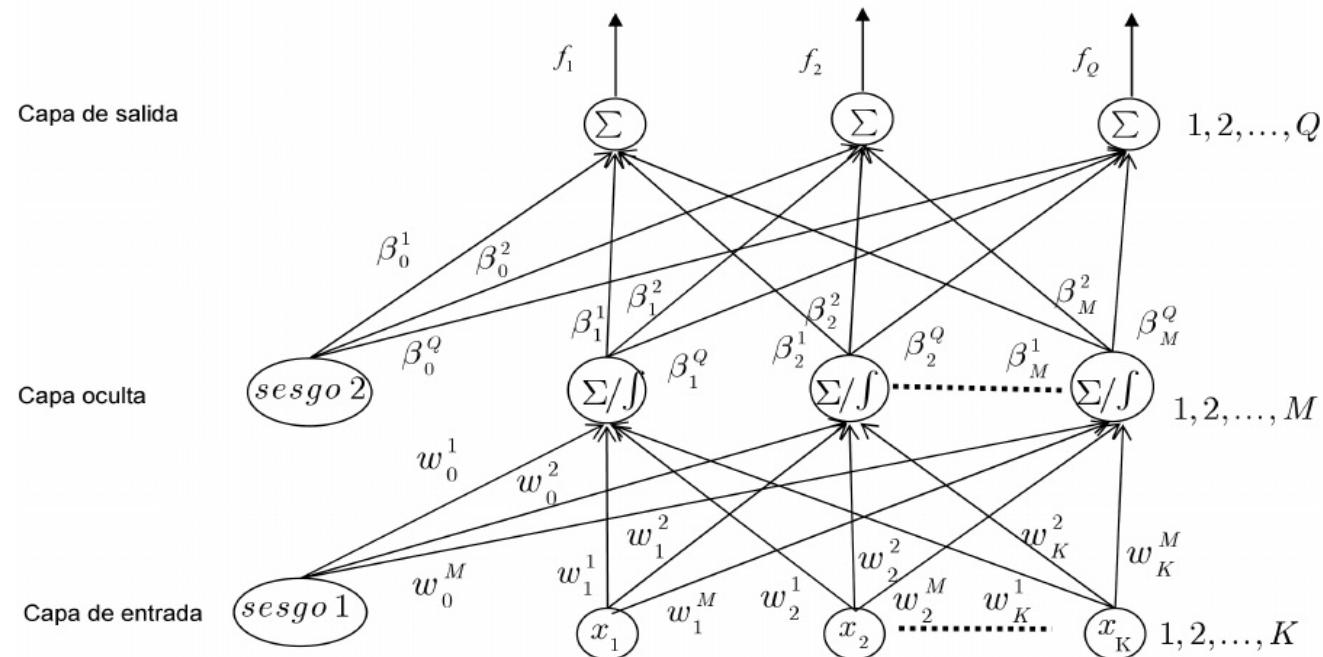


Perceptrón multicapa (Multilayer Perceptron)

- Las neuronas con **SUs** en **capa oculta** son **sumatorias** y tienen la siguiente función de activación:

$$B_j(\mathbf{x}, \mathbf{w}_j) = \frac{1}{1 + \exp\left(-\left(w_0^j + \sum_{i=1}^K w_i^j x_i\right)\right)}, \quad j = 1 \dots M$$

- Los sesgos se establecen al valor de 1, y permiten que haya aprendizaje si las entradas son cercanas a 0.



Perceptrón Multicapa (Multilayer Perceptron)

- Las neuronas en **capa de salida** son también **sumatorias**, pero ya no aplican normalmente SUs, sino la función **identidad**, $f(x) = y$, quedando la siguiente función de activación:

$$f_q(\mathbf{x}, \theta_q) = \beta_0^q + \sum_{j=1}^M \beta_j^q B_j(\mathbf{x}, \mathbf{w}_j), \quad q = 1 \dots Q$$

- ▶ $\theta_q = (\beta_0^q, \beta_1^q, \dots, \beta_M^q, \mathbf{w}_1, \dots, \mathbf{w}_M)$ es el vector de pesos de una neurona q .
 - ▶ $\mathbf{w}_j = (w_0^j, w_1^j, \dots, w_k^j)$ es el vector de pesos de las conexiones entre la capa de entrada y la neurona de la capa oculta j .
 - ▶ \mathbf{x} es el valor de las entradas de la RNA.
 - ▶ $B_j(\mathbf{x}, \mathbf{w}_j)$ es la función de base (salida) de la neurona j de la capa oculta.
 - ▶ β_0^q y w_0^j son los sesgos del modelo asociado a la neurona q en la capa de salida, y a la neurona j de la capa oculta respectivamente.

Perceptrón Multicapa (Multilayer Perceptron)

- A las salidas se les puede aplicar la función **Softmax** para convertirlas en probabilidad (función de **normalización** que asegura que las salidas estarán entre 0 y 1 y que su suma será 1):

$$g_q(\mathbf{x}) = \frac{\exp^{f_q(\mathbf{x})}}{\sum_{i=1}^Q \exp^{f_q(\mathbf{x})}}$$

out_1^H	out_2^H	out_3^H	Clase predicha
0,017	0,047	0,936	3
0,047	0,947	0,006	2
0,066	0,044	0,890	3

Figura: Ejemplo de Softmax para un problema de 3 clases.

Algoritmo de Retropropagación del error

- El objetivo para entrenar una red es conseguir el valor de los **pesos** de los enlaces que **minimice el error de salida**.
- El algoritmo más utilizado para entrenar un MultilayerPerceptron es el algoritmo de Retropropagación del error (*BackPropagation*).
- https://es.wikipedia.org/wiki/Propagaci%C3%B3n_hacia_atr%C3%A1s
- Es un algoritmo que emplea un ciclo **propagación–adaptación** de dos fases.
 1. **FASE 1:** Una vez que se ha aplicado un patrón a la entrada de la red, este se propaga desde la primera capa a través de las capas siguientes, hasta generar una salida. La salida obtenida se compara con la salida deseada y se calcula el error para cada una de las salidas.
 2. **FASE 2:** El error producido en cada salida se propaga hacia atrás mediante lo que se conoce como **regla delta**, hacia todas las neuronas de la capa oculta, modificándose los pesos de los enlaces de la red para que se adapten al error cometido. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido una señal de error que describa su contribución relativa al error total.
 3. Esto se hace para todos los patrones y al finalizar se conoce como **ÉPOCA**. A más épocas o pasadas más entrenamiento (**Ojo con el sobre-entrenamiento**).

Ventajas y desventajas de las RNA

- **Ventajas:**

- ▶ Habitualmente **gran tasa de acierto** en la predicción.
- ▶ **Robustez** ante la presencia de errores (**ruido, outliers, ...**).
- ▶ Gran **versatilidad**: salida nominal, numérica, vectores...
- ▶ Eficiencia (**rapidez**) en la **evaluación de nuevos casos**.

- **Desventajas:**

- ▶ Necesitan **mucho tiempo** para el **entrenamiento**.
- ▶ **Muchos parámetros** (como se verá a continuación). El entrenamiento es, en gran parte, ensayo y error (arquitectura, tasas de aprendizaje, momento).
- ▶ *Poca interpretabilidad* del modelo (modelos de caja negra).
- ▶ Los **atributos** de entrada deben ser **numéricos**.
- ▶ Pueden producir **sobreaprendizaje**.

Árboles de decisión

Entrenamiento en árboles de decisión

Árboles de decisión en Weka

Redes Neuronales

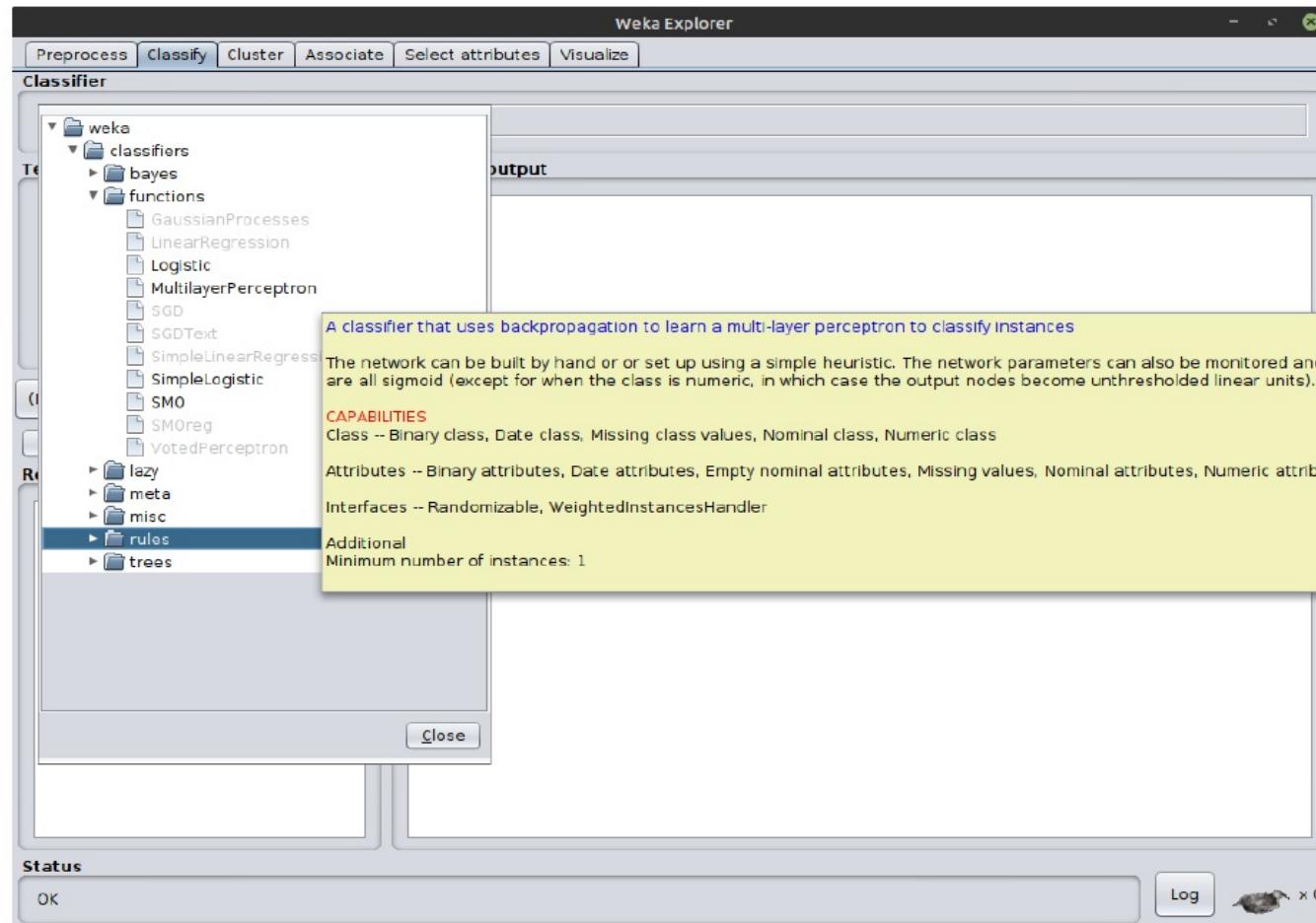
Redes en Weka

Entregables

Bibliografía

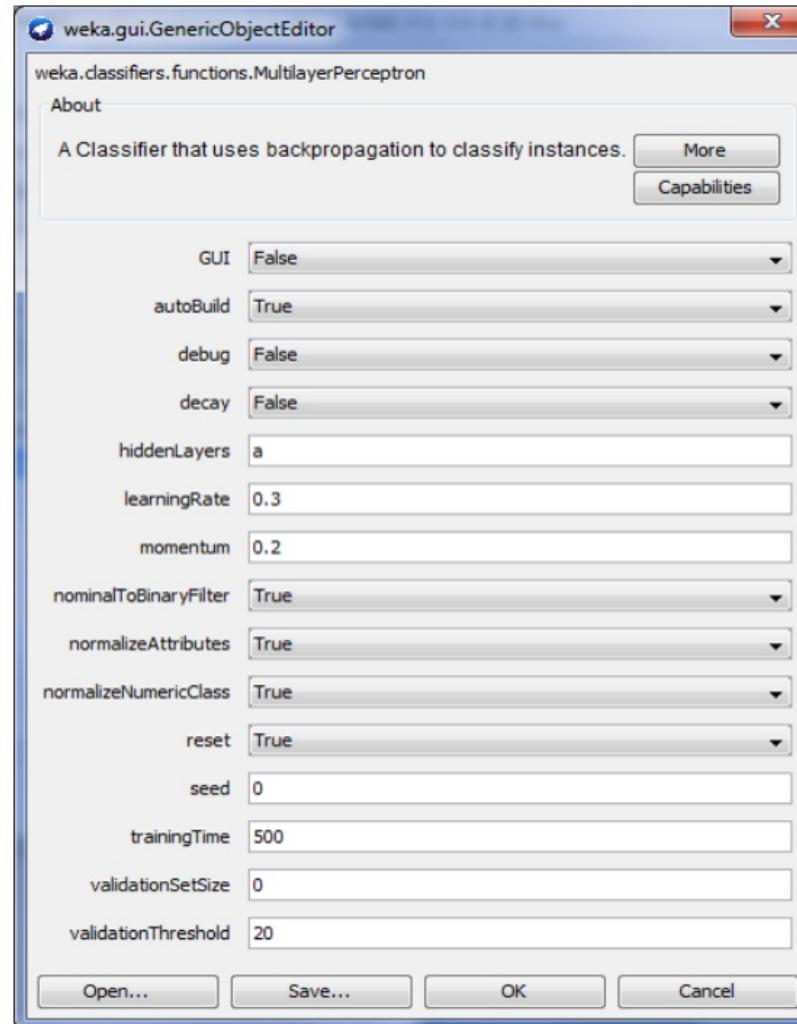
Multilayer Perceptron en Weka: Parámetros

Para Weka el MLP se nombra como **MultilayerPerceptron** y aparece en:
→ *classifiers.functions.MultilayerPerceptron*



MultilayerPerceptron en Weka

Parámetros destacables a configurar para un MLP en Weka:

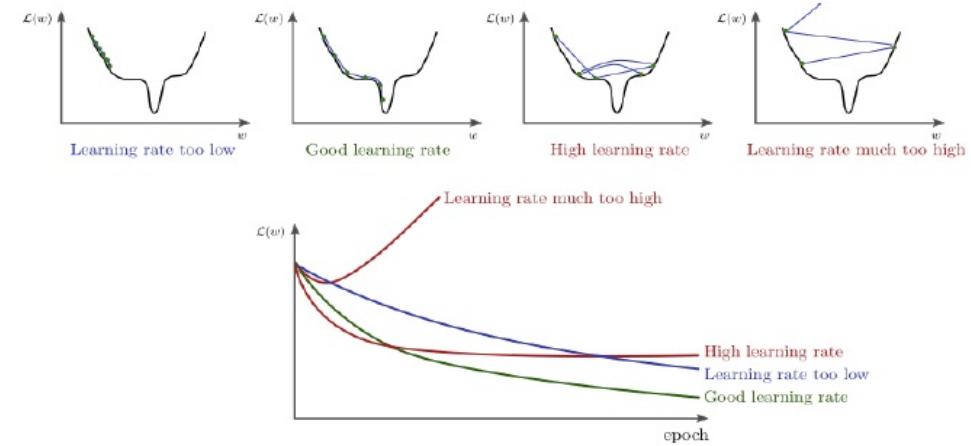
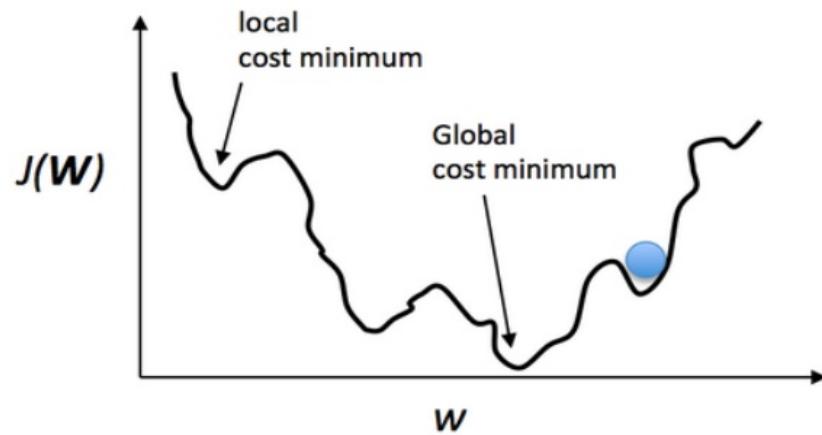


MultilayerPerceptron en Weka: Parámetros

- **GUI:** Permite usar una interfaz interactiva en la construcción de la red (si usamos el GUI, autoBuild nos construye la red automáticamente).
- **decay:** Decrementa el factor de aprendizaje conforme avanzan las épocas.
- **hiddenLayers:** Lista de números positivos que definen:
 - ▶ Número de capas ocultas = tamaño de la lista.
 - ▶ Neuronas en capa oculta i = valor del elemento i de la lista.
 - ▶ **Ejemplo:** 2 capas ocultas con 3 neuronas: 3, 3
 - ▶ Se pueden usar **comodines:**
 - $i = n^o$ de atributos.
 - $o = n^o$ clases.
 - $t = n^o$ ($\text{atributos} + \text{clases}$).
 - $a = n^o$ ($\text{atributos} + \text{clases}$)/2

Multilayer Perceptron en Weka: Parámetros

- **learningRate**: Tasa de aprendizaje o cantidad en la que se actualizan los pesos de los enlaces en cada iteración de aprendizaje respecto al gradiente (pendiente) calculado en el algoritmo de *BackPropagation*. Valor entre [0,1].
 - Controla lo rápido que se adapta el modelo al problema. Las **tasas de aprendizaje más pequeñas** requieren más épocas de entrenamiento, dados los pequeños cambios realizados en los pesos de cada actualización. No dejaremos atrás mínimos locales (queremos minimizar el error), pero se corre el riesgo de que el algoritmo se quede **atascado** en uno de ellos y no sea el global.
 - Una **tasa de aprendizaje demasiado grande** puede hacer que el modelo converja demasiado rápido a una solución subóptima. El algoritmo no se quedará atascado en un mínimo, pero puede que “salte” una **solución óptima o subóptima**.



Multilayer Perceptron en Weka: Parámetros

- **momentum**: El *Momentum* o “impulso” no sirve en sí para aprender, sino que nos permite salir de un subóptimo local alcanzado que tenga un valle grande (terrazas de arroz en la foto). Valor entre [0,1]. Cuanto más cercano a 1, las modificaciones son menos fuertes.
- El algoritmo *BackPropagation* está diseñado para que cuando estamos en un óptimo global, el momentum no permita salir de él.



- *LearningRate* es crítico para actualizar los pesos y minimizar el error. *Momentum* se usa para ayudar a *LearningRate*, pero no para reemplazarla.

MultilayerPerceptron en Weka: Parámetros

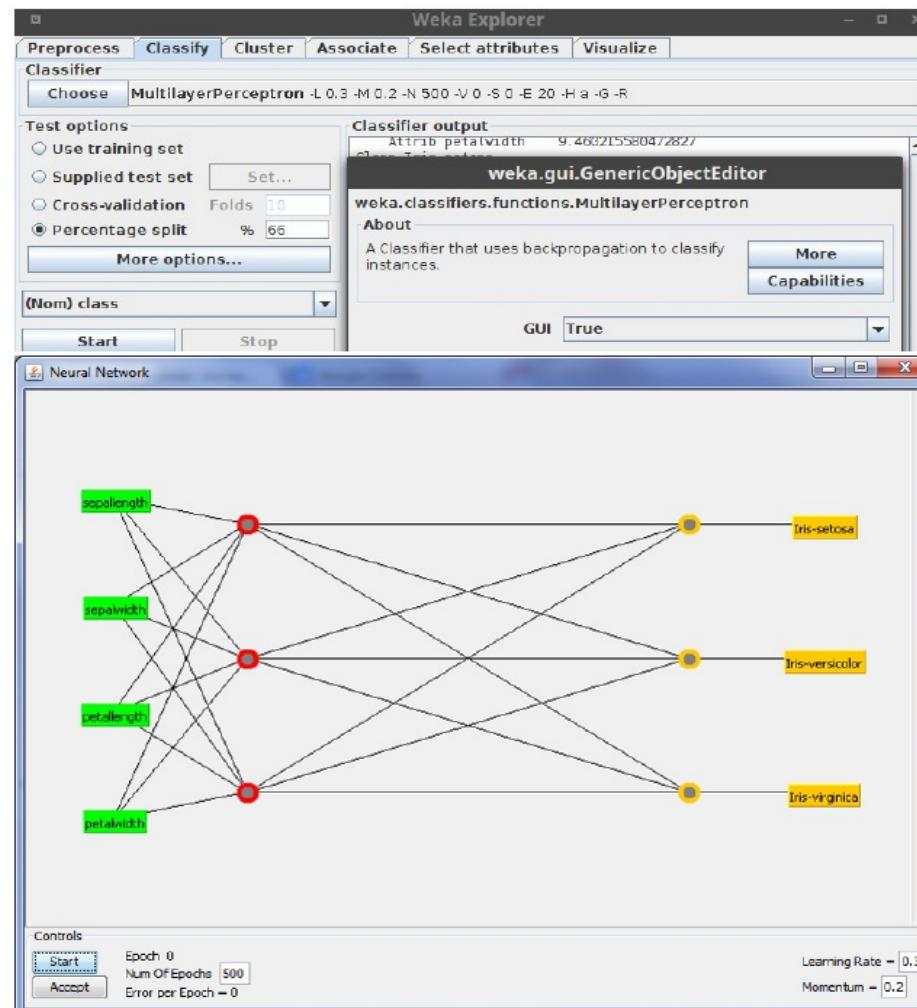
- **nominalToBinaryFilter:** True/False. Por si tenemos atributos nominales sin pasar a binario. MLP trabaja mejor con los atributos numéricos, podría mejorar el rendimiento.
- **normalizeAttributes:** True/False. Por si tenemos los atributos sin normalizar. MLP trabaja mejor con los atributos normalizados.
- **normalizeNumericClass:** True/False
 - ▶ En caso de ser regresión normaliza la variable de salida entre [-1,1].
 - ▶ Ayuda a mejorar el rendimiento de la red.
 - ▶ La normalización es interna, por tanto, la salida será escalada posteriormente al rango original.
- **Reset:** Evita divergencia. Si durante el proceso de aprendizaje, las salidas reales divergen demasiado de los valores estimados, entonces reseteamos el algoritmo bajando el valor del *LearningRate*.
- **Seed:** Semilla para los números aleatorios que se usan para, entre otras cosas, inicializar los pesos.

MultilayerPerceptron en Weka: Parámetros

- **trainingTime o ÉPOCAS:** Número de épocas o pasadas para el entrenamiento (ajuste de los pesos) de la red.
- **validationSetSize:** Porcentaje del conjunto de entrenamiento que será usado para validar. Permitirá parar antes de que se alcance el número total de épocas (“*early stopping*”).
 - Un conjunto de validación es un “trozo” del conjunto de *training* que se usa como “test” cuando se está entrenando la red.
 - Permite parar antes de que se alcancen las épocas indicadas cuando el error empeora de forma continuada pasando ese conjunto de validación.
 - Si el valor introducido es **0**, no usaremos “*early stopping*” o parada temprana y entonces se completarán todas las épocas indicadas.
- **validationThreshold:** Umbral o número de épocas consecutivas durante las que el error de validación puede empeorar o subir. Si se supera dicho umbral, el entrenamiento finalizará aunque no se hayan alcanzado las épocas indicadas en el parámetro *trainingTime*.

Multilayer Perceptron en Weka: GUI

- Se puede modificar la configuración del Perceptrón gráficamente (**GUI** → **True**).



MultilayerPerceptron de Weka con Iris

Modelo obtenido con un *hold-out* 75-25 para Iris. Parámetros por defecto.

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 2

Test options

- Use training set
- Supplied test set Set...
- Cross-validation Folds 10
- Percentage split % 75
- [More options...](#)

(Nom) class

Start Stop

Result list (right-click for options)

18:43:08 - functions.MultilayerPerceptron

Classifier output

```
--- Classifier model (full training set) ---
Sigmoid Node 0
Inputs  Weights
Threshold -3.5015971588434014
Node 3 -1.005811085385995
Node 4 9.0753844669134
Node 5 -4.107890453399234

Sigmoid Node 1
Inputs  Weights
Threshold 1.0692845992273177
Node 3 3.898873687789407
Node 4 -9.768810360340266
Node 5 -8.59913449315135

Sigmoid Node 2
Inputs  Weights
Threshold -1.0071762383436476
Node 3 -4.218406133827042
Node 4 -3.626059686321116
Node 5 8.805122981737854

Sigmoid Node 3
Inputs  Weights
Threshold 3.3824855566856726
Attrib sepallength 0.9098627458622287
Attrib sepalwidth 1.5675138827531245
Attrib petallength -5.037336107319891
Attrib petalwidth -4.915469682506093

Sigmoid Node 4
Inputs  Weights
Threshold -3.3305735922918323
Attrib sepallength -1.116750023770101
Attrib sepalwidth 3.1250096866676538
Attrib petallength -4.133137022912303
Attrib petalwidth -4.075589727871457

Sigmoid Node 5
Inputs  Weights
Threshold -7.496091023618097
Attrib sepallength -1.2158878822058794
Attrib sepalwidth -3.5332821317534946
Attrib petallength 8.401834252274107
Attrib petalwidth 9.460215580472836

Class Iris-setosa
Input
Node 0

Class Iris-versicolor
Input
Node 1

Class Iris-virginica
Input
Node 2
```

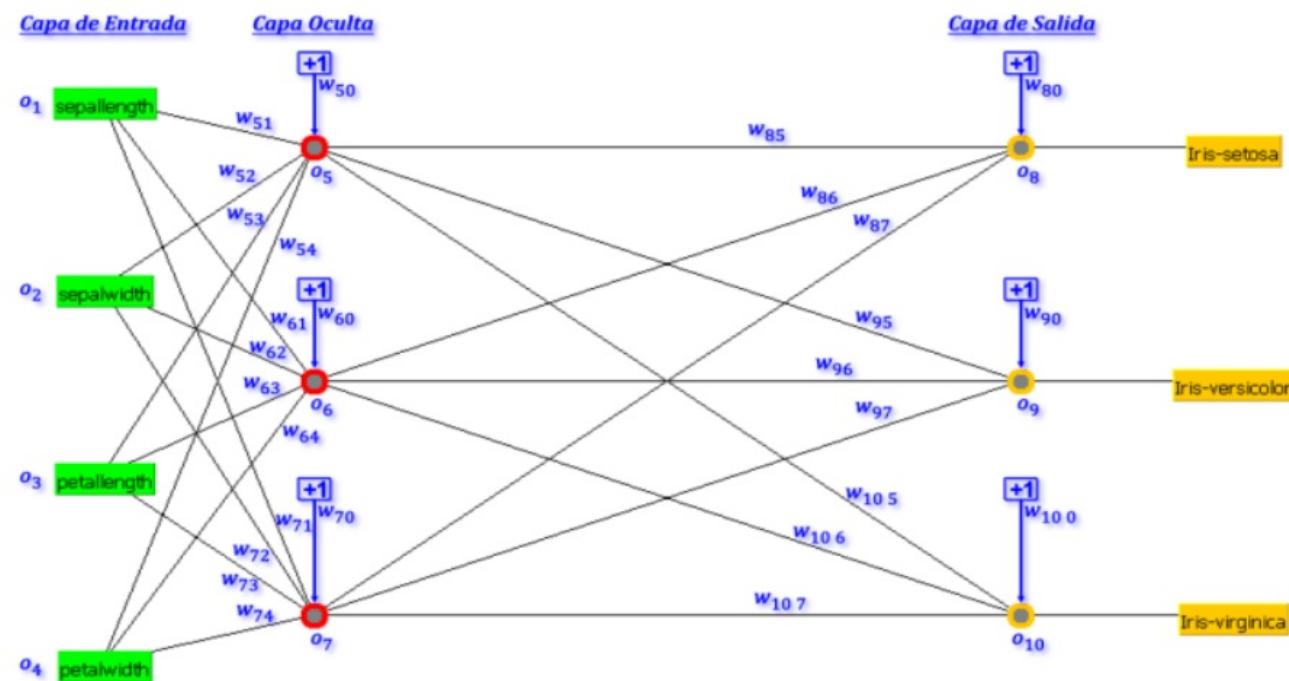
Modelo de MLP

- 4 atributos de entrada.
 - 3 neuronas tipo sigmoide capa oculta.
 - 3 neuronas de salida (una por clase).



MultilayerPerceptron de Weka con Iris

Representación del modelo anterior en **rojo** con un **hold-out 75-25** para Iris.
Los valores en **azul** se detallan en las siguientes diapositivas.



MultilayerPerceptron de Weka con Iris

Modelo obtenido, capa de salida.

	<i>Nodes</i>	<i>Inputs</i>	<i>Weights</i>
<i>Capa de Salida</i>	Sigmoid Node 0 (o_8)	Threshold (+1)	$w_{80} = -3.5015971588434014$
		Node 3 (o_5)	$w_{85} = -1.0058110853859945$
		Node 4 (o_6)	$w_{86} = 9.07503844669134$
		Node 5 (o_7)	$w_{87} = -4.107780453339234$
	Sigmoid Node 1 (o_9)	Threshold (+1)	$w_{90} = 1.0692845992273177$
		Node 3 (o_5)	$w_{95} = 3.8988736877894024$
		Node 4 (o_6)	$w_{96} = -9.768910360340264$
		Node 5 (o_7)	$w_{97} = -8.599134493151348$
	Sigmoid Node 2 (o_{10})	Threshold (+1)	$w_{10\ 0} = -1.007176238343649$
		Node 3 (o_5)	$w_{10\ 5} = -4.2184061338270356$
		Node 4 (o_6)	$w_{10\ 6} = -3.626059686321118$
		Node 5 (o_7)	$w_{10\ 7} = 8.805122981737854$

MultilayerPerceptron de Weka con Iris

Modelo obtenido, capa oculta y capa de entrada.

<i>Capa Oculta</i>	<i>Sigmoid Node 3 (o₅)</i>	<i>Threshold (+1)</i> <i>sepallength (o₁)</i> <i>sepalwidth (o₂)</i> <i>petallength (o₃)</i> <i>petalwidth (o₄)</i>	$w_{50} = 3.382485556685675$ $w_{51} = 0.9099827458022276$ $w_{52} = 1.5675138827531276$ $w_{53} = -5.037338107319895$ $w_{54} = -4.915469682506087$
	<i>Sigmoid Node 4 (o₆)</i>	<i>Threshold (+1)</i> <i>sepallength (o₁)</i> <i>sepalwidth (o₂)</i> <i>petallength (o₃)</i> <i>petalwidth (o₄)</i>	$w_{60} = -3.330573592291832$ $w_{61} = -1.1116750023770083$ $w_{62} = 3.125009686667653$ $w_{63} = -4.133137022912305$ $w_{64} = -4.079589727871456$
	<i>Sigmoid Node 5 (o₇)</i>	<i>Threshold (+1)</i> <i>sepallength (o₁)</i> <i>sepalwidth (o₂)</i> <i>petallength (o₃)</i> <i>petalwidth (o₄)</i>	$w_{70} = -7.496091023618089$ $w_{71} = -1.2158878822058787$ $w_{72} = -3.5332821317534897$ $w_{73} = 8.401834252274096$ $w_{74} = 9.460215580472827$
	<i>Iris-setosa</i> <i>Iris-versicolor</i> <i>Iris-virginica</i>	<i>Node 0 (o₈)</i> <i>Node 1 (o₉)</i> <i>Node 2 (o₁₀)</i>	

Deep Learning: Actualidad

- Conjunto de técnicas que permiten usar redes neuronales de muchas capas y con muchas neuronas.
- Aplicaciones muy importantes en reconocimiento de objetos y en clasificación de imágenes.
- Impulsado también por la disponibilidad de recursos computacionales y el uso de arquitectura hardware específicas (basadas en GPUs - tarjetas gráficas).
- Ejemplo: GoogleNet Inception v4 (red neuronal con muchas capas adaptada para reconocimiento de imágenes):
 - ▶ Red neuronal entrenada con 1.28 millones de imágenes para distinguir entre 1000 tipos de objetos (clases).
 - ▶ Más de 60 millones de pesos en la red.
 - ▶ **GoogleNet Inception**

Árboles de decisión

Entrenamiento en árboles de decisión

Árboles de decisión en Weka

Redes Neuronales

Redes en Weka

Entregables

Bibliografía

Entregables para el guión final

1. Escoja una de las bases de datos de clasificación para el trabajo de las dispuestas en Moodle (Breast cancer, Dermatology, Fantasmas, Glass, Vehicle, Wine, Zoo).

Se entiende que además de pasarla a formato .arff ya ha aplicado el preprocesamiento necesario en función del fichero “**Pistas sobre los datasets con posible preprocesamiento a simple vista.pdf**”, en el caso de que sea una de las bases de datos que lo requiera.

- ▶ Cargue la base de datos y ejecute el **algoritmo C4.5** usando un 75 % para entrenar y un 25 % para generalizar, con los parámetros por defecto.
- ▶ Analice y muestre el árbol obtenido con los parámetros por defecto: nodo principal, número de nodos u hojas, variables presentes y omitidas. Comente también los resultados de las métricas obtenidas.

Entregables para el guión final

2. Escoja una de las bases de datos de clasificación para el trabajo de las dispuestas en Moodle (Breast cancer, Dermatology, Fantasmas, Glass, Vehicle, Wine, Zoo).

Se entiende que además de pasarla a formato .arff ya ha aplicado el preprocesamiento necesario en función del fichero “**Pistas sobre los datasets con posible preprocesamiento a simple vista.pdf**”, en el caso de que sea una de las bases de datos que lo requiera.

- ▶ Cargue la base de datos con un 75/25 % y ejecute el algoritmo *Multilayer-Perceptron* con los valores por defecto.
- ▶ ¿Qué observa al ir modificando solo el *TrainingTime*?.. ¿Cambia el valor de *Correctly Classified instances* al modificar el parámetro?, ¿se estanca el aprendizaje o sobreentrena?.
- ▶ ¿Qué observa al ir modificando solo el *LearningRate*?.. ¿Cambia el valor de *Correctly Classified instances* al modificar el parámetro?, ¿se estanca el aprendizaje o sobreentrena?.

Bibliografía adicional a la de la asignatura y al material de Moodle



Weka: The workbench for machine learning, 2020.
[https://www.cs.waikato.ac.nz/ml/weka/.](https://www.cs.waikato.ac.nz/ml/weka/)

¿Preguntas?