MANUAL DE WEKA

Diego García Morate diego.garcia.morate(at)gmail.com

Acerca de este documento...

Índice

1. Introducción

La Weka (*Gallirallus australis*) es un ave endémica de Nueva Zelanda. Esta Gallinácea en peligro de extinción es famosa por su curiosidad y agresividad. De aspecto pardo y tamaño similar a una gallina las wekas se alimentan fundamentalmente de insectos y frutos.

Figura 1: Imagen de una Weka

Este ave da nombre a una extensa colección de algoritmos de Máquinas de conocimiento desarrollados por la universidad de Waikato (Nueva Zelanda) implementados en Java [1, 2]; útiles para ser aplicados sobre datos mediante los interfaces que ofrece o para embeberlos dentro de cualquier aplicación. Además Weka contiene las herramientas necesarias para realizar transformaciones sobre los datos, tareas de clasificación, regresión, clustering, asociación y visualización. Weka está diseñado como una herramienta orientada a la extensibilidad por lo que añadir nuevas funcionalidades es una tarea sencilla.

Sin embargo, y pese a todas las cualidades que Weka posee, tiene un gran defecto y éste es la escasa documentación orientada al usuario que tiene junto a una usabilidad bastante pobre, lo que la hace una herramienta díficil de comprender y manejar sin información adicional. Este manual tiene por objetivo explicar el funcionamiento básico de este programa (en su versión 3.4-2) y sentar unas bases para que el lector pueda ser autodidacta.

La licencia de Weka es GPL[*], lo que significa que este programa es de libre distribución y difusión. Además, ya que Weka está programado en Java, es independiente de la arquitectura, ya que funciona en cualquier plataforma sobre la que haya una máquina virtual Java disponible.

2. Instalación y Ejecución

Una vez descomprimido Weka y teniendo apropiadamente instalada la máquina de virtual Java, para ejecutar Weka simplemente debemos ordenar dentro del directorio de la aplicación el mandato:

```
java -jar weka.jar
```

No obstante, si estamos utilizando la máquina virtual de Java de Sun (que habitualmente es la

[*] ***G**NU **P**ublic **L**icense*. `http://www.gnu.org/copyleft/gpl.html`

más corriente), este modo de ejecución no es el más apropiado, ya que, por defecto, asigna sólo 100 megas de memoria de acceso aleatorio para la máquina virtual, que muchas veces será insuficiente para realizar ciertas operaciones con Weka (y obtendremos el consecuente error de insuficiencia de memoria); por ello, es altamente recomendable ordenarlo con el mandato:

```
java -Xms<memoria-mínima-asignada>M
    -Xmx<memoria-máxima-asignada>M -jar weka.jar
```

Dónde el parámetro `-Xms` indica la memoria RAM mínima asignada para la máquina virtual y `-Xmx` la máxima memoria a utilizar, ambos elementos expresados en Megabytes si van acompañados al final del modificador "`M`". Una buena estrategia es asignar la mínima memoria a utilizar alrededor de un 60 % de la memoria disponible. Para obtener la lista de opciones estándar, si usamos la máquina virtual de Sun, ordenamos el mandato `java -h`, y para ver la lista de opciones adicionales `java -x`.

Figura 2: Ventana de selección de Interfaz

Una vez que Weka esté en ejecución aparecerá una ventana denominada *selector de interfaces* (figura 2), que nos permite seleccionar la interfaz con la que deseemos comenzar a trabajar con Weka. Las posibles interfaces a seleccionar son *Simple Cli*, *Explorer*, *Experimenter* y *Knowledge flow* que se explicarán detenidamente y de forma individual en secciones siguientes.

3. Ficheros .arff

Nativamente Weka trabaja con un formato denominado `arff` , acrónimo de *Attribute-Relation File Format*. Este formato está compuesto por una estructura claramente diferenciada en tres partes:

1. **Cabecera.** Se define el nombre de la relación. Su formato es el siguiente:

   ```
   @relation <nombre-de-la-relación>
   ```

   Donde `<nombre-de-la-relación>` es de tipo `String`[*]. Si dicho nombre contiene algún espacio será necesario expresarlo entrecomillado.

2. **Declaraciones de atributos.** En esta sección se declaran los atributos que compondrán nuestro archivo junto a su tipo. La sintaxis es la siguiente:

   ```
   @attribute <nombre-del-atributo> <tipo>
   ```

[*]Entendiendo como tipo String el ofrecido por Java.

Donde `<nombre-del-atributo>` es de tipo `String` teniendo las mismas restricciones que el caso anterior. Weka acepta diversos tipos, estos son:

a) **NUMERIC** Expresa números reales[*].

b) **INTEGER** Expresa números enteros.

c) **DATE** Expresa fechas, para ello este tipo debe ir precedido de una etiqueta de formato entrecomillada. La etiqueta de formato está compuesta por caracteres separadores (guiones y/o espacios) y unidades de tiempo:

- dd Día.
- MM Mes.
- yyyy Año.
- HH Horas.
- mm Minutos.
- ss Segundos.

d) **STRING** Expresa cadenas de texto, con las restricciones del tipo `String` comentadas anteriormente.

e) **ENUMERADO** El identificador de este tipo consiste en expresar entre llaves y separados por comas los posibles valores (caracteres o cadenas de caracteres) que puede tomar el atributo. Por ejemplo, si tenemos un atributo que indica el tiempo podría definirse:

```
@attribute tiempo {soleado,lluvioso,nublado}
```

3. **Sección de datos.** Declaramos los datos que componen la relación separando entre comas los atributos y con saltos de línea las relaciones.

```
@data
4,3.2
```

Aunque éste es el modo “completo" es posible definir los datos de una forma abreviada (*sparse data*). Si tenemos una muestra en la que hay muchos datos que sean 0 podemos expresar los datos prescindiendo de los elementos que son nulos, rodeando cada una de las filas entre llaves y situando delante de cada uno de los datos el número de atributo[**].

Un ejemplo de esto es el siguiente

```
@data
{1 4, 3 3}
```

En este caso hemos prescindido de los atributos 0 y 2 (como mínimo) y asignamos al atributo 1 el valor 4 y al atributo 3 el valor 3.

[*]Debido a que Weka es un programa Anglosajón la separación de la parte decimal y entera de los números reales se realiza mediante un punto en vez de una coma.

[**]La numeración de atributos comienza desde el 0, por lo que primero es el 0 y no el 1

En el caso de que algún dato sea desconocido se expresará con un símbolo de cerrar interrogación (“?").

Es posible añadir comentarios con el símbolo “ %”, que indicará que desde ese símbolo hasta el final de la línea es todo un comentario. Los comentarios pueden situarse en cualquier lugar del fichero.

Un ejemplo de un archivo de prueba.

prueba.arff

```
% Archivo de prueba para Weka.
@relation prueba

@attribute nombre STRING
@attribute ojo_izquierdo {Bien,Mal}
@attribute dimension NUMERIC
@attribute fecha_analisis DATE "dd-MM-yyyy HH:mm"

@data
Antonio,Bien,38.43,"12-04-2003 12:23"
'Maria Jose',?,34.53,"14-05-2003 13:45"
Juan,Bien,43,"01-01-2004 08:04"
Maria,?,?,"03-04-2003 11:03"
```

4. Simple CLI

Simple CLI es una abreviación de *Simple Client*. Esta interfaz (figura 3) proporciona una consola para poder introducir mandatos. A pesar de ser en apariencia muy simple es extremadamente potente porque permite realizar cualquier operación soportada por Weka de forma directa; no obstante, es muy complicada de manejar ya que es necesario un conocimiento completo de la aplicación.

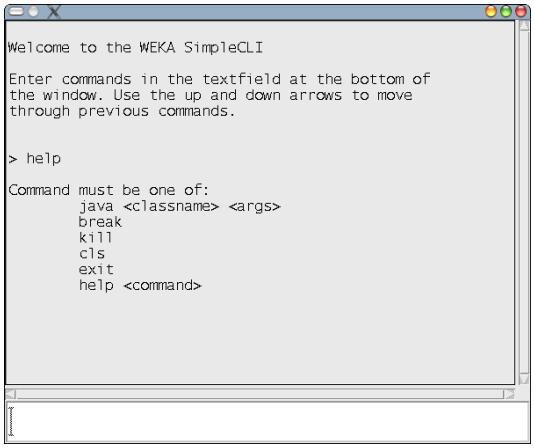

Figura 3: Interfaz de modo consola

Su utilidad es pequeña desde que se fue recubriendo Weka con interfaces. Actualmente ya prácticamente *sólo* es útil como una herramienta de ayuda a la fase de pruebas.

Los mandatos soportados son los siguientes:

`java <nombre-de-la-clase><args>`
: Ejecuta el método “main” de la clase dada con los argumentos especificados al ejecutar este mandato (en el caso de que realmente se haya proporcionado alguno). Cada mandato es atendido en un hilo independiente por lo que es posible ejecutar varias tareas concurrentemente.

`break`
: Detiene la tarea actual.

`kill`
: “Mata” la tarea actual. Este comando es desaconsejable, sólo debe usarse en el caso de que la tarea no pueda ser detenida con el mandato `break`.

`cls` (***Cl**ear **S**creen*).
: Limpia el contenido de la consola.

`exit`
: Sale de la aplicación.

`help <mandato>`
: Proporciona una breve descripción de cada mandato.

5. Explorer

El modo *Explorador* es el modo más usado y más descriptivo[*]. Éste permite realizar operaciones sobre un sólo archivo de datos. La ventana principal es la mostrada en la figura 4.

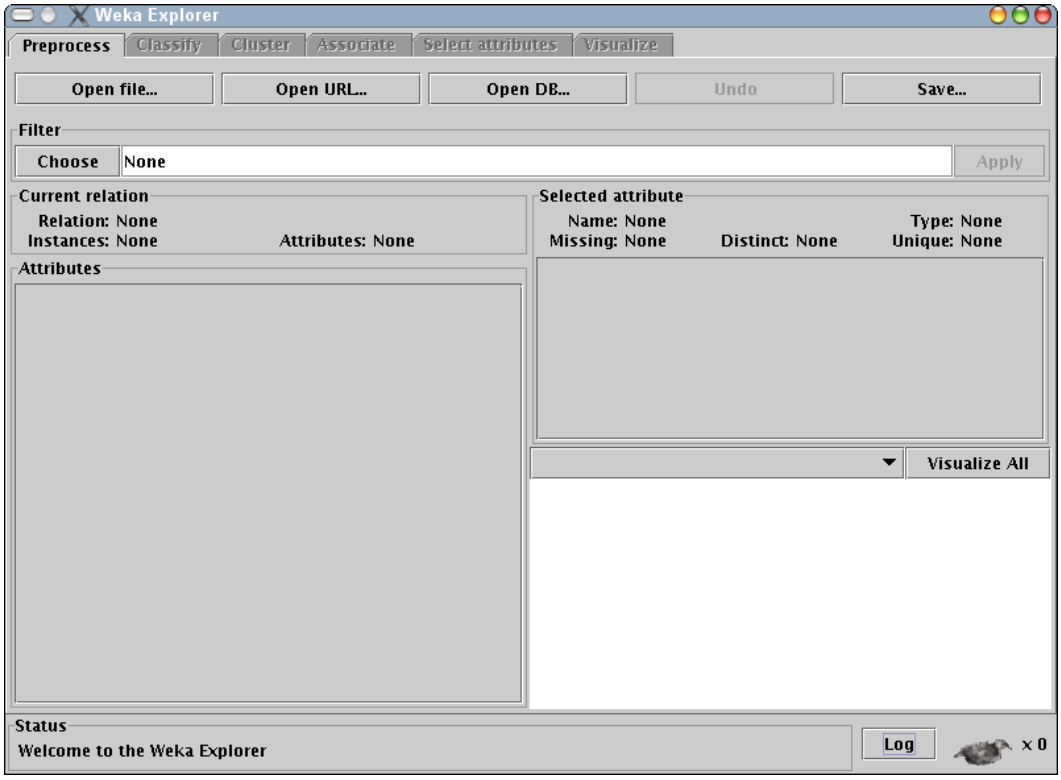

Figura 4: El modo explorador. Ventana inicial.

[*]Es muy importante reseñar que Weka es un programa en continuo desarrollo y cada una de las 4 interfaces que posee evoluciona en separado por lo que actualmente (2004) no son equivalentes por completo, es decir, hay ciertas funcionalidades sólo realizables por algunas de las interfaces y no por otras y viceversa

El explorador permite tareas de:

1. Preprocesado de los datos y aplicación de filtros.
2. Clasificación.
3. *Clustering*.
4. Búsqueda de Asociaciones.
5. Selección de atributos.
6. Visualización de datos.

5.1. Preprocesado

El primer paso para comenzar a trabajar con el explorador es definir el origen de los datos. Weka soporta diferentes fuentes que coinciden con los botones que están debajo de las pestañas superiores mostrados en la ventana 4. Las diferentes posibilidades son las siguientes:

Open File

Al pulsar sobre este botón aparecerá una ventana de selección de fichero. Aunque el formato por defecto de Weka es el `arff` eso no significa que sea el único que admita, para ello tiene interpretadores de otros formatos. Éstos son:

CSV Archivos separados por comas o tabuladores. La primera línea contiene los atributos.

C4.5 Archivos codificados según el formato C4.5. Unos datos codificados según este formato estarían agrupados de tal manera que en un fichero `.names` estarían los nombres de los atributos y en un fichero `.data` estarían los datos en sí. Weka cuando lee ficheros codificados según el formato C4.5 asume que ambos ficheros (el de definición de atributos y el de datos) están en el mismo directorio, por lo que sólo es necesario especificar uno de los dos.

Instancias Serializadas Weka internamente almacena cada muestra de los datos como una instancia de la clase `instance`. Esta clase es *serializable*[*] por lo que estos objetos pueden ser volcados directamente sobre un fichero y también cargados de uno.

Para cargar un archivo `arff` simplemente debemos buscar la ruta donde se encuentra el fichero y seleccionarlo. Si dicho fichero no tiene extensión arff, al abrirlo Weka intentará interpretarlo, si no lo consigue aparecerá un mensaje de error como el de la figura 5.

Pulsando en *Use converter* nos dará la opción de usar un interpretador de ficheros de los tipos ya expuestos.

Open Url

Con este botón se abrirá una ventana que nos permitirá introducir una dirección en la que definir dónde se encuentra nuestro fichero. El tratamiento de los ficheros (restricciones de formato, etc.) es el mismo que el apartado anterior.

[*] En Java un objeto es serializable cuando su contenido (datos) y su estructura se transforman en una secuencia de bytes al ser almacenado. Esto hace que estos objetos puedan ser enviados por algún flujo de datos con comodidad.

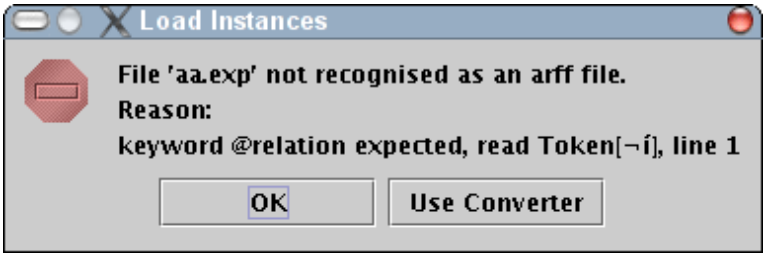

Figura 5: Error al abrir un fichero.

Open DB

Con este botón se nos da la posibilidad de obtener los datos de una base de datos.

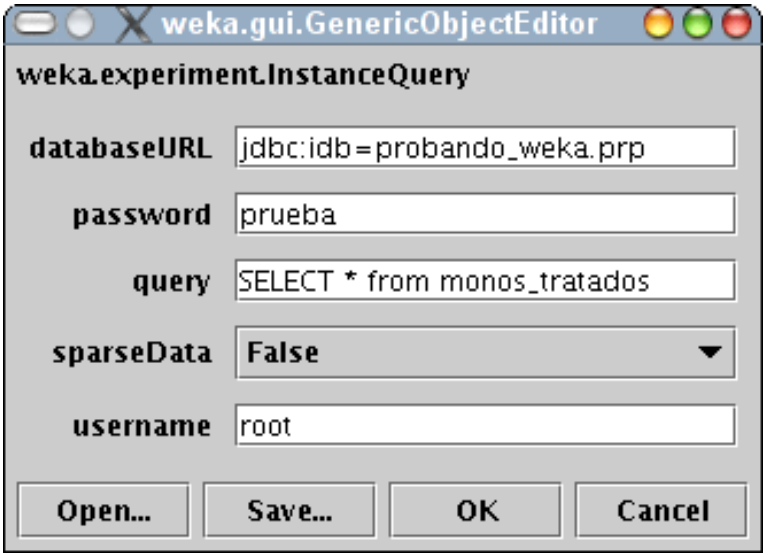

Figura 6: Ventana para seleccionar una base de datos.

Para configurarla lo primero es definir la url por la cual es accesible la base de datos, la contraseña para acceder, el nombre de usuario, la consulta que queremos realizar y si queremos o no usar el modo de datos abreviado (*sparse data*).

Así mismo los botones de la parte inferior dan posibilidad de cargar y guardar esta configuración en un fichero.

Una vez seleccionado el origen de los datos podremos aplicar algún filtro sobre él o bien pasar a las siguientes secciones y realizar otras tareas. Los botones que acompañan a abrir el fichero: **Undo** y **Save**, nos permiten deshacer los cambios y guardar los nuevos datos ya transformados (en formato `arff`). Además, se muestra en la ventana (figura 7) cada uno de los atributos que componen los datos, junto con un resumen con estadísticas de los mismos (media aritmética, rango de los datos, desviación estándar, número de instancias distintas, de qué tipo son, etc.).

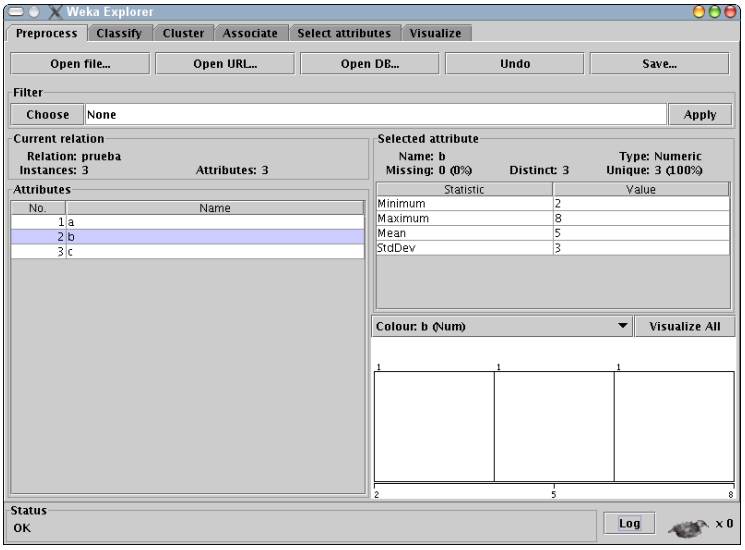

Figura 7: Modo explorador con un fichero de datos cargado.

En el cuadrante inferior derecho aparece una representación gráfica del atributo seleccionado. Encima de ésta hay un menú desplegable que permite variar el atributo de referencia que se representará en color para contrastar ambos atributos. Pulsando en **Visualize all** se abre una ventana desplegable mostrando todas las gráficas pertenecientes a todos los atributos.

5.1.1. Aplicación de filtros

Weka permite aplicar una gran diversidad de filtros sobre los datos, permitiendo realizar transformaciones sobre ellos de todo tipo. Al pulsar el botón **Choose** dentro del recuadro *Filter* se nos despliega un árbol en el que seleccionar los atributos a escoger (figura 8).

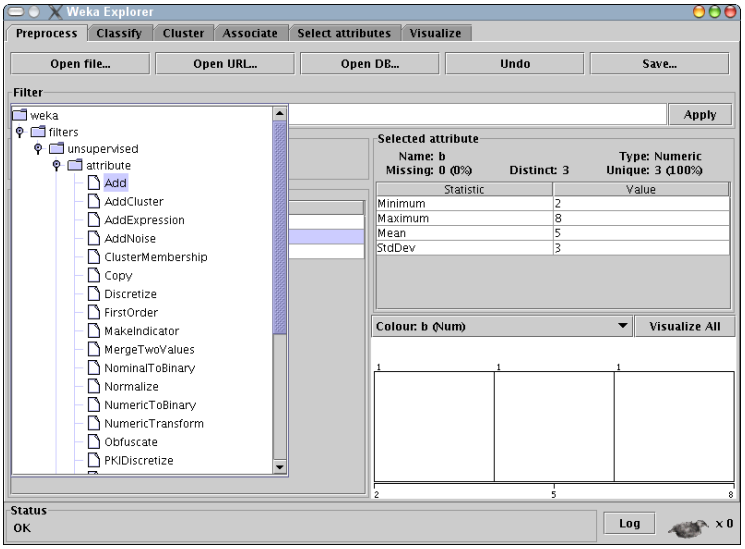

Figura 8: Aplicando un filtro en el modo explorador.

A continuación se hace una breve descripción de cada uno de ellos[*].

Attribute Los filtros agrupados en esta categoría son aplicados a atributos.

Add Añade un atributo más. Como parámetros debemos proporcionarle la posición que va a ocupar este nuevo atributo (esta vez comenzando desde el 1), el nombre del atributo y los posibles valores de ese atributo separados entre comas. Si no se especifican, se sobreentiende que el atributo es numérico.

AddExpression Este filtro es muy útil puesto que permite agregar al final un atributo que sea el valor de una función. Es necesario especificarle la fórmula que describe este atributo, en donde podemos calcular dicho atributo a partir de los valores de otro u otros, refiriéndonos a los otros atributos por "a" seguido del número del atributo (comenzando por 1).

Por ejemplo:

```
(a3^3.4)*a1+sqrt(floor(tan(a4)))
```

Los operadores y funciones que soporta son $+$, $-$, `*`, `/`, `^`, *log*, *abs*, *cos*, *exp*, *sqrt*, *floor* (función techo), *ceil* (función suelo), *rint* (redondeo a entero), *tan*, *sin*, (,).

Otro argumento de este filtro es el nombre del nuevo atributo.

AddNoise Añade ruido a un determinado atributo que debe ser nominal. Podemos especificar el porcentaje de ruido, la semilla para generarlo y si queremos que al introducir el ruido cuente o no con los atributos que faltan.

ClusterMembership Filtro que dado un conjunto de atributos y el atributo que define la clase de los mismos, devuelve la probabilidad de cada uno de los atributos de estar clasificados en una clase u otra.

Tiene por parámetro *ignoredAttributeIndices* que es el rango de atributos que deseamos excluir para aplicar este filtro. Dicho intervalo podemos expresarlo por cada uno de los índices los atributos separados por comas o definiendo rangos con el símbolo guión ("-").

Es posible denotar al primer y último atributo con los identificadores *first* y *last* (en este caso la numeración de los atributos comienza en 1, por lo que *first* corresponde al atributo número 1).

Copy Realiza una copia de un conjunto de atributos en los datos. Este filtro es útil en conjunción con otros, ya que hay ciertos filtros (la mayoría) que destruyen los datos originales. Como argumentos toma un conjunto de atributos expresados de la misma forma que el filtro anterior. También tiene una opción que es *invertSelection* que invierte la selección realizada (útil para copiar, por ejemplo, todos los atributos menos uno).

Discretize Discretiza un conjunto de valores numéricos en rangos de datos. Como parámetros toma los índices de los atributos discretizar (*attribute indices*) y el número de particiones en que queremos que divida los datos (*bins*). Si queremos que las particiones las realice

[*]Por una cuestión de organización interna Weka cataloga todos los filtros dentro de una categoría denominada *Unsupervised*, en futuras versiones se espera que se habiliten filtros de clase *Supervised*.

por la frecuencia de los datos y no por el tamaño de estas tenemos la opción *useEqual-Frecuency*. Si tenemos activa esta última opción podemos variar el peso de las instancias para la definición de los intervalos con la opción *DesiredWeightOfInstancesPerInterval*. Si, al contrario tenemos en cuenta el número de instancias para la creación de intervalos podemos usar *findNumBins* que optimiza el procedimiento de confección de los mismos.

Otras opciones son *makeBinary* que convierte los atributos en binario e *invertSelection* que invierte el rango de los atributos elegidos.

FistOrder Este filtro realiza una transformación de los datos obteniendo la diferencia de pares consecutivos de datos, suponiendo un dato inicial adicional de valor 0 para conseguir que la cardinalidad del grupo de datos resultante sea la misma que la de los datos origen.

Por ejemplo, si los datos son 1 5 4 6, el resultado al aplicar este filtro será 1 4 -1 2. Este filtro toma un único parámetro que es el conjunto de atributos con el que obtener esta transformación.

MakeIndicator Crea un nuevo conjunto de datos reemplazando un atributo nominal por uno booleano (Asignará "1" si en una instancia se encuentra el atributo nominal seleccionado y "0" en caso contrario).

Como atributos este filtro toma el índice el atributo nominal que actuará como indicador, si se desea que la salida del filtro sea numérica o nominal y los índices los atributos sobre los que queremos aplicar el filtro.

MergeTwoValues Fusiona dos atributos nominales en uno solo. Toma como argumentos la posición del argumento resultado y la de los argumentos fuente.

NominalToBinary Transforma los valores nominales de un atributo en un vector cuyas coordenadas son binarias.

Normalize Normaliza todos los datos de manera que el rango de los datos pase a ser $[0,1]$. Para normalizar un vector se utiliza la fórmula:

$$X(i) = \frac{x(i)}{\sqrt{\sum_{i=1}^{n} x(i)^2}}$$

NumericToBinary Convierte datos en formato numérico a binario. Si el valor de un dato es 0 o desconocido, el valor en binario resultante será el 0.

NumericTransform Filtro similar a `AddExpression` pero mucho más potente. Permite aplicar un método java sobre un conjunto de atributos dándole el nombre de una clase y un método.

Obfuscate Ofusca todas las cadenas de texto de los datos. Este filtro es muy útil si se desea compartir una base de datos pero no se quiere compartir información privada.

PKIDiscretize Discretiza atributos numéricos (al igual que `Discretize`), pero el número de intervalos es igual a la raíz cuadrada del número de valores definidos.

RandomProjection Reduce la dimensionalidad de los datos (útil cuando el conjunto de datos es muy grande) proyectándola en un subespacio de menor dimensionalidad utilizando para ello una matriz aleatoria. A pesar de reducir la dimensionalidad los datos resultantes se

procura conservar la estructura y propiedades fundamentales de los mismos.

El funcionamiento se basa en el siguiente producto de matrices:

$$X(i \times n) * R(n \times m) = Xrp(i \times m)$$

Siendo X la matriz de datos original de dimensiones i (número de instancias) $\times$ n (número de atributos), R la matriz aleatoria de dimensión n (número de atributos) $\times$ m (número de atributos reducidos) y Xrp la matriz resultante siendo de dimensión $i \times m$.

Como parámetros toma el número de parámetros en los que queremos aplicar este filtro (*numberOfAttributes*) y el tipo de distribución de la matriz aleatoria que puede ser:

Sparse 1 $-\sqrt{3}$ con probabilidad $\frac{1}{6}$, 0 con probabilidad $\frac{2}{3}$ y $\sqrt{3}$ con probabilidad $\frac{1}{6}$.

Sparse 3 -1 con probabilidad $\frac{1}{2}$ y 1 con probabilidad $\frac{1}{2}$.

Gaussian Utiliza una distribución gaussiana.

Además de estos parámetros, pueden utilizarse también el número de atributos resultantes después de la transformación expresado en porcentaje del número de atributos totales (*percent*), la semilla usada para la generación de números aleatorios (*seed*), y si queremos que aplique antes de realizar la transformación el filtro `ReplaceMissingValues`, que será explicado más adelante.

Remove Borra un conjunto de atributos del fichero de datos.

RemoveType Elimina el conjunto de atributos de un tipo determinado.

RemoveUseless Elimina atributos que oscilan menos que un nivel de variación. Es útil para eliminar atributos constantes o con un rango muy pequeño. Como parámetro toma el máximo porcentaje de variación permitido, si este valor obtenido es mayor que la variación obtenida la muestra es eliminada.

$$\text{Variación obtenida} = \frac{\text{Número de atributos distintos}}{\text{Número de atributos}} * 100$$

ReplaceMissingValues Reemplaza todos los valores indefinidos por la moda en el caso de que sea un atributo nominal o la media aritmética si es un atributo numérico.

Standarize Estandariza los datos numéricos de la muestra para que tengan de media 0 y la unidad de varianza. Para estandarizar un vector x se aplica la siguiente fórmula:

$$x(i) = \frac{x(i) - \overline{x}}{\sigma(x)}$$

StringToNominal Convierte un atributo de tipo cadena en un tipo nominal.

StringToWordVector Convierte los atributos de tipo `String` en un conjunto de atributos representando la ocurrencia de las palabras del texto. Como atributos toma:

- *DFTranform* que indica si queremos que las frecuencias de las palabras sean transformadas según la regla:

$$\text{frec. de la palabra i en la instancia j} * \log \frac{\text{nº de instancias}}{\text{nº de instancias con la palabra i}}$$

- *TFTransform* Otra regla de transformación:

$$\log(1 + \text{frecuencia de la palabra i en la instancia j})$$

- *attributeNamePrefix*, prefijo para los nombres de atributos creados
- Delimitadores (*delimiters*), conjunto de caracteres escape usados para delimitar la unidad fundamental (*token*). Esta opción se ignora si la opción *onlyAlphabeticTokens* está activada, ya que ésta, por defecto, asigna los *tokens* a secuencias alfabéticas usando como delimitadores todos los caracteres distintos a éstos.
- *lowerCaseTokens* convierte a minúsculas todos los tokens antes de ser añadidos al diccionario.
- *normalizeDocLength* selecciona si las frecuencias de las palabras en una instancia deben ser normalizadas o no.
- *outputWordCounts* cuenta las ocurrencias de cada palabra en vez de mostrar únicamente si están o no están en el diccionario.
- *useStoplist* si está activado ignora todas las palabras de una lista de palabras excluidas (*stoplist*).
- *wordsToKeep* determina el número de palabras (por clase si hay un asignador de clases) que se intentarán guardar.

SwapValues Intercambia los valores de dos atributos nominales.

TimeSeriesDelta Filtro que asume que las instancias forman parte de una serie temporal y reemplaza los valores de los atributos de forma que cada valor de una instancia es reemplazado con la diferencia entre el valor actual y el valor pronosticado para dicha instancia.

En los casos en los que la variación de tiempo no se conozca puede ser que la instancia sea eliminada o completada con elementos desconocidos (símbolo "?"). Opciones:

- *attributeIndices* Especifica el rango de atributos en los que aplicar el filtro.
- *FillWithMissing* Las instancias al principio o final del conjunto de datos, donde los valores calculados son desconocidos, se completan con elementos desconocidos (símbolo "?") en vez de eliminar dichas instancias, que es el comportamiento por defecto.
- *InstanceRange* Define el tamaño del rango de valores que se usará para realizar las restas. Si se usa un valor negativo significa que realizarán los cálculos con valores anteriores.
- *invertSelection* Invierte la selección realizada.

Instance Los filtros son aplicados a instancias concretas enteras.

NonSparseToSparse Convierte una muestra de modo completo a modo abreviado.

Randomize Modifica el orden de las instancias de forma aleatoria.

RemoveFolds Permite eliminar un conjunto de datos. Este filtro está pensado para eliminar una partición en una validación cruzada.

RemoveMisclassified Dado un método de clasificación lo aplica sobre la muestra y elimina aquellas instancias mal clasificadas.

RemovePercentage Suprime un porcentaje de muestras.

RemoveRange Elimina un rango de instancias.

RemoveWithValues Elimina las instancias acordes a una determinada restricción.

Resample Obtiene un subconjunto del conjunto inicial de forma aleatoria.

SparseToNonSparse Convierte una muestra de modo abreviado a modo completo.Es la operación complementaria a `NonSparseToSparse`.

5.2. Clasificación

Pulsando en la segunda pestaña (zona superior) del explorador entramos en el modo clasificación (figura 9). En este modo podremos clasificar por varios métodos los datos ya cargados.

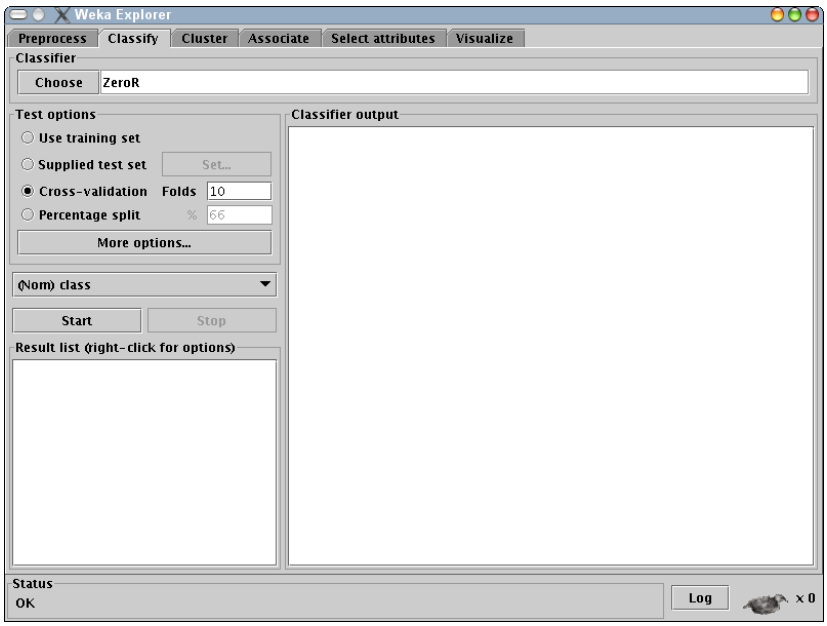

Figura 9: El modo clasificación dentro del explorador.

Si queremos realizar una clasificación lo primero será elegir un clasificador y configurarlo a nuestro gusto, para ello pulsaremos sobre el botón **Choose** dentro del área *Classifier*. Una vez pulsado se desplegará un árbol que nos permitirá seleccionar el clasificador deseado. Una vez seleccionado aparecerá, en la etiqueta contigua al botón **Choose**, el filtro seleccionado y los argumentos con los que se ejecutará. Esta información es muy útil si queremos utilizar el interfaz de consola ya que podremos configurar nuestro filtro con la interfaz y luego obtener el resultado apto para línea de mandato.

Para poder acceder a las propiedades de cada clasificador deberemos hacer *doble-click* sobre la etiqueta antes mencionada. Al darle aparecerá una nueva ventana con las propiedades junto a una breve explicación del mismo. Una vez elegido el clasificador y sus características el próximo paso es la configuración del modo de entrenamiento (*Test Options*). Weka proporciona 4 modos de prueba:

Use training set

Con esta opción Weka entrenará el método con todos los datos disponibles y luego lo aplicará otra vez sobre los mismos.

Supplied test set
Marcando esta opción tendremos la oportunidad de seleccionar, pulsando el botón **Set . . .**, un fichero de datos con el que se probará el clasificador obtenido con el método de clasificación usado y los datos iniciales.

Cross-validation
Pulsando el botón *Cross-validation* Weka realizará una validación cruzada estratificada del número de particiones dado (*Folds*). La validación cruzada consiste en: dado un número n se divide los datos en n partes y, por cada parte, se construye el clasificador con las $n-1$ partes restantes y se prueba con esa. Así por cada una de las n particiones.

Una validación-cruzada es estratificada cuando cada una de las partes conserva las propiedades de la muestra original (porcentaje de elementos de cada clase).

Percentage split
Se define un porcentaje con el que se construirá el clasificador y con la parte restante se probará

Una vez definido el método de prueba Weka nos permite seleccionar algunas opciones más con el botón **More Options** (figura 10).

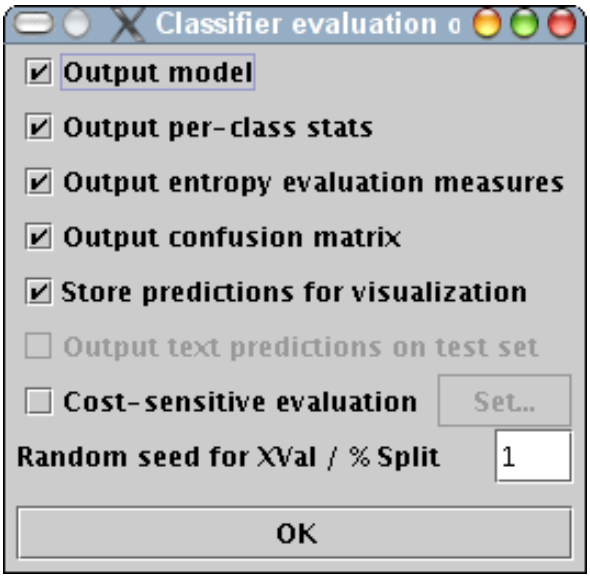

Figura 10: Ventana de opciones adicionales en el modo de clasificación.

Las opciones que se nos presentan son:

Output Model Si la activamos una vez construido y probado el clasificador, nos mostrará en la salida del clasificador (parte media derecha de la ventana 9) el modelo que ha construido.

Output per-class stats Activada muestra estadísticas referentes a cada clase.

Output entropy evaluation mesures Muestra información de mediciones de la entropía en la clasificación.

Output confusion matrix Muestra la matriz de confusión del clasificador. Esta tabla cuyo número de columnas es el número de atributos muestra la clasificación de las instancias. Da una información muy útil porque no sólo refleja los errores producidos sino también informa del tipo de éstos.

Si tuvieramos el problema de clasificar un conjunto de vacas en dos clases gordas y flacas y aplicamos un clasificador, obtendríamos una matriz de confusión como la siguiente.

Gordas	Flacas	
32	4	Gordas
4	43	Flacas

Tabla 1: Ejemplo de una matriz de confusión

Donde las columnas indican las categorías cllasificadas por el clasificador y las filas las categorías reales de los datos. Por lo que los elementos en la diagonal principal son los elementos que ha acertado el clasificador y lo demás son los errores.

Debajo de este cuadro de la ventana existe un menú desplegable que nos permitirá seleccionar un atributo de nuestra muestra. Este atributo es el que actuará como resultado real de la clasificación. Habitualmente este atributo suele ser el último.

Ahora para comenzar un método de clasificación sólo falta pulsar el botón **Start**. Una vez funcionando en la barra de estado aparecerá la información referente al estado del experimento. Cuando acabe, la Weka situada en la esquina inferior derecha dejará de bailar y eso indicará que el experimento ha concluido (figura 11). En la ventana de mensajes del clasificador aparecerá la información referente al desarrollo de éste que hayamos seleccionado.

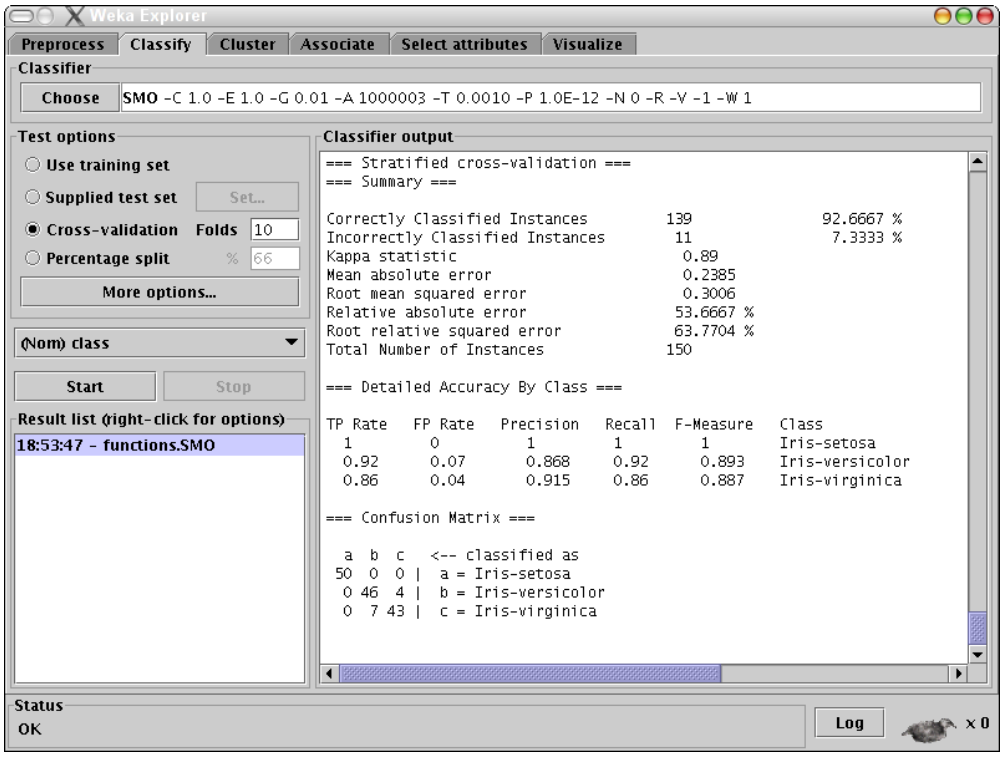

Figura 11: Weka habiendo aplicado un método de clasificación.

En la zona inferior-izquierda se encuentra la lista de resultados en la que aparecerán cada uno de los experimentos que hayamos realizado. Si pulsamos el botón secundario sobre alguno de ellos obtendremos opciones adicionales aplicables al experimento que hayamos seleccionado (figura 12). Éstas permiten visualizar los resultados obtenidos en diferentes variantes, incluyendo gráficas, guardar modelos, etc.

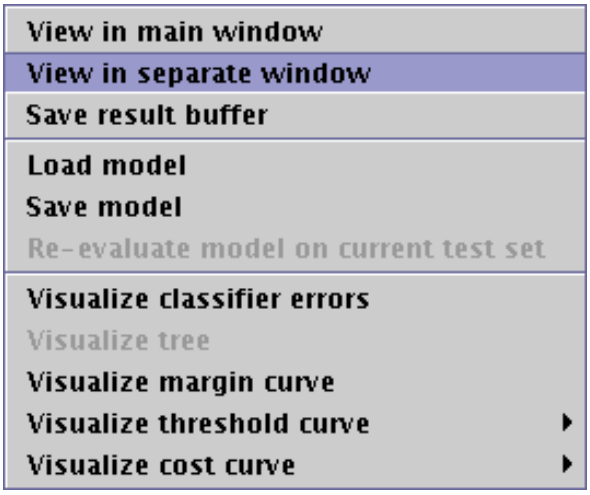

Figura 12: Opciones adicionales de un experimento.

Éstas opciones adicionales son:

View in main window Mostrará el resultado del experimento en la salida estándar del clasificador.

View in separate window Mostrará el resultado del experimento en una nueva ventana.

Save result buffer Guardará el resultado del experimento en un fichero.

Load model Cargará un modelo de clasificador ya construido.

Save model Guardará el modelo de clasificador actual.

Re-evaluate model on current test set Enfrentará un modelo con el conjunto de muestra actual.

Visualize clasiffier errors Se abrirá una nueva ventana (figura 13) en la que nos mostrará una gráfica con los errores de clasificación*.

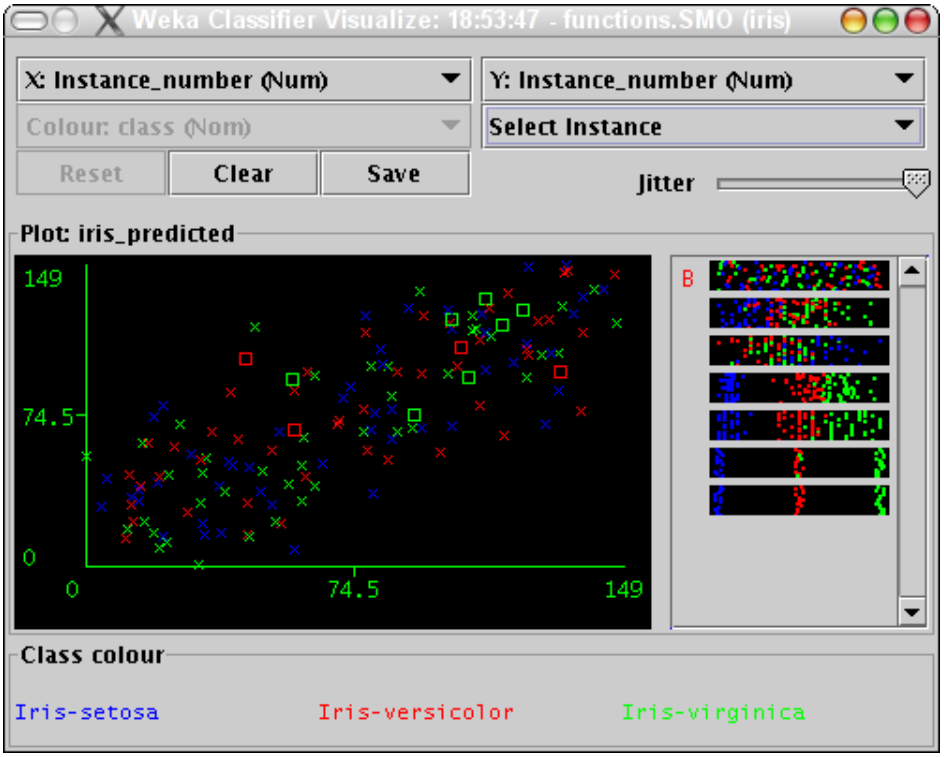

Figura 13: Gráfica de los errores de clasificación.

La ventana del modo gráfica está compuesta por 3 partes. En la parte superior las dos listas desplegables indican el eje X y el eje Y de la gráfica. Es posible variarles a nuestro gusto y obtener todas las combinaciones posibles.

Debajo de la lista desplegable del eje X se encuentra otra en la que podemos definir un atributo que muestre el rango de variación del mismo en colores. El botón contiguo a ésta, en la misma

*Esta opción será sólo accesible si hemos configurado el clasificador para que almacene la clasificación realizada.

línea, sirve para definir el tipo de figura geométrica que queremos utilizar para seleccionar datos en la propia gráfica. Interactuando en la misma seleccionamos un área (figura 14).

Una vez seleccionada a nuestro gusto con el botón **Submit** nos mostrará sólo los datos capturados en el área de la figura que hemos trazado. El botón **Reset** sirve para volver a la situación inicial y el botón **Save** es para guardar los valores de los datos* en un fichero arff.

La barra de desplazamiento, denominada *jitter*, es muy interesante porque permite añadir un ruido aleatorio (en función de lo que se desplace la barra) a los datos. Es útil cuando hay muchos datos juntos y queremos separarlos para poder estimar la cantidad que hay.

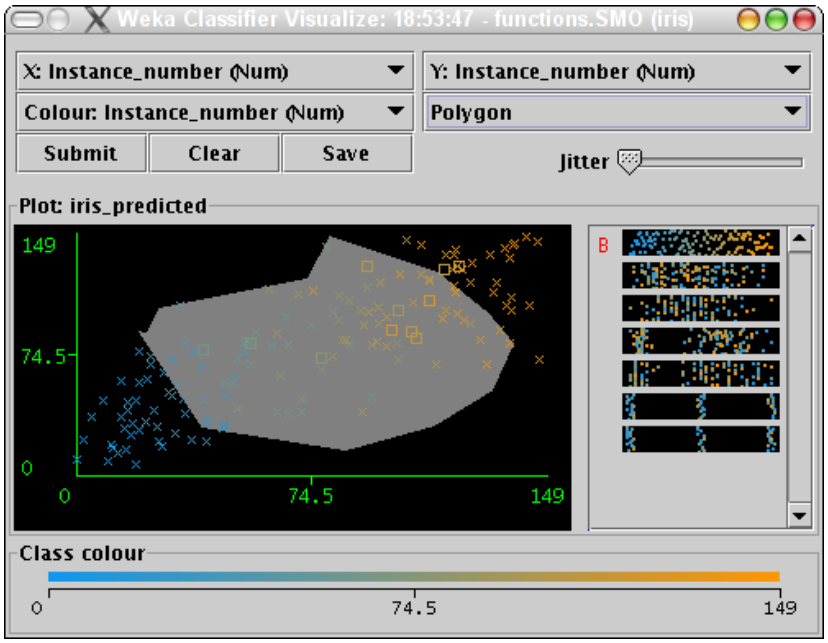

Figura 14: Selección de un área en la gráfica.

En la parte media-derecha de la ventana de la gráfica (fig. 13) se muestra gráficamente, y de una forma muy resumida, la distribución de cada uno de los atributos. Pulsando sobre ellos con el botón principal del ratón definiremos el eje X y con el botón secundario el eje Y.

La última parte (la inferior) de la ventana que muestra gráficas es la leyenda. Pulsando con el ratón en un elemento de la leyenda aparecerá una ventana (figura 15) que nos permitirá modificar los colores en una extensa gama.

*Weka guardará todos los datos de ese apartado, no sólo los pertenecientes a dicha gráfica

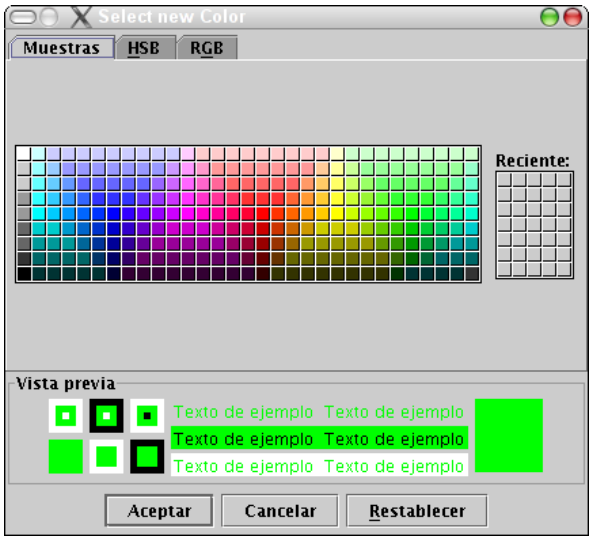

Figura 15: Selección del color para un atributo.

Visualize tree Esta opción mostrará un árbol de decisión, como el de la figura 16, generado por el clasificador, en el caso que lo haya hecho.

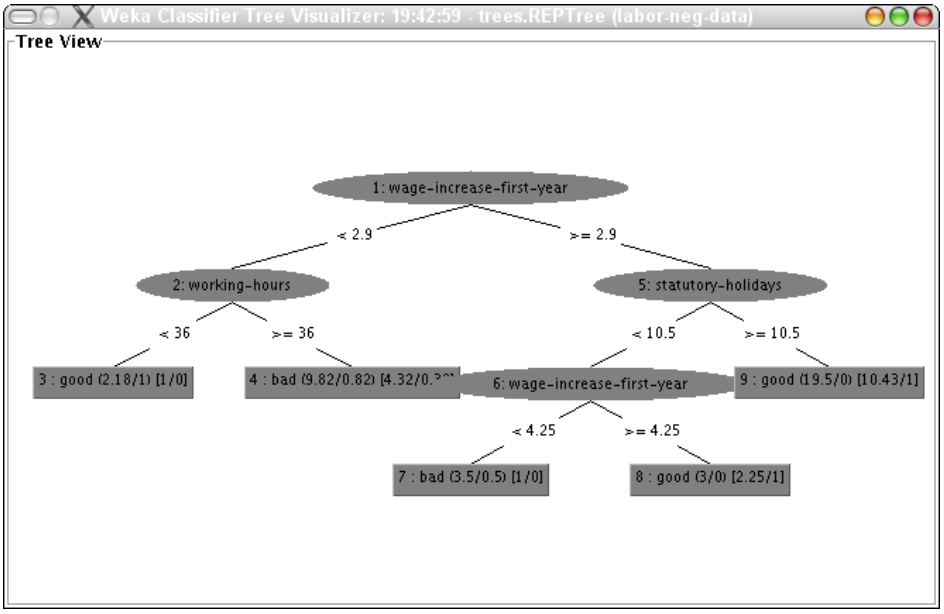

Figura 16: Visualización de árboles de decisión.

Las opciones que existen en esta ventana se despliegan pulsando el botón secundario sobre ella. Éstas son:

Center on Top Node centrará el gráfico en la ventana atendiendo a la posición del nodo superior. Esta opción es muy aconsejable si cambiamos el tamaño de la ventana.

Fit to Screen Ajusta el gráfico a la ventana, no sólo la posición sino también su tamaño.

Auto Scale Escala el gráfico de manera que no haya nodos que colisionen.

Select Font Nos permite ajustar el tamaño del tipo de letra del texto.

Si pulsamos el botón secundario sobre un nodo del grafo nos dará la opción de **Visualize the node**, si hay instancias que correspondan a ese nodo nos las mostrará.

Visualize margin curve Muestra en una curva la diferencia entre la probabilidad de la clase estimada y la máxima probabilidad de otras clases. El funcionamiento de la ventana es igual que la del caso anterior.

Visualize thresold curve Muestra la variación de las proporciones de cada clase. Un truco importante es que si situamos en el eje X los coeficientes falsos positivos y en el eje Y los coeficientes de verdaderos positivos obtendremos una curva ROC, y aparecerá sobre la gráfica el valor del área de ésta.

Visualize cost curve Muestra una gráfica que indica la probabilidad de coste al vavariara sensibilidad entre clases.

5.3. Clustering

Pulsando la tercera pestaña, llamada **Cluster**, en la parte superior de la ventana accedemos a la sección dedicada al clustering (figura 17). El funcionamiento es muy similar al de clasificación: se elije un método de *clustering*, se selecciona las opciones pertinentes y con el botón **Start** empieza el funcionamiento.

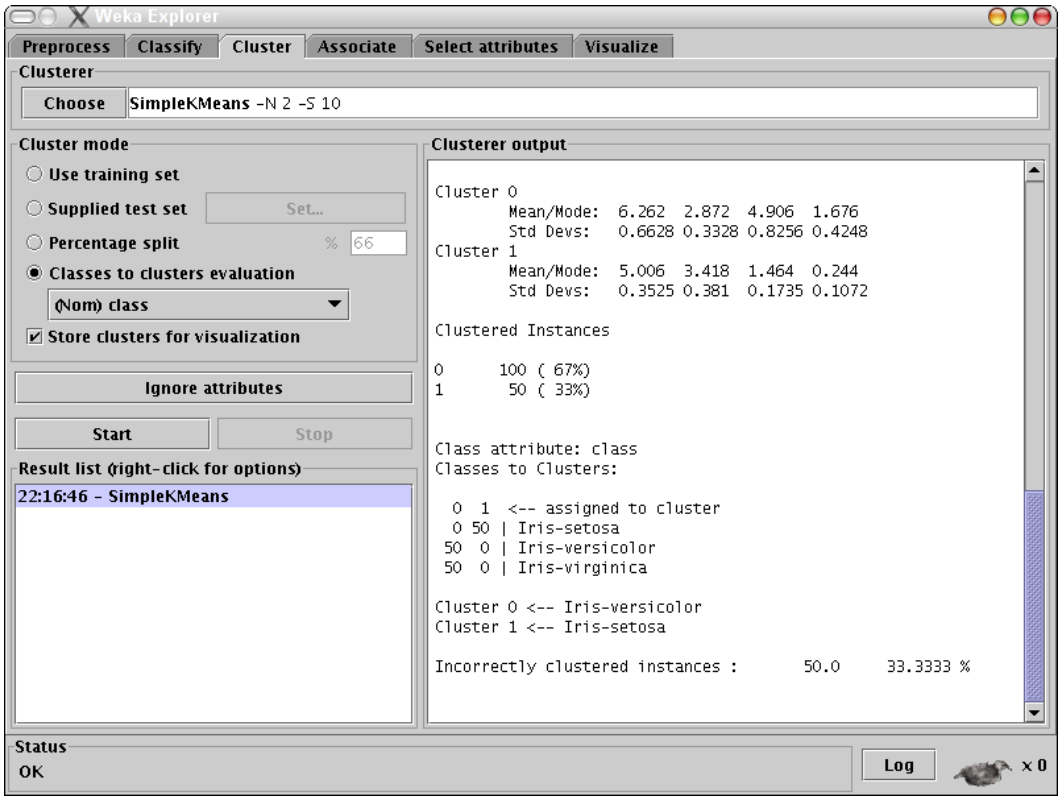

Figura 17: El modo *clustering* dentro del modo explorador.

Una opción propia de este apartado es la posibilidad de ver de una forma gráfica la asignación de las muestras en clusters. Esto se puede conseguir activando la opción *Store cluster for evaluation*, ejecutando el experimento y seguidamente, en la lista de resultados, pulsando el botón secundario sobre el experimento en cuestión y marcando la opción **Visualize cluster assignments** con esto obtendremos una ventana similar a las del modo explorador para mostrar gráficas en el que nos mostrará el *clustering* realizado.

5.4. Búsqueda de Asociaciones

La cuarta pestaña (**Asociate**), muestra la ventana (figura 18) que nos permite aplicar métodos orientados a buscar asociaciones entre datos. Es importante reseñar que estos métodos sólo funcionan con datos nominales. Éste es sin duda el apartado más sencillo y más simple de manejar, carente de apenas opciones, basta con seleccionar un método, configurarlo y verlo funcionar.

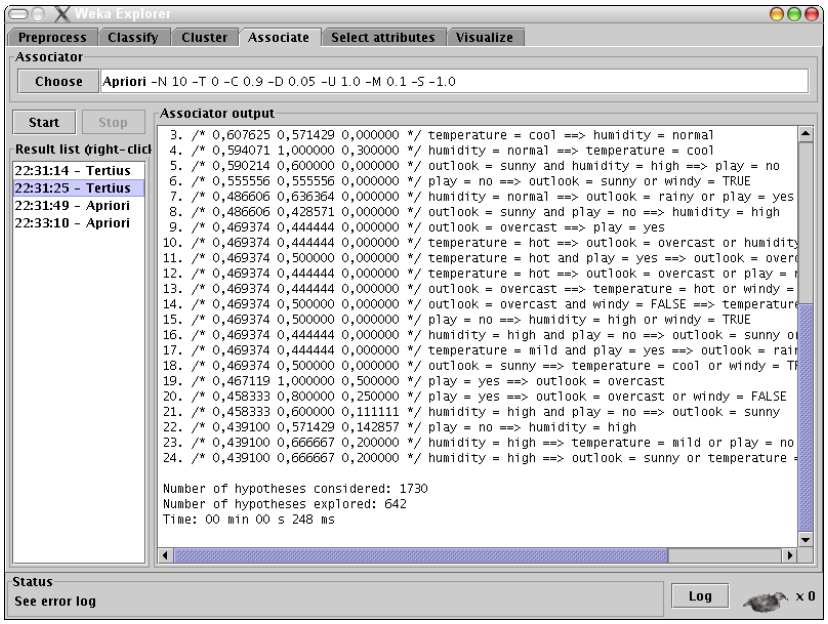

Figura 18: El modo Asociación dentro del modo explorador.

5.5. Selección de Atributos

La pestaña **Select Atributes** nos permite acceder al área de selección atributos (figura 19). El objetivo de estos métodos es identificar, mediante un conjunto de datos que poseen unos ciertos atributos, aquellos atributos que tienen más peso a la hora de determinar si los datos son de una clase u otra.

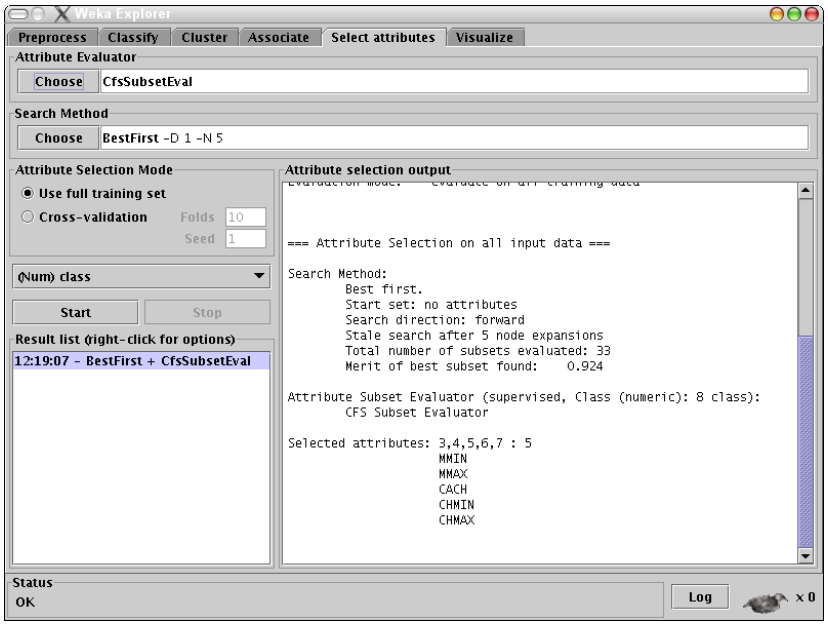

Figura 19: El modo de Selección de Atributos dentro del modo explorador.

Para empezar un método de selección de atributos lo primero es seleccionar el método de evaluación de atributos (*Attribute evaluator*). Este método será el encargado de evaluar cada uno de los casos a los que se le enfrente y dotar a cada atributo de un peso específico. El funcionamiento para seleccionar este método es el mismo que con otros métodos en Weka, se selecciona el método con el botón **Choose** situado dentro del cuadro *Attribute evaluator*. Una vez seleccionado podemos acceder a las propiedades del mismo pulsando sobre el nombre de la etiqueta que muestra el nombre del método seleccionado.

El siguiente paso será elegir el método de búsqueda que será el encargado de generar el espacio de pruebas. El funcionamiento es el mismo al caso anterior. Una vez seleccionado el método de evaluación y el de generación del espacio de pruebas sólo falta elegir el método de prueba, el atributo que representa la clasificación conocida y pulsar **Start**. Una vez acabado el experimento pulsando el botón secundario sobre la etiqueta del experimento en la lista de experimentos realizados tenemos la opción de **Visualize Reduced Data**, que nos mostrará los datos habiendo tomado los mejores atributos en una ventana como la del modo Visualización (*visualize*), que se explicará en el siguiente punto.

5.6. Visualización

El modo visualización (figura 20) es un modo que muestra gráficamente la distribución de todos los atributos mostrando gráficas en dos dimensiones, en las que va representando en los ejes todos los posibles pares de combinaciones de los atributos. Este modo nos permite ver correlaciones y asociaciones entre los atributos de una forma gráfica.

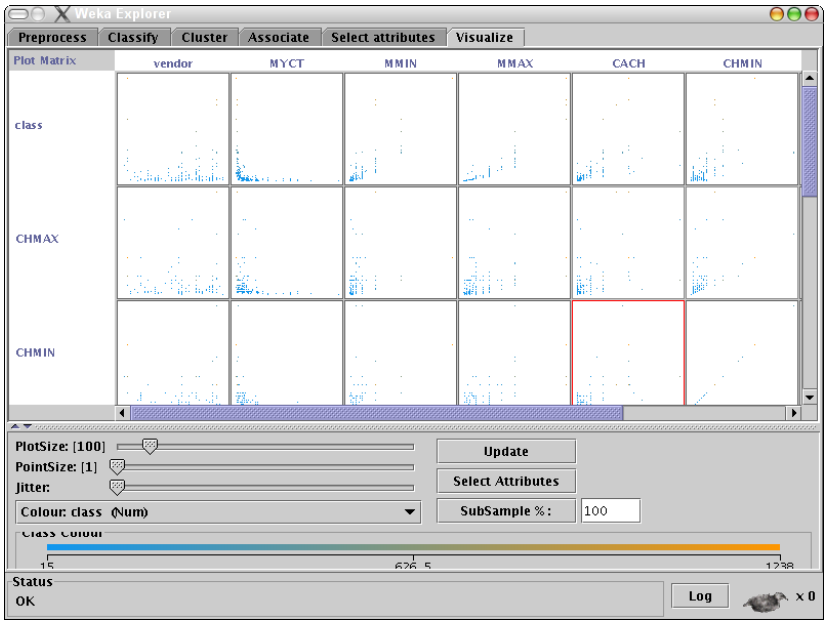

Figura 20: El modo de Visualización dentro del modo explorador.

Pulsando *doble click* sobre cualquier gráfica se nos mostrará en una ventana nueva con el interfaz para gráficas ya explicado. Las opciones que ofrece este modo se activan mediante las barras deslizantes. Las posibles opciones son:

Plotsize define el tamaño del lado de cada una de las gráficas en píxeles, de 50 a 500.

Pointsize define el tamaño del punto expresado en píxeles, de 1 a 10.

Jitter Añade un ruido aleatorio a las muestras, de manera que espacia las muestras que están físicamente muy próximas, esto tiene útilidad cuando se concentran tanto los puntos que no es posible discernir la cantidad de éstos en un área.

Una vez seleccionados los cambios es imprescindible pulsar el botón **Update** para que se representen de nuevo las gráficas. Otro botón útil, es el **Select Atributes**, que nos permite elegir los atributos que se representarán en las gráficas. El último botón que se encuentra en esta ventana es el **Subsample** que permite definir el tanto por ciento de muestras (que escogerá aleatoriamente) que queremos representar.

6. Experimenter

El modo experimentador (*Experimenter*) es un modo muy útil para aplicar uno o varios métodos de clasificación sobre un gran conjunto de datos y, luego poder realizar contrastes estadísticos entre ellos y obtener otros índices estadísticos.

6.1. Configuración

Una vez abierto el modo experimentador obtendremos una ventana como la de la figura 21, que corresponde a la sección *Setup*. Por defecto el modo experimentador está configurado en modo simple, no obstante, esto puede variarse a modo avanzado pulsándo el botón circular acompañado con la etiqueta *Advanced*. Para empezar se explicará el modo simple y posteriormente se pasará a explicar el modo avanzado.

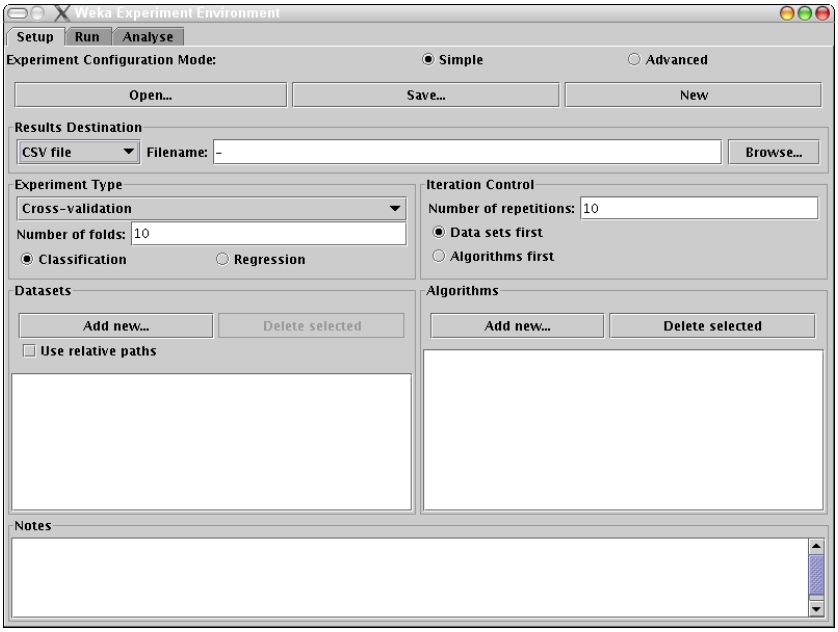

Figura 21: El modo experimentador, modo simple.

Lo primero a realizar es definir un fichero configuración que contendrá todos los ajustes, ficheros involucrados, notas, etc, pertenecientes a un experimento. Con el botón **Open** podemos abrir uno ya creado pero también podremos crear uno nuevo con el botón **New**. Una vez creado podemos guardarlo (Weka por defecto no guarda nada), para lo que usaremos el botón **Save**[*].

El siguiente paso es decidir dónde queremos almacenar los resultados, si es que queremos hacerlo. En caso de decantarnos por no hacerlo, aunque Weka lo permite, luego no se podrán ver los resultados obtenidos. Podemos archivar el resultado del experimento de tres formas distintas, en un fichero arff, en un fichero CSV y en una base de datos. Para ello, simplemente debemos seleccionar la opción que corresponda dentro del cuadro *Results Destination* y a continuación elelegira ruta lógica dónde guardar dicho resultado.

Lo siguiente es definir el tipo de validación que tendrá el experimento, al igual que en el modo *explorer*, tenemos tres tipos de validación: validación-cruzada estratificada, entrenamiento con un porcentaje de la población tomando ese porcentaje de forma aleatoria y entrenamiento con un porcentaje de la población tomando el porcentaje de forma ordenada.

Una vez completados los anteriores pasos, deberemos indicar a Weka qué archivos de datos queremos que formen parte de nuestro experimento. Para ello en el cuadro *Datasets* tenemos un botón llamado **Add new...** que nos permite especificar un archivo de datos a añadir o bien, seleccionando un directorio, nos añadirá todos los archivos que contengan ese directorio de forma recursiva. Junto a ese botón se encuentra la opción *Use relative paths*, que se utiliza para indicar a Weka que almacene las referencias a los ficheros como referencias relativas no indirectas. Puede ser útil si deseamos repetir el experimento en otro ordenador distinto con una estructura de ficheros distinta.

En el cuadro denominado *Iteration control* definiremos el número de repeticiones de nuestro experimento, especificando si queremos que se realicen primero los archivos de datos o los algoritmos. Debajo de este cuadro se encuentra la sección de algoritmos en el que con el botón **Add new...** podremos añadir los algoritmos que ququeramosSi queremos borrar alguno le seleccionamos y pulsamos el botón **Delete selected**.

Si se desea se pueden añadir comentarios y notas al experimento; para ello, Weka dispone de un cuadro de texto en la parte inferior de la ventana.

Con esto ya estaría configurado un experimento en modo simple. En el modo *Advanced* hay ciertas diferencias tanto en los botones (ver figura 22) como en el uso.

[*]Es importante señalar que aunque hayamos guardado una vez, cada vez que se realice un cambio deberemos volver a guardar los cambios con **Save**.

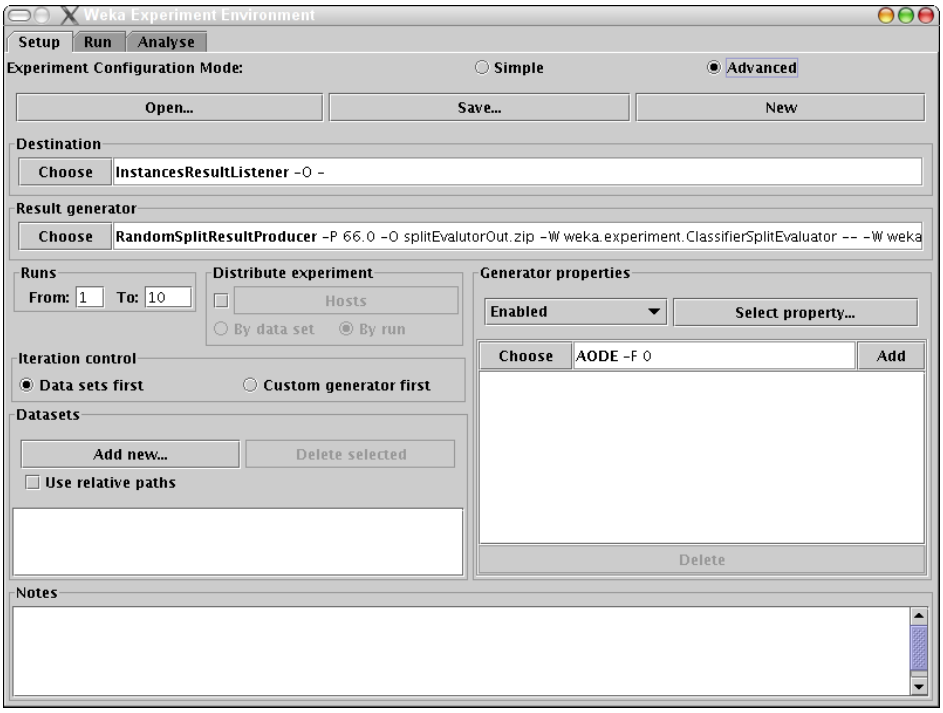

Figura 22: El modo experimentador, modo *Advanced*.

La principal diferencia es que el funcionamiento de este modo está orientado a realizar tareas específicas más concretas que un experimento normal, y una cierta funcionalidad existente en el modo simple se ha traslado al modo avanzado, mostrándola más concreta y explícita al usuario.

El cuadro *Destination* especifica dónde guardará Weka el resultado del experimento. Su funcionalidad es la misma que la del modo simple, pero no su funcionamiento. En este caso deberemos pulsar **Choose** y elegir en el árbol el tipo de método a utilizar para exportar los resultados (Base de datos, fichero arff o fichero csv). Una vez elegido pulsando sobre la etiqueta que indica el método usado, se abrirá una ventana desplegable en donde configuraremos dicho método. Aunque pueda parecer una complicación el cambio en la interfaz para realizar esta operación, ésta segunda permite la adición de nuevos métodos sin variar la interfaz original.

Ahora el cambio fundamental entre una interfaz y otra se basa en el *Result Generator*. En este nuevo modo es necesario seleccionar con el botón **Choose** qué método generador de resultados utilizaremos y seguidamente configurarle. Para ello es necesario seguir siempre el mismo procedimiento, pulsar *doble click* sobre la etiqueta que identifica el método elegido. Los 5 métodos que permite seleccionar Weka son:

- **CrossValidationResultProducer** Genera resultados fruto de una validación cruzada. Tiene por opciones:
  - *numfolds* El número de particiones para la validación cruzada.
  - *outputFile* Fichero donde queremos guardar cada una de las particiones que se realizarán en la validación cruzada. Lo guarda codificado en formato Zip.

Nota: Weka tiene un "fallo" y éste es que al abrir la ventana de diálogo para seleccionar el fichero donde queremos guardar, al elegir uno y pulsar **Select**, lo selecciona pero no cierra dicha ventana. Es por tanto necesario pulsar **Select** y seguidamente cerrar la ventana.

- *rawOutput* Guarda los resultados "tal cual", es decir sin ningún formato fijo. Esta opción es interesante si queremos buscar errores en el programa (*debug mode*).
- *splitEvaluator* El método que seleccionemos para evaluar cada una de las particiones de la validación cruzada. En él especificaremos, de la manera habitual, el clasificador a usar.

- **AveragingResultProducer** Toma los resultados de un método generador de resultados (como podría ser el anterior explicado, *CrossValidationResultProducer*) y cácalculaos promedios de los resultados. Es útil, por ejemplo, para realizar una validación cruzada de muchas particiones por separado y hallar los promedios. Tiene las siguiente opciones:

  - *calculateStdDevs* Seleccionado calcula las desviaciones estándar.
  - *expectedResultsPerAverage* Se elije el número de resultados que se esperan, para dividir la suma de todos los resultados entre este número. Si por ejemplo realizamos una validación cruzada de 10 particiones y asignamos el valor de 10 nos dividirá el resultado de las sumas de cada partición entre 10, es decir, en este caso la media aritmética.
  - *keyFieldname* En esta opción se selecciona el campo que será distintivo y único de cada repetición. Por defecto es "Fold" ya que cada repetición será de una partición distinta (*Fold*).
  - *resultProducer* el método generador de resultados del que tomará los datos.

- **LearningRateResultProducer** Llama a un método generador de resultados para ir repitiendo el experimento variando el tamaño del conjunto de datos. Habitualmente se usa con un *AveragingResultProducer* y *CrossValidationResultProducer* para generar curvas de aprendizaje. Sus opciones son:

  - *lowersize* Selecciona el número de instancias para empezar. Si es 0 se toma como mínimo el tamaño de *stepSize*.
  - *resultProducer* Selecciona el método generador de resultados. Habitualmente para hacer curvas de aprendizaje se usa el *AveragingResultProducer*.
  - *setsize* Número de instancias a añadir por iteración.
  - *upperSize* Selecciona el número máximo de instancias en el conjunto de datos. Si se elije -1 el límite se marca en el número total de instancias.

- **RandomSplitResultProducer** Genera un conjunto de entrenamiento y de prueba aleatorio para un método de clasificación dado.

  - *outputfile* Al igual que antes define el archivo donde se guardarán los resultados.
  - *randomizeData* Si lo activamos cogerá los datos de forma aleatoria.
  - *rawOutput* Activado proporciona resultados sin formato útiles para depurar el programa.
  - *splitEvaluator* Selecciona el método de clasificación a usar.
  - *trainPercent* Establece el porcentaje de datos para entrenar la muestra.

- **DatabaseResultProducer** De una base de datos toma los resultados que coinciden con los obtenidos con un método generador de resultados (cualquiera de los anteriores). Si existen datos que no aparezcan en la base de datos se calculan.

  - *cacheKeyName* Selecciona el nombre del campo de la base de datos a utilizar para formar una caché de la que coger los datos.
  - *databaseURL* Dirección de la base de datos.
  - *resultproducer* Método generador de resultados a utilizar.
  - *username* Nombre de usuario para acceder a la base de datos.
  - *password* Contraseña para acceder a la base de datos.

Una vez que ya está configurado el *Result Generator* podemos utilizar el *Generator properties* para configurar de una forma más sencilla los parámetros seleccionados en el *Result Generator*. Para activarlo, simplemente es necesario pulsar el botón que dentro del campo *Generator properties* pone *Disabled* (desactivado). Una vez pulsado habrá pasado a *Enabled* (activado).

Una vez seleccionado aparecerá una figura 23, en la que podremos seleccionar (marcar con el ratón y pulsar **Select**) algún atributo de los posibles del método. Una vez hecho esto, aparecerá en el recuadro inferior, donde estaba el botón de activación, el valor actual de dicha variable. Escribiendo un nuevo valor y pulsando **Add** podemos añadir uno o más valores y, seleccionándolos y pulsando **Delete** borrarlos. Con el botón **Select property...** seleccionamos otros atributos.

En el caso de que el atributo a modificar sea un método, aparecerá un botón llamado **Choose** que nos permitirá seleccionarlo y modificarlo. Esto es particularmente útil si queremos repetir un experimento con varios clasificadores, pues seleccionado el atributo del clasificador a usar, con ese menú podemos añadir tantos como queramos.

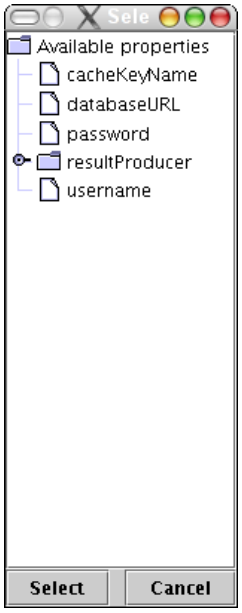

Figura 23: Configurando las opciones de un método de generación de resultados.

El cuadro *runs* permite seleccionar las iteraciones con la que se realizará el experimento.

6.1.1. Weka distribuido

Una de las características más interesantes del modo experimentador es que permite distribuir la ejecución de un experimento entre varios ordenadores mediante Java RMI. El modelo usado es el cliente-servidor. Como primer paso configuraremos un servidor para después explicar la configuración de los clientes.

Configuración del Servidor Los elementos a configurar en el servidor son los mismos que cualquier otro experimento pero además de esto debemos activar el botón de activación del cuadro *Distribute experiment*. Una vez activado es necesario elegir el tipo de distribución, por fichero de datos (*by data set*), o por iteración (*by run*). Un buen consejo es que cuando el número de archivos en el que realizar los experimentos es pequeño la distribución se haga por iteraciones, ya que distribuirá cada una de las tareas fundamentales de un experimento (por ejemplo, en una validación cruzada cada una de las particiones); sin embargo, si usamos un conjunto de archivos grande es recomendable el uso de una distribución por archivos.

Una vez seleccionado pulsaremos en el botón **Hosts** y obtendremos una ventana como la figura 24.

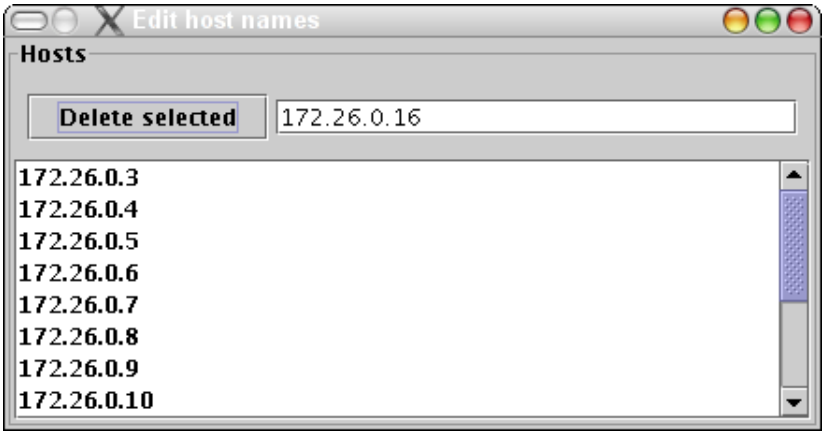

Figura 24: El modo experimentador, configuración de un experimento distribuido

En esta ventana introduciremos en la etiqueta cada uno de los ordenadores que queramos añadir y con el *enter* lo añadiremos. También seleccionando una ip y pulsando **Delete Selected** nos da la opción a borrar un host determinado. Para aceptar los cambios con cerrar la ventana es suficiente.

Con esto ya queda configurado el ordenador que actuará de servidor para la aplicación distribuida.

Configuración de un ordenador cliente Lo primero es copiar el archivo *remoteExperimentServer.jar* situado en la carpeta raíz de Weka, en el ordenador que actuará como cliente. Dicho archivo se encuentra codificado en Jar* pero debemos desempaquetarlo. Para ello se utiliza el siguiente mandato**:

```
jar xvf remoteExperimentServer.jar
```

Con esto queda desempaquetado el archivo `remoteExperimentServer.jar` en el directorio actual. Con esta operación habremos obtenido 3 ficheros: *remoteEngine.jar*, *remote.policy* y *DatabaseUtils.props*. A continuación se hace una breve descripción de los mismos:

*Java ARchives. Este formato de archivos permite empaquetar multitud de ficheros en uno solo para poder almacenar aplicaciones con muchos archivos y que su intercambio y ejecución sea más sencillo.

**Se entiende que la máquina cliente tiene la máquina virtual Java correctamente instalada.

DatabaseUtils.props Habitualmente, cuando se realiza un experimento distribuido lo normal y más cómodo es que los datos estén en una base de datos común a todos los ordenadores que participen en la ejecución del experimento. En este archivo se definen las bases de datos jdbc*, en él lo primero que definiremos es el controlador (*driver*) de jdbc que utilizaremos, que dependerá de la base de datos a utilizar. Las líneas que comiencen con "#" son comentarios.

Lo siguiente será determinar la dirección por la que son accesibles los datos.

remote.policy Archivo que define la política de seguridad. En él se definen las restricciones de acciones a realizar, los puertos y host a utilizar, etc.

remoteEngine.jar Archivo *jar* en el que se encuentra el cliente para realizar experimentos de forma remota.

Una vez configurados *DatabaseUtils.props* y *remote.policy* el siguiente paso es ejecutar el cliente, para ello haremos uso del mandato (todo es la misma línea):

```
java -classpath remoteEngine.jar:/controlador_jdbc \
    -Djava.security.policy=remote.policy \
    -Djava.rmi.server.codebase=
        file:/ruta_al_dir_actual/remoteEngine.jar \
    weka.experiment.RemoteEngine
```

Con esto ya estaría configurado y funcionando un cliente. El siguiente paso sería añadir tantos clientes como deseemos.

6.2. Ejecución

La ventana para comenzar la ejecución del experimento se obtiene pulsando la segunda pestaña superior, etiquetada como *Run*, obtendremos una ventana que será similar a la de la figura 25.

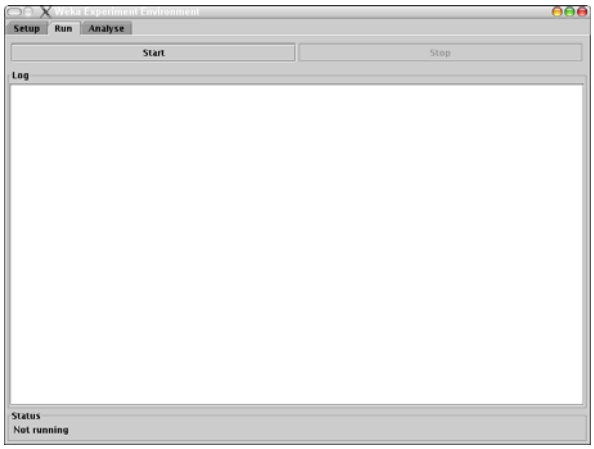

Figura 25: El modo experimentador, ventana de ejecución del experimento.

* *jdbc* es una tecnología que permite la interconexión de bases de datos con aplicaciones Java.

El funcionamiento es obvio, pulsar el botón **Start** para comenzar el experimento y el botón **Stop** si queremos detenerle en plena ejecución. Un detalle importante es que los experimentos no se pueden parar y luego continuar en el punto en que se pararon.

6.3. Análisis de resultados

Pulsando la pestaña *Analyse* entraremos la sección dedicada a analizar los datos (figura 26). Ésta nos permite ver los resultados de los experimentos, realizar contrastes estadísticos, etc.

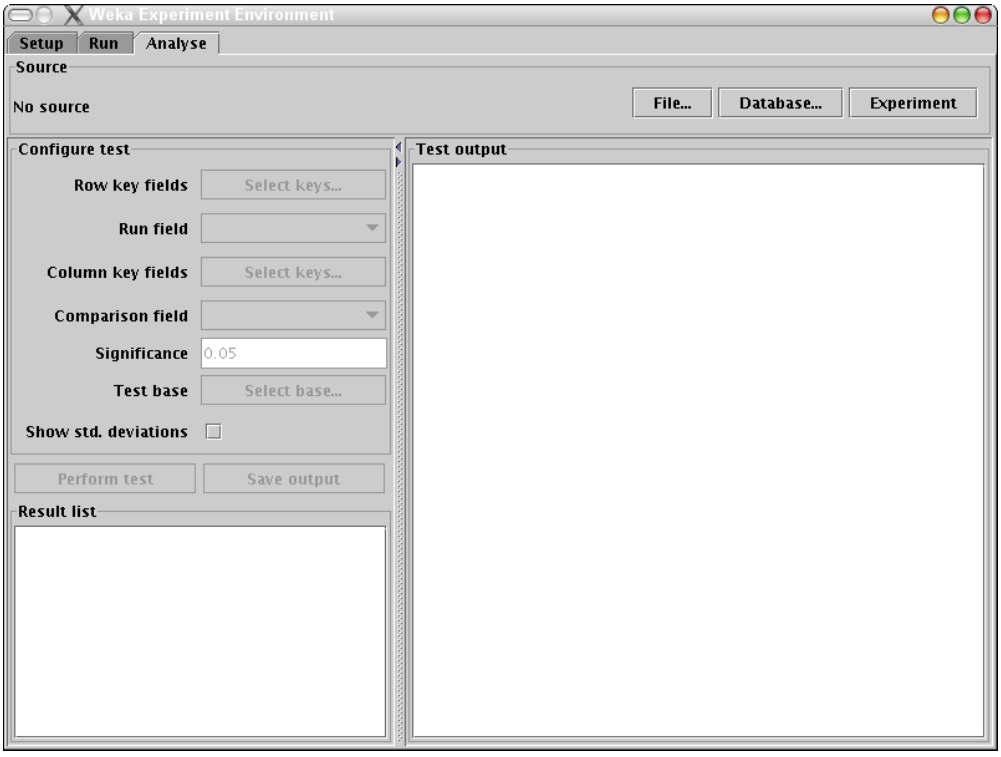

Figura 26: El modo experimentador, ventana de análisis de resultados.

Lo primero a definir es el origen de los datos de los resultados, con los botones **File**, **Database** o **Experiment** (este último procesa los resultados del experimento que está actualmente cargado*).

Una vez hecho esto definiremos el test que queramos realizar; para ello, utilizaremos las opciones que nos ofrece el cuadro *Configure test*. A continuación se realiza una breve descripción de las mismas:

Row key fields Selecciona el atributo o atributos que harán de filas en la matriz de resultados.

Run fields Define el atributo que identifica cada una de las iteraciones.

Column key fields Selecciona los atributos que actuarán de columnas en la matriz de resultados.

Comparison fields El atributo que va a ser comparado en el contraste.

*Es importante señalar que para poder cargar los resultados de un experimento procesado es necesario que hayamos definido un fichero de resultado de los datos.

Significance Nivel de significación para realizar el contraste estadístico. Por defecto es el 0,05.

Test base Seleccionamos qué algoritmo de los utilizados (o el resumen, o el ranking) utilizaremos de base para realizar el test.

Show std. deviations Muestra las desviaciones estándar.

Una vez configurado ya sólo queda utilizar el botón **Perform test** con el que se realizará el test de la t de Student, mostrando el resultado en el cuadro denominado *Test Output*. Con el botón **Save Output** se permite guardar el contenido de la salida del test en un fichero de texto. También, si en la lista de resultados pulsamos el botón secundario del ratón sobre algún elemento, nos mostrará el resultado de ese test en una ventana separada.

Por ejemplo, queremos realizar una comparación entre dos clasificadores, ambos mediante SVM, uno con el kernel lineal y otro gaussiano. Queremos comprobar si hay diferencias significativas entre ambos métodos clasificando sobre la base de datos iris.arff que incluye Weka en el directorio `data`. Para ello una vez realizado el experimento, en el modo análisis de resultados, realizamos un test al 0,05 de confianza, comparando como atributo el porcentaje de acierto (*percent correct*) de ambos clasificadores.

Realizamos el test y obtendremos un resultado como el siguiente:

```
1  Analysing:  Percent_correct
2  Datasets:   1
3  Resultsets: 2
4  Confidence: 0.05 (two tailed)
5  Date:       4/08/04 12:30
6
7
8  Dataset                   (1) functions.SMO ' | (2) functions.SM
9                           ----------------------------------------
10 iris                     (100)   96.27( 4.58) |   88.07(12.62) *
11                          ----------------------------------------
12                                       (v/ /*) |          (0/0/1)
13 Skipped:
14
15 Key:
16
17 (1) functions.SMO '-C 1.0 -E 1.0 -G 0.01 -A 1000003 -T 0.0010
18         -P 1.0E-12 -N 0 -V -1 -W 1' 9220111829601642912
19 (2) functions.SMO '-C 1.0 -E 1.0 -G 0.01 -A 1000003 -T 0.0010
20         -P 1.0E-12 -N 0 -R -V -1 -W 1' 9220111829601642912
```

Como se puede observar en la línea 10, SVM con kernel lineal (función 1) obtiene un 96,27 % (DE. 4,58) de acierto frente a los 88,07 % (DE. 12.62) que se obtienen con un kernel gaussiano. A la derecha del todo aparece un asterisco eso significa que el elemento a comparar es **Peor** que el elemento base. Si hubiera aparecido una "v" significaría lo contrario.

Por tanto, concluimos que hay diferencias significativas entre ambos métodos. Además, en la línea 12, aparece un indicador del tipo `(0/0/1)` siendo (xx,yy,zz), xx el número de veces que el elemento a comparar es mejor que el base, yy el número de veces que son iguales, y zz el número de

veces que es peor. En este caso sólo aparece un 1, debido a que sólo hemos comparado una medida, el porcentaje de aciertos.

7. Knowledge flow

Esta última interface de Weka es quizá la más cuidada y la que muestra de una forma más explícita el funcionamiento interno del programa. Su funcionamiento es gráfico y se basa en situar en el panel de trabajo (zona gris de la figura 27), elementos base (situados en la parte superior de la ventana) de manera que creemos un "circuito" que defina nuestro experimento.

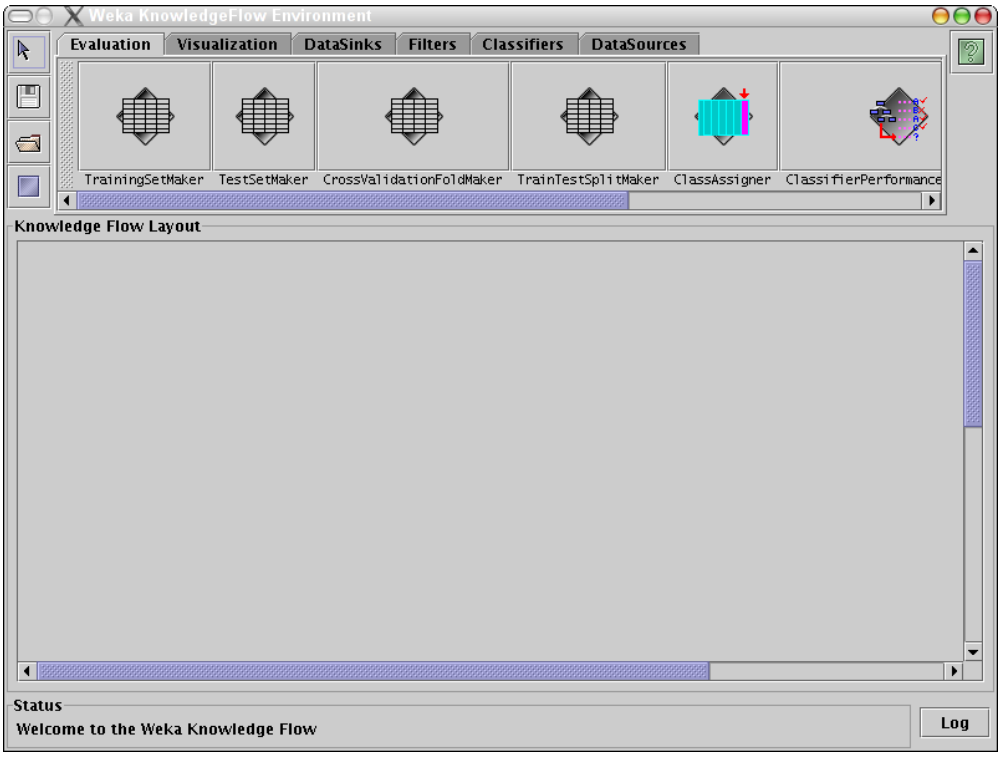

Figura 27: El modo *Knowledge flow*.

Para explicar este modo de funcionamiento se recurrirá a un ejemplo. Supongamos que queremos clasificar mediante máquinas de vectores soporte la muestra (iris.arff) situada en el directorio `data/` del programa. Para ello, lo primero será añadir a nuestro "circuito" una fuente de los datos. Éstas se encuentran en el apartado *Datasources* en la parte superior donde se puede observar que hay cargadores de datos para arff, csv, c45 e instancias serializadas; no obstante, no los hay para bases de datos. Esto es porque Weka es un programa en desarrollo y esta característica aún no está implementada en el modo *Knowledge flow*.

En este ejemplo utilizaremos el cargador *arffloader*. Para utilizarlo pulsaremos sobre él (ver figura 28), el icono del ratón pasará a convertirse en una cruz y podremos situar en la zona de trabajo (zona gris) nuestro cargador. En cualquier momento, con el ratón, podremos arrastrarlo y situarlo dónde creamos conveniente.

Una vez lo hayamos situado es necesario configurarlo. Para ello, pulsando el botón secundario del ratón aparecerá un menú desplegable con varias opciones. Para configurar el archivo `arff` que cargará, pulsamos en *edit/configure* y aparecerá una ventana en la que seleccionaremos el archivo pulsando *doble click* en la etiqueta del archivo. Hecho esto aparecerá una nueva ventana desplegable. En nuestro caso accederemos a `iris.arff`. Si queremos borrar la configuración realizada repetiremos el procedimiento anterior, pulsando en *edit/delete*.

Figura 28: Icono del *arffloader*

El siguiente paso será definir cuál es el atributo que define la clasificación correcta (habitualmente suele ser el último). Para ello haremos uso del objeto *Class asigner* dentro de *Evaluation* que realiza esta tarea. Lo situamos en la zona de trabajo y lo configuramos de la misma forma que el objeto anterior. Una vez ambos objetos estén situados y configurados deberemos unirles de manera que los datos fluyan entre ambos. Abrimos el menú de opciones del cargador (botón secundario sobre el elemento *arffloader*) y mediante la opción *connection/dataset*, Weka nos permitirá dibujar una flecha que una ambos objetos (figura 29). Si queremos eliminarla pulsaremos el botón secundario sobre la flecha y con la opción *Delete* será eliminada.

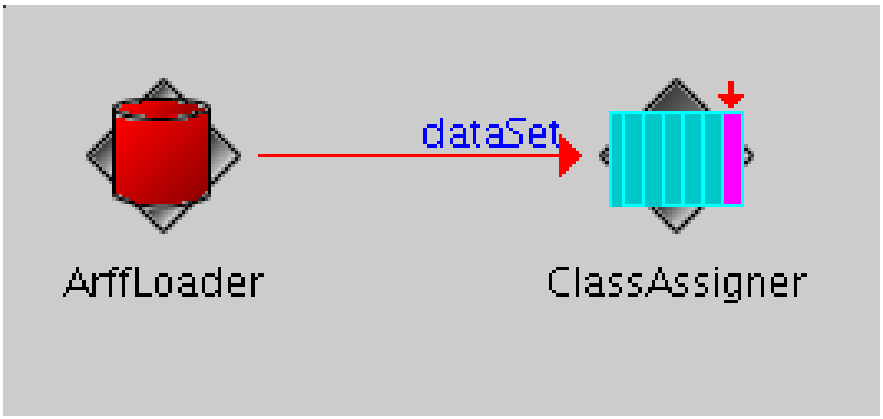

Figura 29: Knowledge flow. El Origen de los datos ya está definido.

En este punto, ya está definido por completo el origen de los datos. El siguiente hito será conseguir realizar una validación cruzada que enfrentar al clasificador. Para ello existe el objeto *Crossvalidationfoldmaker* dentro de *Evaluation*, que será el encargado de realizarnos cada una de las particiones de la validación cruzada. Para configurarlo se utiliza el mismo sistema que antes, pulsar botón secundario y *configure*. Para conectar el *ClassAsigner* con el *Crossvalidationfoldmaker*, seleccionamos la opción *connections/dataset* y dibujamos una flecha al igual que antes.

Ya sólo quedan añadir dos objetos más, el clasificador y algún objeto que nos facilite el resultado, bien sea en modo texto o gráfico. A modo de ejemplo añadiremos dos clasificadores SVM con distintos parámetros. Los objetos que corresponden a los clasificadores se encuentran en el apartado *Classifiers* y son en este caso los *SMO* (que es el algoritmo que utiliza Weka para implementar las SVM). Añadimos dos y los configuramos. Para conectarles pulsaremos en *Crossvalidationfoldermaker* botón secundario y seleccionaremos *connections/testset* uniendo el conjunto de prueba del

generador de pruebas con el clasificador. Es necesario también unir el conjunto de entrenamiento (*connections/trainingset*). Una vez que conectemos por duplicado el generador de particiones a los clasificadores obtendremos el resultado de la figura 30.

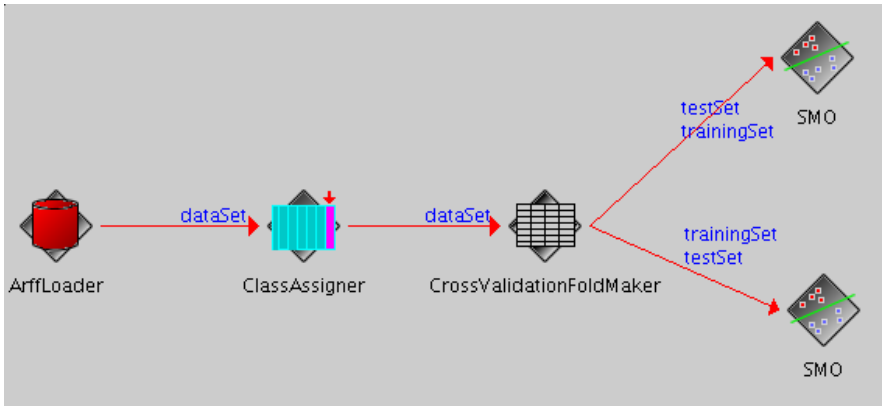

Figura 30: Aún queda definir un elemento que nos muestre el resultado.

El último paso será añadir dos objetos de tipo *textviewer*, que nos permitirán ver el resultado de forma textual. Para conectarlos a los clasificadores *SMO* haremos uso de la opción *connections/text*. Una vez hecho esto ya estaría completada la primera prueba obteniéndose un circuito como el de la figura 31.

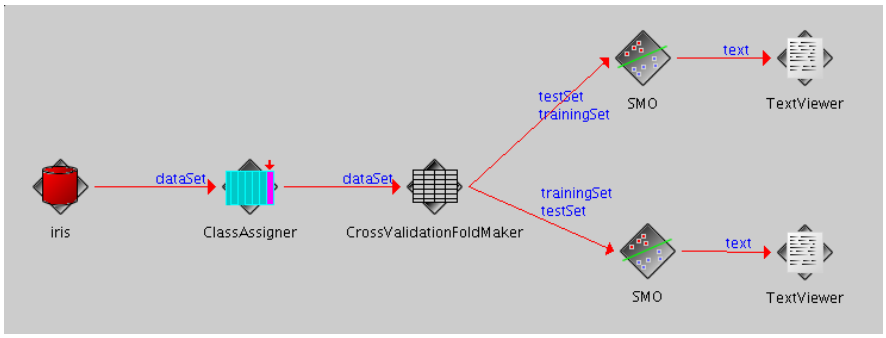

Figura 31: El experimento ya está completo.

Para comenzar con la ejecución del experimento haremos uso de la opción *Actions/Start Loading* dentro del *arffloader*, con esto es suficiente. Para poder observar los resultados, en nuestro caso, con la opción *Actions/Show results* veremos los resultados de cada una de las particiones. De esta forma obtenemos una información mucho más completa que la obtenida del modo explorador.

Esto no hace más que resaltar la necesidad de manejar Weka de una forma completa, porque la comprensión de un sólo modo de funcionamiento es insuficiente para poder obtener el óptimo rendimiento del programa.

8. Modificando Weka

Una de las características más interesantes de Weka es la posibilidad de modificar su código y obtener versiones adaptadas con funcionalidades que no contengan las versiones oficiales. El primer paso para poder modificar Weka es descargarse la última *snapshot* (versión en desarrollo) que se encuentra en el servidor de versiones (CVS). La dirección del mismo es `:pserver:cvs_anon@cvs.scms.waikato.ac.nz:/usr/local/global-cvs/ml_cvs`.

Para operar con dicho servidor es altamente recomendable utilizar algún programa que nos facilite esta tarea. Un buen programa es el *SmartCVS* que es gratuito en su versión normal pero totalmente funcional. Además, está programado en Java lo que lo hace multiplataforma. El sitio web del mismo es `http://www.smartcvs.com`.

Weka hace uso de la herramienta *ant*[*] para la compilación de este proyecto. Una vez que esté todo instalado para compilar Weka haremos uso del mandato `ant` dentro del directorio donde tengamos la versión en desarrollo y obtendremos un resultado como el siguiente:

```
1  hell:/pfc/dev/weka-cvs/weka# ant
2  Buildfile: build.xml
3
4  init_all:
5
6  compile_inline:
7      [javac] Compiling 635 source files to /pfc/dev/weka-cvs
8      [javac] Note: Some input files use or override a deprecated API.
9      [javac] Note: Recompile with -deprecation for details.
10      [rmic] RMI Compiling 1 class to /pfc/dev/weka-cvs
11
12 BUILD SUCCESSFUL
13 Total time: 32 seconds
```

Una vez compilado Weka, podremos ejecutar cada uno de los métodos que implementa por separado (ya que todos pueden ser accesibles vía consola porque tienen método main y paso de parámetros por línea de mandato) o podemos cargar alguno de los interfaces que ofrece. Para poder cargar la interfaz de selección de interfaces (ver figura 2) ordenaremos el mandato siguiente en el directorio inmediatamente superior al de Weka:

```
java weka.gui.GUIChooser
```

La estructura de directorios de Weka está organizada en paquetes de forma que añadir, eliminar o modificar elementos es una tarea sencilla. La estructura física de directorios es la siguiente:

- **associations** Carpeta que contiene las clases que implementan los métodos para realizar asociaciones.
  - **classification** Contiene clases
  - **tertius** Contiene las clases para implementar el método tertius.

[*] Ant es una herramienta para la automatización de la compilación de proyectos Java. Está considerado el *make* para Java. `http://ant.apache.org/ant`

- **attributeSelection**
- **classifiers** Contiene la implementación de los clasificadores existentes en Weka.
  - **bayes** Clases para implementar clasificadores *bayes* como puede ser *naivebayes*.
    - **net** Clases en desarrollo para aplicar el clasificador bayes en búsquedas web.
  - **evaluation** Clases que evevalúanos resultados de un clasificador, como dibujar las matrices de confusión, las curvas de coste, etc.
  - **functions** Contiene métodos de clasificación basados en funciones.
    - **neural** Métodos de clasificación basados en redes neuronales.
    - **pace** Métodos numéricos sobre datos (manipulación de matrices, funciones estadísticas)
    - **supportVector** Métodos de clasificación basados en máquinas de vectores soporte.
  - **lazy** Implementa clasificadores *IB1, IBk, Kstar, LBR, LWL*.
    - **kstar** Métodos necesarios para implementar *Kstar*.
  - **meta** Metaclasificadores.
  - **misc** Clasificadores que se encuentren identificados en otras categorías.
  - **rules** Contiene clases para implementar reglas de decisión.
    - **car** Carpeta que tiene métodos para realizar reglas de asociación sobre clases.
      - **utils**
    - **part** Clases para implementar reglas sobre estructuras parciales de árboles.
  - **trees** Contiene métodos para implementar árboles.
    - **adtree** Implementa árboles *adtree*
    - **j48** Implementa árboles *c45* y otros métodos útiles para tratar con éstos.
    - **lmt** Implementa árboles *lmt*
    - **m5** Implementa árboles *m5*
- **clusterers** Contiene las clases donde se implementa el *clustering*.
- **core** Paquete principal donde se encuentran las clases principales como *Instance, Attribute, Option...*, y las clases que definen "habilidades" (Drawable, Copyable, Matchable, ...).
  - **converters** Clases para convertir entre tipos de archivos y para cargar archivos de forma secuencial.
- **datagenerators** Clases abstractas para generadores de datos y clases para representar tests.
- **estimators** Estimadores de probabilidades condicionadas y de probabilidades simples de diferentes distribuciones (Poisson, Normal ...).
- **experiment** Clases que implementan la entidad experimento. Así mismo también están las que modelan los experimentos remotos.
- **filters** Contiene filtros para aplicar a los datos para preprocesarlos.
  - **unsupervised** Filtros no supervisados.

- **attribute** Filtros para aplicar a atributos.
- **instance** Filtros para aplicar a instancias.

- **supervised** Filtros supervisados. En la última versión oficial (3-4-2) de Weka esta opción no está activada, se prevee que en sucesivas versiones lo esté.
  - **attribute** Filtros para aplicar a atributos.
  - **instance** Filtros para aplicar a instancias.

- **gui** Contiene las clases que implementan las interfaces gráficas.
  - **beans** Clases básicas de Weka de tratamiento de datos y creación de entidades principales del programa referentes a instancias (DataSource, TestsetListener, TetsetProducer, TrainingSet...).
    - **icons** Iconos y demás elementos visuales para aplicar a los beans y al knowledge flow.
  - **boundaryvisualizer** Métodos para crear instancias a partir de otras.
  - **experiment** Interfaz del modo experimentador.
  - **explorer** Interfaz del modo explorador.
  - **graphvisualizer** Clases para dibujar y mostrar gráficas.
    - **icons** Iconos y demás elementos visuales para aplicar al visor de gráficas.
  - **streams** Clases para realizar operaciones sobre flujos de datos, que habitualmente será sobre un objeto de clase *Instance*, que es la clase que almacena las instancias.
  - **treevisualizer** Clases para elaborar árboles y mostrarlos de una forma gráfica.
  - **visualizer** Contiene clases que implementan los métodos para poder seleccionar atributos en las gráficas y las clases principales para dibujar gráficas (Plot2d, VisualizePanel etc.).

- *build.xml* Archivo de reglas de compilación. Define qué archivos es necesario compilar y cómo hacerlo.
- *TODO.xml* Archivo de definición de tareas pendientes.
- *TODO.dtd* Archivo de definición de tipos. Define las construcciones permitidas para hacer listas de tareas pendientes.

8.1. Creando un nuevo clasificador

Programar nuevos clasificadores para Weka es sencillo teniendo conocimientos de Java, ya que Weka está orientado al desarrollo abierto y comunitario y eso se nota; no obstante, su estructura, quizá algo enrevesada, en parte adolece un mal diseño. Para crear un nuevo clasificador han de seguirse las siguientes pautas:

Lo primero es crear una nueva clase y situarla dentro de `weka/classifiers/`. Una vez creada la clase, cualquier clasificador debe heredar de las clases:

- `import weka.classifiers.*`
- `import weka.core.*`

- `import weka.util.*`

Y debe extender la clase `Classifier`. Si el clasificador calcula la distribución de clases debe extender también de `DistributionClassifier`. Una vez completado esto, todo clasificador tiene que tener los siguientes métodos:

- `void buildClassifier(Instances instancias)` Este método debe implementar la construcción del clasificador obteniendo como parámetro las instancias. Este método debe inicializar todas las variables y nunca debe modificar ningún valor de las instancias.
- `float classifyInstance(Instance instancia)` Clasifica una instancia concreta una vez que el clasificador ya está construido. Devuelve la clase en la que se ha clasificado o `Instance.missingValue()` si no se consigue clasificar.
- `double[] distributionForInstance(Instance instancia)` Este método devuelve la distribución de una instancia en forma de vector. Si el clasificador no ha logrado clasificar la instancia dada devolverá un vector lleno de ceros. Si lo consigue y las clases son numéricas devolverá el valor obtenido y en los demás casos devolverá un vector con las probabilidades de cada elemento en cada clase.

Y, en según qué circunstancias, deberá implementar las siguientes interfaces:

- `UpdateableClassifier` Si el clasificador tiene un carácter incremental.
- `WeightedInstanceHandler` Si el clasificador hace uso de los pesos de cada instancia.

Es muy útil conocer las clases `Instance` e `Instances` que son aquellas que almacenan (siempre con `doubles`) todas las instancias, incluso las nominales que utilizan como índice la declaración del atributo y las indexa según este orden. Los métodos más útiles de la clase `Instance` (se encuentra en el paquete weka.core) son:

- `Instances classAttribute()` Devuelve la clase de la que procede el atributo.
- `double classValue()` Devuelve el valor de la clase del atributo.
- `double value(int i)` Devuelve el valor del atributo que ocupa la posición i.
- `Enumeration enumerateAttributes()` Devuelve una enumeración de los atributos que contiene esa instancia.
- `double weight()` Devuelve el peso de una instancia en concreto.

Y de la clase `Instances`:

- `int numInstances()` Devuelve el número de instancias que contiene el conjunto de datos.
- `Instance instance(int)` Devuelve la instancia que ocupa la posición i.
- `Enumeration enumerateInstances()` Devuelve una enumeración de las instancias que posee el conjunto de datos.

Un ejemplo de un clasificador muy sencillo podría ser el siguiente:

EjemploClasificador.java

```
/*
* Ejemplo de clasificador
* -----------------------
* Calcula del conjunto de entrenamiento la media
* de los valores de todos sus datos, y compara con la media
* de la instancia a clasificar. Si es mayor clasifica con la
* clase cuyo índice valor sea 0.0 si no clasifica con la clase
* cuyo índice valor es 1.0.
*/
import weka.classifiers.*
import weka.core.*
import weka.util.*

public class EjemploClasificador extends Classifier {

  private double media_entrenamiento = 0; //Media del conjunto de entrenamiento

  public void buildClassifier(Instances instancias) throw Exception {

    // Recorremos todas las instancias
    for (int i=0; i<instancias.numClasses();m_numclase.length; i++) {
      // Recorremos todos sus atributos
      for (int j=0; j < instancias[i].numAttributes(); j++) {
        media_entrenamiento += instancias[i].value[j]
          / instancias[i].numAttributes();
      }
    }
  } // buildClassifier

  public double classifyInstance(Instance instance) {
    float media_instancia = 0;

  // Recorremos todos sus atributos
  for (int j=0; j < instancias[i].numAttributes(); j++) {
    media_instancia += instancias[i].value[j] / instancias[i].numAttributes();
  }

  if (media_instancia > media_entrenamiento) {
    return 0.0;
  } else if {
    return 1.0;
  }
  } // classifyInstance
} // Fin clase
```

Referencias

[1] Ian H. Witten and Eibe Frank. *Data Mining: Practical machine learning tools with Java implementations*. Morgan Kaufmann, San Francisco, 2000.

[2] Ian H. Witten, Eibe Frank, Len Trigg, Mark Hall Geoffrey Holmes, and Sally Jo Cunningham. Weka: Practical machine learning tools and techniques with java implementations. *Department of Computer Science. University of Waikato. New Zealand*. `http://www.cs.waikato.ac.nz/~ml/publications/1999/99IHW-EF-LT-MH-GH-SJC%-Tools-Java.pdf`.