



Prácticas de Minería de Datos

Grado en Ingeniería Informática

Curso 2013-14

PRÁCTICA 3

Técnicas Básicas de Preprocesamiento con WEKA

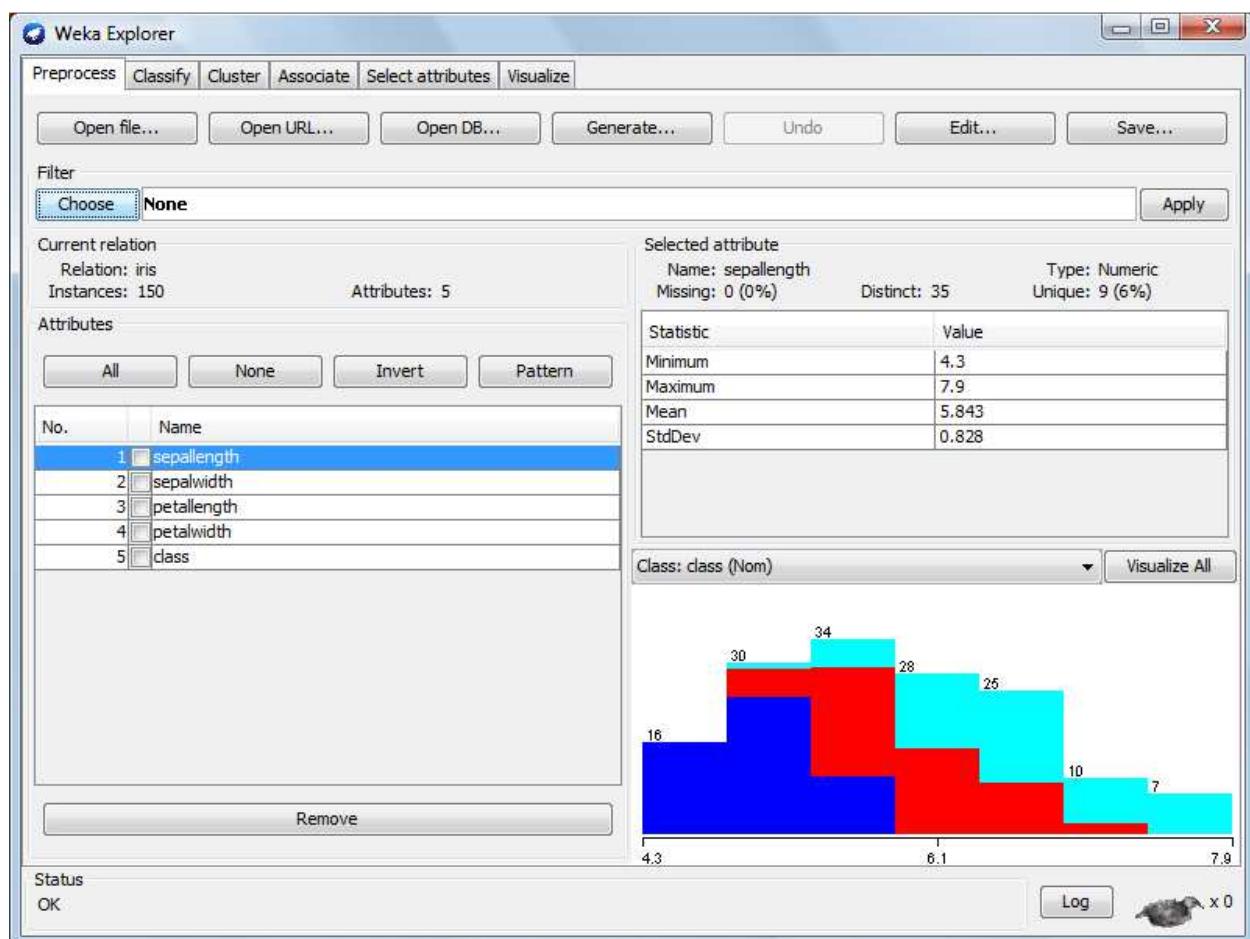


OBJETIVOS

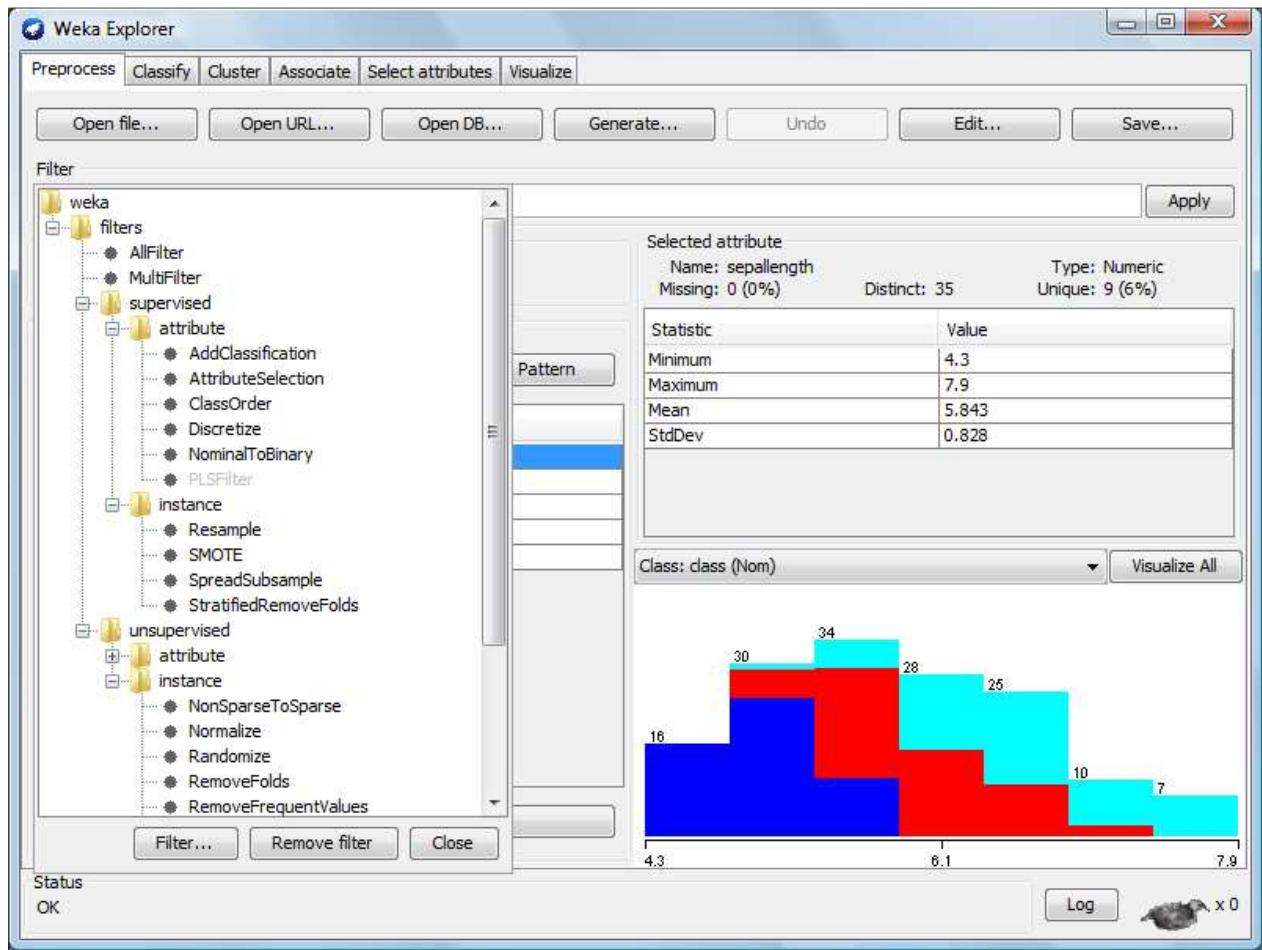
- Aprender a manejar WEKA como herramienta de Preprocesamiento básico para realizar transformaciones en los datos
- Realizar Selección de Características y Selección de Instancias

1. Preprocesamiento en Weka

WEKA incluye distintos métodos para realizar preprocesamiento, los cuales se encuentran, dentro de la pestaña de Preprocesamiento, en el apartado de Filters. La selección de características o atributos se realiza de forma independiente en la pestaña Select attributes, y la de instancias, aunque no se encuentra como tal, puede llevarse a cabo en la pestaña de Preprocesamiento bien de forma manual (editando los datos) o de forma automatizada aplicando filtros.



Weka permite aplicar una gran diversidad de filtros sobre los datos, permitiendo realizar transformaciones sobre ellos de todo tipo a dos niveles: atributos e instancias. Las operaciones de filtrado pueden aplicarse en cascada, de manera que cada filtro toma como entrada el conjunto de datos resultante de haber aplicado un filtro anterior. Al pulsar el botón Choose dentro del recuadro Filter se nos despliega un árbol en el que se pueden encontrar utilidades, transformaciones y procedimientos para el tratamiento de valores perdidos.



Estos son algunos ejemplos:

Utilidades:

- Aplicadas sobre atributos:

- *Add*: Añade un nuevo atributo.
- *AddCluster*: Añade un atributo nominal para representar clusters.
- *AddExpresion*: Añade un atributo que es una expresión en base a otro/s atributos (muy útil para crear nuevos atributos combinación de otros)
- *AddNoise*: Cambia un porcentaje de valores de un atributo.
- *Copy*: Permite duplicar un conjunto de atributos (útil para aplicar los cambios en los atributos copias, conservando los valores de los originales)
- *Remove*: Borra atributos.
- *RemoveType*: Borra atributos de un tipo (bien sea nominal, real, etc.).
- *SwapValues*: Intercambia los valores de dos atributos nominales.

- **Aplicadas sobre instancias:**

- *Randomize*: Desordena el orden de las instancias.
- *StratifiedRemoeFolds*: Devuelve 1 fold de un dataset.
- *RemovePercentage*: Borra un porcentaje del dataset.
- *RemoveRange*: Borra un rango de instancias.
- *RemoveWithValues*: Borra instancias con ciertos valores.
- *Resample*: Obtiene un subconjunto del conjunto inicial de forma aleatoria.

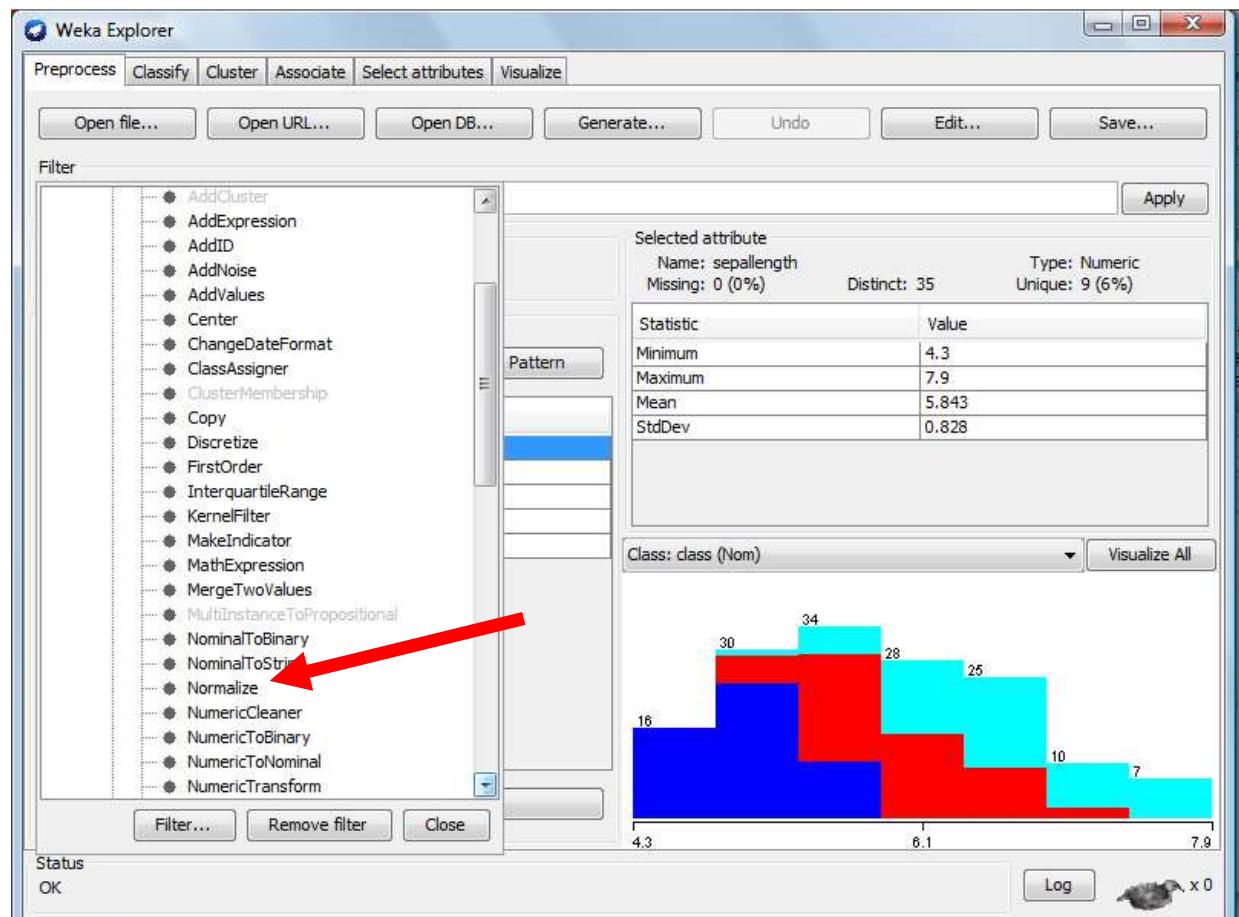
- **Transformaciones:**

- *Normalize*: Escala atributos numéricos al intervalo [0,1].
- *NominalToBinary*: Convierte valores nominales a binarios.
- *RandomProjection*: Proyecta los datos en dimensión n a datos en dimensión m, siendo m<n.
- *Standarize*: Estandariza valores numéricos a media 0 y desviación típica 1.

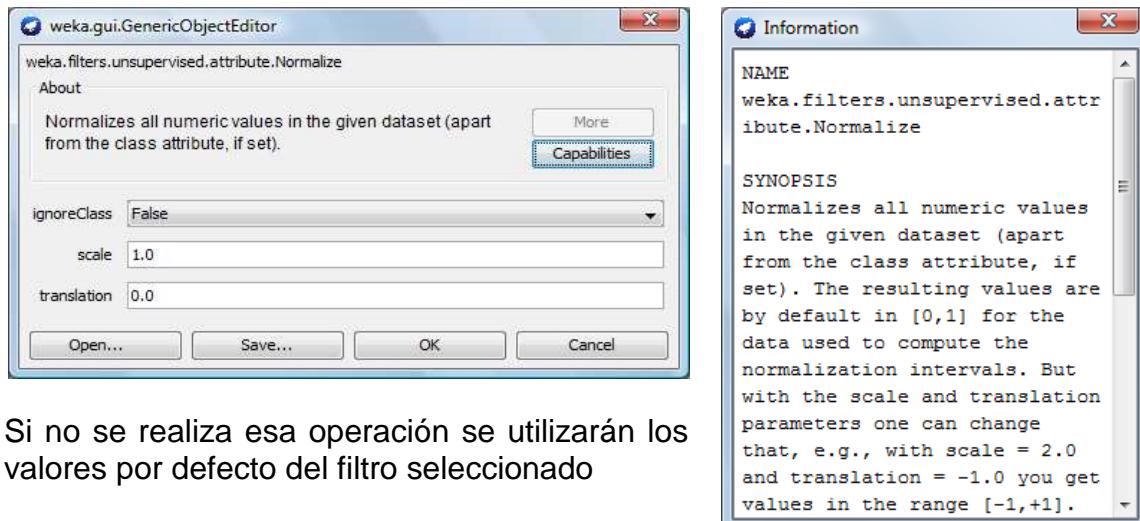
- **Tratamiento de valores perdidos:**

- *ReplaceMissingValues*: Sustituye todos los valores perdidos para atributos nominales y numéricos con las modas y las medias de los datos de entrenamiento.

Para realizar una transformación, por ejemplo escalar atributos a un intervalo determinado (Normalize), escogeremos en primer lugar Filters, seguido de Unsupervised, a continuación Atribute, y finalmente Normalize:



Algunos filtros pueden tener parámetros que requieren ser ajustados. Para ello, una vez seleccionado el filtro se puede configurar sus parámetros haciendo clic sobre su nombre, momento en el que aparece la ventana de configuración correspondiente a ese filtro.



Si no se realiza esa operación se utilizarán los valores por defecto del filtro seleccionado

Tras aplicar el filtro, podemos ver los efectos en los datos a través de las opciones de edición y visualización.

Relation: iris-weka.filters.unsupervised.attribute.Normalize-S1.0-T0.0					
No.	sepallength Numeric	sepalwidth Numeric	petallength Numeric	petalwidth Numeric	class Nominal
1	0.222222222...	0.62499999...	0.0677966...	0.04166666...	Iris-setosa
2	0.166666666...	0.41666666...	0.0677966...	0.04166666...	Iris-setosa
3	0.111111111...	0.5	0.0508474...	0.04166666...	Iris-setosa
4	0.083333333...	0.45833333...	0.0847457...	0.04166666...	Iris-setosa
5	0.194444444...	0.66666666...	0.0677966...	0.04166666...	Iris-setosa
6	0.305555555...	0.79166666...	0.1186440...	0.12500000...	Iris-setosa
7	0.083333333...	0.58333333...	0.0677966...	0.08333333...	Iris-setosa
8	0.194444444...	0.58333333...	0.0847457...	0.04166666...	Iris-setosa
9	0.027777777...	0.37499999...	0.0677966...	0.04166666...	Iris-setosa
10	0.166666666...	0.45833333...	0.0847457...	0.0	Iris-setosa
11	0.305555555...	0.70833333...	0.0847457...	0.04166666...	Iris-setosa
12	0.138888888...	0.58333333...	0.1016949...	0.04166666...	Iris-setosa
13	0.138888888...	0.41666666...	0.0677966...	0.0	Iris-setosa
14	0.0	0.41666666...	0.0169491...	0.0	Iris-setosa
15	0.416666666...	0.83333333...	0.0338983...	0.04166666...	Iris-setosa
16	0.388888888...	1.0	0.0847457...	0.12500000...	Iris-setosa
17	0.305555555...	0.79166666...	0.0508474...	0.12500000...	Iris-setosa
18	0.222222222...	0.62499999...	0.0677966...	0.08333333...	Iris-setosa
19	0.388888888...	0.74999999...	0.1186440...	0.08333333...	Iris-setosa
20	0.222222222...	0.74999999...	0.0847457...	0.08333333...	Iris-setosa
21	0.305555555...	0.58333333...	0.1186440...	0.04166666...	Iris-setosa
22	0.222222222...	0.70833333...	0.0847457...	0.12500000...	Iris-setosa
23	0.083333333...	0.66666666...	0.0	0.04166666...	Iris-setosa
24	0.222222222...	0.54166666...	0.1186440...	0.16666666...	Iris-setosa

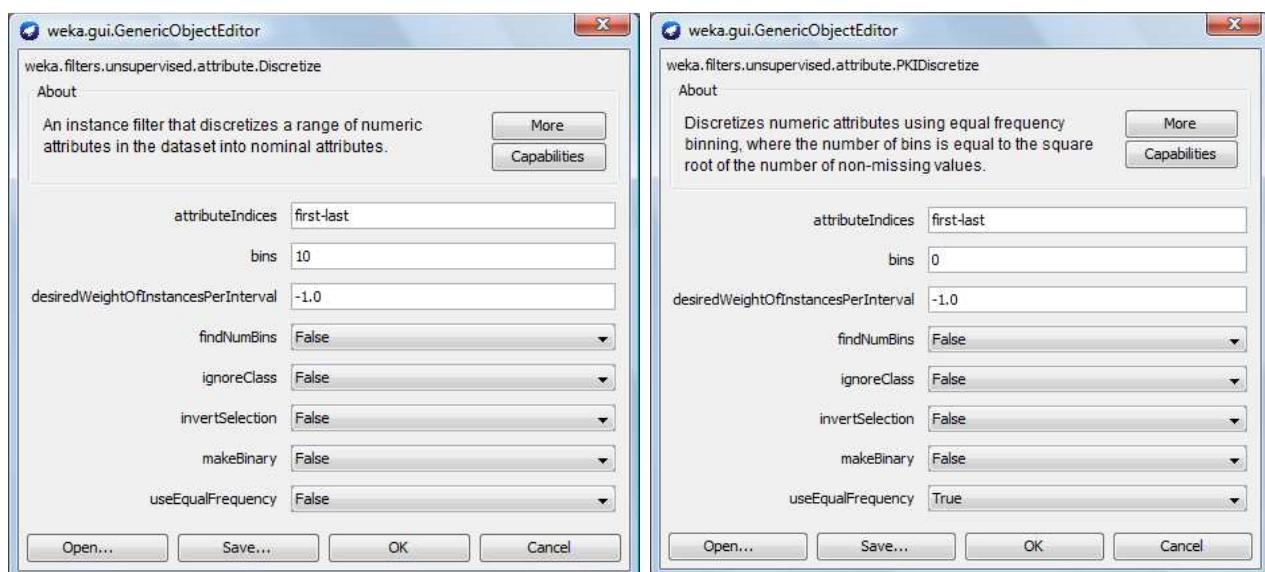
Undo OK Cancel

2. Reducción de Datos

A continuación se va a describir la discretización, la selección de características y la selección de instancias

2.1 Discretización

La discretización se lleva a cabo a través del filtro Discretize o del filtro PKIDiscretize y consiste en convertir atributos numéricos a nominales, preseleccionando los atributos y permitiendo preajustar una serie de parámetros como el número de intervalos, si utilizar intervalos de igual anchura o frecuencia, etc.



El método PKIDiscretize discretiza con intervalos de igual frecuencia, siendo el número de intervalos igual a la raíz cuadrada del número de valores.

2.2 Selección de características o atributos (feature selection)

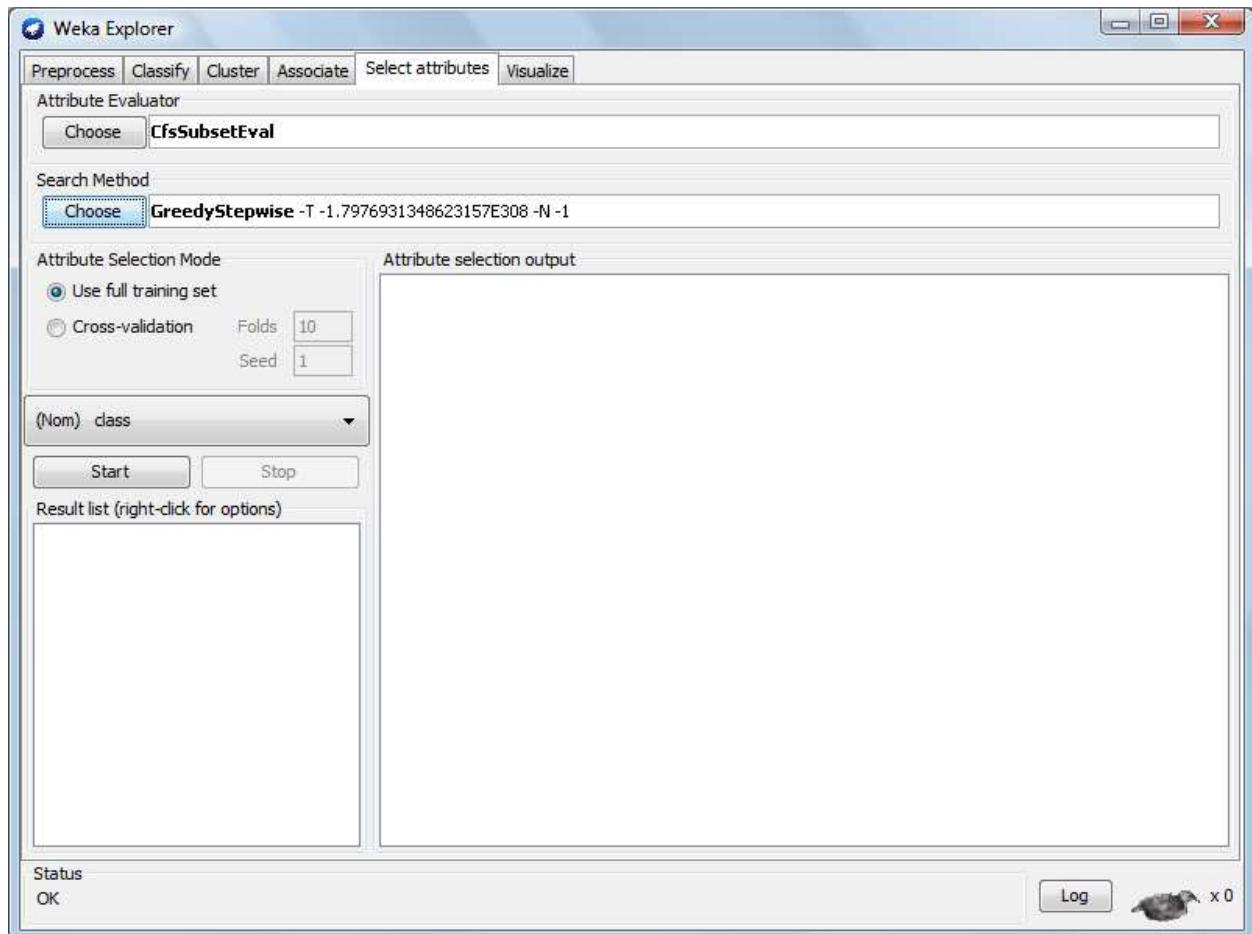
La **selección de características** (eliminación de atributos redundantes e irrelevantes) permite explorar qué subconjuntos de atributos son los que mejor pueden clasificar una clase. Hay que tener en cuenta que si hay un número excesivo de atributos, esto puede hacer que el modelo sea demasiado complejo y se produzca sobreaprendizaje o sobreajuste (overfitting).

El objetivo de la **selección de características** es identificar aquellos atributos que tienen más peso a la hora de determinar si los datos son de una clase o de otra, con idea de encontrar el conjunto mínimo de atributos que puedan separar las clases de la misma forma que lo hace el conjunto completo de atributos. Al reducir el número de atributos se consigue una serie de ventajas: reducir el tamaño de los datos, aumentar la eficiencia y la interpretabilidad y mejorar la calidad del modelo obtenido.

Este proceso se lleva a cabo mediante una búsqueda en el espacio de subconjuntos de características, y evaluando cada uno de ellos, es decir, en los algoritmos de selección

de características hay que definir por un lado la estrategia de búsqueda a utilizar para seleccionar el subconjunto de atributos y la función que evalúa dicho subconjunto. WEKA implementa métodos de búsqueda, y evaluadores, de modo que podemos indicar que métodos de búsqueda utilizar y qué tipo de evaluador de subconjuntos vamos a utilizar para determinar la calidad de dicho subconjunto.

Aunque en Weka, la selección de atributos se puede hacer de varias maneras, la más directa es usando la pestaña de attribute selection, donde seleccionamos el método de búsqueda y el evaluador a utilizar.



En general los métodos de selección de atributos pueden dividirse en dos grandes grupos:

- aquellos que clasifican y evalúan los atributos individualmente
- aquellos que buscan un subconjunto de atributos considerando el efecto combinado de todos ellos. Éstos a su vez se dividen en filtros y wrappers.

Los primeros necesitan usar un evaluador de atributos y un método de búsqueda especial denominado **Ranker**, mientras que los segundos necesitan usar un evaluador de subconjuntos de atributos (hay múltiples métodos de búsqueda para ellos).

2.2.1 Evaluación de conjuntos 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 (subconjuntos de atributos que va probando) a los que se le enfrente y dotar a cada atributo de un peso específico. Se distinguen dos tipos:

- **Métodos de filtro o previos:** Evalúan los subconjuntos basándose en la información que contienen, es decir, el criterio para evaluar los subconjuntos se basa en medidas de calidad previa que se calcula a partir de los datos mismos (distancia, dependencias estadísticas, consistencia, etc.)
- **Métodos de envolvente (wrapper):** Evalúan los subconjuntos mediante una técnica de aprendizaje, es decir, utilizando un clasificador específico que aprende cuál es el subconjunto óptimo, midiendo la calidad del subconjunto a través de la tasa de acierto (o de error) del clasificador.

Los **wrapper** (aunque más exactos) son más lentos que los filtros ya que para cada evaluación de los subconjuntos candidatos hay que aprender un modelo, por ello no son recomendables cuando hay un gran número de atributos o gran cantidad de datos.

Una vez seleccionado el método de evaluación de atributos, el siguiente paso será elegir el método de búsqueda que será el encargado de generar el espacio de pruebas.

Los evaluadores de subconjuntos son por ejemplo:

- *CfsSubsetEval*, que considera el valor o habilidad predictiva individual de cada atributo, junto con el grado de redundancia entre ellos.
- *ClassifierSubsetEval*, el cual utiliza un clasificador para realizar la evaluación.
- *ConsistencySubsetEval*, que mide la consistencia (mismos valores de atributos, pero distinta etiqueta de clase) de las subclases.
- *WrapperSubsetEval*, que utiliza dos elementos, un clasificador y validación cruzada.

Métodos de búsqueda son por ejemplo:

- *BestFirst*, que emplea un algoritmo voraz incremental con backtracking.
- *EshauostiveSearch*, búsqueda exhaustiva empleando fuerza bruta.
- *GeneticSearch*, mediante un algoritmo genético.
- *GreedyStepWise*, voraz incremental sin backtracking.
- *RaceSearch*
- *RandomSearch*, que realiza una búsqueda aleatoria
- *RankSearch*, ordena los atributos y crea un ranking de subconjuntos prometedores, en base al evaluador de subconjuntos de atributos utilizado.

2.2.2 Evaluación de atributos individuales

El método más rápido, pero de menor precisión, consiste sin embargo simplemente en evaluar los atributos individualmente y ordenarlos, de modo que se descartan los atributos que no alcanzan determinado nivel. Esto se lleva a cabo utilizando uno de los evaluadores de atributo simples, y posteriormente el método **Ranker** en la búsqueda.

Los evaluadores de atributos simples implementados son por ejemplo:

- *ChiSquaredAttributeEval*: Calcula la estadística chi-cuadrado de cada atributo con respecto a la clase
- *GainRatioAttributeEval*: Evaluación por tasa de ganancia
- *InfoGainAttributeEval*: Evaluación por ganancia de información
- *OneRAttributeEval*: Metodología OneR
- *PrincipalComponents*: Análisis de componentes principales y transformación
- *ReliefFAttributeEval*: Evaluador basado en instancias
- *SVMAttributeEval*: Usa máquinas de soporte vectorial para calcular el valor de los atributos
- *SymmetricalUncertAttributeEval*: Evalúa atributos basándose en incertidumbre simétrica

En resumen, a grandes rasgos existen 3 posibilidades:

Evaluación de atributos. Por ejemplo:

Método de búsqueda = *Ranker*

Método de evaluación = *InfoGainAttributeEval*

Evaluación de conjuntos de atributos

Filter. Por ejemplo:

Método de búsqueda = *Greedy Stepwise*

Método de evaluación = *CfsSubsetEval*

Método Wrapper. Por ejemplo:

Método de búsqueda = *Greedy Stepwise*

Método de evaluación = *ClassifierSubsetEval*

2.3 Selección de instancias

Los métodos clásicos de **selección de instancias** no están incluidos en Weka.

Solamente tiene incluido el Muestreo Aleatorio, que puede ser simple (no supervisado) o estratificado o balanceado (supervisado)

- Filter -> Unsupervised -> Instance -> Resample (no mantiene la proporción de clase)
- Filter -> Supervised -> Instance -> Resample (mantiene la proporción de clases)

En ambos es posible indicar el porcentaje final de ejemplos que se desea obtener en la muestra.

3. Ejemplo comentado

3.1: Enunciado del Problema: Selección de Fármaco

Queremos predecir el tipo de fármaco (drug) que se debe administrar a un paciente afectado de rinitis alérgica según distintos parámetros/variables. Las variables que se recogen en los historiales clínicos de cada paciente son:

- Age: Edad
- Sex: Sexo
- BP (Blood Pressure): Tensión sanguínea.
- Cholesterol: nivel de colesterol.
- Na: Nivel de sodio en la sangre.
- K: Nivel de potasio en la sangre.

Hay cinco fármacos posibles: DrugA, DrugB, DrugC, DrugX, DrugY. Se han recogido los datos del medicamento idóneo para muchos pacientes en cuatro hospitales. Se pretende, para nuevos pacientes, determinar el mejor medicamento a probar.

3.2: Resolución del Problema

En primer lugar vamos a cargar los datos del primer hospital fichero “drug1n”, ya que al ser el de menor tamaño (200 registros), permite hacer más pruebas inicialmente.

La primera pregunta que nos podemos hacer es ver qué fármacos son más comunes en general, para ver si todos suelen ser igualmente efectivos en términos generales. Para ello seleccionamos el atributo drug, y viendo la distribución por clases podemos concluir que el fármaco más efectivo es el Y, que se administra con éxito en casi la mitad de los pacientes. Una regla vulgar sería aplicar el fármaco Y, en el caso que falle, el fármaco X, y así sucesivamente siguiendo las frecuencias de uso con éxito.

Weka tiene un método que permite generar este modelo tan simple, es decir asignar a todos los ejemplos la clase mayoritaria, recibe el nombre de ZeroR4 en la familia rules. Si vamos a la parte de Classify y ejecutamos este modelo **evaluándolo sobre todos los datos de entrenamiento**, veremos que como era de esperar obtenemos un clasificador de precisión 45.5% y un error de más del 50% (el 54,5% de las veces el medicamento DRUGY no es el adecuado).

Podemos utilizar métodos más complejos como el J48. Si probáis esta técnica veréis que se obtiene una precisión del 97% con un error de sólo el 3% sobre los datos.

Ejercicio 1

1. Examina detenidamente los atributos del problema e intenta mejorar el modelo combinando alguno de los atributos. Utiliza el entorno Visualize para analizar pares de atributos e intenta descubrir si hay alguna relación interesante entre alguno de ellos. Si es así crea un atributo derivado que combine ambos. Usa para ello:

`weka.filters.unsupervised.attribute.AddExpression`

2. Una vez creado el nuevo atributo, vuelve a aplicar de nuevo el método J48 **utilizando otra vez todos los datos para la evaluación** (no olvide indicar que la clase es el atributo drugs, ya que por defecto Weka toma como clase el último atributo, que ahora es el que se acaba de crear).

Responder a las siguientes cuestiones:

- ¿Cuál es la precisión alcanzada ahora?
- ¿Cuál es el tamaño del árbol generado?
- ¿Cuál era el tamaño del árbol generado antes de crear el nuevo atributo?

3. En el problema puede que haya atributos que no son significativos y que pueden afectar al rendimiento de ciertos métodos de aprendizaje. Un ejemplo de método de aprendizaje que reduce su calidad ante la presencia de atributos no relevantes, es el método Naive Bayes. Aplica dicho método con evaluación cruzada (10 pliegues).

- ¿Cuál es la precisión alcanzada por dicho modelo?

4. Realiza una selección de características para eliminar los atributos no relevantes para el problema. Utiliza diferentes técnicas para ello (prueba las indicadas a continuación) y contrasta los resultados obtenidos:

- Evaluación individual de atributos
 - Método de búsqueda = *Ranker*
 - Método de evaluación = *InfoGainAttributeEval*
- Evaluación de conjunto de atributos (filter)
 - Método de búsqueda = *Greedy Stepwise*
 - Método de evaluación = *CfsSubsetEval*
- Evaluación de conjunto de atributos (Wrapper)
 - Método de búsqueda = *ExhaustiveSearch* (búsqueda exhaustiva)
 - Método de evaluación = *WrapperSubsetEval* (utiliza el propio NaiveBayes como clasificador)
- ¿Cuáles son los atributos que recomienda que usemos cada uno de los métodos?

5. Volviendo a la pestaña Preprocess, elimina los atributos descartados por cada de los métodos anteriores (en el caso de Ranker elimina los 3 peores) y vuelve a aplicar el método NaiveBayes con validación cruzada.

- ¿Cuál es la precisión alcanzada ahora con cada uno de los modelos?
- ¿Cuál es el modelo que obtiene los mejores resultados?

Ejercicio 2

6. El modelo inicial obtenido para los fármacos para el J48 tenía casi un 97% de precisión con respecto de los datos de entrenamiento. Esto, en principio, parece óptimo, pero no es así.

Obtener un 97% sobre los datos de entrenamiento es relativamente sencillo y no asegura que el modelo se vaya a comportar bien. Para saber si el modelo se va a comportar bien, debemos comprobarlo con datos “frescos”, es decir, con datos que no hayan intervenido en el aprendizaje del modelo.

Como en la carpeta drugs tenemos datos de cuatro hospitales y hemos utilizado el primer hospital para obtener el modelo, podemos comprobar el modelo con el resto de hospitales para ver si funciona bien.

Prueba el modelo inicial obtenido con J48 para “drug1n” y utiliza como test (“*Supplied test set*”) los datos de los 3 hospitales restantes (deberás primero convertir dichos ficheros a formato .arff).

- ¿Cuál es la precisión alcanzada para cada uno de los hospitales restantes?
7. Aunque el error es pequeño, se podría pensar que sólo se han utilizado 200 datos para generar el modelo y 2.000 para validar. Además hemos usado los datos de un hospital para entrenar “drug1n” y el resto para validar, pudiendo haber diferencias entre los hospitales. Sería mejor unir todos los datos y hacer una partición más razonable.
- Fusiona los datos de los 4 hospitales en un único fichero “drugn.arff”, prueba de nuevo el modelo evaluándolo **sobre todos los datos de entrenamiento** e indica cuál es la precisión alcanzada por el modelo.
 - Crea un conjunto de entrenamiento (80%) y otro de test (20%), uno balanceado (“drugn80b.arff” y “drugn20b.arff”) y otro aleatorio (“drugn80a.arff” y “drugn20a.arff”) utilizando para ello respectivamente los filtros:

```
weka.filters.supervised.instance.StratifiedRemoveFolds  
weka.filters.unsupervised.instance.RemoveFolds
```
 - Prueba el modelo anterior usando como conjunto de entrenamiento y de test los datos anteriores (uno balanceado y otro aleatorio) e indica la precisión obtenida con cada uno de ellos.
 - ¿Influye en este caso mucho si las particiones están balanceadas o no?

¿Cómo entregar la práctica?

- Utilizar un documento de texto para responder a las cuestiones y subirlo a través de la plataforma web. Podéis utilizar pantallas gráficas (histogramas, gráficas 2-D, etc.) para complementar las respuestas.
- Para el ejercicio 6 y 7, suba los ficheros “arff” que habéis creado.