Herramientas de Minería de datos: WEKA (Waikato Environment for Knowledge Analysis)

Juan A. Botía Blaya
juanbot@um.es

November 27, 2007

1 Introducción

En esta práctica, vamos a ilustrar el uso de Weka[1] para el análisis de datos. Si en la primera práctica habíamos estudiado cómo obtener una idea superficial sobre cómo están estructurados los datos a analizar, en esta segunda práctica vamos a ver cómo usar Weka para obtener conjuntos de datos.

Vamos, para ello, a utilizar también el conjunto `iris`, que recordemos, es un conjunto con cuatro atributos en $\bar{x}$ y un atributo de clase y con tres valores diferentes, correspondientes a tres tipos de Iris que se dan en el estudio.

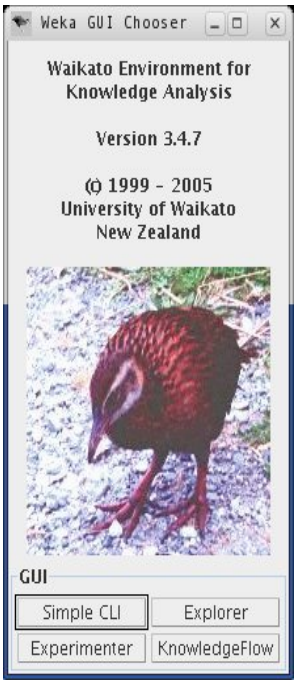
2 La herramienta Weka

La forma más sencilla de ejecutar Weka es haciendo

```
weka-3-4-7> java -jar weka.jar
```

con lo que obtenemos la siguiente ventana

[1] http://www.cs.waikato.ac.nz/ml/weka/



que es un primer paso en donde podemos elegir entre una consola de línea de comandos con `Simple CLI`, acceder a la aplicación principal de Weka que es la que nosotros vamos a utilizar en esta práctica con `Explorer`, comparar la eficiencia de distinos algoritmos con `Experimenter` o diseñar experimentos de minería de datos de manera visual con componentes en `KnowledgeFlow`. La ventana principal del explorer podemos verla en la figura 1. Vemos que está organizada en pestañas y la activada es la correspondiente a preprocesado de los datos.

3 El Explorer

Ya tenemos cargado en el Explorer el conjunto Iris. Vemos que en la parte izquierda aparece la lista de atributos y a la derecha una sencilla descripción de un atributo activado, en este caso es la clase. También podemos ver la distribución de cada uno de los tres valores de clase, en laparte inferior derecha (50 valores para cada uno, un 33%).

3.1 Preproceso

Esta primera pestaña de Weka nos va a servir para visualizar sencillas características sobre los datos, para seleccionar qué atributos vamos a tener en cuenta para el análisis y realizar transformaciones a los mismos, que tenemos a nuestra disposición en el panel etiquetado con `Filter`. Un buen filtro que podríamos aplicar aquí es el encargado de normalizar los datos de entrada ya que no son todos de la misma magnitud, como vimos con `R`. Para ello, solo tenemos que seleccionar `Choose`, y elegirlo de entre los de tipo no supervisado (no hace uso de la clase) y de tipo `Attribute`. Luego, pulsamos en `Apply`. Para comprobar que se ha hecho correctamente, podemos seleccionar cualquier atributo en la lista de la izquierda y los valores que puede tomar en forma de histograma nos aparecerán a la derecha del Explorer. Cada uno de los filtros están documentados en el `javadoc` correspondiente. De esta forma, si buscamos el paquete

`weka.filters.unsupervised.attribute,`

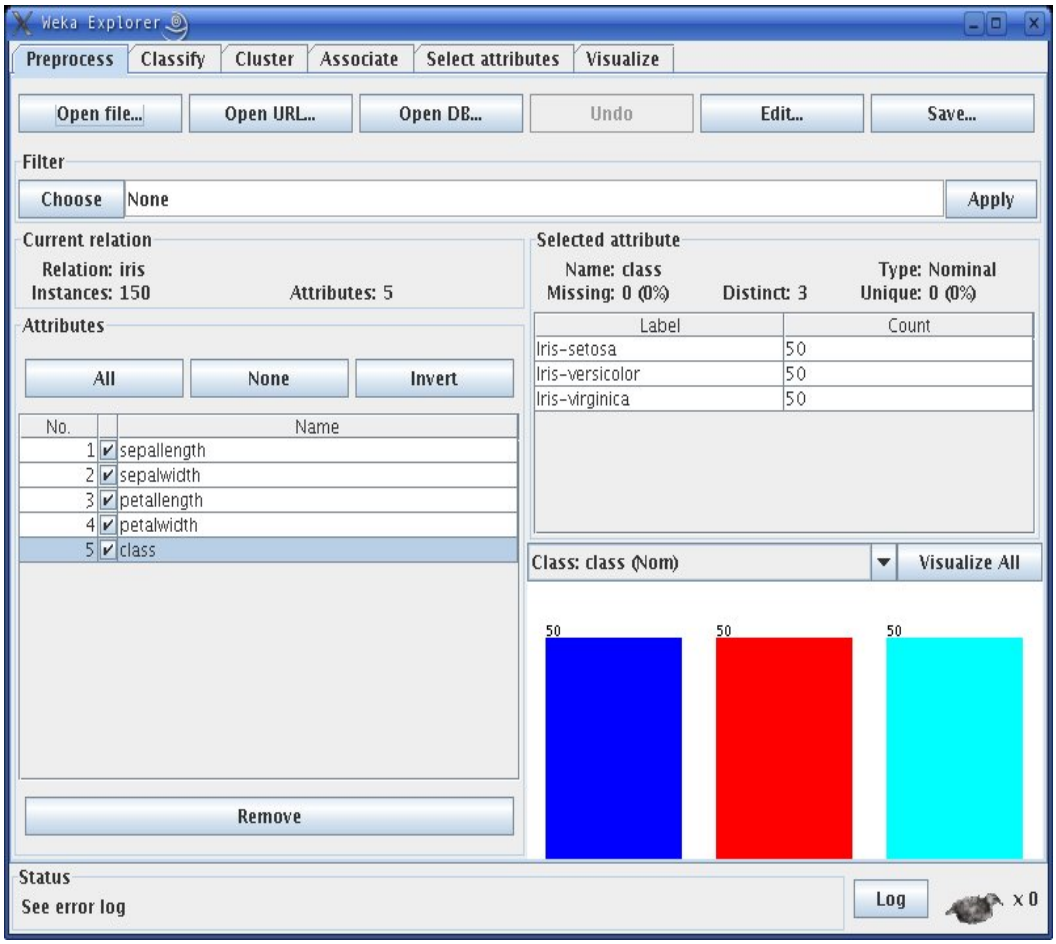


Figure 1: La ventana del explorer de Weka, con la pestaña de preprocesamiento activada y el conjunto Iris cargado

bajo el mismo encontraremos cada uno de los filtros, en forma de clase, y con una explicación de lo que hace cada uno. Podemos consultar el filtro `Normalize` y obtener la siguiente descripción

> `Normalizes all numeric values in the given dataset. The resulting values are in` $[0, 1]$ `for the data used to compute the normalization intervals.`

3.2 Generando modelos a partir de datos

3.2.1 Modelos de clasificación

Vamos a generar distintos modelos para el conjunto de datos Iris. Supongamos primeramente que en el problema necesitamos un clasificador para, dada una nueva observación $\bar{x} \in R^4$, determinar si las correspondientes medidas para pétalo y sépalo, poder clasificar nuevas observaciones de la planta sin la ayuda de un experto, solo con la ayuda de la máquina.

En el caso de que no necesitemos preguntarnos cómo se clasifican los nuevos $\bar{x}$, podemos usar una red neuronal. En este caso, nos iremos a la pestaña de `Classify` en la que veremos un árbol en la que aparece la lista de clasificadores que tenemos. Los vamos a tener basados en la regla de bayes, basados en

funciones (como redes neuronales), clasificadores *lazy*, árboles y reglas de decisión. Pinchamos en `functions` y escogemos el perceptrón multi-capa. Una vez hecho esto, en el panel de texto que está dentro del panel `Classifier` veremos que aparece la leyenda

MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H a

y si pinchamos ahí nos aparecerá una ventana que nos permite configurar el algoritmo, con parámetros como el ratio de aprendizaje, el momentum, el tiempo destinado para aprender, etc. Si asumimos que los valores por defecto son los correctos, podemos pasar a otra parte.

Una vez configurado el algoritmo, debemos configurar el proceso de testeo del mismo, en el palen `Test Options`. En esta hay cuatro opciones. Utilizar conjunto de entrenamiento (i.e. `Use training set`) implica usar todo el conjunto de datos para aprender y para obtener un estimador de la capacidad de generalización; es por tanto, una medida optimista del rendimiento del clasificador. Si usamos un conjunto de test distinto del que hemos cargado al principio (`Supplied test set`), que en su totalidad se usaría para el aprendizaje, debemos indicarlo. También tenemos la opción de validación cruzada que hemos visto en clase y, por último, la opción *hold-out* con `Percetage split`, de tal forma que un porcentaje especificado lo usamos para el entrenamiento y el otro para evaluación. En este caso, dado que tenemos un total de 150 instancias (i.e. un conjunto de datos bastante pequeño), vamos a usar validación cruzadad, de 10 pliegues.

Después de esto, seleccionamos el atributo de clase con la pestaña inferior al panel que acabamos de dejar y pulsamos `Start`, con lo que obtendremos unos resultados como los que aparecen en la consola de la derecha y que reproducimos seguidamente:

```
=== Run information ===

Scheme:       weka.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20
Relation:     iris-weka.filters.unsupervised.attribute.Normalize
Instances:    150
Attributes:   5
              sepallength
              sepalwidth
              petallength
              petalwidth
              class
Test mode:    10-fold cross-validation
```

En esta primera parte vemos la información que describe el experimento cuando se iba a ejecutar.

Seguidamente vemos el modelo de red neuronal resultante, en términos de los pesos $w_{i,j}$ de la red neuronal. Obsérvese que tiene cuatro nodos de entrada (uno por cada atributo) y tres nodos a la salida (uno por cada clase).

```
=== Classifier model (full training set) ===

Sigmoid Node 0
    Inputs    Weights
    Threshold    -3.501597158843403
    Node 3    -1.0058110853859925
    Node 4    9.07503844669134
    Node 5    -4.107780453339232
Sigmoid Node 1
    Inputs    Weights
```

```
    Threshold    1.0692845992273083
    Node 3    3.8988736877894143
    Node 4    -9.76891036034027
    Node 5    -8.599134493151334
Sigmoid Node 2
    Inputs    Weights
    Threshold    -1.0071762383436411
    Node 3    -4.2184061338270515
    Node 4    -3.6260596863211156
    Node 5    8.805122981737851
Sigmoid Node 3
    Inputs    Weights
    Threshold    3.382485556685686
    Attrib sepallength    0.9099827458022247
    Attrib sepalwidth    1.5675138827531327
    Attrib petallength    -5.037338107319896
    Attrib petalwidth    -4.915469682506112
Sigmoid Node 4
    Inputs    Weights
    Threshold    -3.3305735922918305
    Attrib sepallength    -1.1116750023770083
    Attrib sepalwidth    3.125009686667652
    Attrib petallength    -4.133137022912302
    Attrib petalwidth    -4.079589727871454
Sigmoid Node 5
    Inputs    Weights
    Threshold    -7.496091023618097
    Attrib sepallength    -1.2158878822058812
    Attrib sepalwidth    -3.5332821317534897
    Attrib petallength    8.401834252274115
    Attrib petalwidth    9.460215580472829
Class Iris-setosa
    Input
    Node 0
Class Iris-versicolor
    Input
    Node 1
Class Iris-virginica
    Input
    Node 2


Time taken to build model: 0.97 seconds
```

Seguidamente, se incluyen una medida de errores que pueden dar una idea de la efectividad del clasificador que se incluye arriba. Algunos son evidentes, sin embargo llama la atención el estadístico Kappa.

```
=== Stratified cross-validation ===
=== Summary ===
```

```
Correctly Classified Instances         146               97.3333 %
Incorrectly Classified Instances         4                2.6667 %
Kappa statistic                          0.96
Mean absolute error                      0.0327
Root mean squared error                  0.1291
Relative absolute error                  7.3555 %
Root relative squared error             27.3796 %
Total Number of Instances              150
```

que vamos a explicar con un ejemplo. Pero primero una definición del estadístico:

> El estadístico Kappa es un índice que compara el nivel de coincidencia entre varios expertos con el nivel de coincidencia que se podría dar por casualidad.

En realidad, puede verse como el nivel de coincidencia entre los expertos, al cual se le resta el que se da por casualidad, para tener una medida totalmente indicativa. Valores de $+1$ indicar total coincidencia (el valor ideal), valores de 0 indican no más coincidencia de la que puede esperarse por casualidad y valores de -1 total desacuerdo. Supongamos que tenemos un total de 29 pacientes[2], examinados por dos doctores de manera independiente y se obtienen los siguientes resultados:

		Doctor A		
		No	**Yes**	**Total**
Doctor B	**No**	**10 (34.5%)**	**7 (24.1%)**	**17 (58.6%)**
	Yes	**0 (0.0%)**	**12 (41.4%)**	**12 (41.4%)**
Total		**10 (34.5%)**	**19 (65.5%)**	**29**

de tal forma tenemos el siguiente resultado:

```
Kappa = (Observed agreement - Chance agreement)/(1 - Chance agreement)
Observed agreement = (10 + 12)/29 = 0.76
Chance agreement = 0.586 * 0.345 + 0.655 * 0.414 = 0.474
Kappa = (0.76 - 0.474)/(1 - 0.474) = 0.54
```

Las medidas que parecen seguidamente, se refieren a cada una de las clases. TP hace referencia a `true positive` (observaciones $\bar{x}$ que el modelo clasificó correctamente para cada clase), FP a `false positives` (observaciones $\bar{x}$ que el modelo clasificó como de esa clase cuando en realidad no lo eran).

```
=== Detailed Accuracy By Class ===

TP Rate   FP Rate   Precision   Recall  F-Measure   Class
  1         0          1         1         1        Iris-setosa
  0.96      0.02       0.96      0.96      0.96     Iris-versicolor
```

[2] http://www.dmi.columbia.edu/homepages/chuangj/kappa/

```
  0.96      0.02       0.96      0.96      0.96     Iris-virginica

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 48  2 |  b = Iris-versicolor
  0  2 48 |  c = Iris-virginica
```

Con precisión nos referimos a la fracción de ejemplares que se han clasificado como de la clase correspondiente y que, en realidad, son de esa clase. Por tanto,

$$Precision = \frac{TP}{TP + FP},$$

y luego tenemos el *recall* (sensibilidad) que se refiere a la fracción de ejemplos de la clase de todo el conjunto que se clasifican correctamente,

$$Recall = \frac{TP}{TP + FN}.$$

Por último, tenemos una combinación de ambas,

$$f - measure = \frac{2 \times Precision \times Recall}{Precision + Recall}.$$

La última información que aparece es la de la matriz de confusión. En esta, se muestra en forma de matriz, en qué han consistido, concretamente, las equivocaciones del clasificador ya que en $m[i, j]$ se tiene el número de ejemplares que son de la clase i y se han clasificado como de la clase j. Por tanto, es de esperar que la diagonal principal contenta enteros altos y el resto de celdas números próximos a cero o cero. En este ejemplo concreto podemos ver cómo se han clasificado dos ejemplares de la iris virgínica como si fueran iris versicolor. Lo mismo ha pasado a la inversa.

3.2.2 Modelos de clustering

Si intentamos generar un clustering k-means, teniendo cuidado de que el parámetro k tenga valor 3, obtendremos la siguiente salida

```
=== Run information ===

Scheme:       weka.clusterers.SimpleKMeans -N 3 -S 10
Relation:     iris-weka.filters.unsupervised.attribute.Remove-R5
Instances:    150
Attributes:   4
              sepallength
              sepalwidth
              petallength
              petalwidth
Test mode:    evaluate on training data

=== Model and evaluation on training set ===


kMeans
```

```
======

Number of iterations: 6
Within cluster sum of squared errors: 6.9981140048267605

Cluster centroids:

Cluster 0
Mean/Mode:  5.8885 2.7377 4.3967 1.418
Std Devs:   0.4487 0.2934 0.5269 0.2723
Cluster 1
Mean/Mode:  5.006  3.418  1.464  0.244
Std Devs:   0.3525 0.381  0.1735 0.1072
Cluster 2
Mean/Mode:  6.8462 3.0821 5.7026 2.0795
Std Devs:   0.5025 0.2799 0.5194 0.2811

Clustered Instances

0       61 ( 41%)
1       50 ( 33%)
2       39 ( 26%)
```

Recordemos que el clustering no es un problema de clasificación sino de agrupación de datos. En el caso del K-means, agrupamos los datos mediante su representación por tres centroides (uno por cada clase ya que $k = 3$). Después de seis iteraciones del algoritmo de ajuste de los centroides, se muestra cada uno de los puntos mediante sus cuatro dimensiones y la agrupación que ha descubierto el clustering. Podemos visualizar los clusters que ha descubierto el algoritmo si pinchamos con el botón derecho en el panel que muestra la lista de resultados, en el correspondiente a nuestro algoritmo y, posteriormente, seleccionamos la visualización de clusters. Obtendremos las ventanas de la figura 2.

4 Ejercicios

Ejercitar lo aprendido con los conjuntos de datos que vienen, en formato arff, con Weka. Por ejemplo, intentar responder a las siguientes preguntas:

1. Determinar el tipo de lentes de contacto para un determinado paciente a partir de medidas de los pacientes diversas.

2. Determinar qué factores pueden influir en la consideración para obtener una hipoteca.

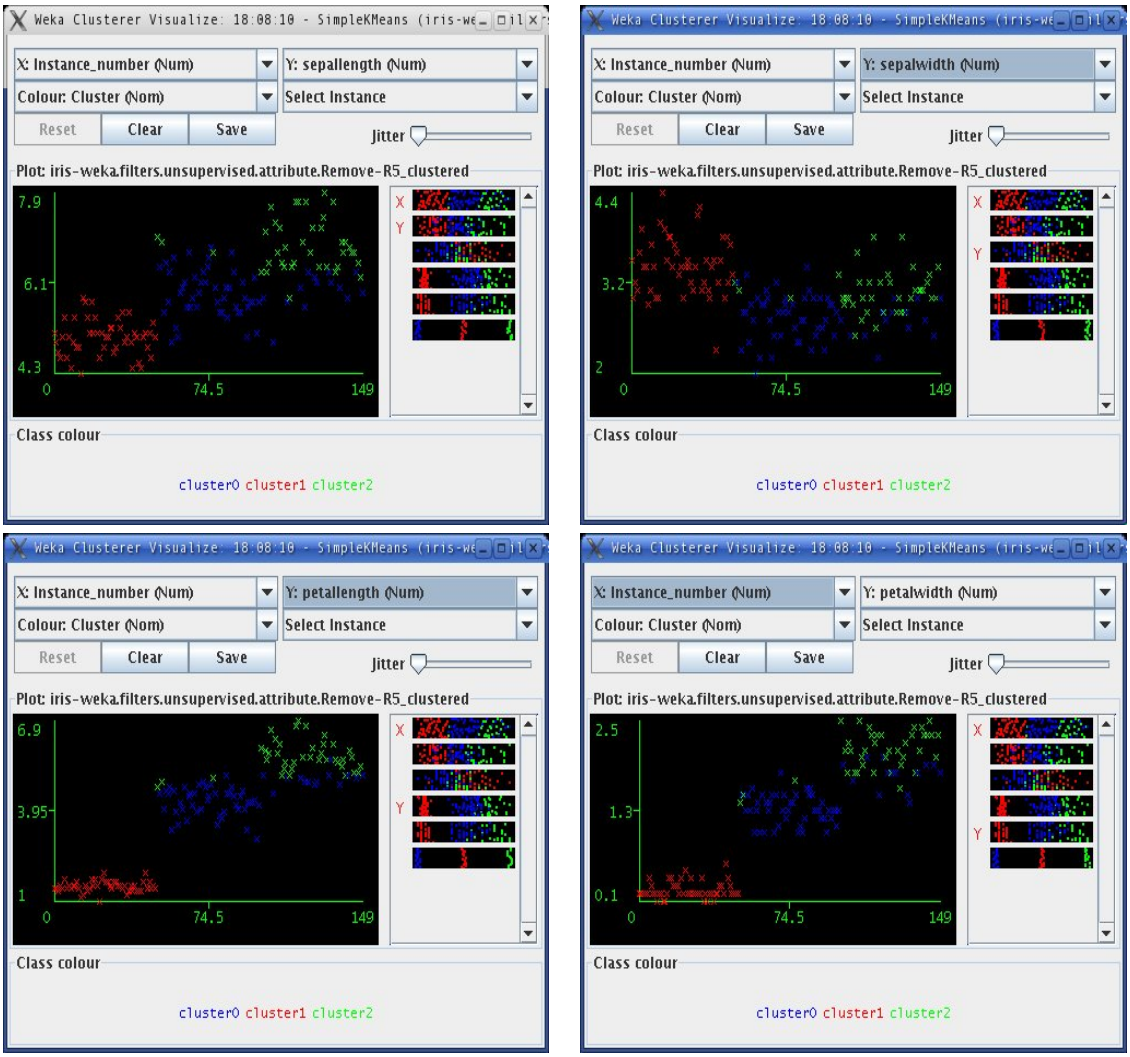


Figure 2: Visualización de los clusters encontrados para el conjunto iris, según cada uno de los cuatro atributos