Tutorial. Uso básico del API de WEKA

MRA, 2011/12

25 de febrero de 2012

Índice

1. Introducción

Todas las funcionalidades disponibles en los distintos interfaces gráficos (GUIs) de WEKA pueden ser usadas desde código Java, junto con algunas otras no accesibles directamente desde el GUI (normalmente funciones de bajo nivel).

1.1. Listado de paquetes

- **weka.core**: Paquete con las clases e interfaces que conforman la infraestructura de WEKA. Son comunes a los distintos algoritmos implementados en WEKA.
  - Define las estructuras de datos que contienen los datos a manejar por los algoritmos de aprendizaje
    - Clase **Instances**: encapsula un dataset (conjunto de datos) junto con los métodos para manejarlo (creación y copia, división en subdatasets [entrenamiento y prueba], aleatorización, gestión de pesos, ...)
    - Clase **Attribute**: encapsula los atributos que definen un dataset (nombre de atributo, tipo [nominal, numérico, string], valores posibles).
    - Clase **Instance**: encapsula cada uno de los ejemplos individuales que forman un dataset, almacenando los valores de los respectivos atributos.
  - Subpaquete **weka.core.converters**: clases auxilares para leer y escribir datasets desde distintas fuentes de datos (ficheros ARFF, bases de datos, etc)
  - Subpaquete **weka.core.neighboursearch**: implementaciones de algoritmos y estrucutras de datos para la búsqueda eficiente a instancias similares ("vecinas")

- **weka.classifiers**: Paquete con las implementaciones de algoritmos de clasificación (tanto a métodos de clasificación discreta como de predicción numérica).
  - Subpaquetes: **weka.classifiers.bayes**,**weka.classifiers.rules**, **weka.classifiers.lazy**, **weka.classifiers.trees**, **weka.classifiers.functions**, **weka.classifiers.meta**, etc
  - Clase abstracta **Classifier**: métodos comunes a todos los clasificadores
- **weka.clusterers**: Paquete con las implementaciones de algorimtos de clustering.
  - Clase abstracta **AbstractClusterer**: métodos comunes a todos los algoritmos
  - Clase **ClusterEvaluation**: evaluador de clusters
- **weka.attributeSelection**: Paquete con métodos de selección de atributos.
  - El proceso de selección de atributos involucra 2 tipos de clases:
    1. Evaluadores de atributos: heredan de la clase abstracta: **ASEvaluation**. Son de 2 tipos
       - miden la relevancia de atributos aislados
       - miden la relevancia de combinaciones de 2 o más atributos
    2. Métodos de búsqueda (selectores): algoritmos de búsqueda que empleando los evaluadores usan diversas estrategias para comprobar la bondad de distintas combinaciones de atributos (los más simples sólo hacen un ranking). Heredan de la clase abstracta: **ASSearch**
  - Clase **AttributeSelection**: Encapsula el proceso de selección de atributos relevantes, combinando un evaluador y un selector.
- **weka.filters**: Paquete con diversos filtros para procesar los datos. Normalmente son usados para preprocesar los datos de entrenamiento/evaluación antes de utilizar los algoritmos de aprendizaje de WEKA, aunque en algunos casos los filtros tienen una utilidad por si mismos.
  - Pueden aplicarse para tratar instancias o atributos
    - Filtrado de instancias: procesan las instancias de un dataset realizando transformaciones sobre ellas: selección de instancias, borrado, generación de nuevas instancias, ...
    - Filtrado de atributos: procesan los atributos de un dataset realizando transformaciones sobre ellos: selección de atributos (delegan el trabajo en clases **AttributeSelection**), cambio de formato de atributos (numérico a nominal, texto a nominal, etc), normalización de valores, generación de nuevos atributos, etc
  - En función de la información que manejan:
    - Filtros supervisados: usan información sobre la clase de las instancias
    - Filtros no supervisados: no usan información sobre la clase de las instancias
  - Clase abstracta **Filter**: define las funcionalidades básicas de todos los filtros, junto con métodos estáticos útiles para crear y ejecutar filtros.
- **weka.associations**: Paquete con las implementaciones de algoritmos de aprendizaje de reglas de asociación
- **weka.gui**: Paquete con la implementación de los interfaces gráficos de WEKA
- **weka.datagenerators**, **weka.estimators**, **weka.experiment**: Paquete con clases e interfaces para la generación de datasets "artificiales" y la realización de experimentos y evaluaciones.

1.2. Referencias

- Página de WEKA
- Javadoc del API de WEKA
- Wiki de WEKA

- Programación con WEKA
- Clasificación con WEKA
- Uso de clustering

1.3. Código de ejemplo (para Weka 3.7.x)

- Clustering de ficheros ARFF
  - Muestra la carga de ficheros ARFF y el uso de algoritmos de clustering
  - Compilación: **javac -cp .:weka.jar ClusteringSimple.java**
  - Ejecución: **java -cp .:weka.jar ClusteringSimple iris.arff**
- Clasificador de SPAM
  - Muestra la creación de datasets (atributos+instancias), la generación de vectores numéricos a partir de texto con el filtro **StringToWordVector** y el entrenamiento y uso de un clasificador de tipo k-NN.
  - Compilación: **javac -cp .:weka.jar ClasificadorSPAM.java**
  - Ejecución: **java -cp .:weka.jar ClasificadorSPAM 3 train test**

2. Clases básicas: datasets, atributos e instancias

Para crear y gestionar los conjuntos de entrenamiento o validación en memoria es necesario utilizar las clases **Instances**, **Attribute** e **Instance** del paquete **weka.core**.

2.1. Clase Instances

Representación en memoria de una colección de ejemplos (dataset).

- Descrito por un conjunto de atributos (**Attribute**).
- Contiene un conjuntos de instancias/ejemplos (**Instance**) que almacenan conteniendo los valores de sus atributos.
- Opcionalmente uno de los atributos podrá estar marcado como atributo clase.
  - en **clasificación** el atributo clase debe de ser de tipo **Nominal**
  - en **predicción numérica** el atributo clase debe de ser de tipo **Numérico**

Constructores y métodos de interés:

- **Instances(String nombre,ArrayList¡Attribute¿atributos,int capacidad)**: Crea un dataset con el nombre y la capacidad indicada, asignándole la lista de atributos que recibe en el **ArrayList**
- **Instances(java.io.Reader reader)**: Crea un dataset y lo carga desde el fichero ARFF al que apunta el **Reader**.
- Manejar atributos
  - Buscar por posición: **Attribute attribute(int index)**
  - Buscar por nombre: **Attribute attribute(String name)**
  - Establecer atributo clase: **void setClass(Attribute att)**, **void setClassIndex(int classIndex)**
- Manejar instancias
  - Añadir una instancia: **void add(Instance instance)**

- Recupear instancias: **Instance instance(int index)**, **Instance firstInstance()**, **Instance lastInstance()**, **Enumeration enumerateInstances()**, ...

- Estadísticas: **kthSmallestValue(Attribute att, int k)**, **meanOrMode(Attribute att)**, **numDistinctValues(Attribute att)**, ..
- Manejar el dataset: **delete()**, **delete(int index)**, **randomize(java.util.Random random)**, **stratify(int numFolds)**, **Instances resample()**, **Instances testCV(int numFolds, int numFold)**, **Instance trainCV(int numFolds, int numFold)**, ...

2.2. Clase Attribute

El API de WEKA contempla cinco tipos de atributos.

- **numérico:** representa un valor de tipo real
  - **Constructor: Attribute(String nombre)**
- **nominal:** representa un valor tomando de un conjunto discreto de valores posibles
  - **Constructor: Attribute(String nombre, List¡String¿valores)**
  - Recibe un **List** de **String** con las etiquetas de los valores posibles
- **string:** representa una cadena de caracteres
  - **Constructor: Attribute(String nombre, (List¡String¿) null)**
- **fecha:** representa una fecha
- **relacional:** representa un valor que a su vez estará estructurado en distintos atributos

Métodos:

- **int index()**: índice del atributo dentro de un dataset (**Instances**)
- **String name()**: etiqueta del atributo
- **int numValues()**: número de valores posibles de un atributo Nominal
- **String value(int valIndex)**: devuelve la etiqueta indicada de un atributo Nominal
- **void setWeight(double value)**, **double weight()**: establece y recupera el peso del atributo

2.3. Clase Instance

Almacena los valores de un ejemplo (instancia).
Internamente los valores de los atributos de cada instancia se representan como un vector de números reales (**double[]**), independientemente del tipo de los atributos.
Normalmente estará asociado a un dataset (**Instances**) que determina el formato y el tipo de los atributos cuyos valores almacena la instancia.

Constructor y métodos:

- **Instance(int numAttributes)**: construye una instancia con el número de atributos indicados
- **setDataset(Instances instances)**: indica el dataset (**Instance**) del cual esta instancia almacena valores, describe el formato de la instancia (número y tipo de atributos)
- **Attribute attribute(int index)**: devuelve el atributo indicado
- **Attribute classAttribute()**: devuelve el atributo clase (si está definido)

- **double classValue()**: devuelve el valor almacenado en el atributo clase (es el índice de la etiqueta de la clase)
- **setClassValue(String value)**: establece el valor del atributo clase
- **double value(Attribute att)**, **double value(int index)**: devuelve el valor de un atributo numérico (o el índice del valor en los nominales)
- **String stringValue(Attribute att)**, **String stringValue(int index)**: devuelve el valor de un atributo nominal o string
- **setValue(Attribute att, double value)**, **setValue(int attIndex, double value)**: establece el valor de un atributo Numérico
- **setValue(Attribute att, String value)**, **setValue(int attIndex, String value)**: establece el valor de un atributo Nominal o String

Existe una subclase **SparseInstance** orientada a almacenar vectores de atributos con muchos valores nulos de forma compacta. En las instancias **SparseInstance** sólo se almacenan los atributos (numéricos o nominales) distintos de cero.

3. Uso de filtros

Los filtros proveen un mecanismo para procesar datasets e instancias y realizar transformaciones sobre ellos. Los filtros tienen dos modos de funcionamiento: modo entrenamiento y modo uso.

- **Modo entrenamiento** (o modo construcción):
  - El filtro recibe instancias mediante el método **input()** que van siendo acumuladas
  - Al invocar al método **batchFinished()** se finaliza la fase de construcción y se realizan los cálculos y operaciones necesarias para realizar el filtrado.
    - Se crean las estructuras de datos precisas para el filtrado.
    - Las instancias recibidas se filtran y se ponen en la cola de salida de instancias filtradas (accesibles mediante la función **output()** en el mismo orden en que fueron introducidas).
- **Modo de uso normal**:
  - El filtro recibe instancias con **input()**, las procesa y las hace disponibles en la cola de salida mediante la función **output()**.

Nota: Para realizar la fase de construcción y filtrar un dataset complero se puede utilizar el método estático **Instances useFilter(Instances data, Filter filter)** de la clase abstracta **Filter**.

```
...
public static Instances useFilter(Instances data, Filter filter) {
    for (int i = 0; i < data.numInstances(); i++) {
      filter.input(data.instance(i));
    }
    filter.batchFinished();
    Instances newData = filter.getOutputFormat();
    Instance processed;
    while ((processed = filter.output()) != null) {
      newData.add(processed);
    }
    return newData;
}
...
```

Normalmente la estructura (múmero y tipo de atributos) de las instancias originales y la de las instancias filtradas no coincidirán.

- El método **setInputFormat(Instances instanceInfo)** establece el formato de las instancias de entrada.
- El método **Instances getOutputFormat()** devuelve el formato de las instancias filtradas de salida.

3.1. StringToWordVector

La mayoría de los algoritmos de WEKA no pueden manejar directamente atributos de tipo String.
El filtro **StringToWordVector** (en el paquete **weka.filters.unsupervised.attribute**) transforma los atributos de tipo string en vectores numéricos.

- Tokeniza los atributos de tipo string para extraer las palabras
- Opcionalmente normaliza las palabras (extrae las raices *stemming*) y elimina las palabras presentes en una lista de palabras frecuentes (*stopwords*).
- Selecciona las palabras/tokens a utilizar para construir los vectores
- Define el dataset de salida, donde a cada una de las palabras supervivientes le corresponderá un atributo numérico.
- El valor de esos atributos numéricos se calcula a partir de la frecuencia de la correspondiente palabra en las instancias de entrada.

La selección de palabras se realiza durante la fase de construcción del filtro al ejecutar **batchFinished()** (o indirectamente mediante **Filter.useFilter()**).
Nota:

- Se puede especificar los índices de los atributos sobre los que se aplicará el filtro con **setSelectedRange(String rango)**
  - Recibe un String con los índices (o rangos de índices) de los atributos a filtrar separados por comas
  - La numeración de los índices empieza por 1, no por 0
- Los atributos no filtrados (incluido el atributo clase si lo hubiera) se colocarán al inicio del dataset resultante del filtrado.

Métodos:

- **setSelectedRange(java.lang.String newSelectedRange)**: especifica los atributos a filtrar
- **setTokenizer(Tokenizer value)**: configura el tokenizador a emplear
- **setStemmer(Stemmer value)**: configura el stemmer a emplear
- **setUseStoplist(boolean useStoplist)**, **setStopwords(java.io.File value)**: configuran el uso de listas de stopwords
- **setWordsToKeep(int newWordsToKeep)**: número de palabras (atributos) a mantener en el dataset final
- **setIDFTransform(boolean IDFTransform)**, **setNormalizeDocLength(SelectedTag newType)**, **setOutputWordCounts(boolean outputWordCounts)**, **setTFTransform(boolean TFTransform)**: configuran cómo se calcularán los valores (pesos) de los atributos numéricos resultantes

4. Algoritmos de clasificación

Todos los algoritmos de clasificación heredan de **weka.classifiers.Classifier** y deben de implementar los siguientes métodos básicos:

- **void buildClassifier(Instances data)**: entrena el clasificador con el conjunto de entrenamiento (**Instances**) indicado
- **double classifyInstance(Instance instance)**: clasifica la instancia que recibe como parámetro. [Exige haber invocado antes a **buildClassifier()**]
  - La estructura de la instancia (número y tipo de atributos) debe coincidir con la del objeto **Instances** usado en el entrenamiento
  - El valor devuelto (de tipo *double*) indica la clase predicha. Se corresponde con el índice de su etiqueta en el objeto **List** asociado al atributo clase.
- **double[] distributionForInstance(Instance instance)**: clasifica la instancia y devuelve un vector **double[]** con un componente para cada valor del atributo clase que cuantifica su probabilidad o importancia relativa (dependiendo del método de clasificación). [Exige haber invocado antes a **buildClassifier()**]

La clase abstracta **Classifier** también ofrece el método **Classifier forName(String classifierName, String[] options)** que crea un clasificador de la clase indicada con los parámetros que se le pasan como array de String (el javadoc de cada método de clasificación especifica el formato de las opciones que esperan)

5. Algoritmos de clustering

Todos los algoritmos de clustering implementan el interfaz **weka.clusterers.Clusterer** y deben de aportar los siguientes métodos básicos:

- **void buildClusterer(Instances data)**: calcula los clusters (grupos) para el dataset de entrenamiento indicado
- **int numberOfClusters()**: número de clusters resultantes (dependiendo del método concreto se especifica antes de entrenar o se calcula durante el entrenamiento)
- **int clusterInstance(Instance instance)**: indica el cluster al que pertenece la instancia pasada como argumento. [Exige haber invocado antes a **buildClusterer()**]
  - La estructura de la instancia (número y tipo de atributos) debe coincidir con la del objeto **Instances** usado en el entrenamiento
- **double[] distributionForInstance(Instance instance)**: devuelve un vector **double[]** donde cada uno de sus componentes cuantifica el grado de pertenecia de la instancia al cluster correspondiente. [Exige haber invocado antes a **buildClusterer()**]

6. Selección de atributos

El proceso de selección de atributos se divide en dos tareas:

- Evaluar la bondad de cada atributo o combinación de atributos. Se delega en un objeto que herede de la clase **weka.attributeSelection.ASEvaluation**
  - Evaluación atributos simples: **ChiSquaredAttributeEval**, **GainRatioAttributeEval**, **InfoGainAttributeEval**, **CostSensitiveAttributeEval**, ...
  - Evaluación grupos de atributos: **ClassifierSubsetEval**, **ConsistencySubsetEval**, **CostSensitiveSubsetEval**, ...
- Búsqueda y selección de la lista de mejores atributos. Se delega en un objeto que herede de la clase **weka.attributeSelection.ASSearch**

- Búsqueda de atributos simples: **Ranker**, ...
- Búsqueda de grupos de atributos: **BestFirst**, **GreedyStepwise**, **ExhaustiveSearch**, **GeneticSearch**, ...

El método de búsqueda/selección determina los evaluadores que son admitidos (ver javadoc o comprobar compatibilidad desde el interfaz gráfico de WEKA)

Métodos:

- **void setEvaluator(ASEvaluation evaluator)**: establece el evaluador de atributos (o conjuntos de atributos)
- **void setSearch(ASSearch search)**: establece el método de búsqueda/selección
- **void SelectAttributes(Instances data)**: aplica la selección de atributos con el dataset indicado
  - Para realizar la selección de atributos mediante Validación Cruzada (*cross validation*) es necesario especificar un valor de *true* a **setXval(boolean x)** e indicar el número de pasadas con **setFolds(int folds)**
- **Instance reduceDimensionality(Instance in)**: reduce la dimensión de una instancia, incluyendo únicamente los atributos seleccionados en la última ejecución de **SelectAtributes**
- **Instances reduceDimensionality(Instances in)**: reduce la dimensión de un conjunto de instancias (dataset), incluyendo únicamente los atributos seleccionados en la última ejecución de **SelectAtributes**
- **int numberAttributesSelected()**: devuelve el número de atributos seleccionados en la última ejecución de **SelectAtributes**
- **int[] selectedAttributes()**: devuelve (en orden) el índice de los atributos seleccionados en la última ejecución de **SelectAtributes**
- **double[][] rankedAttributes()**: devuelve (en orden) el índice de los atributos seleccionados en la última ejecución de **SelectAtributes**, junto con su relevancia en un array **double[][]**

Ejemplo: ordenar atributos por ganancia de formación

```
Instances dataset = new Instances(new FileReader(args[0]));
dataset.setClassIndex(dataset.numAttributes()-1);

AttributeSelection as = new AttributeSelection();
as.setEvaluator(new InfoGainAttributeEval());
as.setSearch(new Ranker());
// // Para usar crosswalidation
// as.setFolds(10);
// as.setXval(true);
as.SelectAttributes(dataset);

System.out.println("Atributos ordenados:");
for(int i: as.selectedAttributes()) {
    System.out.println(dataset.attribute(i).name());
}
```