MINERÍA DE DATOS COMPLEJOS. CUADERNILLO DE PRÁCTICAS

1.TUTORIAL BÁSICO DE WEKA

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
    - *Instances*: Clase que encapsula un *dataset* de datos junto con los métodos para manejarlo (e.j. creación y copia, división en subdatasets [entrenamiento y prueba] etc.).
    - *Attribute*: Clase que encapsula los atributos que definen un *dataset*
    - *Instance*: Clase que encapsula cada uno de los ejemplos individuales que forman un *dataset*, almacenando los valores de los respectivos atributos.
  - ***weka.core.neighboursearch***: subpaquete con la implementación de algoritmos y estructuras de datos para la búsqueda eficiente de vecinos. La clase que vamos a utilizar es similar y se llama `moa.classifiers.lazy.neighboursearch.LinearNNSearch`.

1.2.CLASES BÁSICAS

CLASE INSTANCES

Representación en memoria de una colección de ejemplos (*dataset*) descrito por un conjunto de atributos (*Attribute*). Contiene un conjunto de instancias/ejemplos (*Instance*) que almacenan conteniendo los valores de sus atributos.

Opcionalmente (ej. En clasificación) uno de los atributos podrá estar marcado como ***atributo de clase***. Esto habrá que tenerlo en cuenta al ejecutar algoritmos de clasificación. Habrá que configurar en el *main* cuál es el atributo de clase.

Métodos

- Constructores:
  - *Instances(java.io.Reader reader)*: Crea un *dataset* y lo carga desde el fichero ARFF al que apunta el *Reader*.
  - *Instances(Instances dataset)*: Crea un *dataset* copiando las instancias del *dataset* que se pasa como parámetro.
  - Instances(Instances dataset, int capacity): Crea un *dataset* vacío con la estructura del *dataset* que se pasa como parámetro.

    - Instances(String relationName, FastVector attrInfo, int capacity): Crea un *dataset* vacío con la estructura atributos que se pasa como parámetro.
- Manejar atributos
    - *void setClassIndex(int classIndex):* Establece el atributo de clase (valor en [0, numAttributes-1]).
- Manejar instancias
    - Instance instance(int index)**:** Recupera la instancia index-ésima.
    - Instance remove(int index): Elimina la instancia index-esima.
- Estadísticas: *numInstances()*,*numAttributes()***,** *num.Classes()*.

CLASE INSTANCE

Almacena un ejemplo o patrón (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.

Métodos

- *double classValue()*: devuelve el valor almacenado en el atributo clase en formato interno (es el índice de la etiqueta de la clase).
- *double value(int index)*: devuelve el valor de un atributo numérico (o el índice del vector del valor en los nominales (ej. classValue).
- void setValue(int index, double value): establece un valor determinado para un atributo.

CLASE LINEARNNSEARCH

Implementa la búsqueda de los vecinos más cercanos por fuerza bruta.

Métodos

- LinearNNSearch(Instances insts). Constructor con vecindario.
- void setInstances(Instances insts). Establece las instancias que conformarán el vecindario (*neighbourhood*).
- setSkipIdentical(boolean skip). Establece la propiedad que incluye (o no) dentro de los vecinos devueltos aquellas instancias idénticas a la instancia objetivo (aquellas con distancia cero a la instancia).
- Instances kNearestNeighbours(Instance target, int kNN). Devuelve un nuevo *dataset* con los kNN vecinos de la instancia objetivo.
    - Puede devolver más de k vecinos en caso de que haya empates de distancia.
    - Los vecinos los devuelve ordenados por distancia.
- Instance nearestNeighbour(Instance target) Devuelve el vecino más cercano a la instancia objetivo en el vecindario.
- double[] getDistances(). Devuelve las distancias a los k vecinos más cercanos. Requiere haber llamado previamente a kNearestNeighbours

Aprovecharemos la clase **LinearNNSearch** para la búsqueda de vecinos y el cálculo de las distancias.

```
LinearNNSearch S = new LinearNNSearch(dataset);
S.setSkipIdentical(true); // No utiliar en MOA
Instances kNN = S.kNearestNeighbours(instancia, k);
```

DESARROLLAR UN CLASIFICADOR KNN EN MOA

FUENTES

- http://moa.cms.waikato.ac.nz/documentation/

CLASE MOA.CLASSIFIERS.ABSTRACTCLASSIFIER

Todos los clasificadores MOA heredan de **moa.classifiers.AbstractClassifier**

- **InstancesHeader getModelContext().** Gets the reference to the header of the data stream. The header of the data stream is extended from WEKA Instances. This header is needed to know the class attribute and the attributes. **Returns:** the reference to the data stream header.
- **boolean correctlyClassifies(Instance instance)**. Gets whether the classifier correctly classifies an instance. Uses getVotesForInstance to obtain the prediction.

Habrá que implementar los siguientes métodos:

- **public void resetLearningImpl()**. Este método se llama cuando comienza el aprendizaje. Debe configurar el clasificador con el estado correspondiente a no haber recibido aún ninguna instancia.
- **public void trainOnInstance(Instance inst)**. Este método se llama cada vez que llega una nueva instancia.
- **double[] getVotesForInstance(Instance inst)**. Este método utiliza el modelo generado para predecir la clase de una instancia de clase desconocida.

IMPLEMENTACIÓN DEL CLASIFICADOR

La clase se llamará *StreamKnn*, extenderá *AbstractClassifier* y tendrá los siguientes parámetros:

- **k**. Representa el número de vecinos.
- **useReservoir**. Indica si el algoritmo trabajará o no con un *reservoir*. De tipo boolean.
- **maxSize**. Tamaño máximo de la ventana o del *reservoir*.

La clase tendrá las siguientes propiedades privadas

- **Window**. Ventana con las instancias más recientes. De tipo *Instances*
- **maxClassValue**. Mayor valor de clase que haya aparecido, para contar el número total de clases. De tipo entero.
- **numProcessedInstances**: Número de instancias procesadas. De tipo entero.

EJERCICIO 1
- **Crea un proyecto java para realizar esta práctica.**
- **Crea una clase de nombre *StreamKNN* que herede de moa.classifiers.AbstractClassifier. La clase tendrá las propiedades públicas y privadas descritas arriba.**
- **Crea el constructor de la clase (public StreamKNN(lista de parámetros)) para que acepte como parámetros las tres propiedades públicas y las inicialice.**

MÉTODO resetLearningImpl()

Se encargará de inicializar la ventana como un *dataset* vacío. Además inicializará los valores de *classCount* y *numInstances*.

EJERCICIO 2
- **Completa el método void resetLearningImpl() con la iniciación de maxClassValue y numInstances.**

MÉTODO trainOnInstanceImpl(Instance instance)

Este método actualiza el número de clases y, en función de que se trabaje o no con *reservoir*, actualiza la ventana.

EJERCICIO 3
- **Completa el método updateWindow(Instance instance) para que añada una nueva instancia a la venta (window.add(instance)). Si la ventana está llena, habrá que eliminar elemento más antiguo (window.remove(0)).**

EJERCICIO 4
- **Completa el método updateReservoir para que añada una nueva instancia a la ventana.**
    - **Si la ventana no está llena, se insertará la instancia en el reservoir.**
    - **Si la ventana está llena:**
        - **Se obtendrá un número aleatorio para determinar si la instancia debe insertarse en la ventana.**
        - **En caso afirmativo. El número aleatorio determina qué elemento del reservoir eliminar.**

MÉTODO getVotesForInstances(instance)

Este método realiza la predicción para una nueva instancia.

EJERCICIO 5
- **Implementa la búsqueda de los vecinos. Tienes un ejemplo al inicio de este tutorial**

EJERCICIO 6
- **Crear el vector resultado de tamaño número de clases: classCount+1**
- **Recorrer los vecinos (un dataset de Instances)**
  - **Actualizar el vector de resultados con el voto de cada vecino: votes[(int)neighbours.instance(i).classValue()]++;**

GENERAR UNA CLASE CON UN MÉTODO main

EJERCICIO 7.
- **Para probar la clase haremos una clase *Ejecutar* que tenga un *main*:**

```
public class Ejecutar {
	public Ejecutar() {
		// TODO Auto-generated constructor stub
	}
	public static void main(String[] args) {
		//Prepares the data generator
		int numInstances = 10000;
		RandomRBFGenerator stream = new RandomRBFGenerator();
		stream.prepareForUse();
		//Prepares the classifier
		StreamKNN knn = new StreamKNN(10, 1000, true);
		knn.setModelContext(stream.getHeader());//Sets the
reference to the header of the data stream
	knn.prepareForUse();

	long time_start, time_end;
	double accuracy;
	int correct=0, samples =0;
	time_start = System.currentTimeMillis();
	while(stream.hasMoreInstances() & samples<numInstances)
	{
		Instance inst = stream.nextInstance();
		if(knn.correctlyClassifies(inst))
		  correct++;
		samples++;
		knn.trainOnInstance(inst);
		accuracy  = 100.0*(double)correct/(double)samples;
		System.out.println("#samples: "+samples+ " #correct:
"+correct+" accuracy: "+accuracy);
	}
	time_end = System.currentTimeMillis();
	System.out.println("the task has taken "+ ( time_end -
time_start ) +" milliseconds");
}
```