Introducción

En la investigación, la tarea de la ejecución de experimentaciones con un gran número de datasets es normalmente una tarea ardua y tediosa. Asímismo, para el usuario no experto o científico, el uso de herramientas de minería de datos que se ejecutan en línea de comandos es complicado y a veces imposible, pues el usuario desconoce los parámetros del algoritmo y no puede acceder al artículo para comprobarlos.

Debido a este problema se crea EPM-Framework, el cual permite por un lado a los investigadores ejecutar repetidas veces un algoritmo en varios conjuntos de datos sencillamente, y por otro para los usuario no expertos el uso de la interfaz gráfica facilita la utilización de algoritmos de minería de datos.

Los algoritmos incluidos en este software son algoritmos ubicados en la tarea de minería de datos llamada "minería de patrones emergentes" (EPM, por sus siglas en inglés). Estos algoritmos no se encuentran actualmente unificados en ningún software público, por lo que es adecuado la unificación de estos métodos en un único paquete para ser presentados al público en general.

Este documento refleja como utilizar la herramienta EPM-Framework, en donde se describe como utilizar la herramienta e instrucciones para desarrolladores para añadir nuevos algoritmos.

Uso de la aplicación

La aplicación puede ser utilizada de dos formas:

- Mediante línea de comandos.
- Mediante una interfaz gráfica.

Uso mediante línea de comandos. Ejecución de un único algoritmo

El uso en línea de comandos permite la ejecución de un único algoritmo en un único par de conjunto de datos (entrenamiento - prueba). El uso mediante línea de comandos es el siguiente:

```
java -jar framework.jar /ruta/a/ficheroParametros.txt
```

Se puede observar que para utilizar la aplicación desde línea de comandos se pasa un único párametro. Este es una ruta a un fichero que contendrá los parámetros para ejecutar un algoritmo que se encuentre en el framework. Por ejemplo, para el algoritmo BCEP, este fichero contiene:

```
algorithm = BCEP
training = breast-5-1tra.dat
```

```
test = breast-5-1tst.dat
Minimum support = 0.01
Minimum GrowthRate = 100
```

Los únicos argumentos obligatorios que deben estar presentes en este fichero son "algorithm" que define el nombre del algoritmo a utilizar y cuyo nombre debe concordar con alguno de los nombres de los algoritmos del framework, "training" y "test" que definen, respectivamente, las rutas hacia los ficheros de entrenamiento y test. Se pueden utilizar tanto rutas relativas como absolutas en estos campos.

Una vez el algoritmo termina su ejecución, se almacenarán en la misma carpeta donde se ubica el fichero de entrenamiento cinco ficheros. Estos contendran:

1. La representación en forma de regla de los patrones obtenidos. Este fichero se llamará `RULES.txt`.

2. Las medidas de calidad para el entrenamiento de cada patrón. La última línea muestra los resultados medios, así como el accuracy y el área bajo la curva (AUC). Este fichero se llamará `TRA_QUAC_NOFILTER.txt`.

3. Igual que el anterior pero se muestra los resultados de un conjunto en donde se obtienen únicamente aquellos patrones que son minimales. Este fichero se llama `TRA_QUAC_MINIMAL.txt`.

4. Un fichero similar al anterior, pero en este caso se obtendrán aquellos patrones que sean maximales. El fichero se llama `TRA_QUAC_MAXIMAL.txt`.

5. Por último, un fichero en donde se obtendrán aquellas reglas que superen un valor de 0.6 en confianza en los datos de test. Este fichero se llama `TRA_QUAC_CONFIDENCE.txt`.

Así mismo, en la carpeta donde se ubica el fichero de test, se guardarán también estos ficheros, a excepción del fichero de reglas, empezando por el prefijo "TST".

Uso mediante línea de comandos. Ejecución sobre varios conjuntos de datos.

Es posible también ejecutar un único algoritmo sobre varios conjuntos de datos particionados siguiendo un esquema de validación cruzada y obtener únicamente unas medidas de calidad de test. Este método es ideal para investigadores que pretenden obtener resultados para comparaciones con los algoritmos del framework. Para poder ejecutar en este modo es necesario cumplir con varios requisitos previos:

1. Una carpeta que contendrá todos los conjuntos de datos. Está carpeta será llamada directorio raíz.

2. De la carpeta raíz colgará una carpeta por conjunto de datos. Dentro de cada carpeta se hallarán los conjunto de datos.

3. Los ficheros que corresponden al conjunto de datos particionado deberán terminar con el siguiente nombre: `NUM_FOLDS-i-tra.dat` para el fichero de entrenamiento de para la partición i, y `NUM_FOLDS-i-tst.dat` para el fichero de test asociado a al fichero de entrenamiento anterior. El valor `NUM_FOLDS` se corresponde el número de folds utlizados. Por ejemplo, si los datos están particionados usando 5-fold cross validation, este número será 5.

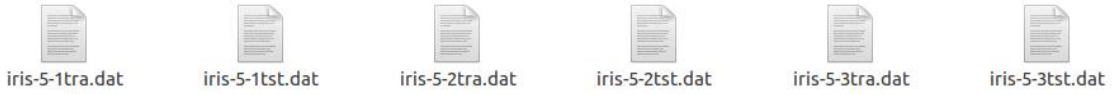

Figura 1: Ejemplo de nombrado de los ficheros para una partición del conjunto de datos iris con 5-fold cross validation. En esta imagen no se muestran los dataset correspondientes a las particiones 4 y 5.

Una vez se tienen estos requisitos previos cumplimentados, se puede ejecutar con la misma orden que en el método anterior, es decir, pasandole como argumento una ruta a un fichero con parámetros. Dicho parámetros deberá contener los una serie de campos. Siguiendo el ejemplo del método BCEP, el fichero de parámetros para ejecutar por lotes (o batch) es el siguiente:

```
algorithm = BCEP
directory = /home/agarcia/EPM_Seleccion_parametros/BCEP
number of folds = 5
Minimum support = 0.01
Minimum GrowthRate = 40
```

Nótese que el único cambio introducido ha sido la sustitución de los campos "training" y "test" por los parametros "directory", que indica la ruta hacia la carpeta raíz que contiene los conjuntos de datos, y "number of folds" que indica el número de particiones que existen en los conjuntos de datos. En este caso, es 5.

EL framework ejecutará el algoritmo sobre cada partición de todos los conjuntos de datos, cuando finalice con todas las particiones, obtendrá la media aritmética de todos los patrones obtenidos por ejecución para cada medida de calidad usada, aplicando una serie de filtros y guardará un fichero de medidas de calidad para cada uno de los filtros aplicados. En este sentido, los resultados son un único valor por medida de calidad, que indica el valor medio. En concreto, los ficheros que se generan son:

- El fichero `QM_unfiltered.txt` que contiene las medidas de calidad de todos los patrones sin filtrar.

- El fichero `QM_MINIMALS.txt` que contiene las medidas de calidad de aquellos patrones que son minimales.

- El fichero `QM_MAXIMALS.txt` que contiene las medidas de calidad de aquellos patrones que son maximales.

- El fichero `QM_BYCONFIDENCE.txt` que contiene las medidas de calidad de aquellos patrones que superan un valor 0.6 de confianza en los datos de entrenamiento.

Estos ficheros contienen los resultados de los datos de test.

NOTA IMPORTANTE: es necesario remarcar que este framework, actualmente, **no realiza el procedimiento de particionado mediante validación cruzada.** Por lo tanto, los conjuntos de datos deben ser proporcionados **ya divididos.**

Uso mediante interfaz gráfica

Para utilizar la interfaz gráfica es necesario que no se introduzcan parámetros en la llamada al ejecutable. La interfaz es la mostrada en la Figura 2.

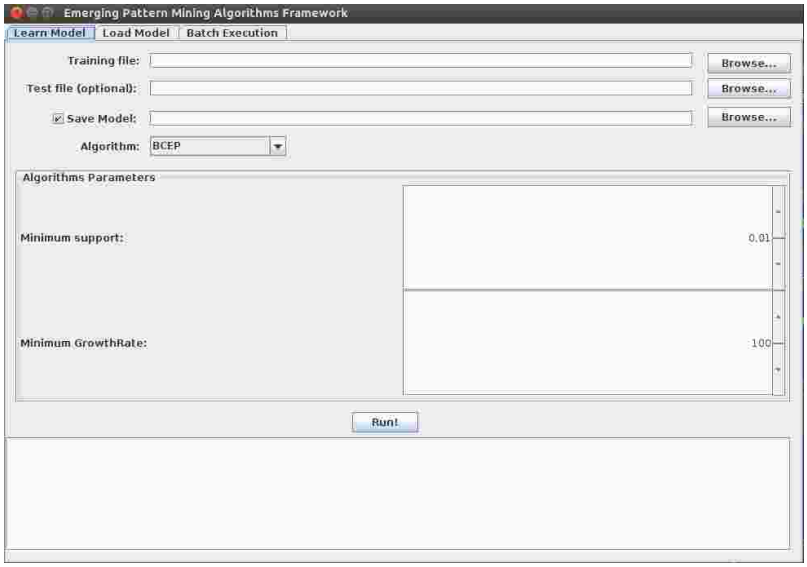

Figura 2: Pantalla inicial de la interfaz de usuario.

Pestaña *Learn Model*

Esta pestaña es la que se nos muestra inicialmente al abrir la interfaz de usuario. Su funcionalidad es similar a la ejecución por línea de comandos sobre un único par entrenamiento-prueba. Podemos ejecutar un único algoritmo (el seleccionado) con los parámetros establecidos sobre los ficheros de entrenamiento y prueba seleccionados. Sin embargo, en este último caso no es obligatorio el uso de un fichero de prueba. Esto se debe a que esta pestaña permite, además de la funcionalidad anterior, entrenar un modelo con el fin de guardarlo para posteriormente realizar predicciones en la pestaña *Load Model*. Si tenemos marcada la opción *Save Model*, podremos guardar este modelo entrenado en la ubicación que hayamos elegido bajo la extensión ".ser".

Una vez hayamos realizado la configuración del experimento, pulsamos el botón "Run!" para ejecutar el algoritmo. Cuando el algoritmo finalice, este devolverá el mismo conjunto de ficheros que el que se devuelve en la ejecución por línea de comandos en los directorios donde se almacenan los conjuntos de entrenamiento y test respectivamente.

Pestaña *Load Model*

Esta pestaña nos permitirá cargar un modelo previamente guardado con extensión ".ser" para predecir nuevas instancias en un conjunto de datos con instancias de test.

Una vez se pulse el boton "Run!" se iniciará el procedimiento de predicción. Cuando finalice, en el cuadro de texto que se encuentra debajo se mostrará o bien las predicciones para la clase, en caso de que sean instancias no etiquetadas, o bien una comparativa "clase real vs clase predicha" para cada instancia de test etiquetada. Este resultado se puede almacenar si se pulsa en el botón "Save to file...".

Pestaña *Batch Execution*

El objetivo de esta pestaña es el mismo que en el funcionamiento por línea de comandos. Por lo tanto, los requisitos previos y funcionamiento es idéntico.

Desarrollo de algoritmos para el framework

Todo el trabajo de obtención de medidas de calidad y filtrado de patrones, así como la ejecución en modo *batch*, forma parte del propio framework. Este sistema facilita mucho el desarrollo de métodos para el framework ya que el desarrollador debe centrarse únicamente en la implementación de su método de aprendizaje y predicción de instancias.

Para ello se ha realizado una estructura de paquetes para mantener organizadas los diferentes elementos del framework:

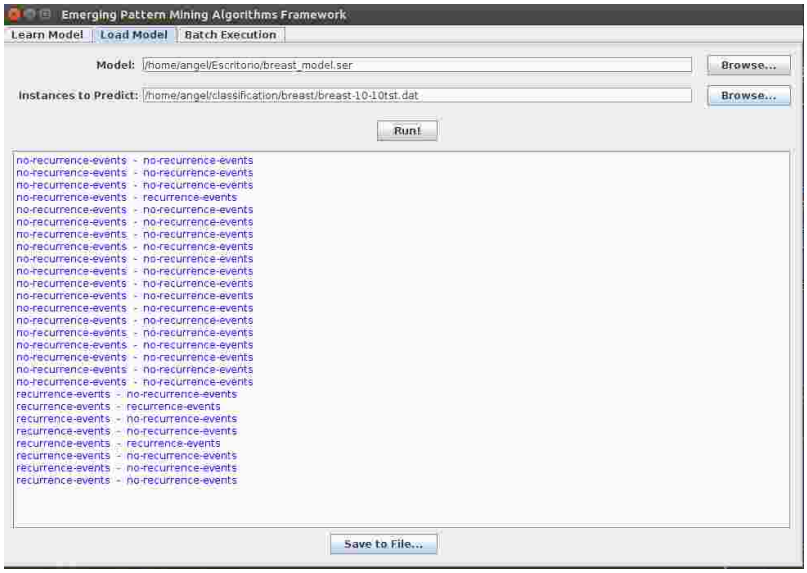

Figura 3: Pestaña *Load model* con los resultados de una predicción. Comparativa tipo "clase real vs clase predicha".

- Paquete "framework": De este paquete cuelgan diferentes subpaquetes con utilidades para el framework y los algoritmos que se desarrollen. Actualmente se encuentran los siguientes subpaquetes:
  - GUI. Este paquete contiene la interfaz gráfica, la clase Main que se ejecutará al iniciar el framework y la clase Model, de la que se hablará más adelante.
  - exceptions. Contiene diferentes clases para excepciones.
  - deprecated. Contiene las clases de un antiguo primer diseño del framework que son necesarias para los primeros métodos añadidos al framework.
  - items. Contiene las clases para representar los patrones. Un diagrama UML para representar esta clase se encuentra reflejado en la Figura 4.
  - utils. Contiene clases con métodos y/o estructuras de datos de utilidad para los algoritmos y el framework. Dentro de este paquete se encuentran los paquetes que contienen las estrcutras de datos CP-Tree y BSC-Tree.
- Paquete .ªlgorithms": Se encuentran los diferentes algoritmos desarrollados para el framework. Cada algoritmo representará un subpaquete.
- Paquete "keel": Tiene la funcionalidad necesaria para leer los ficheros de datos en formato de KEEL.

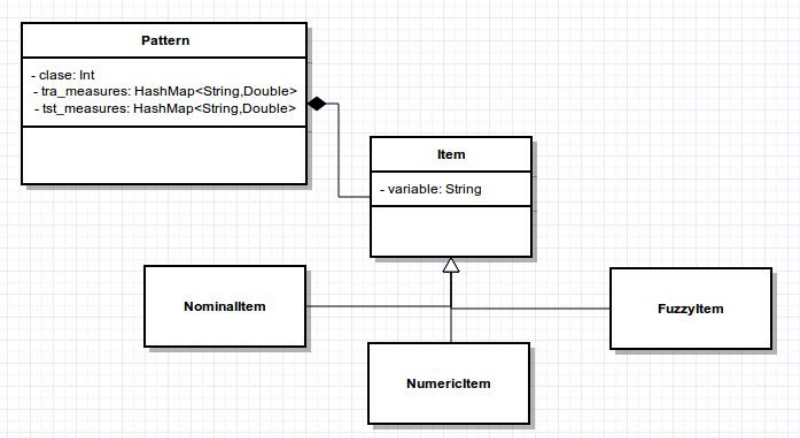

Figura 4: Diagrama de clases para representar un patrón.

Funcionamiento interno del framework

¿Cómo funciona el framework una vez se ejecuta en uno de los dos modos descritos anteriormente? El funcionamiento interno del framework desde que se carga el conjunto hasta que se obtienen los resultados es el siguiente:

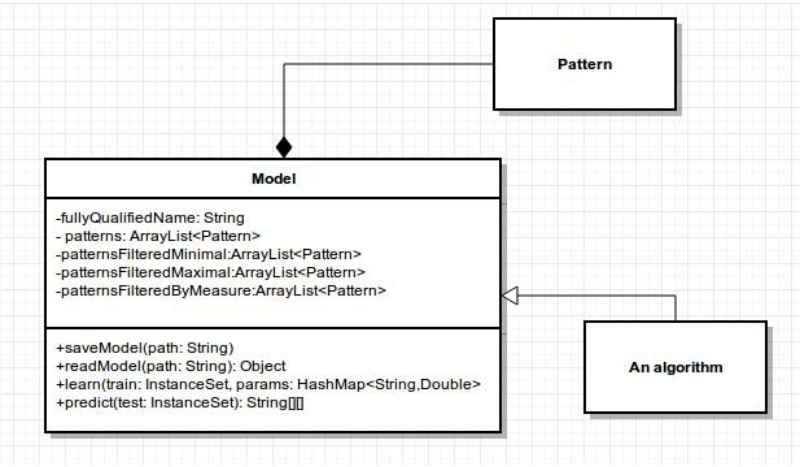

Figura 5: Diagrama de la clases de la clase Model. Nótese que un algoritmo desarrollado para el framework hereda de ésta.

1. Se leen los ficheros de entrenamiento y prueba y se almacena en clases InstanceSet. Asimismo, se almacenan los parámetros en una estructura HashMap<String,String>. En donde la clave será el nombre del parámetro tal y como se indica en el fichero "algorithms.xml" que se detallará más adelante.

2. Se crea una nueva instancia del algoritmo a ejecutar de manera dinámica a partir del valor que se encuentra en el valor **fullyQualifiedName** del fichero "algorithms.xml". Esta variable deberá seguir el formato **nombrePaquete.nombreClaseMain** donde nombrePaquete es el nombre del paquete y nombreClaseMain es el nombre de la clase que hereda de la clase Model y que contendrá los métodos learn() y predict(). A modo de ejemplo, en BCEP, este nombre es: **algorithms.bcep.BCEP_Model**.

3. A continuación se llamará al método learn() propio del algoritmo, al que se le pasan el fichero de entrenamiento y los parámetros. Este método learn **debe guardar los patrones obtenidos en la variable "patterns" propia de la clase Model.** Por lo tanto, el desarrollador puede utilizar la representación interna que le interese, sin embargo, al finalizar su método deberá realizar la conversión de sus resultados a la representación del framework.

4. Una vez finalizado el método learn(), el framework obtiene los patrones mediante un casting de la instancia del algoritmo a Model. A continuación, el framework se encarga de obtener las medidas de calidad para entrenamiento, realizar el filtrado de los patrones y guardar resultados de entrenamiento en fichero.

5. Una vez realizadas estas operaciones, el framework llama al método predict() del algoritmo. Este deberá obtener predicciones para los cuatro conjuntos de patrones i.e., el conjunto completo de patrones y cada uno de los filtrados.

6. Con estas predicciones, el framework se encarga de calcular las medidas descriptivas y predictivas en todos los conjuntos de patrones para el conjunto de test, tal y como se ha explicado anteriormente.

7. Opcionalmente, si se marca la opción "Save Model" se llamará a la funcion saveModel() que se encarga de guardar el método en un fichero.

8. En modo batch, este proceso se realiza iterativamente hasta que se procesan todas las particiones. Después, se promedian los resultados y se almacenan en disco.

Herencias

Tal y como se puede observar en el diagrama UML, la clase principal (aquella que contiene los métodos learn() y predict()) del método a desarrollar debe heredar de la clase Model. Asimismo, para poder almacenar en un fichero el modelo es necesario que se hereda también de la clase abstracta Serializable de Java. Esta herencia es necesaria hacerla en todas las clases del método, para que así se puedan almacenar el modelo en un fichero correctamente.

Registro en el framework

Para que un algoritmo pueda ser utilizado por el framework, es necesario que este esté registrado en el fichero "algorithms.xml", este fichero XML contiene toda la información necesaria del algoritmo. Por ejemplo, el algoritmo BCEP tiene la siguiente entrada:

```
<algorithm>
<name>BCEP</name>
<class>bcep.BCEP_Model</class>
<parameter>
<name>Minimum support</name>
<type>real</type>
<domain>
<min>0</min>
<max>1</max>
</domain>
<default>0.01</default>
</parameter>
<parameter>
<name>Minimum GrowthRate</name>
<type>real</type>
<domain>
<min>1</min>
<max>10000</max>
</domain>
<default>100</default>
</parameter>
</algorithm>
```

Las posibles etiquetas son:

1. `algorithm`: Define un algoritmo.

2. `name`: Define el nombre de un método o parámetro. Este nombre será el mostrado en la interfaz y el que se comparará para poder ejecutar el método en el modo línea de comandos. Asimismo el nombre del parámetro será la clave del mismo.

3. `class`: Define el *fully qualified name* del método.

4. `parameter`: Empieza un parámetro del agoritmo.

5. `type`: Tipo de dato del parámetro, los posibles valores son: `real`, `integer` o `nominal`.

6. `domain`: Empieza la definición del dominio del parámetro. Para tipos reales y enteros, se deben definir las entradas `min` y `max` que definen el valor mínimo y máximo del parámetro respectivamente. Para tipos nominales, cada valor nominal de la variable deberá ir entre etiquetas `item`.

7. `default`: Define el valor por defecto, puede ser un valor numérico o bien el índice donde se encuentre el valor por defecto, empezando estos índice en 1.

Ampliación

Todos las clases del framwork pueden ampliarse añadiendo nuevas funcionalidades. Sin embargo, **no se debe modificar lo ya hecho**, pues puede existir dependencias que puede hacer que algún algoritmo falle.

GitHub

El proyecto se encuentra disponible en GitHub y se puede clonar de la siguiente URL: https://github.com/aklxao2/epm-framework.git