Introducción

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 una interfaz gráfica y mediante línea de comandos. 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 únicamente 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.

Una vez el algoritmo termina su ejecución, se almacenarán en la misma carpeta donde se ubica el fichero de entrenamiento cuatro 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 las n mejores reglas según una medida de calidad. Actualmente se obtienen las 3 mejores reglas por confianza. Este fichero se llamará `TRA_QUAC_BEST.txt`.

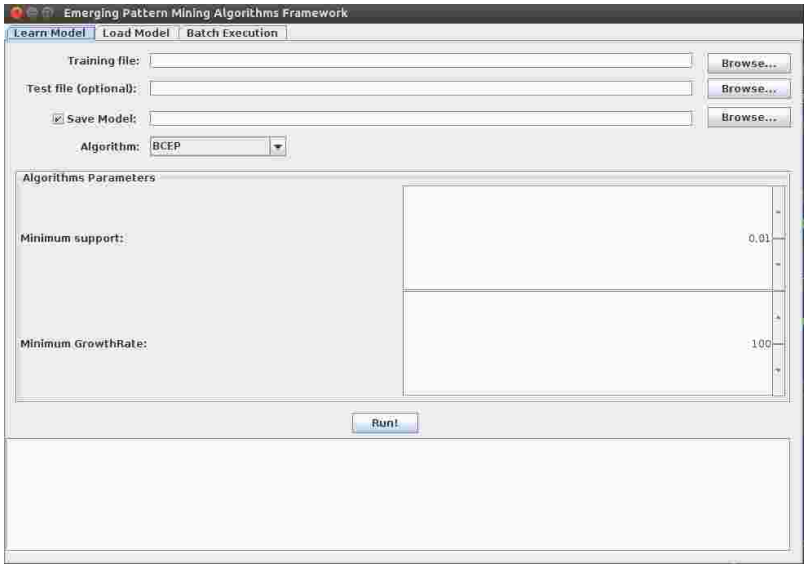

Figura 1: Pantalla inicial de la interfaz de usuario.

4. El último fichero corresponderá a la aplicación del filtro anterior a cada clase. Este fichero se llamará `TRA_QUAC_BESTCLASS.txt`

Por su parte, en la ubicación del fichero de test se almacenarán los resultados obtenidos por estos patrones en el fichero de test, por lo tanto, se almacenarán tres ficheros cuya representación es la misma que para entrenamiento. Estos ficheros se llaman `TST_QUAC_NOFILTER.txt`, `TST_QUAC_BEST.txt` y `TST_QUAC_BESTCLASS.txt`.

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 1.

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. Podemos ejecutar un único algoritmo (el seleccionado), con los argumentos 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, pues esta pestaña permite únicamente 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 de abajo se mostrará o bien las predicciones para la clase, en caso de que no exista atributos de clase o bien una comparativa "clase real vs clase predicha" para cada instancia de test. Este resultado se puede almacenar si se pulsa en el botón "Save to file...".

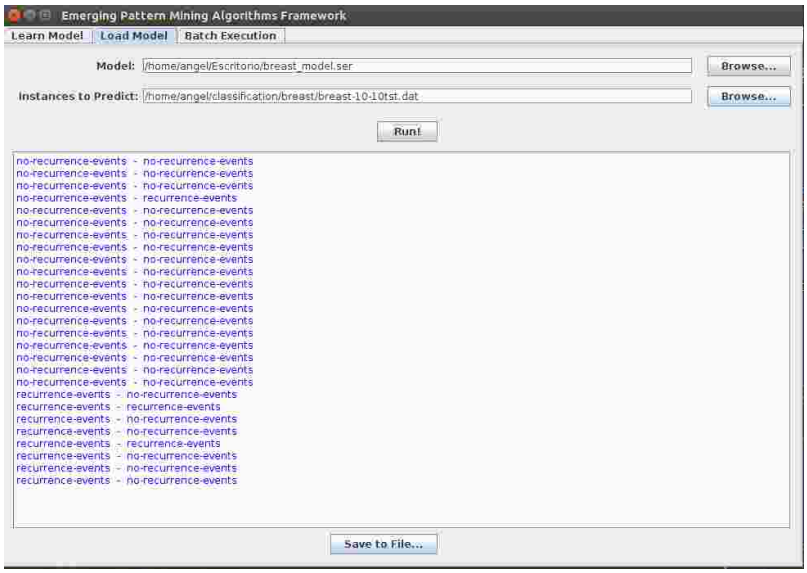

Figura 2: Pestaña *Load model* con los resultados de una predicción.

Pestaña *Batch Execution*

El objetivo de esta pestaña es la aplicación de un algoritmo sobre una serie de conjuntos de datos divididos según un esquema *k-fold cross validation* en donde se devuelve para conjunto de datos, los resultados medios para test en cada *fold*.

Para poder realizar una ejecución *batch* es necesario seguir el siguiente esquema de funcionamiento:

- Se seleccionará una carpeta que contendrá una carpeta por conjunto de datos. Por ejemplo, para ejecutar sobre tres conjuntos de datos diferentes debo tener una carpeta "Experimentación", por ejemplo, que contenga tres carpetas. Es recomendable nombrar a estas carpetas con el nombre del conjunto de datos.

- Dentro de cada carpeta de datos, se encontrarán los ficheros que tendrán las diferentes particiones del esquema *k-fold cross validation*. Estos ficheros deben ser nombrados de forma que finalicen siguiendo el formato **Nº_FOLDS-i-tra.dat** para ficheros de entrenamiento o **Nº_FOLDS-i-tst.dat** para ficheros de test en donde **Nº_FOLDS** representa el valor de k para *k-fold cross validation* e **i** es un número que va desde 1 a Nº_FOLDS donde se indica el número de partición. Por ejemplo, si quiero ejecutar el conjunto de datos *breast* con *5-fold cross validation*, se debe tener un fichero **breast-5-1-tra.dat**, **breast-5-3-tst.dat**, etc. En caso de ser *10-fold cross validation* será **breast-10-1-tra.dat**, **breast-10-3-tst.dat**, etc.

Una vez se tienen todos los ficheros configurados y el algoritmo selecionado y configurado, se debe seleccionar la carpeta raíz (en este caso, "Experimentación") y pulsar "Run!". Este método crea en cada carpeta de conjuntos de datos tres ficheros, que contendrán la media de todas las medidas de calidad **para test** de las k ejecuciones realizas para ese conjunto de datos. Estos ficheros son:

- `QM_Unfiltered.txt`. Contiene las medidas medias de todas las reglas, sin aplicar filtros.

- `QM_FilteredALL.txt`. Contiene las medidas medias de las n mejores reglas de cada ejecución según una medida de calidad. Por defecto, los tres mejores patrones por confianza.

- `QM_FilteredBYCLASS.txt`. Igual al anterior, pero con los n mejores de cada clase.

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.

Esto se consigue gracias a la clase **Model** de la cual deben heredar todos los métodos implementados para este framework. En la Figura 3 se puede ver un diagrama UML de implementación del método BCEP. Cada uno de los diferentes algoritmos se encontrará en un paquete diferente, en donde se encontrarán todas aquellas clases auxiliares del mismo.

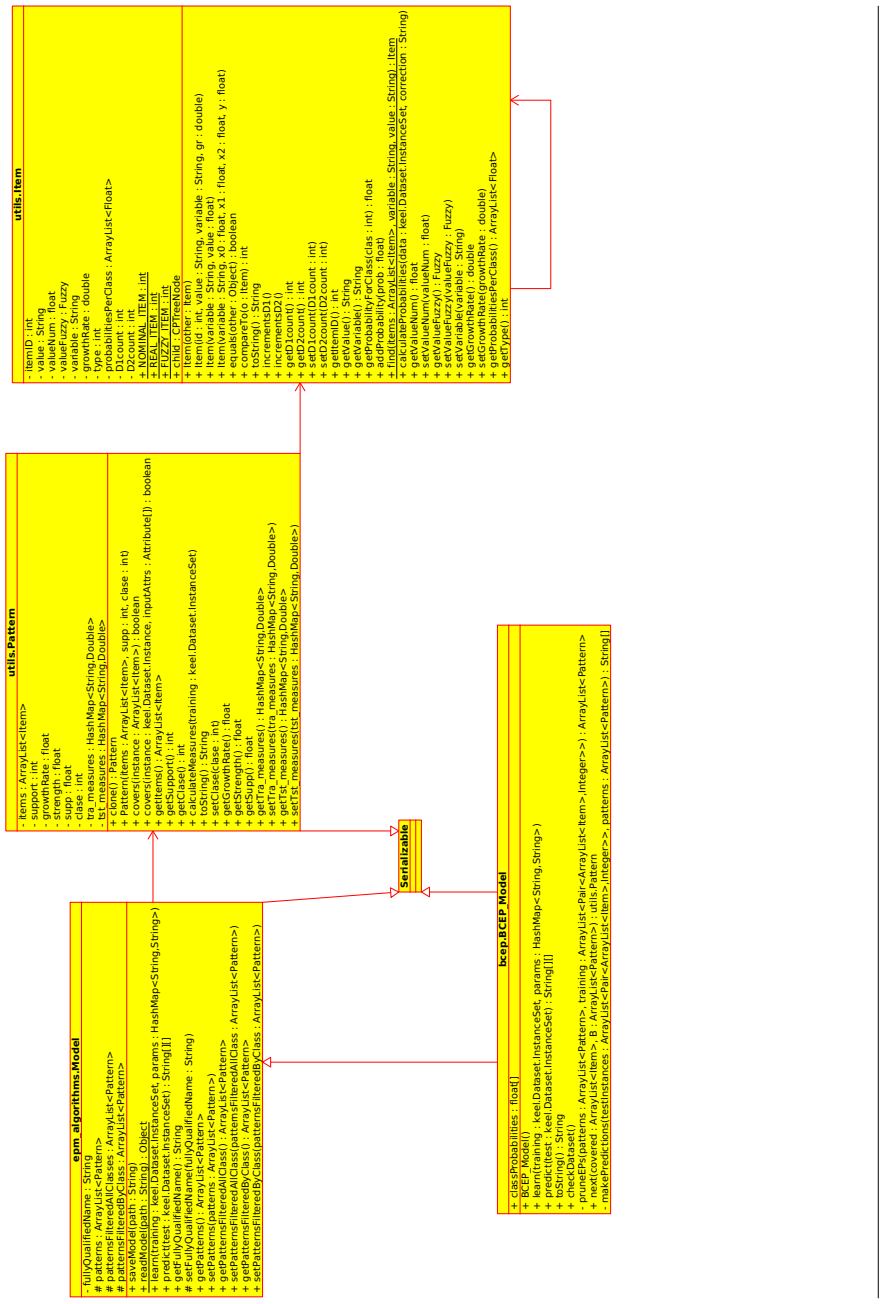

Figura 3: Diagrama UML de implementación del algoritmo BCEP. Se han ignorado las dependencias a las clases de KEEL.

Funcionamiento interno del framework

Con este diagrama UML en mente, las llamadas a los algortimos de aprendizaje se realizan de la siguiente manera:

1. Se leen los ficheros de entrenamiento y prueba. 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".

2. Se crea una nueva instancia de manera dinámica a partir del valor que se encuentra en la variable **fullyQualifiedName**. 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: **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 interesa, 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 en fichero.

5. Una vez realizadas estas operaciones, el framework llama al método predict() del algoritmo. Este deberá obtener predicciones para los tres conjuntos de patrones, es decir, todos los patrones, los n mejores patrones en una medida de calidad y los n mejores mejores patrones en una medida de calidad para cada clase.

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.

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 utilizada 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

Tanto la clase Pattern como Item pueden ser ampliadas, añadiendo variables o métodos adicionales para ajustarse a las necesidades de los desarrolladores.

GitHub

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