TÍTULO DE GRADO EN
INGENIERÍA INFORMÁTICA

Desarrollo de Sistemas Inteligentes

Documentación de la lectura del fichero y primeras reglas

CURSO 2019 / 2020
CONVOCATORIA DE ENERO

Eduardo Salmerón Castaño — eduardo.salmeronc@um.es
Victor García Puche — victor.garciap@um.es
Nicolás Fuentes Turpín — nicolas.fuentest@um.es

Índice general

1 - Introducción

La segunda fase del proyecto consiste en desarrollar un Sistema Inteligente capaz de leer y sacar todos los datos necesarios de un fichero *log* de un ECG mediante un motor de reglas.

La tercera y última fase del proyecto (parte obligatoria) consiste en desarrollar el resto de reglas, en este caso de diagnosis, para poder especificar qué enfermedades puede padecer el paciente dependiendo de su ECG. Finalmente, una vez analizado todo el electrocardiograma, crearemos un fichero de salida y mostraremos las posibles enfermedades que el paciente puede padecer.

2 - Interpretar el fichero

El primer paso que hemos tomado para realizar esta fase, ha sido crear un fichero java que lea el fichero log, y saque todos los datos de este. El fichero en cuestión se llama "Parser.java". Además, mientras se leen los datos, se van insertando como hechos en la base de hechos.

Este proceso de parseo del fichero consta de varias partes:

2.1 - Creación e inicio de la "*KieSession*"

Para empezar el parseo, antes de nada debemos crear una base de hechos e inicializarla con el nombre de la sesión ("ksession-rules").

2.2 - Parseo del ECG (ciclos y pulsaciones por minuto)

A continuación, empezamos leyendo el fichero y leyendo las primeras lineas (La primera linea de todas la, obviamos ya que contiene el "*ECG pattern*" del fichero). Estas lineas se corresponden con el número de ciclos y el número de pulsaciones por minuto del ECG. para obtener ambos valores, utilizamos una expresión regular para eliminar todo lo que no sean dígitos ("\D+"):

```
// Obtenemos el numero de ciclos
line = reader.readLine();
Cycles cycles = new Cycles(Integer.valueOf(line.replaceAll("\\D+", "")));
// Obtenemos el ritmo cardiaco
line = reader.readLine();
BPM bpm = new BPM(Integer.valueOf(line.replaceAll("\\D+", "")));
```

Ahora insertamos estos valores en la BH:

```
kSession.insert(cycles);
kSession.insert(bpm);
```

2.3 - Parseo del ECG (ondas)

En esta parte leemos cada onda y creamos una instancia de esta, dependiendo de cuál sea. Esto lo hacemos de forma dinámica, es decir, tenemos una variable de tipo *Wave* y dependiendo de la letra que leamos, creamos la onda de un tipo u otro. Esto podemos hacerlo ya que todos los tipos de onda heredan de *Wave*.

Para calcular en qué ciclo está cada onda, tenemos un contador inicializado a cero, que se incrementa cada vez que el parser encuentra una onda P (la primera onda de un ciclo).

Una vez tenemos la onda creada con todos los datos, ya podemos insertarla también en la BH.

```
char letter = line.charAt(0);
if (letter == 'P')
    actualCycle++;
String[] dataArray = line.replaceAll("[A-Z()]", "").split(",");

Wave wave;
switch (letter) {
case 'P':
    wave = new P(Integer.valueOf(dataArray[0]), Integer.valueOf(dataArray[1]),
            Float.valueOf(dataArray[2]), actualCycle);
    break;
case 'Q':
    wave = new Q(Integer.valueOf(dataArray[0]), Integer.valueOf(dataArray[1]),
            Float.valueOf(dataArray[2]), actualCycle);
    break;
case 'R':
    wave = new R(Integer.valueOf(dataArray[0]), Integer.valueOf(dataArray[1]),
            Float.valueOf(dataArray[2]), actualCycle);
    break;
case 'S':
    wave = new S(Integer.valueOf(dataArray[0]), Integer.valueOf(dataArray[1]),
            Float.valueOf(dataArray[2]), actualCycle);
    break;
case 'T':
    wave = new T(Integer.valueOf(dataArray[0]), Integer.valueOf(dataArray[1]),
            Float.valueOf(dataArray[2]), actualCycle);
    break;
default:
    wave = null;
    break;
}
```

2.4 - Lanzar las reglas

Llegamos a la última parte, en la que lo único que debemos hacer es lanzar todas las reglas con "*fireAllRules*" para que el motor de reglas empiece a lanzar reglas con los hechos que le hemos insertado. Las reglas que hemos creado se explicarán a continuación.

3 - Creación de reglas

En este apartado vamos a explicar las reglas que hemos creado y qué función cumple cada una. Además, En cada regla se notifica por consola que se ha ejecutado la regla. El fichero de reglas se llama "*DomainRules*".

3.1 - Creación del complejo QRS

En esta regla se crean los complejos QRS, es decir, tomamos las ondas Q, R y S pertenecientes a

```
rule "QRS Complex Creation"
    when
        $q: Q($c : cycle)
        $r: R(cycle == $c)
        $s: S(cycle == $c)
    then
        QRSComplex qrs = new QRSComplex($q.getStart(), $s.getEnd(), $c);
        insert(qrs);
        System.out.println("Se ha creado el complejo QRS del ciclo numero " + $c);
end
```

un mismo ciclo, creamos el complejo QRS y lo insertamos como hecho.

3.2 - Creación del intervalo QT

```
rule "QT Intervale Creation"
    when
        $q: Q($c : cycle)
        $t: T(cycle == $c)
    then
        QTIntervale qt = new QTIntervale($q.getStart(), $t.getEnd(), $c);
        insert(qt);
        System.out.println("Se ha creado el intervalo QT del ciclo numero " + $c);
end
```

Aquí se crean los intervalos QT de cada ciclo de forma similar a la anterior, y se inserta en la BH.

3.3 - Creación del segmento ST

```
rule "ST Segment Creation"
    when
        $s: S($c : cycle)
        $t: T(cycle == $c)
    then
        STSegment st = new STSegment($s.getStart(), $t.getEnd(), $c);
        insert(st);
        System.out.println("Se ha creado el segmento ST del ciclo numero " + $c);
end
```

Por último, tenemos la regla de creación de segmentos ST. De forma similar a las anteriores, crea el segmento con las ondas S y T de cada ciclo y lo inserta como hecho.

4 - NormalValues

Antes de explicar la creación e identificación de cada una de nuestras reglas, debemos apuntar que, para poder comparar los datos de nuestro ECG con los datos de uno "normal", hemos creado una clase denominada "*NormalValues*". Esta clase actúa única y exclusivamente de contenedor y se le añade a nuestra *kSession* al inicio de esta:

```
public class NormalValues {

    private final int minBPM = 60;
    private final int maxBPM = 100;
    private final float maxPeakWaveT = 0.55f;
    private final float minPeakWaveT = 0.1f;
    private final int maxQTDuration = 420;
    private final int minQRSDuration = 100;

    public int getMaxBPM() {
        return maxBPM;
    }
    public int getMinBPM() {
        return minBPM;
    }
    public float getMaxPeakWaveT() {
        return maxPeakWaveT;
    }
    public float getMinPeakWaveT() {
        return minPeakWaveT;
    }
    public int getMaxQTDuration() {
        return maxQTDuration;
    }
    public int getMinQRSDuration() {
        return minQRSDuration;
    }

}
```

5 - Reglas de diagnosis

En este apartado procederemos a la explicación de las reglas que son capaces de diagnosticar nuestro ECG. Como cada enfermedad muestra unos síntomas distintos (o no) hemos considerado crear una regla por cada potencial enfermedad a analizar.

Explicaremos cada regla por separado (estas reglas se encuentran en el fichero "*IssueRules.drl*"):

5.1 - Hipopotasemia e Isquemia Coronaria:

Esta enfermedad se caracteriza porque las ondas T de un ECG adquieren valores negativos y sus ondas U valores altos y positivos. Sus síntomas son similares a la Isquemia Coronaria, ya que nuestro sistema no analiza las ondas U de un ECG, por lo que podemos explicar ambas en un mismo apartado

```
rule "Hypokalemia"
    when
        not (exists (Hypokalemia()))
        $nv : NormalValues()
        $t: T(peak < $nv.minPeakWaveT, $c : cycle)
    then
        insert(new Hypokalemia("Tienes indicios de hipopotasemia en el ciclo numero ", $c));
end


rule "Coronarian Ischemia"
    when
        not (exists (CoronarianIsq()))
        $nv : NormalValues()
        $t: T(peak < $nv.minPeakWaveT, $c : cycle)
    then
        insert(new CoronarianIsq("Tienes indicios de isquemia coronaria en el ciclo numero ", $c));
end
```

5.2 - Hipocalcemia:

Esta regla permite la identificación de Hipocalcemia. Esta enfermedad se caracteriza porque los intervalos QT tardan demasiado en terminar su recorrido.

```
rule "Hypokalcemia"
    when
        not (exists (Hypokalcemia()))
        $nv : NormalValues()
        $qt : QTIntervale(duration > $nv.maxQTDuration, $c : cycle)
    then
        insert(new Hypokalcemia("Tienes indicios de hipocalcemia en el ciclo numero ", $c));
end
```

5.3 - Infarto de miocardio agudo:

Esta enfermedad denominada IAM se identifica porque su ECG presenta valores altos y positivos en sus ondas T.

```
rule "AMI"
    when
        not (exists (AMI()))
        $nv : NormalValues()
        $t : T(peak > $nv.maxPeakWaveT, $c : cycle)
    then
        insert(new AMI("Tienes indicios de paceder un infarto de miocardio agudo en el ciclo numero ", $c));
end
```

5.4 - Bradicardia:

```
rule "Bradycardia"
    when
        not (exists (Bradycardia()))
        $nv : NormalValues()
        $bpm : BPM($b : bpm , $b < $nv.minBPM)
    then
        insert(new Bradycardia("Tienes indicios de bradicardia, ya que la media de tus pulsaciones por minuto es de  ", $b));
end
```

La bradicardia se caracteriza porque el paciente presenta unas pulsaciones por minuto inferiores a lo normal.

5.5 - Taquicardia:

```
rule "Tachycardia"
    when
        not (exists (Tachycardia()))
        $nv : NormalValues()
        $bpm : BPM($b: bpm, $b > $nv.maxBPM)
    then
        insert(new Tachycardia("Tienes indicios de taquicardia, ya que la media de tus pulsaciones por minuto es de  ", $b));
end
```

Esta enfermedad es análoga a la anterior. Los pacientes que la padecen se caracterizan por presentar unas pulsaciones por minuto por encima a la media.

5.6 - Contracción prematura del Ventrículo:

Esta enfermedad, denominada PVC, presenta complejos QRS de muy corta duración.

```
rule "Premature Ventricular Contraction"
    when
        not (exists (PVC()))
        $nv : NormalValues()
        $qrs : QRSComplex(duration < $nv.minQRSDuration, $c : cycle)
    then
        insert(new PVC("Tienes indicios de una contraccion prematura del ventriculo en el ciclo numero ", $c));
end
```

5.7 - Aleteo Auricular:

Por último, identificamos esta enfermedad debido a que sus ondas T y P son anormales y estas parecen a priori que son una sola (el final de la onda T se une con el inicio de la P).

```
rule "Atrial Flutter"
    when
        not (exists (AtrialFl()))
        $t : T($c : cycle)
        $p : P(cycle == $c+1, $t.end == $p.start)
    then
        insert(new AtrialFl("Tienes indicios de un aleteo auricular en el ciclo numero ", $c));
end
```

6 - Regla de impresión

Hemos conseguido crear una regla genérica para la impresión del diagnóstico del ECG. Lo hemos conseguido gracias a la creación de una jerarquía de enfermedades que separa estas dependiendo de sus síntomas. Así, tenemos una clase denominada *"Issue"* que solo necesita un parámetro denominado *message*. Este parámetro será el usado para la impresión del mensaje en nuestro fichero de salida:

```
public abstract class Issue {

    private String message;

    public Issue (String message) {
        this.message = message;
    }

    public String getMessage() {
        return message + data();
    }

    protected abstract String data();

    public void setMessage(String message) {
        this.message = message;
    }

}
```

Es destacable el hecho de que, para imprimir el mensaje, es necesario implementar una función abstracta denominada data, la cual permitirá a nuestras diversas clases de enfermedades decidir que dato característico se imprime junto con su mensaje:

```
@Override
protected String data() {
    return "(" + cycle + ")";
}
```

En la imagen anterior mostramos como nuestra clase denominada *"CycleIssue"* implementa la función en cuestión. (mostrando así el ciclo donde se encontró el síntoma para la activación de su regla)

```
rule "Print Issue"
    when
        $w : PrintWriter()
        $i : Issue()
    then
        $w.println($i.getMessage());
        System.out.println($i.getMessage());
end
```

Nuestra regla en cuestión lo único que busca es si existe alguna enfermedad añadida a la base de hechos, recupera la instancia de "*PrintWriter*" encargada de la impresión y añade a nuestro fichero de salida el mensaje de la enfermedad en sí.

Por ultimo, es necesario explicar que todas nuestras reglas de diagnosis añaden una instancia de la enfermedad a la base de hechos, para así impedir que esa regla se dispare más de una vez.