Doble Grado en
Ingeniería Informática y Matemáticas

INGENIERÍA DEL CONOCIMIENTO
(E. Computación y Sistemas Inteligentes)

PRÁCTICA-1: Asesorar a un alumno de Ingeniería Informática para elegir Rama

UNIVERSIDAD
DE GRANADA

Carlos Santiago Sánchez Muñoz

Grupo de prácticas 1 - Viernes

Email: carlossamu7@correo.ugr.es

7 de abril de 2020

Índice

1. Adquisición del Conocimiento **2**

2. Modularidad del sistema **3**

3. Conversión de la calificación media **5**

4. Finalizar la ejecución y obtener consejo **6**

5. Permitir respuestas de información parcial **6**

1. Adquisición del Conocimiento

La práctica consiste en un sistema experto que asesore a un estudiante de Ingeniería Informática tal y como lo haría un compañero. El planteamiento que he llevado para la implementación de esta práctica ha sido utilizando el árbol de decisión que tuve que construir para la entrega de teoría de Técnicas de Adquisición del Conocimiento. La mayoría de la metodología de trabajo y los Consejos que se toman son en base a ese árbol a excepción de aquellas nuevas funcionalidades que ahora tenemos y en ese momento no se habían contemplado.

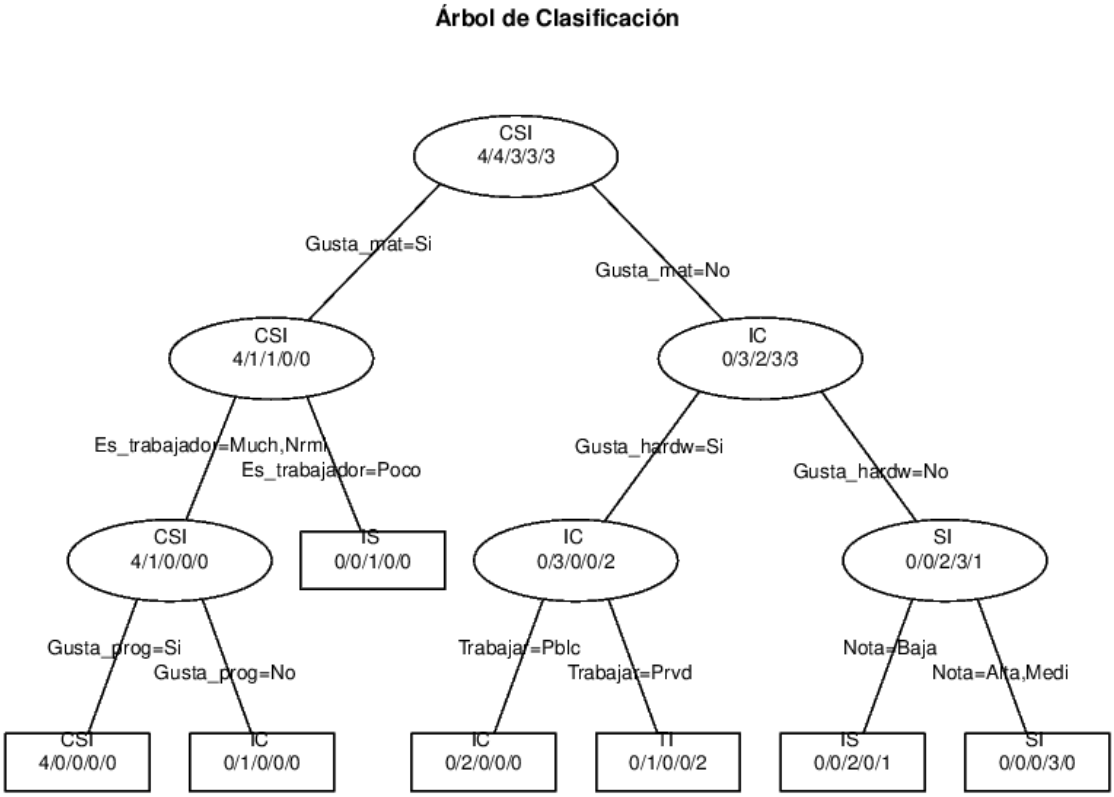

Imagen 1: Árbol de decisión

Mediante este árbol las reglas de decisión asociadas son:

```
IF Gusta_mat=Si AND Es_trabajador=Poco THEN IS
IF Gusta_mat=Si AND (Es_trabajador=Normal OR Es_trabajador=Mucho) AND
    Gusta_prog=Si THEN CSI
IF Gusta_mat=Si AND (Es_trabajador=Normal OR Es_trabajador=Mucho) AND
    Gusta_prog=No THEN IC
IF Gusta_mat=No AND Gusta_hardware=Si AND Trabajar=Publica THEN IC
IF Gusta_mat=No AND Gusta_hardware=Si AND (Trabajar=Privada OR
    Trabajar=Docencia) THEN TI
IF Gusta_mat=No AND Gusta_hardware=No AND Nota=Baja THEN IS
IF Gusta_mat=No AND Gusta_hardware=No AND (Nota=Alta OR Nota=Media) THEN SI
```

2. Modularidad del sistema

Es momento de comenzar con la implementación del sistema. Nuestro objetivo es programar un sistema experto con una buena modularización, funcionalidades separadas y así será más fácil añadir nuevas funcionalidades y depurar posibles fallos.

En primer lugar se da un **mensaje de bienvenida** informando de las Ramas posibles y de cómo funciona el sistema.

```
(defrule Da_bienvenida
  (declare (salience 10))
=>
  (printout t "Bienvenido al sistema de asesoramiento de ramas del
  Grado en I. Informatica. Las ramas son:" crlf)
  (printout t "- Computacion y Sistemas Inteligentes." crlf)
  (printout t "- Ingenieria del Software." crlf)
  (printout t "- Ingenieria de Computadores." crlf)
  (printout t "- Sistemas de Informacion." crlf)
  (printout t "- Tecnologias de la Informacion." crlf)
  (printout t "En cualquier momento puede responder Fin para obtener
  el consejo sin responder el resto de preguntas." crlf)
)
```

Las posibles **preguntas** que el usuario recibirá son:

- ¿Te gustan las matemáticas? (Si/No)
- ¿Eres trabajador? (Mucho/Normal/Poco)
- ¿Te gusta programar? (Si/No/No se)
- ¿Te gusta el hardware? (Si/No/No se)
- ¿En que te gustaría trabajar? (Pública/Privada/Docencia/No se)
- Introduce tu calificación media de expediente.

Con estas preguntas iremos creando los hechos: `(Gusta_mat ?r)`, `(Es_trabajador ?r)`, `(Gusta_prog ?r)`, `(Gusta_hardw ?r)`, `(Trabajar ?r)` y `(Calificacion_media ?r)`. Esta será discretizada más adelante en el hecho `(Nota ?r)`. Un ejemplo de inserción de un hecho en el sistema es:

```
;;; Si le gustan las matematicas ;;;

(defrule Gusta_mat
  (declare (salience 9))
=>
  (printout t "Te gustan las matematicas? (Si/No)" crlf)
  (assert (Gusta_mat (read)))
)
```

Lo primero que tenemos que observar es que el usuario puede introducir un valor no válido o simplemente equivocarse al escribir. El sistema debe ser capaz de resolver estos fallos. Por eso todas las reglas tienen una función **check** asociada que comprueba que lo

introducido es válido. En este caso sería:

```
(defrule Gusta_mat_check
  (declare (salience 100))
  ?f <- (Gusta_mat ?r)
  (test (and (neq ?r Si) (neq ?r No) (neq ?r Fin)))
=>
  (printout t "Respuesta no valida. Te gustan las matematicas? (Si/No)" crlf)
  (retract ?f)
  (assert (Gusta_mat (read)))
)
```

El resto de las preguntas son análogas por lo que no voy a añadir su código. La única diferente es la de la calificación media que se tratará más adelante.

La siguiente cuestión a abordar es cómo toma el sistema las decisiones en función de estos hechos para dar un **consejo**. Los consejos los representaremos por los hechos `(Consejo ?rama ?texto ?experto)` que es lo que nos han pedido. Lo primero es implementar una regla que imprima por pantalla estos consejos. En dicho método también compruebo que la rama aconsejada exista:

```
;;; Metodo para imprimir un consejo ;;;

(defrule imprime_consejo
  (Rama ?rama)          ; sirve para asegurarse de que la rama existe
  (Consejo ?rama ?texto ?experto)
=>
  (printout t ?experto " te aconseja la Rama de " ?rama "." crlf)
  (printout t "Motivo: " ?texto "." crlf)
)
```

A continuación expongo una de las diversas reglas que hay para dar un consejo a un alumno que le gusten las matemáticas, que sea trabajador y le guste programar:

```
;;; Rama de Computacion y Sistemas Inteligentes ;;;

(defrule Rama_CSI
  (declare (salience 10))
  (Gusta_mat Si)
  (or (Es_trabajador Mucho) (Es_trabajador Normal))
  (Gusta_prog Si)
=>
  (assert (Consejo Computacion_y_Sistemas_Inteligentes "te gustan las
  matematicas y programar y ademas eres trabajador" "CLISP"))
)
```

Para el resto de las ramas es similar y omito poner el código para no alargar la memoria.

3. Conversión de la calificación media

Antes de hablar de la conversión es fundamental comentar el `check` de este hecho. Es fundamental que lo que introduce sea un número, ya sea flotante o entero, pues si voy a operar con < y > entre otros y es una cadena de texto el programa aborta. Esto lo podemos hacer con `numberp` y lo haremos con un `while`.

A parte de esta comprobación exigiremos que el número introducido esté en el intervalo `[5,10]` ya que en la media cuentan las asignaturas aprobadas hasta la fecha.

```
;;; La nota media que ha obtenido en las asignaturas que ha cursado ;;;

(defrule Nota
  (declare (salience 9))
  (Gusta_mat No)
  (Gusta_hardw No)
=>
  (printout t "Introduce tu calificacion media de expediente" crlf)
  (bind ?num (read))
  (while (and (not (numberp ?num)) (not (eq ?num Fin)))
    (printout t "Respuesta incorrecta (debe de ser flotante o entero).
    Introduce tu calificacion media de expediente" crlf)
    (bind ?num (read)))
  (if (eq ?num Fin) then (assert (Consejo Sistemas_de_Informacion "no te
  gustan las matematicas ni el hardware" "CLISP"))
  else (assert (Calificacion_media ?num)))
)

(defrule Nota_check_interval
  (declare (salience 99))
  ?f <- (Calificacion_media ?r)
  (test (or (> ?r 10) (< ?r 5)))
=>
  (printout t "Respuesta incorrecta (fuera de [5,10]).
  Introduce tu calificacion media de expediente" crlf)
  (retract ?f)
  (bind ?num (read))
  (while (and (not (numberp ?num)) (not (eq ?num Fin)))
    (printout t "Respuesta incorrecta (debe de ser flotante o entero).
    Introduce tu calificacion media de expediente" crlf)
    (bind ?num (read)))
  (if (eq ?num Fin) then (assert (Consejo Sistemas_de_Informacion "no
  te gustan las matematicas ni el hardware" "CLISP"))
  else (assert (Calificacion_media ?num)))
```

La conversión es relativamente sencilla, si está en `[8,10]` es `Alta`, para `[6.5,8]` es `Media` y para `[5,6.5]` es `Baja`.

```
;;; Convertir la Calificacion_media a Nota ;;;

(defrule Nota_conversion_alta
  (declare (salience 20))
  (Calificacion_media ?r)
  (test (>= ?r 8))
=>
  (assert (Nota Alta))
```

```
)

(defrule Nota_conversion_media
  (declare (salience 20))
  (Calificacion_media ?r)
  (test (and (>= ?r 6.5) (< ?r 8)))
=>
  (assert (Nota Media))
)

(defrule Nota_conversion_baja
  (declare (salience 20))
  (Calificacion_media ?r)
  (test (and (>= ?r 5) (< ?r 6.5)))
=>
  (assert (Nota Baja))
)
```

4. Finalizar la ejecución y obtener consejo

En cualquier momento se puede responder a absolutamente cualquier pregunta con la respuesta `Fin` y el sistema nos proporciona el consejo. Hay que incluir esta peculiaridad en los `check` y posteriormente implementar reglas para dar el consejo en ese momento. El consejo se da conforme a la etiqueta del nodo del árbol de decisión en el que se encuentre el sistema en ese momento.

```
(defrule Fin_1
  (declare (salience 100))
  (Gusta_mat Fin)
=>
  (assert (Consejo Computacion_y_Sistemas_Inteligentes "no me has dado
  ninguna informacion, esta es la especialidad mas escogida" "CLISP"))
)
```

Los demás casos son análogos a este.

5. Permitir respuestas de información parcial

Se da la posibilidad de responder `No se` en algunas preguntas. La que más sentido tiene es la pregunta de dónde quiere trabajar el usuario pues una gran parte de los alumnos estarán dudosos. La mayoría de los alumnos de Ingeniería Informática después de casi dos años en la carrera saben si les gusta programar o las matemáticas o el hardware. A pesar de esto he dado la posibilidad de responder `No se` a dos cuestiones más y la forma de resolverlo es muy diferente.

- Empiezo con el caso de en qué trabajar. Almaceno, en este caso con `(explode$ (readline))` la información pues pueden ser dos palabras. Lo resuelvo insertando dos consejos que se imprimirán por pantalla.

```
;;; No sabe a que dedicarse ;;;

(defrule Rama_TI_No_se
  (declare (salience 10))
  (Gusta_mat No)
  (Gusta_hardw Si)
  (Trabajar No se)
=>
  (assert (Consejo Tecnologias_de_la_Informacion "no sabes donde
  trabajar pero te gusta el hardware y no las mates" "CLISP"))
  (assert (Consejo Ingenieria_de_Computadores "te gusta el hardware
  por lo que esta es una buena opcion si descubres que te gustaria
  trabajar en la Empresa Publica" "CLISP"))
)
```

- El segundo caso es que no sabe si le gusta programar. Lo resuelvo insertando dos consejos que se imprimirán por pantalla.

```
;;; No sabe si le gusta programar ;;;

(defrule Rama_CSI_No_se
  (declare (salience 10))
  (Gusta_mat Si)
  (or (Es_trabajador Mucho) (Es_trabajador Normal))
  (Gusta_prog No se)
=>
  (assert (Consejo Computacion_y_Sistemas_Inteligentes "aunque no sabes
  si te gusta programar te gustan las matematicas y eres trabajador" "CLISP"))
  (assert (Consejo Ingenieria_de_Computadores "si descubres que no te
  gusta programar, esta puede ser una buena eleccion" "CLISP"))
)
```

- En el último caso vamos a tomar una idea original pues si el usuario no sabe si le gusta el hardware le preguntaremos si le gustaron determinadas asignaturas de la carrera y en función de eso rellenaremos el hecho `Gusta_hardw`.

```
;;; No sabe si le gusta el hardware ;;;

(defrule Gusta_hardw_No_se
  (declare (salience 10))
  ?f <- (Gusta_hardw No se)
=>
  (printout t "Te gustaron asignaturas como FFT, TOC y EC? (Si/No)" crlf)
  (retract ?f)
  (bind ?a (read))
  (while (and (neq ?a Si) (neq ?a No) (neq ?a Fin))
    (printout t "Respuesta incorrecta. Te gustaron asignaturas como
    FFT, TOC y EC? (Si/No)" crlf)
    (bind ?a (read)))
  (assert (Gusta_hardw ?a))
)
```