Genetic Tuning on Fuzzy Systems Based on the Linguistic 2-Tuples Representation

Inteligencia computacional

Profesor: Francisco Alfredo Márquez Hernández y
Antonio Peregrín Rubio.
Alumnos: Ana Godoy Pérez y José Manuel Betanzos

Máster en Ingeniería Informática
Escuela Politécnica Superior de Ingeniería
Universidad de Huelva

Índice

1. Creación de un Software para el diseño y lanzamiento de un Controlador Difuso de Tipo Mamdani **2**

1.1. Base de conocimiento 3

1.1.1. Base de Datos 3

1.1.2. Base de Reglas 3

1.2. Interfaz de fuzzificación 4

1.3. Sistema de inferencia 4

1.4. Interfaz de defuzzificación 4

1.5. Read 5

1.6. Files 5

1.7. FuzzySystem 5

2. Ejecución del software **6**

Objetivos

El objetivo de esta práctica es la creación de un software que permita el diseño y lanzamiento de un controlador difuso MISO (Multiple input Simple Output) sobre un conjunto de valores de variables, que además permita un aprendizaje de las etiquetas lingüísticas (tuning) sobre un fichero de entrenamiento.

1. Creación de un Software para el diseño y lanzamiento de un Controlador Difuso de Tipo Mamdani

En esta primera parte se pretender crear un software que permita el diseño y lanzamiento de un controlador difuso de tipo Mandani. Este software permitirá definir el número de variables de entrada, la variable de salida, el universo de discurso, el número de etiquetas lingüísticas para cada una de las variables, así como el operador de implicación, conjunción y el método de inferencia. Además permitirá introducir una base de reglas en un formato predefinido. Para simplificar esta primera parte se utilizarán etiquetas triangulares. Además, como operadores de conjunción e implicación la t-norma del mínimo y como método de defuzzificación un Modo B.

A continuación se van a explicar los paquetes de los que está compuesto este software, que se corresponden con los componentes forman un sistema difuso de este tipo: base de conocimiento, interfaz de fuzzificación, sistema de inferencia e interfaz de defuzzificación. Asímismo cuenta con otros paquetes auxiliares que ayudarán a cargar los datos necesarios en el sistema. Todos ellos cuenta con su JavaDoc y el código está comentado para facilitar la compresión del mismo.

También es necesario tener en consideración que la relación entre las partes del sistema será la mostrada en la siguiente imagen **??** :

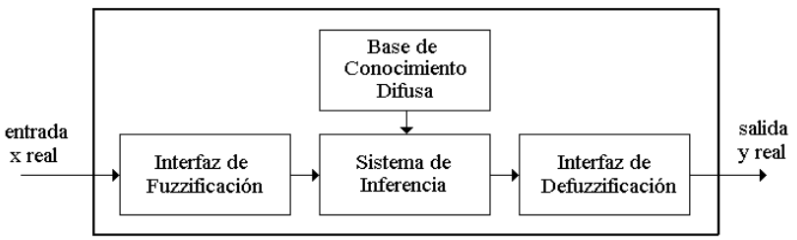

Figura 1: Flujo de un sistema difuso de tipo Mamdani.

1.1. Base de conocimiento

La base de conocimiento almacena el conocimiento particular del problema a resolver. Consta de dos elementos: la base de reglas y la base de conocimiento. Tanto la estructura de datos de la BC como la información que la formará debe ser especificada antes de que el SBRD comience a funcionar.

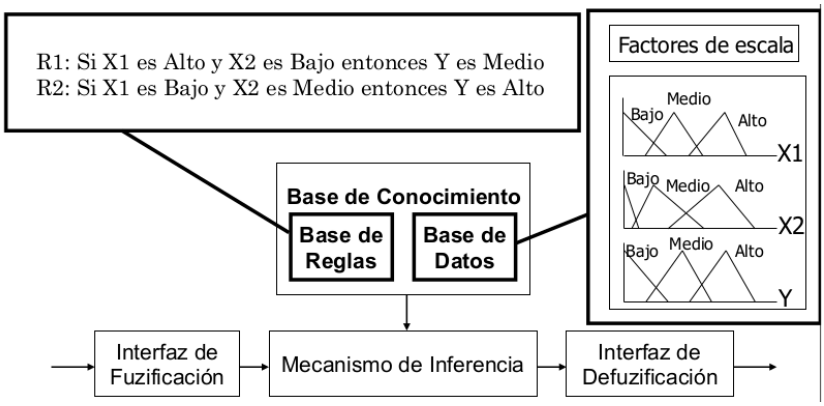

Figura 2: Base de conocimiento del sistema difuso.

1.1.1. Base de Datos

La base de datos contiene la definición de los conjuntos difusos asociados a los términos lingüísticos empleados en las reglas de la BR.
Para su definición se ha optado por una tabla hash en la que cada valor queda definido por el triángulo que representa a una partición difusa del conjunto difuso de las variables. Este "triángulo" no sólo almacena las coordenadas de la etiqueta si no que también almacena a qué variable del conjunto difuso pertenece. Como key de la tabla hash se empleará un número en el intervalo [1,num_etiquetas].
Los datos necesarios para dar contenido a esta estructura serán tomados del fichero con extensión *.pwm*

1.1.2. Base de Reglas

La base de reglas está formada por un conjunto de reglas lingüísticas de tipo "Si - entonces". En este caso las reglas contenidas en el fichero de extensión *.wm* vienen definidas por las coordenadas de las etiquetas a las que hacen referencia.
Cada regla será almacenada como el conjunto de referencias a las etiquetas que la forman.

1.2. Interfaz de fuzzificación

Este componente es uno de los que permite al SBRD de tipo Mamdani manejar entradas y salidas reales. Su tarea es la de establecer una aplicación que haga corresponder un conjunto difuso, definido en el universo de discurso de la entrada en cuestión, a cada valor preciso del espacio de entrada.
Para almacenar esta correspondencia se ha optado de nuevo por una tabla hash que tendrá por clave la variable de entrada que se está considerando, y como valor las referencias a las particiones difusas (etiquetas) a las que pertenece.

1.3. Sistema de inferencia

El Sistema de Inferencia es el componente encargado de llevar a cabo el proceso de inferencia difusa. Para ello el Sistema de Inferencia deberá realizar las siguientes tareas sobre cada regla de la Base de Reglas:

- Determinar $\mu_A(x_0)$, mediante el Operador de Conjunción (en esta práctica se empleará el mínimo).El resultado, denominado grado de emparejamiento (en adelante lo notaremos por h) de dichas entradas con la regla que se está evaluando, representa una medida de "coincidencia" de los valores que toman las variables de entrada con los valores lingüísticos que describen el antecedente de esa regla. Estos valores se irán almacenando en el array *matching* para su posterior uso.

- Aplicar del Operador de Implicación, F (en esta práctica se empleará el mínimo). En el caso que nos ocupa, debido a que sólo existirá un consecuente, F = min, es decir, directamente se guardará el punto de máximo valor de la variable de salida del consecuente de la regla que se está evaluando en el vector *pmv*.

1.4. Interfaz de defuzzificación

Convierte la salida del sistema de inferencia (lo inferido por cada regla) en un único valor real final. En este caso el método que se empleará para calcular este valor único será el Modo B, que consiste en convertir la aportación de cada regla en un número, que en esta práctica se ha hecho mediante el PMV (punto de máximo valor) y calcular un único valor final como salida empleando la siguiente fórmula:

$$y_0 = \frac{\sum_i h_i \cdot PMV_i}{\sum_i h_i}$$

1.5. Read

Este paquete cuenta con las clases necesarias tanto para leer la información relativa al problema y almacenarla en las estructuras correspondientes, como para escribir las salidas en el formato deseado.
A continuación se explica la finalidad de cada una de ellas:

- **ReadPWM:** lee un fichero de extensión .pwm que contiene la información relativa a la base de datos y la almacena.
- **ReadRB:** lee un fichero de extensión .wm que contiene las reglas que definen al sistema difuso. Este fichero también contiene la salida por defecto y el error cuadrático medio de los fichero tanto de training como de test.
- **ReadExamples:** lee un fichero de training o test y guarda las entradas y salidas en dos estructuras respectivamente. Además esta clase permite calcular el error cuadrático medio del fichero leído, así como calcular las salidas para un conjunto de valores de entradas. También permite escribir las salidas obtenidas tras el entrenamiento o la fase de test siguiendo la estructura de estos ficheros.

1.6. Files

El paquete Files contiene los ficheros necesarios para la ejecución del sistema fuzzy: base de datos (.pwm), base de reglas (.wm), fichero de entrenamiento (.tra) y test (.tst). También albergará las salidas producidas tras una ejecución.

1.7. FuzzySystem

Este paquete contiene tanto la clase principal que contiene al *main()*, como una clase en la que se han almacenado los parámetros que se emplearán a lo largo de todo el sistema.

2. Ejecución del software

Para ejecutar el software habrá que moverse a la carpeta *FuzzySystem/dist*. Esta carpeta deberá contener el *.jar* y la carpeta *src/Files* con al menos los siguientes archivos: base de reglas, base de datos y fichero de entrenamiento. Además, esta carpeta contará con un README.TXT en el que se explica cómo ejecutar el software.
El controlador difuso se puede lanzar de dos formas:

- Dando un ejemplo de entrada, en cuyo caso devolverá la salida que quedará almacenada en *src/Files/Output.txt*. La instrucción para ejecutar esta opción sería:

  java -jar "FuzzySystem.jarin valor1 valor2 valori

- Dando un fichero de entrenamiento *.tra* con una secuencia de valores de entrada salida (una línea por cada secuencia), en cuyo caso cogerá por cada línea del fichero de entrenamiento los valores de las variables de entrada y devolverá los valores de la ejecución del controlador. Además, devolverá el error cuadrático medio (ECM). La salida de esta ejecución se almacenará en *src/Files/Output_tra.txt* siguiendo el mismo formato que el que tiene el fichero de entrenamiento.

  java -jar FuzzySystem.jar -tra ruta_del_fichero.tra