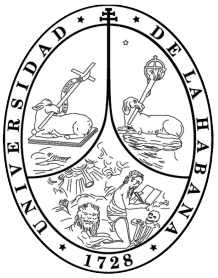

Proyecto de Simulación
"Sistema de Inferencia difuso"

Est. **Lázaro Raúl Iglesias Vera** L.IGLESIAS@ESTUDIANTES.MATCOM.UH.CU

C-412

Índice

1 Introducción

El objetivo de este proyecto es desarrollar una biblioteca que sirva de soporte para el trabajo con sistemas de inferencia difusos.

Con tal objetivo se diseñó una librería de *python* para manejar variables, conjuntos y cualquier elemento de un sistema difuso; en particular se creó una sintaxis especial para la creación de reglas que permite la utilización de más de una variable de salida mediante el uso de varios consecuentes.

2 Desarrollo

2.1 Elementos del sistema

A continuación serán descritos los elementos del sistema. Cada uno de ellos tiene su submódulo correspondiente en el módulo *pyfuzzysystem* que puede encontrar en el repositorio del proyecto [1].

2.1.1 Funciones de Membresía

El sistema posee una implementación general de una función de membresía, que permite utilizar cualquier función como una función de membresía. Para su utilización se deben proporcionar los puntos clave de la función dada para, a través de ellos, poder definir el dominio de la misma.

Las funciones implementadas son:

1. Gamma
2. L
3. Triangular (Lambda)
4. Trapezoidal (Pi)
5. S
6. Z
7. Gausiana

2.1.2 Funciones de defusificación

Para defusificar se implementaron un grupo de métodos que reciben un conjunto difuso y devuelven el valor correspondiente.

Las funciones implementadas son:

1. Bisección
2. Centroide
3. Mayor de los máximos (LOM)
4. Menor de los máximos (SOM)
5. Media de los máximos (MOM)

2.1.3 Conjuntos difusos

Los conjuntos difusos implementados son creados a partir de un nombre, una función de membresía y una función de agregación(*T-conorma*) mediante la cual se podrá aplicar la unión entre ellos. También haciendo uso de función de membresía se podrá obtener su dominio de definición.

2.1.4 Variables

Se implentaron variables difusas y lingüísticas las cuales se pueden encontrar en el submódulo *pyfuzzysystem.variable* y se explicarán a continuación.

1. **Lingüística**
   Esta variable facilita la creación de reglas tratando de lograr una sintaxis lo más parecida posible al lenguaje natural. Lo que sería en lenguage natural una regla:

   $$(A\ es\ bueno\ AND\ B\ es\ regular)\ OR\ C\ es\ malo\ \ =>\ \ D\ es\ poco,\ E\ es\ mucho$$

   Se representaría en el código como:

   $$((A_{bueno}\ \&\ B_{regular})\ |\ C_{malo})\ \ >>\ \ +D_{poco}\ +E_{mucho}$$

   Siendo cada $Letra_{categoria}$ una variable lingüística.
   Al crear este tipo de variable se debe especificar las funciones de unión(*T-conorma*) e intersección(*T-norma*) que utilizará para hacer las operaciones.

2. **Formal**
   Por formal se nombró a un conjunto difuso que forma parte de una variable difusa. Entonces reutilizando la notación vista anteriormente, todos los $Letra_{categoria}$ serían Formales. Estas variables son variables lingüísticas así que requieren de una función de intersección y una de unión, además del nombre de la variable difusa que representa y el conjunto que la conforma. Una invocación de un formal recibiendo la entrada de un sistema difuso devuelve la evaluación de la función de membresía de su conjunto tomando el valor de entrada que representa a su variable.

3. **Difusa**
   Una variable difusa no es más que un nombre junto con un grupo de conjuntos que la representan. Sea A una variable difusa, se le pueden añadir nuevos conjuntos mediante su función *add*, la cual recibe los parámetros de inicialización de un conjunto. Esta función fue creada para soportar el recurso de programación llamado interface fluida, el cual permite añadir varios conjuntos a la vez de la siguiente manera:

   $$A.add(...).add(...).add(...)...$$

   Para acceder a un conjunto de la variable se puede hacer a tavés de su nombre($A['bueno']$) lo cual devuelve una variable formal.

Haciendo uso de estas 3 variables mostradas previamente ya se puede definir de manera exacta el mecanismo de definición de reglas. El ejemplo mostrado durante la explicación de la variable lingüística quedaría de la siguiente manera.

$$((A['bueno']\ \&\ B['regular'])\ |\ C['malo'])\ \ >>\ \ +D['poco']\ +\ E['mucho']$$

Siendo A, B, C, D, E variables difusas de las cuales se extraen formales.

Por tanto de manera general una regla lingüística es cualquier cadena que cumpla la siguiente expresión regular:

$$Var['nombre']((\&\ \vee\ |)Var['nombre'])^{*}\ >>\ (+Var['nombre'])^{+}$$

Es importante mencionar que para la correcta formación del antecedente debe hacerse uso de paréntesis. Además, dada la precedencia de los operadores en *python* es obligatorio colocar paréntesis alrededor del antecedente, lo cual quedaría como:

$$(\ antecedente\ )\ >>\ consecuentes$$

2.1.5 Reglas

Las reglas reciben una proposición lingüística (resultado de utilizar el operador >>) y una función, la cual será utilizada para obtener el resultado del consecuente. La función brindada debe recibir un conjunto difuso, un valor (que será el resultado de la evaluación del antecedente de la regla) y devolver un conjunto difuso. Al invocar una regla tomando la entrada de un sistema difuso se aplicará la regla y se devolverán los conjuntos resultantes de sus consecuentes.

Reglas implementadas:

1. Mamdani (Corte)
2. Larsen (Escala)

2.1.6 Sistema difuso

Un sistema difuso requiere de un tipo de regla (clase de la regla) que usa y una función de defusificación. Para añadir una regla a un sistema S se puede utilizar la siguiente sintaxis:

$$S \mathrel{+}= (\ antecedente\ )\ >>\ consecuentes$$

Como se puede apreciar el operador '+=' permite añadir una proposición lingüística al sistema, mediante la cual se construirá una regla del tipo que usa el sitema.

En caso de querer añadir alguna regla previamente construida se puede utilizar el método *add_rule*. Añadir reglas de diferentes tipos utilizando esta vía puede conllevar comportamientos inesperados en los resultados.

La función $infer$ permite realizar el proceso de inferencia y recibe como parámetro un diccionario con el valor de cada variable de entrada.

Análogamente a la sección de las reglas fueron creados el sistema de Mamdani y de Larsen los cuales usan sus respectivas reglas.

2.2 Problema de ejemplo

El ejemplo puede encontrase en el archivo *main.py* del proyecto [1].

2.2.1 Definición

Una empresa está haciendo cambios en los salarios de sus empleados y quiere hallar el salario adecuado para cada uno en base a su puntualidad, desempeño y nivel de relaciones sociales. En particular se prefiere un nivel medio de relaciones sociales, tanto un trabajor muy activo que distraiga a los demás como un trabajador aislado son mal vistos por la empresa. Todas las métricas están dadas en un valor en el rango de 1 a 10.

2.2.2 Variables del sitema

Se crearon las siguientes variables:

- **puntualidad** {mala, buena}
- **eficiencia** {poca, estándar, excelente}
- **nivel de relaciones** {bajo, normal, alto}
- **salario** {tipo1, tipo2, tipo3, tipo4, tipo5, tipo6}

2.2.3 Reglas

$$
\begin{array}{rcrcrl}
Puntualidad : mala & \wedge & Eficiencia : poca & \wedge & Relaciones : baja & => salario : tipo1 \\
Puntualidad : mala & \wedge & Eficiencia : poca & \wedge & Relaciones : normal & => salario : tipo2 \\
Puntualidad : mala & \wedge & Eficiencia : poca & \wedge & Relaciones : alta & => salario : tipo1 \\
Puntualidad : mala & \wedge & Eficiencia : estándar & & & => salario : tipo3 \\
Puntualidad : mala & \wedge & Eficiencia : excelente & \wedge & Relaciones : baja & => salario : tipo4 \\
Puntualidad : mala & \wedge & Eficiencia : excelente & \wedge & Relaciones : normal & => salario : tipo5 \\
Puntualidad : mala & \wedge & Eficiencia : excelente & \wedge & Relaciones : alta & => salario : tipo4 \\
Puntualidad : buena & \wedge & Eficiencia : poca & \wedge & Relaciones : baja & => salario : tipo2 \\
Puntualidad : buena & \wedge & Eficiencia : poca & \wedge & Relaciones : normal & => salario : tipo3 \\
Puntualidad : buena & \wedge & Eficiencia : poca & \wedge & Relaciones : alta & => salario : tipo3 \\
Puntualidad : buena & \wedge & Eficiencia : estándar & \wedge & Relaciones : baja & => salario : tipo3 \\
Puntualidad : buena & \wedge & Eficiencia : estándar & \wedge & Relaciones : normal & => salario : tipo5 \\
Puntualidad : buena & \wedge & Eficiencia : estándar & \wedge & Relaciones : alta & => salario : tipo4 \\
Puntualidad : buena & \wedge & Eficiencia : excelente & \wedge & Relaciones : baja & => salario : tipo5 \\
Puntualidad : buena & \wedge & Eficiencia : excelente & \wedge & Relaciones : normal & => salario : tipo6 \\
Puntualidad : buena & \wedge & Eficiencia : excelente & \wedge & Relaciones : alta & => salario : tipo6
\end{array}
$$

2.2.4 Resultados

- **Puntualidad: 10, Eficiencia: 10, Relaciones: 5**
  Mamdani: 157.62
  Larsen: 158.14
- **Puntualidad: 10, Eficiencia: 10, Relaciones: 10**
  Mamdani: 148.44
  Larsen: 148.44
- **Puntualidad: 5, Eficiencia: 8, Relaciones: 10**
  Mamdani: 96.91
  Larsen: 96.61
- **Puntualidad: 5, Eficiencia: 10, Relaciones: 0**
  Mamdani: 106.5
  Larsen: 106.03
- **Puntualidad: 3, Eficiencia: 2, Relaciones: 7**
  Mamdani: 70
  Larsen: 69.99

3 Conclusiones

Utilizando el sistema implementado se pudo observar cómo mediante el uso del sitema de Larsen y Mamdani se obtienen valores similares como resultados de las inferencias.

También durante la realización de las pruebas y la selección del problema, se notó la importancia de seleccionar correctamente la función de defusificación. La buena utilización de las funciones de membresía y la definición de las reglas, pues de no hacerlo adecuadamente se pueden obtener resultados poco acordes con los requerimientos del problema, o simplemente tener un grupo de valores inalcanzables, como es el caso del ejemplo mostrado previamente donde el salario máximo no se puede alcanzar haciendo uso de la función de defusificación centroide.

Dicho lo anterior, contar con un experto en el tema para el diseño del sistema es importante para garantizar la validez del mismo.

Referencias

[1] Repositorio del proyecto (ir)