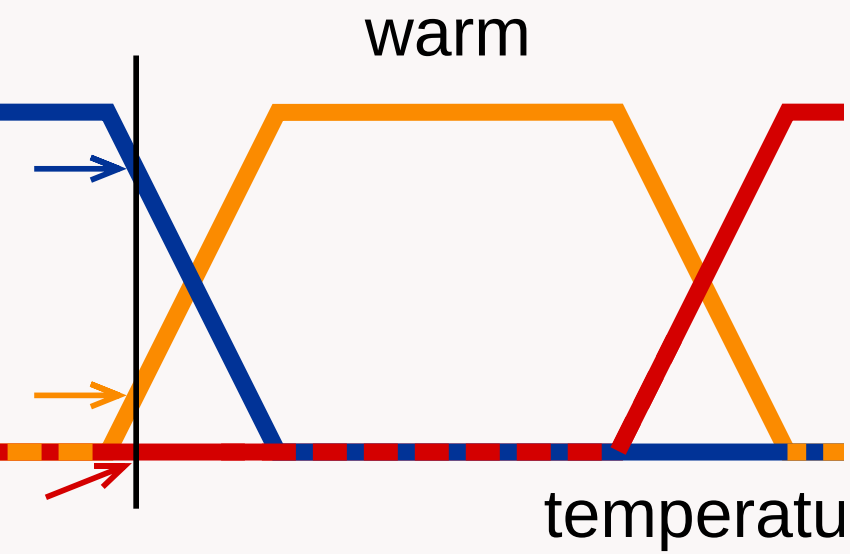


Control borroso

Robótica

Alberto Díaz y Raúl Lara

Curso 2022/2023

Departamento de Sistemas Informáticos

Recordatorio de lógica borrosa

Se puede considerar una extensión de la teoría clasica de conjuntos:

- En esta teoría, los elementos pertenecen o no a un conjunto
- Función característica: $f(x) = 1$ si $x \in A$ y $f(x) = 0$ si $x \notin A$

Trata información a priori imprecisa en términos de conjuntos borrosos:

- Los elementos pertenecen a un conjunto con un grado de pertenencia.
- Función de pertenencia: $f(x) = \mu(x) \in [0, 1]$

Los conjuntos borrosos se agrupan en particiones

- Una partición se define sobre una variable denominada lingüística.

Definiciones

Variable lingüística: Variable cuyos valores son términos en lenguaje natural.

Partición borrosa: Todos los conjuntos borrosos de una variable lingüística.

Función de pertenencia: Determina el grado de pertenencia de un elemento a un conjunto borroso (en tanto por uno).

Ejemplo: La variable lingüística $precio$ *puede tomar los valores* $precio \equiv \{barato, normal, caro\}$. *Estos serán tres conjuntos borrosos, cada uno con las funciones de pertenencia* $\{f_{barato}(x),\ f_{normal}(x)$ *y* $f_{caro}(x)\}$.

Operaciones borrosas

Complemento: $f'_{barato}(x) = 1 - f_{barato}(x)$

t-normas (intersección)

- Mínimo: $f_{barato} \cap f_{normal} = min(f_{barato}, f_{normal})$
- Producto algebráico: $f_{barato} \cap f_{normal} = f_{barato} \cdot f_{normal}$

t-conormas (unión)

- Máximo: $f_{barato} \cap f_{normal} = max(f_{barato}, f_{normal})$
- Suma algebráica: $f_{barato} \cap f_{normal} = f_{barato} + f_{normal} - f_{barato} \cdot f_{normal}$

La **inferencia** ($\rightarrow$) se suele definir como la operación de **intersección**.

Reglas borrosas

Son reglas que relacionan varios antecedentes con consecuentes, donde:

- Antecedentes: Conjuntos borrosos de entrada
- Consecuentes: Conjuntos borrosos de salida

Si *el precio es barato* ***Y*** *la calidad es mala* ***entonces*** *la satisfacción es baja.*

Se agrupan en una **base de reglas**, las cuales pueden ser de varios tipos:

- De tipo Mandani: **Si** V_1 es $F_i^{V_1}$ **Y** V_2 es $F_j^{V_2}$ **Y** $\ldots$ **entonces** V_o es $F_k^{V_o}$
- De tipo Sugeno: **Si** V_1 es $F_i^{V_1}$ **Y** V_2 es $F_j^{V_2}$ **Y** $\ldots$ **entonces** $V_o = f(\vec{x})$

Fuzzification y defuzzification

Fuzzification : Convertir valores de entrada concretos en conjuntos borrosos.

- Es basicamente aplicar las funciones de pertenencia a los valores de entrada.

Defuzzification : Convertir conjuntos borrosos en valores de salida concretos.

- Existen muchas técnicas para realizar esta operación.
- Las más comunes son el centroide y el centroide simplificado

Centroide

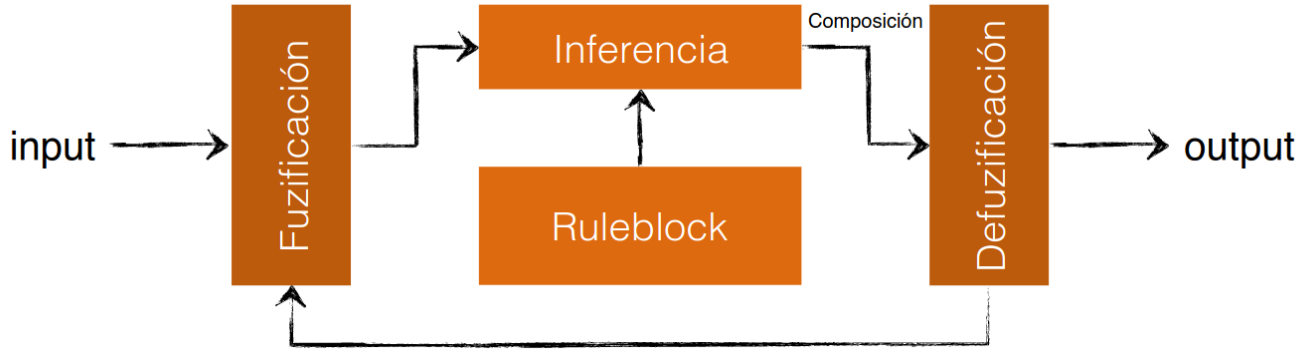
$$y = \frac{\int y \cdot \mu(y) dy}{\int \cdot \mu(y) dy}$$

Centroide simplificado

$$y \approx \frac{\sum y \cdot \mu(y) dy}{\sum \cdot \mu(y) dy}$$

Controlador borroso

Es un sistema de control que se apoya en la lógica borrosa como sigue:



1. Toma la entrada al sistema.
2. Pasa los valores a pertenencia a conjuntos borrosos (*fuzzification*)
3. Infiere conjuntos de borrosos de salida haciendo uso de las reglas borrosas.
4. Pasa los conjuntos borrosos de salida en valores concretos (*defuzzification*)
5. Aplica la salida al sistema a controlar.

Ejemplo de implementación de un controlador borroso

Diseño de un controlador borroso

Para diseñar un controlador borroso, se debe seguir el siguiente proceso:

1. Identificar variables de entrada y de salida.
2. Determinar los conjuntos borrosos para cada variable
3. Definir las reglas borrosas que van a regir el comportamiento del controlador.
4. (Opcional) Normalización y escalado de entradas y salidas.

Implementaremos un controlador para el *problema de las propinas:*

- Problema clasico de control borroso.
- ¿Cuánto dar de propina en función de la calidad del servicio y de la comida?
- Usaremos la biblioteca `skfuzzy` para implementar un controlador borroso.

Formulación del problema

Antecedentes (entradas):

- Servicio (de 0 a 10): $malo$, $normal$, $bueno$
- Calidad (de 0 a 10): $mala$, $aceptable$, $buena$

Consecuentes (salidas):

- Propina (de 0 a 25): $baja$, $media$, $alta$

Reglas:

1. **Si** Servicio $bueno$ o Calidad $buena$ **entonces** Propina $alta$
2. **Si** Servicio $normal$ **entonces** Propina $media$
3. **Si** Servicio $malo$ y Calidad $mala$ **entonces** Propina $baja$.

Implementación de las variables lingüísticas

El primer paso es definir las variables de entrada y salida del controlador.

```
import numpy as np
from skfuzzy import control as ctrl

# Antecedentes
servicio = ctrl.Antecedent(np.arange(0, 11, 1), 'servicio')
calidad = ctrl.Antecedent(np.arange(0, 11, 1), 'calidad')
# Consecuente
propina = ctrl.Consequent(np.arange(0, 26, 1), 'propina')
```

Definición de los conjuntos borrosos

Para cada variable, se definen los conjuntos borrosos que la componen.

```
import skfuzzy as fuzz

# Conjuntos borrosos de servicio
servicio['malo'] = fuzz.trimf(servicio.universe, [0, 0, 5])
servicio['normal'] = fuzz.trimf(servicio.universe, [0, 5, 10])
servicio['bueno'] = fuzz.trimf(servicio.universe, [5, 10, 10])
# Conjuntos borrosos de calidad
calidad['mala'] = fuzz.trimf(calidad.universe, [0, 0, 5])
calidad['aceptable'] = fuzz.trimf(calidad.universe, [0, 5, 10])
calidad['buena'] = fuzz.trimf(calidad.universe, [5, 10, 10])
# Conjuntos borrosos de propina
propina['baja'] = fuzz.trimf(propina.universe, [0, 0, 13])
propina['media'] = fuzz.trimf(propina.universe, [0, 13, 25])
propina['alta'] = fuzz.trimf(propina.universe, [13, 25, 25])
```

Se puede usar el método `.automf(n)` para definirlos de forma automática.

Visualización de los conjuntos borrosos

Para visualizar los conjuntos borrosos, se puede usar la función `view()`.

```
servicio.view()
calidad.view()
propina.view()
```

Concretamente mostrará la variable lingüística junto con:

- Las **funciones de pertenencia** que caracterizarán a cada conjunto borroso.
- El **dominio** de la variable lingüística.

Definición de las reglas

Para definir las reglas, se debe usar la función `ctrl.Rule()`.

```
rulebase = [
    ctrl.Rule(servicio['bueno'] | calidad['buena'], propina['alta'])
    ctrl.Rule(servicio['normal'], propina['media'])
    ctrl.Rule(servicio['malo'] & calidad['mala'], propina['baja'])
]
```

Suele ser buena costumbre definir las reglas en una lista.

Definición del controlador

Para definir el controlador, se debe usar la función `ctrl.ControlSystem()`.

```
>>> controlador = ctrl.ControlSystem([r1, r2, r3])
```

Luego se simula con la función `ctrl.ControlSystemSimulation()`.

- Este objeto se encarga de implementar casos concretos sobre un controlador.

```
>>> simulacion = ctrl.ControlSystemSimulation(controlador)
```

- El caso concreto se simulará con la función `compute()`.

```
>>> simulacion.input['quality'] = 6.5
>>> simulacion.input['service'] = 9.8
>>> simulacion.compute()
>>> print(simulacion.output['tip'])
19.847607361963192
```

¡GRACIAS!