PROYECTO 2 Problema de la propina modo complejo

El problema de la propina: la manera difícil

Nota: Este método calcula todo a mano, paso a paso, Para la mayoría de las personas, la nueva API para sistemas difusos sera preferible. El mismo problema se resuelve con la nueva API en este ejemplo.

El "problema de la propina" se usa comúnmente para ilustrar el poder de los principios lógicos difusos para generar un comportamiento completo a partir de un conjunto compacto de reglas

Recapitulando

Variables de entrada
Un numero de variables juegan en la decisión de cuanto propina dejar mientras se cena. Consideremos dos de ellas

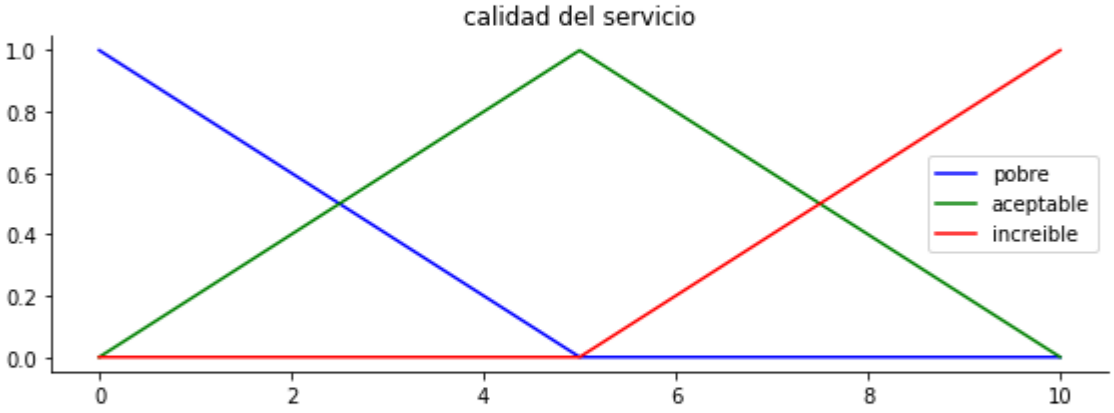
Calidad: calidad de la comida
Servicio: calidad del servicio

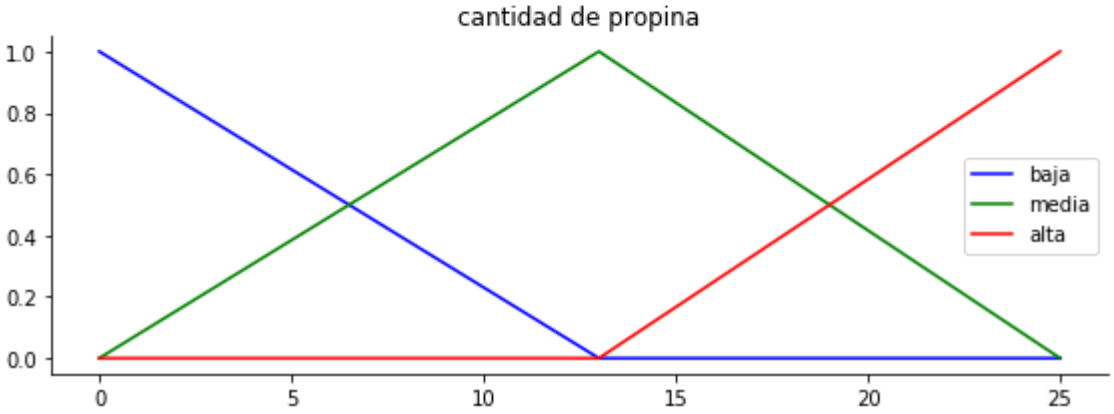
Variable de salida
La variable de salida es simplemente el monto de la propina en puntos porcentuales

Propina: porcentaje de la factura que se agrega como propina

Para los fines de discusión, supongamos que necesitamos funciones de membresía "alta", "media" y "baja" para las variables de entrada y nuestra variable de salida. Estos se definen en Scikit-Fuzzy de la siguiente manera:

```python
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt

# Generar variables del universo discurso
# Calidad y servicio se encuentran en rangos subjetivos
# Propina tiene un rango de 0 a 25  en unidades porcentuales

x_cldad = np.arange(0, 11, 1)
x_serv = np.arange(0, 11, 1)
x_prpna = np.arange(0, 26, 1)

# generar funciones de pertenencia difusa

cldadbaja = fuzz.trimf(x_cldad, [0, 0, 5])
cldadmedia = fuzz.trimf(x_cldad, [0, 5, 10])
cldadalta = fuzz.trimf(x_cldad, [5, 10, 10])
servbajo = fuzz.trimf(x_serv, [0, 0, 5])
servmedio = fuzz.trimf(x_serv, [0, 5, 10])
servalto = fuzz.trimf(x_serv, [5, 10, 10])
prpnabaja = fuzz.trimf(x_prpna, [0, 0, 13])
prpnamedia = fuzz.trimf(x_prpna, [0, 13, 25])
prpnaalta = fuzz.trimf(x_prpna, [13, 25, 25])

# Visualizar estos universos y funciones de pertenencia
fig, (ax0, ax1, ax2) = plt.subplots(nrows=3, figsize=(8, 9))

ax0.plot(x_cldad, cldadbaja, 'b', linewidth=1.5, label = 'mala')
ax0.plot(x_cldad, cldadmedia, 'g', linewidth=1.5, label = 'decente')
ax0.plot(x_cldad, cldadalta , 'r', linewidth=1.5, label = 'genial')
ax0.set_title('calidad de la comida')
ax0.legend()

ax1.plot(x_serv, servbajo , 'b', linewidth=1.5, label = 'pobre')
ax1.plot(x_serv, servmedio , 'g', linewidth=1.5, label = 'aceptable')
ax1.plot(x_serv, servalto , 'r', linewidth=1.5, label = 'increible')
ax1.set_title('calidad del servicio')
ax1.legend()

ax2.plot(x_prpna, prpnabaja , 'b', linewidth=1.5, label = 'baja')
ax2.plot(x_prpna, prpnamedia , 'g', linewidth=1.5, label = 'media')
ax2.plot(x_prpna, prpnaalta , 'r', linewidth=1.5, label = 'alta')
ax2.set_title('cantidad de propina')
ax2.legend()

# desactivar los ejes superior/derecho

for ax in (ax0, ax1, ax2):

    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()

plt.tight_layout()
```





Reglas difusas

Ahora, para que estos triángulos sean útiles, definimos la relación difusa entre las variables de entrada y de salida. Para nuestro ejemplo, consideremos tres reglas simples:

If la comida es mala O el servicio es deficiente, then la propina será baja.
If el servicio es aceptable, then la propina será media
If la comida es buena O el servicio es excelente, then la propina será alta.

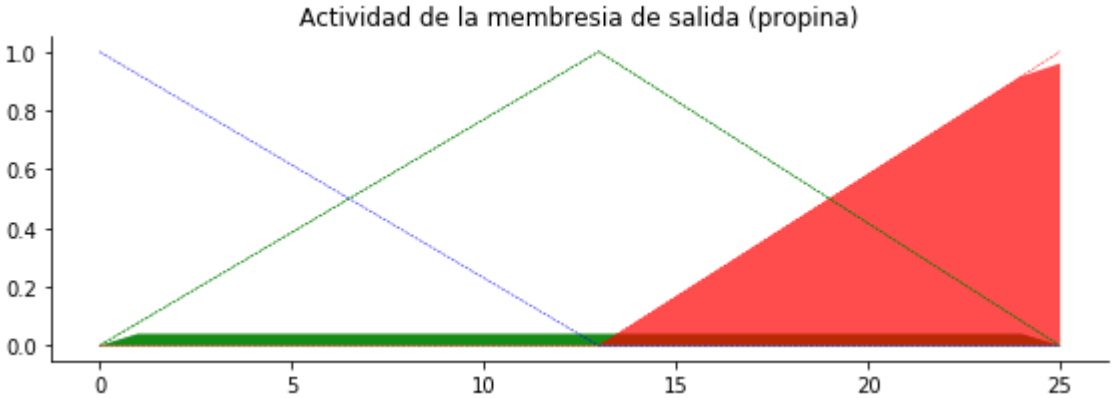
La mayoría de la gente estaría de acuerdo con estas reglas, pero son imprecisas. Convertir las reglas imprecisas en una propina definida y procesable es todo un reto. Este es el tipo de tarea en el que destaca la lógica difusa.

Aplicación de la regla

¿Cual seria la propina en la siguiente circunstancia?

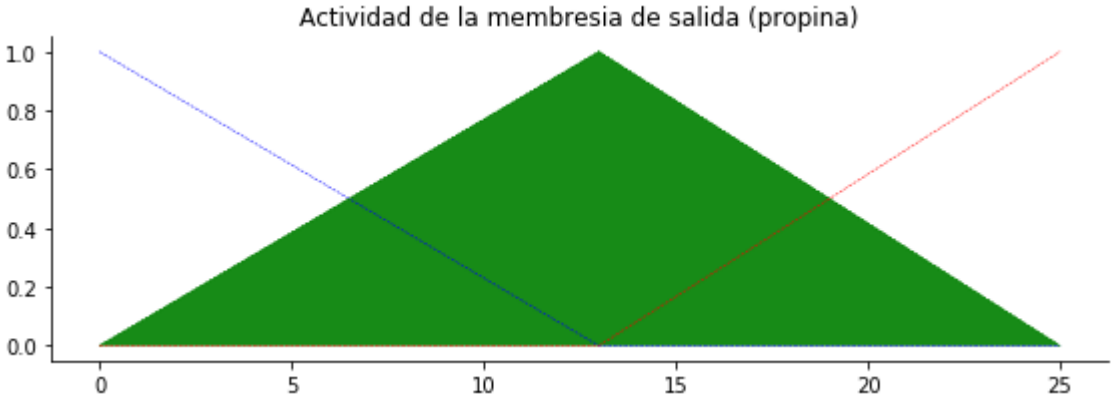
La calidad de la comida fue de 6.5
El servicio fue de 9.8

```
# necesitamos la activación de nuestras funciones de pertenencia difusas en los valores de 6.5 y 9.8
# como los valores de 6.5 y 9.8 no estan en nuestros universos se utilizara la funcion fuzz.interp_membership!

nivelcldadbajo = fuzz.interp_membership(x_cldad, cldadbaja, 6.5)
nivelcldadmedio = fuzz.interp_membership(x_cldad, cldadmedia, 6.5)
nivelcldadalto = fuzz.interp_membership(x_cldad, cldadalta, 6.5)

nivelservbajo = fuzz.interp_membership(x_serv, servbajo, 9.8)
nivelservmedio = fuzz.interp_membership(x_serv, servmedio, 9.8)
nivelservalto = fuzz.interp_membership(x_serv, servalto, 9.8)

# ahora creemos nuestras reglas y apliquemoslas. La regla 1 corresponden a mala comida o servicio
#el operador OR significa que tomaremos el maximo de los dos

activar_regla1 = np.fmax(nivelcldadbajo, nivelservbajo)

# ahora aplicamos esto recortando la parte superior de la funcion de membresia de salida correspondiente con np.fmin

activarpropinabaja = np.fmin(activar_regla1, prpnabaja) # eliminado completamente a 0

# para la regla 2 relacionamos el servicio aceptable a la propina media activar_regla2

activarpropinamedia = np.fmin(nivelservmedio, prpnamedia)

# para la regla 3 relacionamos el buen servicio o la buena comida con propina alta

activar_regla3 = np.fmax(nivelcldadalto, nivelservalto)

# ahora aplicamos esto recortando con np.fmin

activarpropinaalta = np.fmin(activar_regla3, prpnaalta)
propina0 = np.zeros_like(x_prpna)    # es un array de zeros solamente

# visualizemos esto
fig, ax0 = plt.subplots(figsize=(8, 3))

ax0.fill_between(x_prpna, propina0, activarpropinabaja, facecolor = 'b', alpha=0.7)
ax0.plot(x_prpna, prpnabaja, 'b', linewidth = 0.5, linestyle = '--', )
ax0.fill_between(x_prpna, propina0, activarpropinamedia, facecolor = 'g', alpha=0.7)
ax0.plot(x_prpna, prpnamedia, 'g', linewidth = 0.5, linestyle = '--', )
ax0.fill_between(x_prpna, propina0, activarpropinamedia, facecolor = 'g', alpha=0.7)
ax0.plot(x_prpna, prpnamedia, 'g', linewidth = 0.5, linestyle = '--', )
ax0.fill_between(x_prpna, propina0, activarpropinaalta, facecolor = 'r', alpha=0.7)
ax0.plot(x_prpna, prpnaalta, 'r',   linewidth = 0.5, linestyle = '--', )
ax0.set_title('Actividad de la membresia de salida (propina)')

# desactivar los ejes superior/derecho y plt graficar los resultados hasta despues del for obtenidos
for ax in (ax0,):

    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()

plt.tight_layout()
```

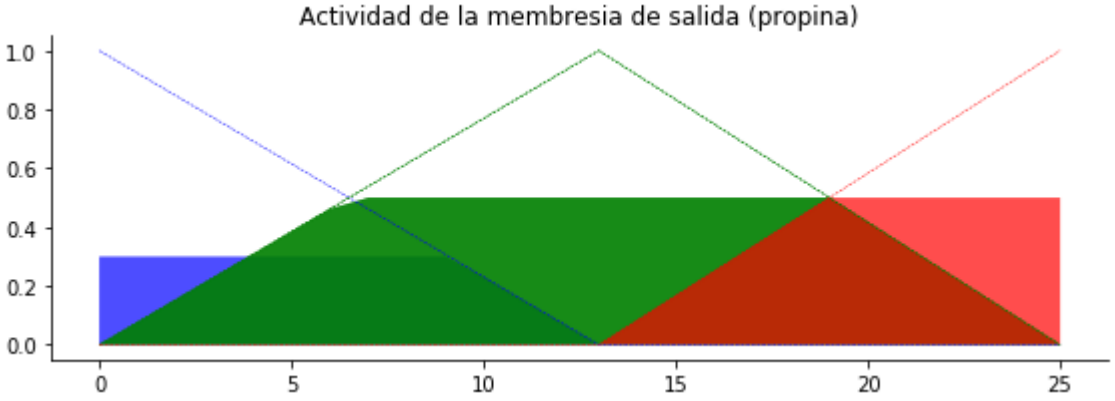


Ahora observaremos las graficas con los valores sugeridos en el caso general, (proyecto1.pdf)

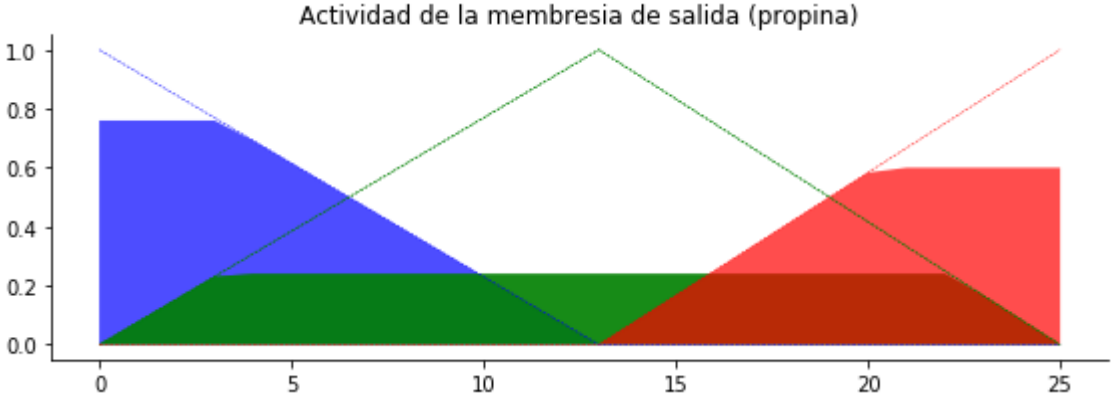
Calidad y servicio con 5



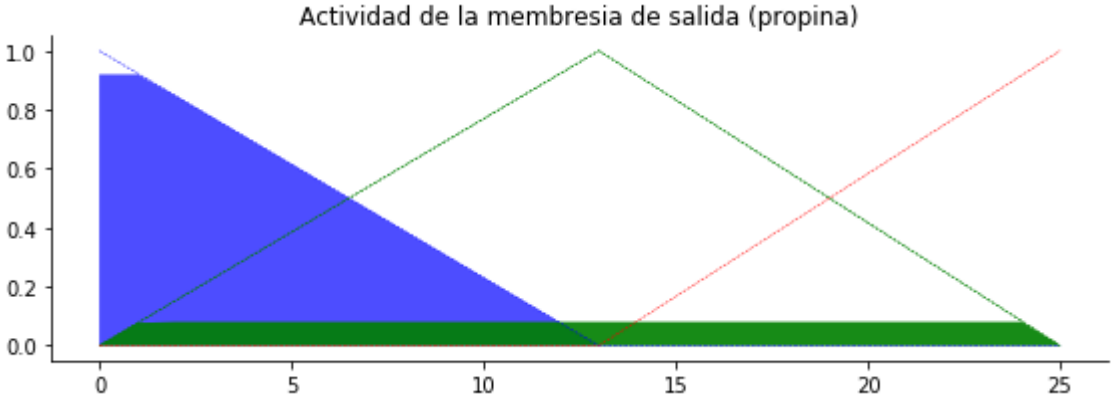
calidad con 3.5 y servicio con 7.5



calidad con 8 y servicio con 1.2



calidad con 1.1 y servicio con 0.4



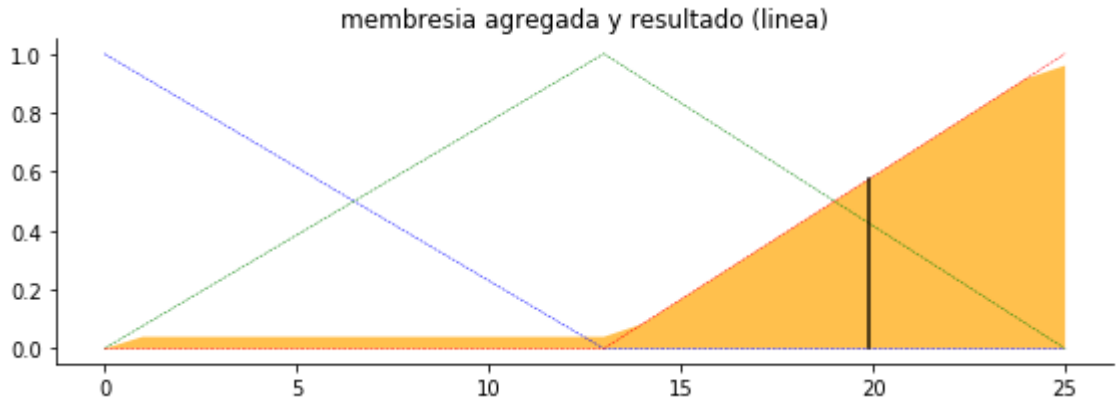
Agregación de reglas

Una vez conocida la actividad de cada función de pertenencia de salida, deben combinarse todas las funciones de pertenencia de salida. Para ello se suele utilizar un operador de máximo. este paso también se conoce como agregación

Defuzzificacion

Por ultimo, para obtener una respuesta del mundo real, volvemos a la lógica crisp(nítida) desde el mundo de las funciones de pertenencia difusas. Para este ejemplo se utilizara el método del centroide

```
# agregando las 3 funciones de membresia de salida
#En este caso, se está calculando el máximo grado de pertenencia entre tip_activation_lo y
#el máximo grado de pertenencia entre tip_activation_md y tip_activation_hi. Esto asegura que el
#resultado final tenga el grado de pertenencia más alto de las tres funciones de membresía.

agregadas = np.fmax(activarpropinabaja, np.fmax(activarpropinamedia, activarpropinaalta))


# calcular resultado defuzzificado
propina = fuzz.defuzz(x_prpna, agregadas, 'centroid')
activacion_propina = fuzz.interp_membership(x_prpna, agregadas, propina) # para la grafica

# visualizar esto
fig, ax0 = plt.subplots(figsize=(8, 3))

ax0.plot(x_prpna, prpnabaja, 'b', linewidth=0.5, linestyle='--', )
ax0.plot(x_prpna, prpnamedia, 'g', linewidth=0.5, linestyle='--', )
ax0.plot(x_prpna, prpnaalta, 'r', linewidth=0.5, linestyle='--', )
ax0.fill_between(x_prpna, propina0, agregadas, facecolor='Orange', alpha=0.7)
ax0.plot([propina, propina], [0, activacion_propina], 'k', linewidth=1.5, alpha=0.9)
ax0.set_title('membresia agregada y resultado (linea)')

# desactivar los ejes superior/derecho y plt graficar los resultados hasta despues del for obtenidos
for ax in (ax0,):
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.get_xaxis().tick_bottom()
    ax.get_yaxis().tick_left()

plt.tight_layout()
```



Pensamientos (autor del blog)

El poder de los sistemas difusos esta permitiendo un comportamiento complicado e intuitivo basado en un sistema escaso de reglas con una sobre carga mínima. Tenga en cuenta que nuestros universos de función de membresía eran gruesos, solo definidos en los enteros, pero Fuzz.Inerp_Membership permitió que la resolución efectiva aumente la demanda. Este sistema puede responder a cambios arbitrariamente pequeños en las entradas, y la carga de procesamiento es mínima