Logika rozmyta - regułowy system wnioskowania

Joanna Kołodziejczyk

1 Wprowadzenie - cel zajęć

Celem zajęć jest implementacja rozmytego systemu wnioskowania (Fuzzy Inference System) z użyciem biblioteki scikit-fuzzy. (`https://scikit-fuzzy.readthedocs.io/en/latest/`) Wykonane zostaną dwa przykłady demonstrujące system jedno i dwu-wejściowy.

2 Przykład z jednym wejściem

Przykład pierwszy przewiduje liczbę oczekiwanych klientów w lodziarni przy określonej temperaturze zewnętrznej. Przykład ten został pochodzi z prezentacji `https://www.youtube.com/watch?v=qUQf1JxnTnY`.

Głównym elementem systemu rozmytego jest baza wiedzy w postaci reguł w formie:

IF założenie (antecedent) THEN wniosek (consequent)

Zakładamy, że właściciel lodziarni wyraził swoją wiedzę ekspercką w postaci prostych reguł:

1. Jeśli temperatura jest wysoka to lodziarnia jest zatłoczona.
2. Jeśli temperatura jest umiarkowana, to lodziarnia ma przeciętną liczbę klientów.
3. Jeśli temperatura jest niska, to lodziarnia jest pusta.

W systemie zatem mamy dwie zmienne. Na wejściu do systemu jest temperatura *temp* a na wyjściu liczba klientów *customers*.

Do wykonania zadania konieczne jest załadowanie bibliotek.

Listing 1: Niezbędne biblioteki

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

2.1 Opis system rozmytego

Pierwszym krokiem jest określenie rozmytych funkcji przynależności, które reprezentują stopień prawdziwości dla danej liczbowej zmiennej wejściowej i wyjściowej.

Listing 2: Definiowanie zmiennych

```
# Określenie dziedziny dla zmiennych:wejściowej i wyjściowej
temp = np.arange(-10, 40, 1)
customers = np.arange(0, 35, 1)

# Przyjęto 3 wartości lingwistyczne dla temperatury: hot, mederate, cool
# Funkcja przynależności dla temperatury
t_hot = fuzz.trimf(temp, [25, 40, 40]) #funkcja trójkątna
t_moderate = fuzz.trimf(temp, [10, 25, 40]) #funkcja trójkątna
t_cool = fuzz.trapmf(temp, [-10, -10, 10, 25]) #funkcja trapezoidalna

# Przyjęto 3 wartości lingwistyczne dla liczby klientów: crowded, busy,
    quiet
# Funkcja przynależności dla dla liczby klientów
c_crowded = fuzz.trimf(customers, [24, 35, 35])
c_busy = fuzz.trimf(customers, [0, 24, 35])
c_quiet = fuzz.trimf(customers, [0, 0, 24])
```

2.1.1 Wizualizacja zmiennych systemu

W FIS można przedstawić model zmiennej wejściowej i wyjściowej demonstrując rozkład funkcji przynależności na wykresach.

Listing 3: Wykresy zmiennych

```
# Wizualizacja przynależności dla temperatury
fig, [ax1, ax2] = plt.subplots(nrows=2, ncols=1)

ax1.plot(temp, t_hot, 'r', temp, t_moderate, 'm', temp, t_cool, 'b')
ax1.set_ylabel('Przynależność')
ax1.set_xlabel('Temperatura (Celsius)')
ax1.set_ylim(-0.1, 1.1)

# Wizualizacja przynależności dla liczby klientów
ax2.plot(customers, c_quiet, 'c', customers, c_busy, 'm', customers,
    c_crowded, 'ForestGreen')
ax2.set_ylabel('Przynależność')
ax2.set_xlabel('Liczba klientów')
ax2.set_ylim(-0.1, 1.1)

plt.show()
```

2.1.2 Reguły i wnioskowanie

Ważnym elementem systemu FIS jest baza reguł. Zawiera ona implikacje opisujące jakie wartości wejść implikują lingwistyczne wartości wyjść. Używa się rozmytej implikacji

jako np. relacji product **fuzz.relation_product**. Wnioskowanie wykonywane jest oddzielnie dla każdej reguły w systemie, więc powstaną trzy oddzielne macierze. Scikit-fuzzy ma zaimplementowaną inną relację wnioskowanie, jako min (klasyczne podejście Mamdani), **fuzz.relation_min**.

Listing 4: Reguły w systemie

```
# Wnioskowanie
# Reguły rozmyte iloczyn (nie ma znaczenia, bo na wejściu jest 1 zmienna)
# R1 - Jeżeli gorąco, to tłoczno
R1 = fuzz.relation_product(t_hot, c_crowded)
# R2 - Jeżeli umiarkowanie, to normalne zainteresowanie
R2 = fuzz.relation_product(t_moderate, c_busy)
# R3 - Jeżeli zimno, to pusto
R3 = fuzz.relation_product(t_cool, c_quiet)
```

Agregacja reguł jest procesem złożenia wszystkich odpalonych reguł w jedną rozmytą konkluzję. Jedną z możliwości jest realizacja agregacji jako funkcji max (suma).

Listing 5: Agregacja reguł

```
# Złożenie reguł (agregacja) z użyciem operatora max
R_combined = np.fmax(R1, np.fmax(R2, R3))
```

Płaszczyzna modelu rozmytego to zależność pomiędzy wejściem i wyjściem dla pełnej dziedziny. Modele zazwyczaj są nieliniowe.

Listing 6: Powierzchnia odpowiedzi

```

#Wizualizacja powierzchni decyzyjnej/modelu/prognostycznej
plt.figure(2)
plt.imshow(R_combined)
cbar = plt.colorbar()
cbar.set_label('Przynależność')
plt.yticks([i * 10 for i in range(6)], [str(i * 10 -10) for i in range(6)])
plt.ylabel('Temp')
plt.xlabel('Liczba klientów')
```

2.2 Wyostrzanie

Rozmyta konkluzja pokazuje, jak system reaguje na wszystkie temperatury i wszystkich klientów. Aby sporządzić konkretną prognozę dla konkretnej temperatury należy wrócić z przestrzeni liczb rozmytych do liczb ostrych.

Chcemy się dowiedzieć ilu klientów może spodziewać się lodziarnia przy założeniu, że są 35 stopnie Celsiusza.

Listing 7: Wyostrzanie metodą środka ciężkości

```
# fuzz.defuzz Wyostrzanie metodą środka ciężkości 'centroid'
print("Liczba klientów przy temp. 35 st. Celsiusza:",
      fuzz.defuzz(customers, R_combined[temp == 35], 'centroid').round())
```

2.3 Wyniki dla całej przestrzeni wejść

Możliwe jest wygenerowanie zbioru odpowiedzi dla całej dziedziny wejścia. Można wówczas korzystać z niego np. do tworzenia wizualizacji.

Listing 8: Wizualizacja wyników

```
# Wygeneruj tablicę dla wszystkich temperatur z dziedziny
predicted_customers = np.zeros_like(temp)

# wypełnij tablicę wyliczoną liczbą klientów dla każdej temperatury
for i in range(len(temp)):
    predicted_customers[i] = fuzz.defuzz(customers, R_combined[i, :], '
    centroid')

# Wizualizacja
plt.figure(3)
plt.subplot(1, 2, 1)
# Liczba klientów dla temp. 35 st C
plt.plot(temp, predicted_customers, 'k')
plt.vlines(35, 5, predicted_customers[temp == 35], color='DarkOrange',
    linestyle='dashed', lw=2)
plt.hlines(predicted_customers[temp == 35], -10, 35, color='DarkOrange',
    linestyle='dashed', lw=2)
plt.xlabel('Temp')
plt.ylabel('Liczba klientów')

plt.subplot(1, 2, 2)
# Liczba klientów dla temp. -1 st C
plt.plot(temp, predicted_customers, 'k')
plt.vlines(-1, 5, predicted_customers[temp == -1], color='DarkOrange',
    linestyle='dashed', lw=2)
plt.hlines(predicted_customers[temp == -1], -10, -1, color='DarkOrange',
    linestyle='dashed', lw=2)
plt.xlabel('Temp')
plt.ylabel('Liczba klientów')

plt.show()
```

3 Przykład dwu-wejściowy

Utworzony zostanie rozmyty system wnioskowania, który modeluje, w jaki sposób można kalkulować napiwki w restauracji. Jako parametr wejściowy przyjmuje się obsługę i jakość jedzenia, oceniane w skali od 0 do 10. Napiwek stanowi procent wydanej kwoty i waha się między 0 a 25%.

- Antecedents (Wejścia):
  - Service: wartości oceny należą do przedziału od 0 do 10 a wartości lingwistyczne to np. poor, average, good
  - Food: wartości oceny należą do przedziału od 0 do 10 a wartości lingwistyczne to np. poor, average, good

- Consequents (Wyjście)
  - Napiwek: procent kwoty, przy możliwych wartościach od 0 do 25% a wartości lingwistyczne to np. low, medium, high.
- Reguły
  IF service = poor OR food = poor, THEN tip = low.
  IF service = average, THEN tip = medium.
  IF service = good OR food = good THEN tip = high.

- Użycie:
  - Oceniono service = 9.8
  - Oceniono food = 6.5
  - Obliczyć rekomendację napiwku: ??%.

Listing 9: Skrypt do obliczania napiwku

```
import numpy as np
import skfuzzy as fuzz
from numpy.core._multiarray_umath import ndarray
from skfuzzy import control as ctrl
import matplotlib.pyplot as plt
from matplotlib import cm

# Wejścia do systemu – deklarowana dziedzina
quality = ctrl.Antecedent(np.arange(0, 11, 1), 'quality')
service = ctrl.Antecedent(np.arange(0, 11, 1), 'service')
# Wyjście z systemu – deklarowana dziedzina
tip = ctrl.Consequent(np.arange(0, 26, 1), 'tip')

# Automatyczne generowanie funkcji przynależności  dla wejść w liczbie . automf(3, 5, or 7)
# Liczba funkcji wpływa na liczbę reguł
quality.automf(3)
service.automf(3)

# Ręczne budowanie funkcji przynależności dla wyjścia
tip['low'] = fuzz.trimf(tip.universe, [0, 0, 13])
tip['medium'] = fuzz.trimf(tip.universe, [0, 13, 25])
tip['high'] = fuzz.trimf(tip.universe, [13, 25, 25])

# Wizualizacja wejść i wyjścia
service.view()
quality.view()
tip.view()
```

Kolejnym krokiem jest wykonanie wnioskowania na regułach. Baza wiedzy z regułami rozmytymi jest deklarowana następującym kodem:

Listing 10: Reguły dla napiwku

```
rule1 = ctrl.Rule(quality['poor'] | service['poor'], tip['low'])
rule2 = ctrl.Rule(service['average'], tip['medium'])
rule3 = ctrl.Rule(quality['good'] | service['good'], tip['high'])
```

Przesłanki w regułach mogą być łączone operatorem OR (symbol |) i operatorem AND (symbol &).

Posiadając zestaw reguł można utworzyć model.

Listing 11: Model

```
# Model rozmyty w oparciu o trzy reguły
tipping_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])

#Obliczenie wyjść (symulacja systemu) dla obiektu tipping_ctrl
tipping = ctrl.ControlSystemSimulation(tipping_ctrl)
```

Jeżeli chcemy obliczyć napiwek dla poszczególnych wejść można podać jakie wartości zostają podane na wejściu do systemu i zostanie obliczona wartość napiwku:

Listing 12: Wyliczenie napiwku

```
# Do ControlSystem należy wprowadzić wartości wejść
tipping.input['quality'] = 6.5
tipping.input['service'] = 9.8

# Oblicz wyjście
tipping.compute()

print ("Obliczony napiwek wynosi:",  tipping.output['tip'].round(2))
tip.view(sim=tipping)
```

Lepsze zrozumienie działania modelu można uzyskać poprzez wygenerowanie powierzchni odpowiedzi — wizualizacji obliczonego napiwku dla każdej możliwej kombinacji wejść. Ze względu na to, że wejścia są wartościami ze zbioru liczb rzeczywistych dokonujemy próbowania przedziału z krokiem co 0.25.

Listing 13: Wizualizacja powierzchni 3D

```
# Wykreślenie płaszczyzny modelu
qu = np.arange(0, 10.25, 0.25)
ser = np.arange(0, 10.25, 0.25)
qu, ser = np.meshgrid(qu, ser)

predicted_val: ndarray = np.zeros(shape=(len(ser),len(qu)))
for i in range(len(qu)):
    for j in range(len(ser)):
        #print(qu[i,j],ser[i,j])
        tipping.input['quality'] = qu[i,j]
        tipping.input['service'] = ser[i,j]
        tipping.compute()
        predicted_val[i,j] = tipping.output['tip']

# Wykreślenie wykresu trójwymiarowego
fig = plt.figure()
ax = fig.gca(projection='3d')
```

```
surf = ax.plot_surface(qu, ser, predicted_val, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)

fig.colorbar(surf, shrink=0.5, aspect=5)

plt.show()
```

4 Sprawozdanie

Wykonać sprawozdanie zawierające opracowanie własnego systemu wnioskowania rozmytego dwu (lub więcej) wejściowego. Należy wykonać zadanie w krokach i opisać je w opracowaniu:

1. Wybrać problem i zidentyfikować dane wejściowe i wyjście. Opisać jaką ekspertyzę będzie wykonywać system.

2. Określić dziedzinę (zakres przyjmowanych wartości) dla zmiennych wejściowych i wyjściowych systemu.

3. Wybrać nazwy/etykiety lingwistyczne dla zbiorów rozmytych dla zmiennych wejściowych i wyjścia. UWAGA: jeżeli zastosowany zostanie automat nazwy będą nadane automatycznie. Nawet przy wyborze automatu wartości lingwistyczne powinny zostać opisane w sprawozdaniu. Przedstawić w sprawozdaniu wykresy zmiennych wejściowych i wyjścia.

4. Opracować reguły wiążące wejścia z wyjściem (W sprawozdaniu opisać je słownie).

5. Zbudować model z użyciem biblioteki scikit-fuzzy (tylko w kodzie).

6. Przetestować model na kilku różnych wartościach wejściowych. Przedstawić wyniki i opisać w sprawozdaniu.

7. Wygenerować wykres 3D płaszczyzny odpowiedzi. UWAGA: Jeżeli zostanie wybrane więcej niż w wejścia, to wykres 3D można wykreślić tylko parami pomiędzy wejściami. Ocenić poprawność modelu analizując zależności wejścia-wyjście. Napisać krótko w sprawozdaniu jak zachowuje się model dla małych i dużych wartości wejść.

8. Jeżeli to konieczne (kratery w modelu 3D lub niewłaściwy kierunek zależności), dokonać poprawek, tak bo model może być niezgodny z oczekiwaniami. W takim przypadku krótko opisać poprawki w sprawozdaniu, lub wskazać, że poprawki nie były potrzebne.

Sprawozdanie i kod implementujący własny system wnioskowania rozmytego proszę podłączyć w Teams. Proszę zadbać, by sprawozdanie i kod zawierały nazwisko autorki/a. Sprawozdanie może być przygotowane w formie notatnika w Jupyter Notebook (format .ipynb)