

Wstęp do Sztucznej Inteligencji - rok akademicki 2023/2024

Przed rozpoczęciem pracy z notatnikiem zmień jego nazwę zgodnie z wzorem:
NrAlbumu_Nazwisko_Imie_PoprzedniaNazwa.

Przed wystąpieniem notatnika upewnij się, że rozwiązałeś wszystkie zadania/ćwiczenia.

Temat: Wnioskowanie oparte na wiedzy niepewnej. Wnioskowanie rozmyte

Zapoznaj się z treścią niniejszego notatnika czytając i wykonując go komórka po komórce. Wykonaj napotkane zadania/ćwiczenia.

Na niniejszych laboratoriach będziemy wykorzystywać bibliotekę `fuzzython` (<https://github.com/yudivian/fuzzython>). Oryginalny moduł zawierał kilka błędów przez co nie dało się go zainstalować za pomocą `pip`. Aby móc z niego skorzystać należy pobrać zmodyfikowany moduł `fuzzython.zip` (plik dostępny na delcie), rozpakować go i wrzucić na dysk google, a następnie wykonać poniższą komórkę montującą dysk google:

```
import sys
from google.colab import drive
drive.mount('/content/drive')
# upewnij się że poniższa ścieżka jest poprawna
path_nb = r'/content/drive/My Drive/Colab
Notebooks/WdSI_2023/T4_Fuzzy'
sys.path.append(path_nb)
```

Przy importowaniu elementów z modułu `fuzzython` używamy konstrukcji:

```
from fuzzython.nazwa_podmodulu import jakies, funkcje, klasy, itp.
```

Zadanie 1 (obowiązkowe, 5pkt)

Zaprojektuj system rozmyty typu Mamdani, który będzie oceniał stopień przekonania o możliwości spowodowania wypadku podczas jazdy samochodem.

Zmienne wejściowe:

- prędkość jazdy (10 – 200km/h): mała, średnia, szybka, bardzo szybka
- widoczność (0.05 – 4km): bardzo słaba, średnia, dobra.

Wyjście systemu:

- stopień przekonania o możliwości spowodowania wypadku (0 – 1): **bardzo małe, małe, średnie, duże.**

Zdefiniuj własne kształty dla zbiorów rozmytych, stwórz własne reguły oraz przygotuj wykresy ilustrujące funkcje przynależności zbiorów rozmytych jak również powierzchnię decyzyjną sterownika rozmytego.

TWÓJ PROGRAM:

#YOUR CODE HERE

```
from fuzzython.adjective import Adjective
from fuzzython.systems.mamdani import MamdaniSystem
from fuzzython.ruleblock import RuleBlock
from fuzzython.variable import Variable
from fuzzython.fsets.triangular import Triangular
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np

speedLow = Triangular((-0.1, 0.0), (10, 1), (30, 0))
speedMedium = Triangular((10, 0), (30, 1), (90, 0))
speedHigh = Triangular((30.0, 0), (90, 1), (150, 0))
speedpriceYesterdayHigh = Triangular((90, 0.0), (150, 1), (200.1, 1))
adjectiveSpeedLow = Adjective('speedLow', speedLow)
adjectiveSpeedMedium = Adjective('speedMedium', speedMedium)
adjectiveSpeedHigh = Adjective('speedHigh', speedHigh)
adjectiveSpeedpriceYesterdayHigh =
Adjective('speedpriceYesterdayHigh', speedpriceYesterdayHigh)
speed = Variable('speed', 'kmh', adjectiveSpeedLow,
adjectiveSpeedMedium, adjectiveSpeedHigh,
adjectiveSpeedpriceYesterdayHigh)

visionLow = Triangular((0.04, 1), (1, 1), (2, 0.0))
visionMedium = Triangular((1, 0), (2, 1), (4, 0))
visionHigh = Triangular((2.0, 0), (4, 1), (4.1, 1))
adjectiveVisionLow = Adjective('visionLow', visionLow)
adjectiveVisionMedium = Adjective('visionMedium', visionMedium)
adjectiveVisionHigh = Adjective('visionHigh', visionHigh)
visible = Variable('visible', 'km', adjectiveVisionLow,
adjectiveVisionMedium, adjectiveVisionHigh)

crashLow = Triangular((-0.1, 0), (0, 1), (50, 0))
crashMedium = Triangular((0,0), (45, 1), (66, 0))
crashHigh = Triangular((25,0), (75,1), (100, 0))
crashVisionHigh = Triangular((70, 0), (82, 1), (100.1, 1))
adjectiveCrashLow = Adjective('crashLow', crashLow)
adjectiveCrashMedium = Adjective('crashMedium', crashMedium)
adjectiveCrashHigh = Adjective('crashHigh', crashHigh)
adjectiveCrashVisionHigh = Adjective('crashVisionHigh',
```

```

crashVisionHigh)
crash = Variable('crash', '%', adjectiveCrashLow,
adjectiveCrashMedium, adjectiveCrashHigh, adjectiveCrashVisionHigh,
defuzzification='COG', default = 0)

scope = locals()

rule1 = 'if speed is adjectiveSpeedpriceYesterdayHigh and visible is
adjectiveVisionLow then crash is adjectiveCrashVisionHigh'
rule2 = 'if speed is adjectiveSpeedLow or visible is
adjectiveVisionHigh then crash is adjectiveCrashLow'
rule3 = 'if speed is adjectiveVisionHigh or visible is
adjectiveVisionLow then crash is adjectiveCrashHigh'
rule4 = 'if visible is adjectiveVisionMedium or speed is
adjectiveSpeedMedium then crash is adjectiveCrashMedium'
rule5 = 'if visible is adjectiveCrashLow and speed is
adjectiveSpeedLow then crash is adjectiveCrashMedium'
rule6 = 'if visible is adjectiveVisionHigh and speed is
adjectiveSpeedLow crash is adjectiveCrashLow'

block = RuleBlock('rb_mamdani', operators=('MIN', 'MAX', 'ZADEH'),
activation='MIN', accumulation='MAX')
block.add_rules(rule1, rule2, rule3, rule4, rule5, scope=scope)

mamdani = MamdaniSystem('mamdani_model', block)

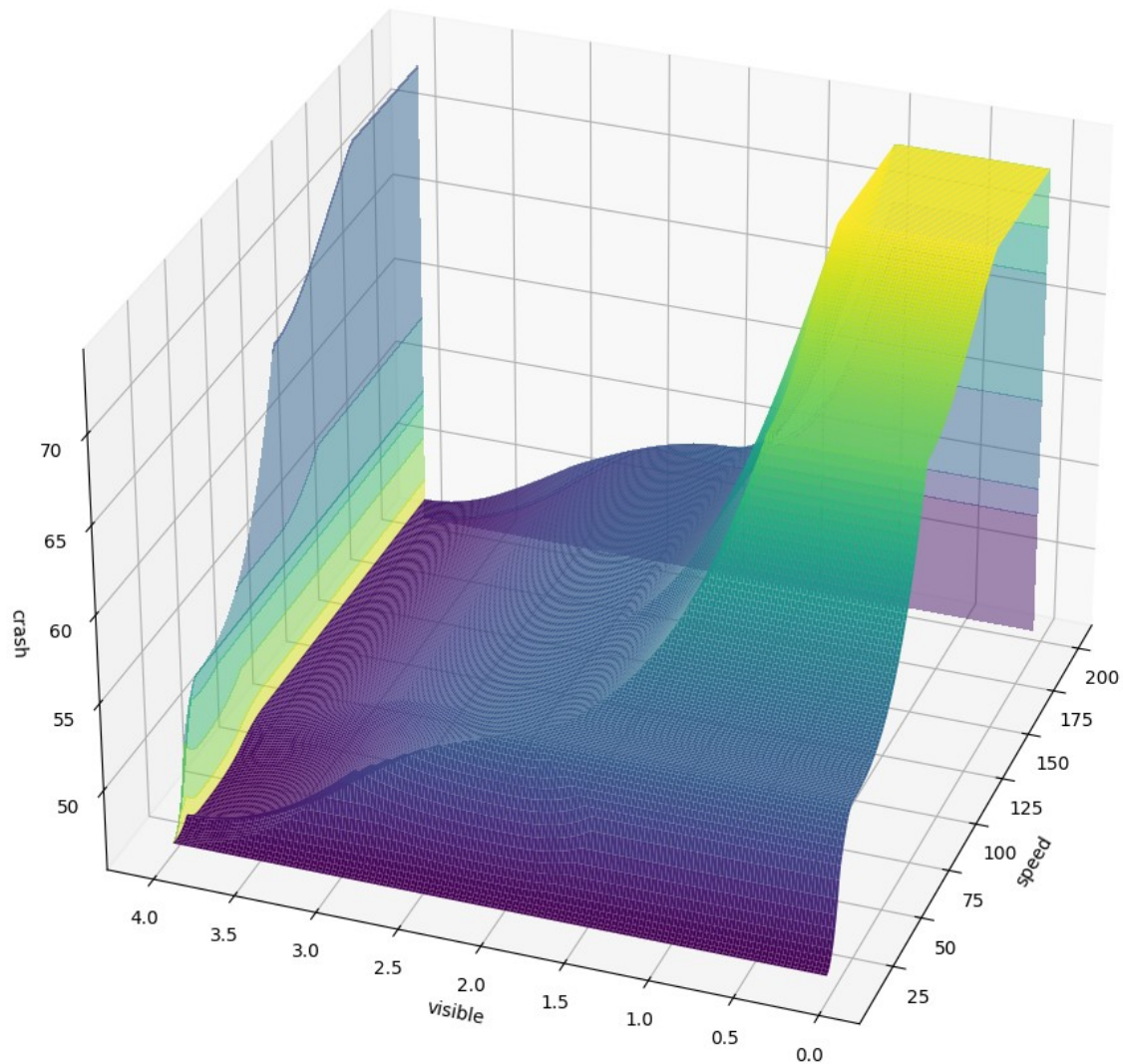
sampledx = np.linspace(10, 200, 200)
sampledy = np.linspace(0.05, 4, 200)
x, y = np.meshgrid(sampledx, sampledy)
z = np.zeros((len(sampledx), len(sampledy)))

for i in range(len(sampledx)):
    for j in range(len(sampledy)):
        inputs = {'speed': x[i, j], 'visible': y[i, j]}
        res = mamdani.compute(inputs)
        z[i, j] = res['rb_mamdani']['crash']

fig = plt.figure(figsize=(12,12))
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap='viridis',
linewidth=0.4, antialiased=True)
cset = ax.contourf(x, y, z, zdir='z', offset= 0, cmap='viridis',
alpha=0.5)
cset = ax.contourf(x, y, z, zdir='x', offset= 200, cmap='viridis',
alpha=0.5)
cset = ax.contourf(x, y, z, zdir='y', offset= 4, cmap='viridis',
alpha=0.5)
ax.set_xlabel('speed')
ax.set_ylabel('visible')

```

```
ax.set_zlabel('crash')  
ax.view_init(30, 200)
```



Zadanie 2 (obowiązkowe, 5pkt)

Zaprojektuj system rozmyty typu Takagi-Sugeno, który będzie prognozował cenę akcji na jeden dzień do przodu.

Zmienne wejściowe:

- cena akcji wczoraj (1 – 20zł): niska, średnia, wysoka,

- cena akcji dziś (1 – 20zł): niska, średnia, wysoka.

Wyjście systemu:

- cena akcji jutro dana równaniem regresji: $\alpha \cdot x + \beta \cdot y + \gamma$

Reguły postaci:

- Jeśli cena akcji wczoraj była A_i i cena akcji dziś jest B_i to cena akcji jutro będzie $\alpha_k \cdot x + \beta_k \cdot y + \gamma_k$

gdzie:

- x to cena akcji wczoraj (ostra wartość)
- y to cena akcji dziś (ostra wartość)
- A_i, B_i to zbiory rozmyte odpowiednio dla ceny akcji wczoraj i dziś
- $\alpha_k, \beta_k, \gamma_k$ - współczynniki w równaniu regresji (liczby rzeczywiste)
- k - numer reguły

Zdefiniuj własne kształty dla zbiorów rozmytych, stwórz własne reguły (dobór współczynników regresji) oraz przygotuj wykresy ilustrujące funkcje przynależności zbiorów rozmytych jak również powierzchnię decyzyjną sterownika rozmytego.

TWÓJ PROGRAM:

YOUR CODE HERE

```
from fuzzython.systems.sugeno import SugenoSystem

priceYesterdayLow = Triangular((0.9, 0.0), (5, 1), (10, 0))
priceYesterdayMedium = Triangular((1, 0.0), (10, 1), (20, 0))
priceYesterdayHigh = Triangular((10, 0.0), (20, 1), (20.1, 1))
adjectivepriceYesterdayLow = Adjective('priceYesterdayLow',
priceYesterdayLow)
adjectivepriceYesterdayMedium = Adjective('priceYesterdayMedium',
priceYesterdayMedium)
adjectivepriceYesterdayHigh = Adjective('priceYesterdayHigh',
priceYesterdayHigh)
yesterday = Variable('yesterday', 'zl', adjectivepriceYesterdayLow,
adjectivepriceYesterdayMedium, adjectivepriceYesterdayHigh)

priceTodayLow = Triangular((0.9, 0.0), (5, 1), (10, 0))
priceTodayMedium = Triangular((1, 0.0), (10, 1), (20, 0))
priceTodayHigh = Triangular((10, 0.0), (20, 1), (20.1, 1))
adjectivePriceTodayLow = Adjective('priceYesterdayLow',
priceYesterdayLow)
adjectivePriceTodayMedium = Adjective('priceYesterdayMedium',
priceYesterdayMedium)
```

```

adjectivePriceTodayHigh = Adjective('priceYesterdayHigh',
priceYesterdayHigh)
today = Variable('today', 'z1', adjectivePriceTodayLow,
adjectivePriceTodayMedium, adjectivePriceTodayHigh)

scope = locals()

rule1 = 'if yesterday is adjectivepriceYesterdayLow and today is
adjectivePriceTodayLow then z=yesterday*0.1+today*0.2+2'
rule2 = 'if yesterday is adjectivepriceYesterdayHigh and today is
adjectivePriceTodayHigh then z=yesterday*0.7+today*0.8+3.4'
rule3 = 'if yesterday is adjectivepriceYesterdayLow and today is
adjectivePriceTodayHigh then z=yesterday*0.3+today*0.3+4.5'
rule4 = 'if yesterday is adjectivepriceYesterdayHigh and today is
adjectivePriceTodayLow then z=yesterday*0.1+today*0.2+1.1'
rule5 = 'if yesterday is adjectivepriceYesterdayMedium or today is
adjectivePriceTodayMedium then z=yesterday*0.3+today*0.3+2.2'

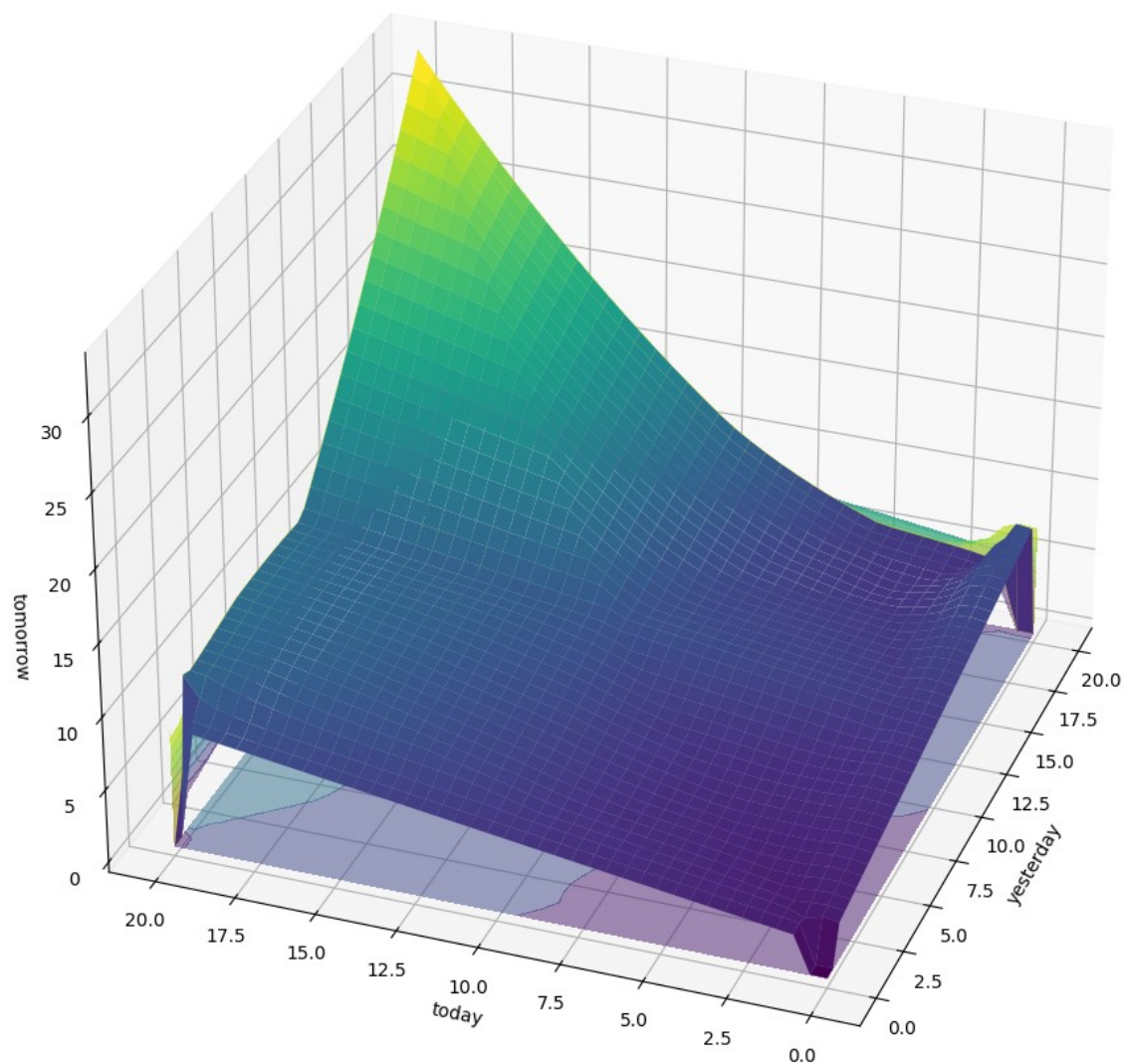
block = RuleBlock('rb_takagi', operators=('MIN', 'MAX', 'ZADEH'),
activation='MIN', accumulation='MAX')
block.add_rules(rule1, rule2, rule3, rule4, rule5, scope=scope)
sugeno = SugenoSystem('model_takagi', block)

sampled = np.linspace(0, 20, 40)
x, y = np.meshgrid(sampled, sampled)
z = np.zeros((len(sampled),len(sampled)))

for i in range(len(sampled)):
    for j in range(len(sampled)):
        inputs = {'yesterday': x[i, j], 'today': y[i, j]}
        res = sugeno.compute(inputs)
        z[i, j] = res['rb_takagi']

fig = plt.figure(figsize=(12,12))
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap='viridis',
linewidth=0.4, antialiased=True)
cset = ax.contourf(x, y, z, zdir='z', offset= 0, cmap='viridis',
alpha=0.5)
cset = ax.contourf(x, y, z, zdir='x', offset= 20, cmap='viridis',
alpha=0.5)
cset = ax.contourf(x, y, z, zdir='y', offset= 20, cmap='viridis',
alpha=0.5)
ax.set_xlabel('yesterday')
ax.set_ylabel('today')
ax.set_zlabel('tomorrow')
ax.view_init(30, 200)

```

© Katedra Informatyki, Politechnika Krakowska