Intelligent Systems - Fuzzy Logic

Lars Grit

December 3, 2025

Abstract

Fuzzy logic biedt een manier om menselijke redenering te modelleren in systemen waar onzekerheid en ruis bestaat. Met Python-gebaseerd fuzzy logic framework (`fuzzylogic`) en een voorbeeld uit Negnevitsky's *Artificial Intelligence: A Guide to Intelligent Systems*. Bouwen wij het volledige *Spare Parts Service Centre* expertsystem.

1 De Demo: Implementatie van Hoofdstuk 4.7

In onderstaande code implementeren we volledig het fuzzy expert system uit Sectie 4.7. De inputs zijn genormaliseerd naar het interval $[0, 1]$ en de rule-base bestaat uit 27 regels zoals beschreven in de literatuur.

1.1 Code

```
from fuzzylogic.classes import Domain, Set, Rule
from fuzzylogic.functions import S, R # S = left shoulder, R = right shoulder (as in
    your example)
from fuzzylogic.hedges import very
from typing import Any


mean_delay = Domain("Mean␣delay␣m", 0.0, 1.0, res=0.001) # VS, S, M
servers = Domain("Number␣of␣servers␣s", 0.0, 1.0, res=0.001) # S, M, L
util = Domain("Utilisation␣factor␣", 0.0, 1.0, res=0.001) # L, M, H
spares = Domain("Number␣of␣spares␣n", 0.0, 1.0, res=0.001) # VS, S, RS, M, RL, L, VL


def _to_set(x: Any) -> Set:
    return x if isinstance(x, Set) else Set(x)

def middle(left: Any, right: Any) -> Set:
    return _to_set(left) & _to_set(right)

mean_delay.VS = S(0.0, 0.3)
mean_delay.M = R(0.4, 0.7)
mean_delay.S = middle(R(0.1, 0.3), S(0.3, 0.5))

servers.S = S(0.0, 0.35)
servers.L = R(0.60, 1.0)
servers.M = middle(R(0.30, 0.50), S(0.50, 0.70))

util.L = S(0.0, 0.6)
util.H = R(0.6, 1.0)
```

```
util.M = middle(R(0.4, 0.6), S(0.6, 0.8))

spares.VS = S(0.0, 0.30)
spares.VL = R(0.70, 1.0)
spares.S = S(0.0, 0.40)
spares.L = R(0.60, 1.0)
spares.RS = middle(R(0.25, 0.35), S(0.35, 0.45))
spares.M = middle(R(0.30, 0.50), S(0.50, 0.70))
spares.RL = middle(R(0.55, 0.65), S(0.65, 0.75))


rules = Rule({
    (mean_delay.VS, servers.S, util.L): spares.VS,
    (mean_delay.S, servers.S, util.L): spares.VS,
    (mean_delay.M, servers.S, util.L): spares.VS,

    (mean_delay.VS, servers.M, util.L): spares.VS,
    (mean_delay.S, servers.M, util.L): spares.VS,
    (mean_delay.M, servers.M, util.L): spares.VS,

    (mean_delay.VS, servers.L, util.L): spares.S,
    (mean_delay.S, servers.L, util.L): spares.S,
    (mean_delay.M, servers.L, util.L): spares.VS,

    (mean_delay.VS, servers.S, util.M): spares.S,
    (mean_delay.S, servers.S, util.M): spares.VS,
    (mean_delay.M, servers.S, util.M): spares.VS,

    (mean_delay.VS, servers.M, util.M): spares.RS,
    (mean_delay.S, servers.M, util.M): spares.S,
    (mean_delay.M, servers.M, util.M): spares.VS,

    (mean_delay.VS, servers.L, util.M): spares.M,
    (mean_delay.S, servers.L, util.M): spares.RS,
    (mean_delay.M, servers.L, util.M): spares.S,

    (mean_delay.VS, servers.S, util.H): spares.VL,
    (mean_delay.S, servers.S, util.H): spares.L,
    (mean_delay.M, servers.S, util.H): spares.M,

    (mean_delay.VS, servers.M, util.H): spares.M,
    (mean_delay.S, servers.M, util.H): spares.M,
    (mean_delay.M, servers.M, util.H): spares.S,

    (mean_delay.VS, servers.L, util.H): spares.RL,
    (mean_delay.S, servers.L, util.H): spares.M,
    (mean_delay.M, servers.L, util.H): spares.RS,
})

if __name__ == "__main__":

    values = {
        mean_delay: 0.25,
        servers: 0.50,
        util: 0.70,
    }

    result = rules(values)
```

```
print(f"Recommended number of spares: {result:.3f}")
```

1.2 Resultaat

Bij de voorbeeldinput geeft het systeem:

```
Recommended normalised spares:  0.307
```

Dit komt overeen met een lage, maar niet minimale voorraadniveaus, precies zoals beschreven in het oorspronkelijke hoofdstuk.

2 Evaluatie van het framework

2.1 Is het simpel te gebruiken?

Ja. Met het framework kan je makkelijk domains, fuzzy sets en rules definiëren. De terminologie volgt ook de fuzzy literatuur.

2.2 Is de code begrijpelijk?

De structuur is duidelijk:

1. Definieer de domains
2. Definieer de membership functions
3. Definieer de rules
4. Voer een defuzzificatie uit

2.3 Is het snel?

Ja. De inference werkt met max-min aggregatie en numerieke intergratie voor defuzz.

2.4 Is de onderliggende code begrijpelijk ?

Ja en nee. Het gebruik van `S`-, `R`-functies en `very()`-hedges volgt traditionele fuzzy logic literatuur. Wel zijn alle functions recursive closures, de main classes gebruiken veel dunder methoden.

2.5 Is grafische output mogelijk?

Niet ingebouwd, en ook niet gebruikt maar wel mogelijk met matplotlib of andere plotting libraries.

2.6 Is het veilig voor robotica?

Voor niet-safety-critical toepassingen wel. De code is niet geoptimaliseerd voor real-time performance.