

Analiza Techniczna Systemu Ekspertowego BrewSense

Analiza Kodu Źródłowego i Logiki Rozmytej

16 grudnia 2025

Streszczenie

Niniejszy dokument stanowi szczegółową analizę techniczną projektu BrewSense – systemu oceny jakości kawy opartego na logice rozmytej (ang. *fuzzy logic*). Analiza koncentruje się na wewnętrznej architekturze oprogramowania, implementacji algorytmów sterowania rozmytego przy użyciu biblioteki `scikit-fuzzy`, strukturach danych oraz przepływie informacji pomiędzy warstwą prezentacji a warstwą logiki biznesowej.

Spis treści

1 Architektura Systemu	1
1.1 Przepływ Danych	1
2 Analiza Modułu Logiki Rozmytej (fuzzy_system.py)	2
2.1 Zmienne Lingwistyczne i Przestrzenie Rozważenia	2
2.2 Funkcje Przynależności (Membership Functions)	2
2.3 Baza Reguł i Wnioskowanie	2
2.3.1 Struktura Reguł	2
2.3.2 Mechanizm Catch-All	3
2.4 Symulacja i Defuzzification	3
3 Analiza Warstwy Prezentacji (gui.py)	3
3.1 Integracja z Systemem Rozmytym	3
3.2 Profile Kawy	3
3.3 Wizualizacja (Custom Widget)	3
4 Podsumowanie Techniczne	4

1 Architektura Systemu

Projekt BrewSense został zrealizowany w języku Python, a jego architektura odzwierciedla klasyczny wzorzec separacji logiki od interfejsu użytkownika. Możemy wyróżnić trzy główne komponenty (pliki):

1. `main.py` – Punkt wejścia (Bootstrap).
2. `gui.py` – Warstwa Prezentacji (View/Controller). Odpowiada za akwizycję danych od użytkownika oraz wizualizację wyników.
3. `fuzzy_system.py` – Warstwa Logiki (Model). Zawiera silnik wnioskowania rozmytego.

1.1 Przepływ Danych

Globalny przepływ danych w systemie wygląda następująco:

1. Użytkownik manipuluje suwakami (Sliders) w interfejsie graficznym.
2. Wartości surowe (np. liczby całkowite) są normalizowane do zakresów zmiennych lingwistycznych.
3. Znormalizowane dane trafiają do metody `evaluate()` w klasie `CoffeeQualitySystem`.

4. Następuje proces **rozmywania** (fuzzification), **wnioskowania** (inference) na podstawie bazy reguł oraz **agregacji**.
5. Wynik jest poddawany **wyostrzaniu** (defuzzification) do jednej wartości skalarnej (ocena 0-100).
6. Wynik wraca do GUI, gdzie steruje parametrami wizualizacji (np. kolorem cieczy w wirtualnym kubku).

2 Analiza Modułu Logiki Rozmytej (fuzzy_system.py)

Plik ten jest sercem systemu. Implementuje on klasę `CoffeeQualitySystem`, która wykorzystuje bibliotekę `scikit-fuzzy` (importowaną jako `skfuzzy` oraz `control`).

2.1 Zmienne Lingwistyczne i Przestrzenie Rozważan

System definiuje cztery zmienne wejściowe (Antecedents) oraz jedną zmienną wyjściową (Consequent). Przestrzenie rozważan (Universes of Discourse) są zdefiniowane za pomocą `numpy.arange`.

Zmienna	Typ	Zakres	Opis
<code>bitterness</code>	Wejściowa	0.0 – 10.0	Poziom goryczy
<code>acidity</code>	Wejściowa	0.0 – 10.0	Poziom kwasowości
<code>aroma</code>	Wejściowa	0.0 – 10.0	Intensywność aromatu
<code>temperature</code>	Wejściowa	60.0 – 95.0	Temperatura parzenia (°C)
<code>quality</code>	Wyjściowa	0.0 – 100.0	Ocena końcowa jakości

Tabela 1: Definicja zmiennych systemu

W kodzie realizowane jest to poprzez instrukcje:

```
1 self.bitterness = ctrl.Antecedent(np.arange(0, 10.1, 0.1), 'bitterness')
2 # ...
3 self.quality = ctrl.Consequent(np.arange(0, 100.1, 0.1), 'quality', defuzzify_method='
  centroid')
```

2.2 Funkcje Przynależności (Membership Functions)

System wykorzystuje dwa rodzaje funkcji przynależności:

- **Trójkątne (trimf)** – dla wartości środkowych (np. 'medium', 'optimal').
- **Trapezowe (trapmf)** – dla wartości brzegowych, aby zapewnić pełną przynależność (wartość 1.0) na krańcach zakresów.

Przykład implementacji dla temperatury:

```
1 # Temperatura (Temperature)
2 self.temperature['low'] = fuzz.trapmf(self.temperature.universe, [60, 60, 70, 75])
3 self.temperature['optimal'] = fuzz.trimf(self.temperature.universe, [72, 80, 88])
4 self.temperature['high'] = fuzz.trapmf(self.temperature.universe, [85, 90, 95, 95])
```

Użycie `trapmf` dla wartości 'low' ([60, 60, 70, 75]) oznacza, że dla temperatur od 60 do 70 stopni przynależność do zbioru 'low' wynosi 1.0, a następnie spada liniowo do 0 przy 75 stopniach.

2.3 Baza Reguł i Wnioskowanie

Metoda `_create_rules` definiuje bazę wiedzy eksperckiej. System zawiera ponad 40 reguł.

2.3.1 Struktura Reguł

Reguły są tworzone przy użyciu obiektu `ctrl.Rule`. Operator **AND** jest realizowany w bibliotece `scikit-fuzzy` domyślnie jako minimum (t-norma min).

Przykład reguły dla "Wybitnej Kawy":

```

1 ctrl.Rule(self.aroma['strong'] & self.bitterness['medium'] &
2           self.acidity['medium'] & self.temperature['optimal'],
3           self.quality['excellent'])

```

Oznacza to matematycznie:

$$\mu_{excellent}(y) = \min(\mu_{strong}(x_1), \mu_{medium}(x_2), \mu_{medium}(x_3), \mu_{optimal}(x_4))$$

2.3.2 Mechanizm Catch-All

Autor kodu zaimplementował tzw. reguły "fallback" oraz "catch-all" (na końcu listy reguł), które obsługują przypadki brzegowe lub nietypowe kombinacje, zapobiegając sytuacji, w której żadna reguła nie jest aktywna. Przykładowo, słaby aromat (`weak`) niemal zawsze degraduje jakość, niezależnie od pozostałych parametrów.

2.4 Symulacja i Defuzzification

W metodzie `_create_control_system`:

```

1 self.control_system = ctrl.ControlSystem(self.rules)
2 self.simulator = ctrl.ControlSystemSimulation(self.control_system)

```

Obiekt `ControlSystemSimulation` jest "maszyną", która przetwarza konkretne dane wejściowe.

Metoda `defuzzification` została jawnie ustawiona na '`centroid`' (metoda środka ciężkości). Oblicza ona środek ciężkości pola pod krzywą wynikową funkcji przynależności zbioru wyjściowego.

$$z^* = \frac{\int \mu_{agg}(z) \cdot z \, dz}{\int \mu_{agg}(z) \, dz}$$

W metodzie `evaluate`: 1. Wprowadzane są dane (`self.simulator.input[...] = ...`). 2. Wywoływane jest obliczenie (`self.simulator.compute()`). 3. Pobierany jest wynik (`self.simulator.output['quality']`).

3 Analiza Warstwy Prezentacji (gui.py)

Interfejs graficzny zbudowany jest w oparciu o bibliotekę **PyQt5**. Plik ten nie tylko wyświetla okna, ale pełni rolę kontrolera pośredniczącego.

3.1 Integracja z Systemem Rozmytym

Wewnątrz klasy głównego okna następuje inicjalizacja systemu:

```

1 try:
2     from fuzzy_system import CoffeeQualitySystem
3 except ImportError:
4     # ... Klasa Dummy dla test w ...

```

Zastosowano tu mechanizm zabezpieczający (blok `try-except`), który pozwala uruchomić GUI nawet w przypadku błędu importu logiki (np. brak biblioteki `scikit-fuzzy`), podstawiając atrapę (mock).

3.2 Profile Kawy

Zdefiniowano słownik `COFFEE_PROFILES`, który mapuje nazwy napojów (np. "Espresso Italiano", "Cold Brew") na konkretne zestawy parametrów. Wybór profilu z listy rozwijanej (QComboBox) automatycznie ustawia suwaki na zdefiniowane wartości, co natychmiast wyzwala przeliczenie jakości przez system rozmyty.

3.3 Wizualizacja (Custom Widget)

Klasa `CoffeeVisualizer` to niestandardowy widget Qt (`QWidget`), który nadpisuje metodę `paintEvent`.

- Animacja:** Wykorzystano `QPropertyAnimation` do płynnej zmiany poziomu cieczy w kubku (`fillLevel`).
- Mapowanie Jakości na Kolor:** Metoda `_get_coffee_color` zmienia barwę cieczy w zależności od zwróconego przez system rozmyty wyniku `quality` (od jasnobrązowego "lury" po głęboki, ciemny brąz dla wysokiej jakości).

- **Para:** Jeśli parametr wejściowy temperatury przekracza 75°C , rysowana jest animowana para nad kubkiem. Jest to bezpośrednie wizualne sprzężenie zwrotne jednej ze zmiennych wejściowych systemu rozmytego.

4 Podsumowanie Techniczne

Projekt **BrewSense** jest przykładem poprawnej implementacji systemu sterowania rozmytego typu Mam-dani.

- **Logika:** Opiera się na deterministycznych regułach lingwistycznych, które są przetwarzane w sposób ciągły (niebinarny). Pozwala to na ocenę niuansów, np. "kawa trochę za gorzka, ale o idealnej temperaturze".
- **Kod:** Użycie **scikit-fuzzy** zwalnia programistę z ręcznej implementacji matematyki zbiorów rozmytych, pozwalając skupić się na definicji funkcji przynależności i reguł.
- **Interakcja:** GUI nie zawiera własnej logiki biznesowej oceny kawy – jest jedynie "terminalem" do wprowadzania danych do silnika rozmytego i wyświetlania jego wyników.

System jest skalowalny – dodanie nowego parametru (np. "stopień zmielenia") wymagałoby jedynie dodania nowej zmiennej **Antecedent**, zdefiniowania dla niej funkcji przynależności i aktualizacji bazy reguł w pliku **fuzzy_system.py**.