Tema 3 Sistem de nutriție și fitness bazat pe fapte și reguli

Sisteme Semantice

Draghici Andreea-Maria
Inginerie Software
IS 2.1

Contents

1. Introducere

Acest proiect implementează un sistem pentru a oferi recomandări nutriționale și de fitness. Sistemul utilizează reguli de producție pentru a deduce concluzii bazate pe un set de fapte inițiale și reguli definite.

2. Metodele de raționament utilizate

Sistemul folosește două metode principale de raționament:

- **Inferență Înainte (Forward Chaining):** Pornește de la faptele inițiale și deduce toate concluziile posibile aplicând regulile disponibile.
- **Inferență Înapoi (Backward Chaining):** Pornește de la o concluzie țintă și verifică dacă aceasta poate fi validată pe baza faptelor și regulilor existente.

3. Descrierea implementării

Implementarea este realizată în Python, având următoarele componente principale:

1. Fapte inițiale: Sistemul pornește cu un set de 20 de fapte care reprezintă condiții sau caracteristici ale utilizatorului, cum ar fi 'sedentar', 'hipertensiune' sau 'stres, etc'. Acestea sunt definite sub formă de dicționar.

2. Reguli de producție: 10 reguli de producție de tip IF-THEN sunt utilizate pentru a defini relațiile dintre condițiile inițiale și concluziile ce pot fi deduse, utilizând inferență înainte și înapoi.

3. Motor de inferență: Motorul de inferență aplică regulile folosind cele două metode de raționament descrise anterior.

4. Explicații: Fiecare concluzie dedusă este însoțită de o explicație detaliată despre cum a fost obținută.

4. Concluziile deducerii

Sistemul oferă următoarele concluzii, evidențiind dacă acestea sunt obținute direct din faptele inițiale sau prin intermediul unor concluzii intermediare.

Exemplu (Inferență Înainte):
- Concluzie: 'risc_colesterol_crescut' (CF: 0.9).

Explicație: Regula aplicată: Daca ['sedentar', 'consum_grasimi_saturate'] atunci risc_colesterol_crescut. Concluzie obținută direct din faptele inițiale.

- Concluzie: 'recomandare_reducere_grasimi_exercitii' (CF: 0.8).

Explicație: Regula aplicată: Daca ['risc_colesterol_crescut', 'hipertensiune'] atunci recomandare_reducere_grasimi_exercitii. Concluzie obținută prin concluzii intermediare.

Exemplu (Inferență Înapoi):

- Interogare: 'recomandare_dieta_fitness'.

Rezultat: Adevarat (CF: 0.85).

Explicație: Regula aplicată: Daca ['doreste_pierdere_in_greutate', 'imc_peste_30'] atunci recomandare_dieta_fitness. Concluzie obținută direct din faptele inițiale.

5. Simulare sistem

Inferența înainte pornește de la un set de **fapte inițiale** și aplică regulile de producție pentru a deduce noi **concluzii**. Sistemul utilizează condițiile specificate în fiecare regulă pentru a valida și genera concluzii, care la rândul lor pot deveni condiții pentru alte reguli.

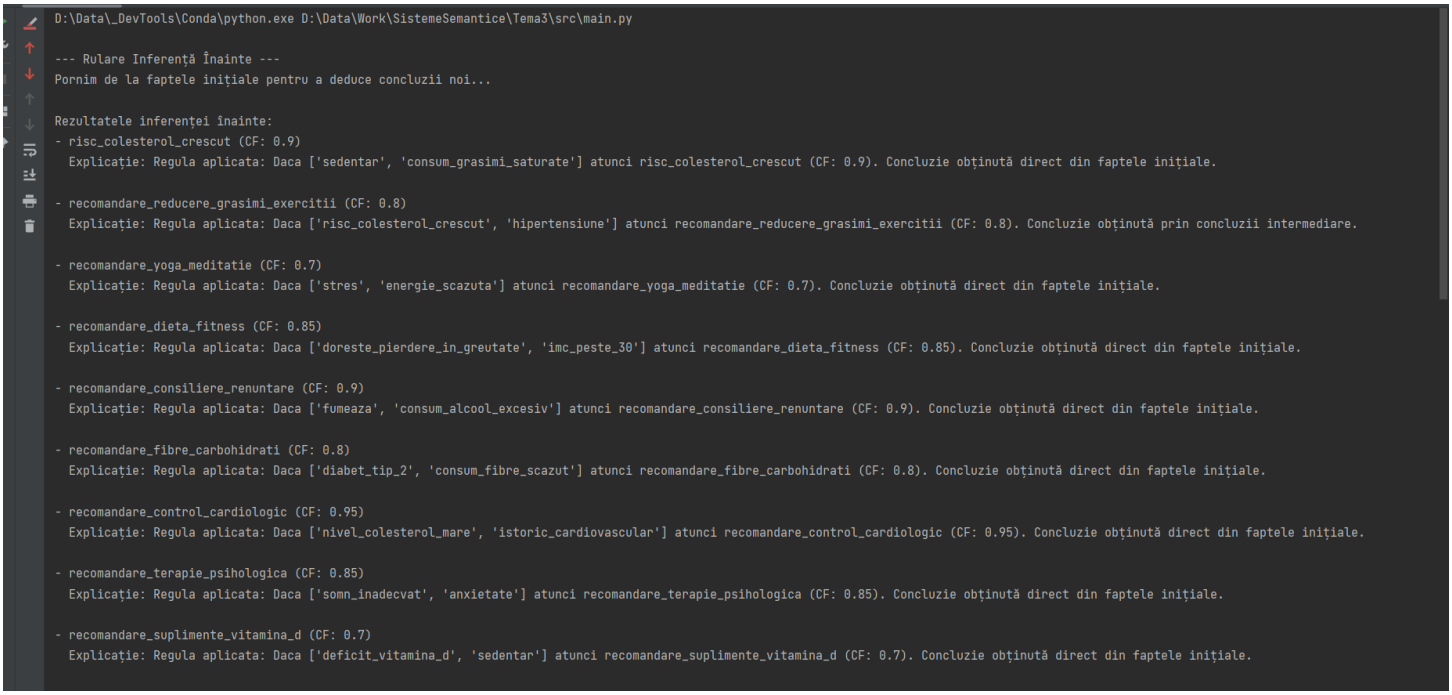
```
# Motor de inferenta: Forward chaining
def inlantuire_inainte(self) :
    reguli_aplicate = True
    while reguli_aplicate :
        reguli_aplicate = False
        for regula in self.reguli :
            if all(self.fapte.get(conditie, False) or conditie in self.fapte_deduse for conditie in
regula["if"]) :
                concluzie = regula["then"]
                if concluzie not in self.fapte_deduse :
                    # Deduce conclusion
                    self.fapte_deduse[concluzie] = regula["cf"]
                    reguli_aplicate = True
                    # Check if conclusion is derived directly or via intermediates
                    direct = all(cond in self.fapte for cond in regula["if"])
                    tip = "direct din faptele inițiale" if direct else "prin concluzii intermediare"
                    self.explicatii[
                        concluzie] = f"Regula aplicata: Daca {regula['if']} atunci {concluzie} (CF:
{regula['cf']}). Concluzie obținută {tip}."

# Metoda pentru rularea inferenței înainte
def rulare_inferenta_inainte(self) :
    print("\n--- Rulare Inferență Înainte ---")
```

```
print("Pornim de la faptele inițiale pentru a deduce concluzii noi...\n")
self.inlantuire_inainte()
if self.fapte_deduse :
    print("Rezultatele inferenței înainte:")
    for concluzie in self.fapte_deduse :
        print(f"- {concluzie} (CF: {self.fapte_deduse[concluzie]})")
        print(f"  Explicație: {self.explica(concluzie)}\n")
else :
    print(" Nu s-au putut deduce concluzii noi.")
```

Rezultatele sunt prezentate sub forma:

- **Concluzie dedusă**: Este afișată concluzia obținută, însoțită de un **Factor de Certitudine (CF)**, care indică nivelul de încredere al deducției.
- **Factorul de Certitudine (CF)** este definit direct în regulile de producție și reprezintă un nivel predefinit de încredere asociat fiecărei reguli. Acesta **nu este calculat dinamic** în actuala versiune a sistemului.
- **Explicație**: Este specificată regula aplicată și modul în care concluzia a fost obținută:
    - **Concluzie obținută direct din faptele inițiale**: Regula a fost validată doar pe baza faptelor furnizate inițial.
    - **Concluzie obținută prin concluzii intermediare**: Regula a utilizat atât faptele inițiale, cât și concluzii deduse anterior.



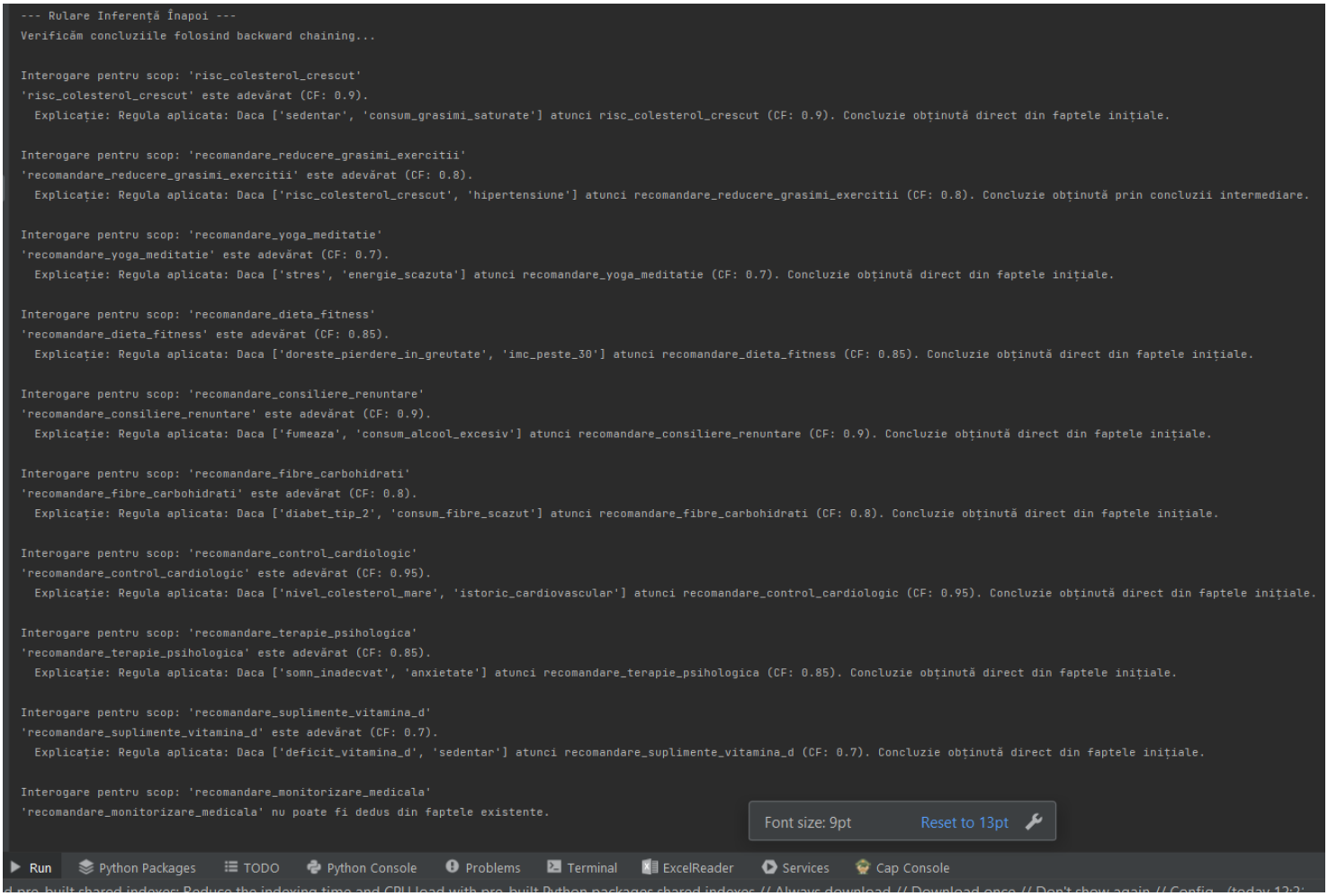
Inferența înapoi funcționează pornind de la o **concluzie țintă** (scop) și verificând dacă există suficiente date pentru a o deduce:

```
# Motor de inferenta: Backward chaining
def inlantuire_inapoi(self, scop) :
    if scop in self.fapte :
        return self.fapte[scop]
    if scop in self.fapte_deduse :
        return self.fapte_deduse[scop]

    for regula in self.reguli :
        if regula["then"] == scop :
            if all(self.inlantuire_inapoi(conditie) for conditie in regula["if"]) :
                self.fapte_deduse[scop] = regula["cf"]
                # Check if conclusion is derived directly or via intermediates
                direct = all(cond in self.fapte for cond in regula["if"])
                tip = "direct din faptele inițiale" if direct else "prin concluzii intermediare"
                self.explicatii[
                    scop] = f"Regula aplicata: Daca {regula['if']} atunci {scop} (CF: {regula['cf']}).
Concluzie obținută {tip}."
                return regula["cf"]
    return False

# Metoda pentru rularea inferenței înapoi
def rulare_inferenta_inapoi(self) :
    print("\n--- Rulare Inferență Înapoi ---")
    print("Verificăm concluziile folosind backward chaining...\n")
    for regula in self.reguli :
        scop = regula["then"]
        print(f"Interogare pentru scop: '{scop}'")
        rezultat = self.inlantuire_inapoi(scop)
        if rezultat :
            print(f"'{scop}' este adevărat (CF: {self.fapte_deduse.get(scop, 'N/A')}).")
            print(f"  Explicație: {self.explica(scop)}\n")
        else :
            print(f"'{scop}' nu poate fi dedus din faptele existente.\n")
```

1. **Pentru fiecare scop**, sistemul:
    - Verifică dacă scopul este direct validat de faptele inițiale.
    - Dacă nu, verifică dacă regulile pot valida scopul prin concluzii intermediare.
2. **Rezultatul**:
    - Sistemul decide dacă scopul este adevărat sau nu pe baza faptelor existente și a regulilor aplicabile.
    - Explicația oferită precizează regula utilizată și originea deducției.



6. Concluzii

Utilizarea inferenței înainte și înapoi permite o abordare flexibilă, în funcție de cerințele utilizatorului. Evidențierea originii concluziilor (directe sau intermediare) oferă o perspectivă clară asupra procesului decizional.

Am ales domeniul sănătății și fitness-ului deoarece este un subiect de interes general, iar deciziile corecte în acest domeniu pot avea un impact pozitiv major asupra vieții utilizatorilor. Universul semantic definit include fapte legate de obiceiurile de viață (cum ar fi fumatul, sedentarismul), condițiile de sănătate (cum ar fi hipertensiunea, diabetul de tip 2) și obiectivele utilizatorului (cum ar fi scăderea în greutate).

7. Anexă: Codul Sursă al Proiectului

```
class SistemNutritieFitness :
    def __init__(self) :
        # 20 Initial Facts
        self.fapte = {
            "sedentar" : True,
            "consum_grasimi_saturate" : True,
            "hipertensiune" : True,
            "stres" : True,
            "doreste_pierdere_in_greutate" : True,
            "energie_scazuta" : True,
            "consum_proteine_scazut" : True,
            "consum_fibre_scazut" : True,
            "imc_peste_30" : True,
            "fumeaza" : True,
            "consum_alcool_excesiv" : True,
            "diabet_tip_2" : True,
            "consum_zahar_ridicat" : True,
            "exercitii_regulate" : False,
            "somn_inadecvat" : True,
            "nivel_colesterol_mare" : True,
            "anxietate" : True,
            "varsta_peste_50" : False,
            "istoric_cardiovascular" : True,
            "deficit_vitamina_d" : True
        }

        # 10 Rules
        self.reguli = [
            {"if" : ["sedentar", "consum_grasimi_saturate"], "then" : "risc_colesterol_crescut", "cf" : 0.9},
            {"if" : ["risc_colesterol_crescut", "hipertensiune"], "then" : "recomandare_reducere_grasimi_exercitii",
             "cf" : 0.8},
            {"if" : ["stres", "energie_scazuta"], "then" : "recomandare_yoga_meditatie", "cf" : 0.7},
            {"if" : ["doreste_pierdere_in_greutate", "imc_peste_30"], "then" : "recomandare_dieta_fitness",
             "cf" : 0.85},
            {"if" : ["fumeaza", "consum_alcool_excesiv"], "then" : "recomandare_consiliere_renuntare", "cf" : 0.9},
            {"if" : ["diabet_tip_2", "consum_fibre_scazut"], "then" : "recomandare_fibre_carbohidrati", "cf" : 0.8},
            {"if" : ["nivel_colesterol_mare", "istoric_cardiovascular"], "then" : "recomandare_control_cardiologic",
             "cf" : 0.95},
            {"if" : ["somn_inadecvat", "anxietate"], "then" : "recomandare_terapie_psihologica", "cf" : 0.85},
            {"if" : ["deficit_vitamina_d", "sedentar"], "then" : "recomandare_suplimente_vitamina_d", "cf" : 0.7},
            {"if" : ["varsta_peste_50", "hipertensiune", "istoric_cardiovascular"],
             "then" : "recomandare_monitorizare_medicala", "cf" : 0.9},
        ]

        # Deduced facts and explanations
        self.fapte_deduse = {}
        self.explicatii = {}

    # Motor de inferenta: Forward chaining
    def inlantuire_inainte(self) :
        reguli_aplicate = True
```

```
        while reguli_aplicate :
            reguli_aplicate = False
            for regula in self.reguli :
                if all(self.fapte.get(conditie, False) or conditie in self.fapte_deduse for conditie in regula["if"]) :
                    concluzie = regula["then"]
                    if concluzie not in self.fapte_deduse :
                        # Deduce conclusion
                        self.fapte_deduse[concluzie] = regula["cf"]
                        reguli_aplicate = True
                        # Check if conclusion is derived directly or via intermediates
                        direct = all(cond in self.fapte for cond in regula["if"])
                        tip = "direct din faptele inițiale" if direct else "prin concluzii intermediare"
                        self.explicatii[
                            concluzie] = f"Regula aplicata: Daca {regula['if']} atunci {concluzie} (CF: {regula['cf']}). Concluzie obținută {tip}."

    # Motor de inferenta: Backward chaining
    def inlantuire_inapoi(self, scop) :
        if scop in self.fapte :
            return self.fapte[scop]
        if scop in self.fapte_deduse :
            return self.fapte_deduse[scop]

        for regula in self.reguli :
            if regula["then"] == scop :
                if all(self.inlantuire_inapoi(conditie) for conditie in regula["if"]) :
                    self.fapte_deduse[scop] = regula["cf"]
                    # Check if conclusion is derived directly or via intermediates
                    direct = all(cond in self.fapte for cond in regula["if"])
                    tip = "direct din faptele inițiale" if direct else "prin concluzii intermediare"
                    self.explicatii[
                        scop] = f"Regula aplicata: Daca {regula['if']} atunci {scop} (CF: {regula['cf']}). Concluzie obținută {tip}."
                    return regula["cf"]
        return False

    # Metoda pentru rularea inferenței înainte
    def rulare_inferenta_inainte(self) :
        print("\n--- Rulare Inferență Înainte ---")
        print("Pornim de la faptele inițiale pentru a deduce concluzii noi...\n")
        self.inlantuire_inainte()
        if self.fapte_deduse :
            print("Rezultatele inferenței înainte:")
            for concluzie in self.fapte_deduse :
                print(f"- {concluzie} (CF: {self.fapte_deduse[concluzie]})")
                print(f"  Explicație: {self.explica(concluzie)}\n")
        else :
            print(" Nu s-au putut deduce concluzii noi.")

    # Metoda pentru rularea inferenței înapoi
    def rulare_inferenta_inapoi(self) :
        print("\n--- Rulare Inferență Înapoi ---")
        print("Verificăm concluziile folosind backward chaining...\n")
```

```
        for regula in self.reguli :
            scop = regula["then"]
            print(f"Interogare pentru scop: '{scop}'")
            rezultat = self.inlantuire_inapoi(scop)
            if rezultat :
                print(f"'{scop}' este adevărat (CF: {self.fapte_deduse.get(scop, 'N/A')}).")
                print(f"  Explicație: {self.explica(scop)}\n")
            else :
                print(f"'{scop}' nu poate fi dedus din faptele existente.\n")
        # Explicatii pentru concluzii

    def explica(self, concluzie) :
        return self.explicatii.get(concluzie, "Nu exista explicatie pentru concluzia ceruta.")


# Functia main
from src.solution import SistemNutritieFitness


# Functia main
def main():

    # Instantierea sistemului
    sistem = SistemNutritieFitness()

    # Rularea inferenței înainte
    sistem.rulare_inferenta_inainte()

    # Rularea inferenței înapoi
    sistem.rulare_inferenta_inapoi()


# Apelam functia main
if __name__ == "__main__" :
    main()
```