Universidade Federal de Alagoas - UFAL 2025.1
Instituto de Computação - IC Inteligencia Artificial
Aluno(s): Pedro Henrique Vieira Giló e Caio Oliveira França dos Anjos
Matrícula(s): 22210732 e 22210578

Lista 2 de Exercícios de IA

Link para o repositorio: https://github.com/PedroGilo12/InteligenciaArtificial-2025.1-IC-UFAL.git

1. Construa um agente conversacional (Chatbot) para uma aplicação a ser definida em sala de aula, usando um framework de sua livre escolha.

Foi escolhido o uso do **framework Rasa** para desenvolver um chatbot capaz de marcar reuniões, a ideia apenas de exemplo serve para que futuramente possa aproveitar parte desse chatbot para criar um bot real que possa gerenciar calendário e marcação de reuniões em servidores em redes sociais.

1.1 Introdução ao Framework Rasa:

O Rasa é um framework open-source para construção de assistentes conversacionais inteligentes. Ele permite criar chatbots capazes de entender linguagem natural e manter conversações contextualizadas.

Principais Componentes do Rasa:

- **Rasa NLU**: Responsável pelo entendimento da linguagem natural (intenções e entidades)
- **Rasa Core**: Gerencia o fluxo da conversa e tomada de decisões

Arquivos Principais em um Projeto Rasa:

1. `domain.yml`: Define o "universo" do chatbot:

```
intents:
  - cumprimentar
  - marcar_reuniao
  - despedir

responses:
  utter_cumprimentar:
    - "Olá! Como posso ajudar?"
  utter_perguntar_data:
    - "Para quando gostaria de marcar a reunião?"

actions:
  - action_confirmar_reuniao
```

2. `nlu.yml`: Contém exemplos de treinamento para reconhecimento de intenções:

```
nlu:
- intent: marcar_reuniao
  examples: |
    - Quero marcar uma reunião
```

```
            - Podemos agendar um encontro?
            - Preciso marcar uma call
```

3. `stories.yml`**:** Define fluxos de conversa típicos:

```
stories:
- story: fluxo_marcacao_reuniao
  steps:
  - intent: cumprimentar
  - action: utter_cumprimentar
  - intent: marcar_reuniao
  - action: utter_perguntar_data
```

4. `rules.yml`**:** Regras para comportamentos fixos:

```
rules:
- rule: responder_saudacao
  steps:
  - intent: cumprimentar
  - action: utter_cumprimentar
```

5. `actions.py`**:** Código Python para ações customizadas:

```
class ActionConfirmarReuniao(Action):
    def name(self) -> Text:
        return "action_confirmar_reuniao"

    def run(self, dispatcher, tracker, domain):
        # Lógica para confirmar reunião
        dispatcher.utter_message("Reunião marcada com sucesso!")
        return []
```

6. `config.yml`**:** Configuração do pipeline de NLU e políticas:

```
pipeline:
  - name: WhitespaceTokenizer
  - name: RegexFeaturizer
  - name: DIETClassifier

policies:
  - name: MemoizationPolicy
  - name: TEDPolicy
```

Vantagens do Rasa para Chatbots

- **Privacidade**: Todo processamento ocorre localmente
- **Customização**: Total controle sobre o fluxo conversacional

- **Integração**: Fácil conexão com calendários (Google Calendar, Outlook)
- **Aprendizado**: Melhora continuamente com mais interações

1.2 Resultados:

Os resultados obtidos estão a seguir:

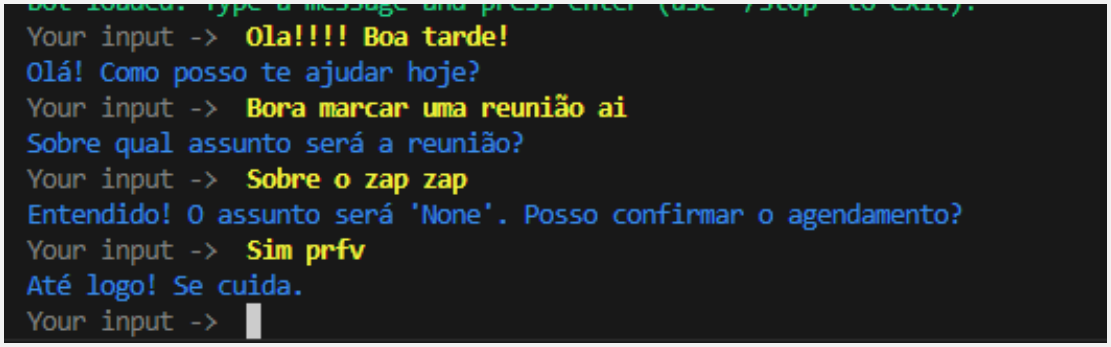

Figure 1: Primeira interação com o chatbot

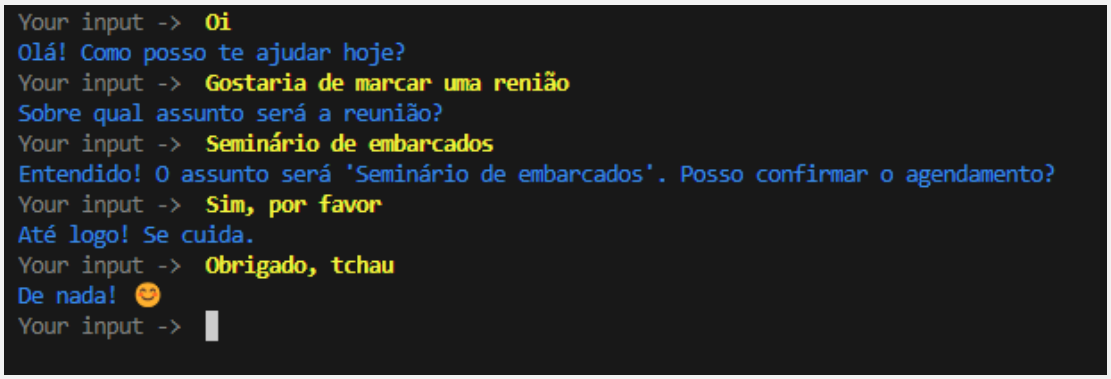

Figure 2: Segunda interação com chatbot após correções.

2. Desenvolva, usando a ferramenta Scikit Fuzzy (ou uma outra de sua livre escolha), um sistema baseado em regras fuzzy para controle de Temperatura de um Chuveiro, contendo uma evolução da base de conhecimento a seguir. Sobre a solução a ser desenvolvida, apresente:

2.a) Descrição do Processo de Modelagem:

O objetivo da modelagem é controlar a **abertura da válvula** do chuveiro para manter uma temperatura confortável da água, ajustando automaticamente conforme a **temperatura** da água e o **fluxo** de água.

O sistema utiliza um **controlador fuzzy Mamdani**, onde:

- Entradas: Temperatura da água e fluxo de água
- Saída: Abertura da válvula

Fuzzificação: Variáveis e Funções de Pertinência

Entradas:

- **Temperatura (0°C a 100°C):**
  - Baixa: Representada pela função triangular $[0, 0, 50]$
  - Média: Representada por $[25, 50, 75]$
  - Alta: Representada por $[50, 100, 100]$
- **Fluxo de Água (0 a 20 litros/minuto):**
  - Baixo: $[0, 0, 10]$
  - Médio: $[5, 10, 15]$
  - Alto: $[10, 20, 20]$

Saída:

- **Abertura da Válvula (0% a 100%):**
  - Pequena: $[0, 0, 50]$
  - Moderada: $[25, 50, 75]$
  - Grande: $[50, 100, 100]$

Cada entrada e saída foi mapeada utilizando **funções de pertinência triangulares**, por serem simples e eficientes para representar o comportamento gradual das variáveis.

Abaixo o plote das fuções de pertinência que foram utilizadas:

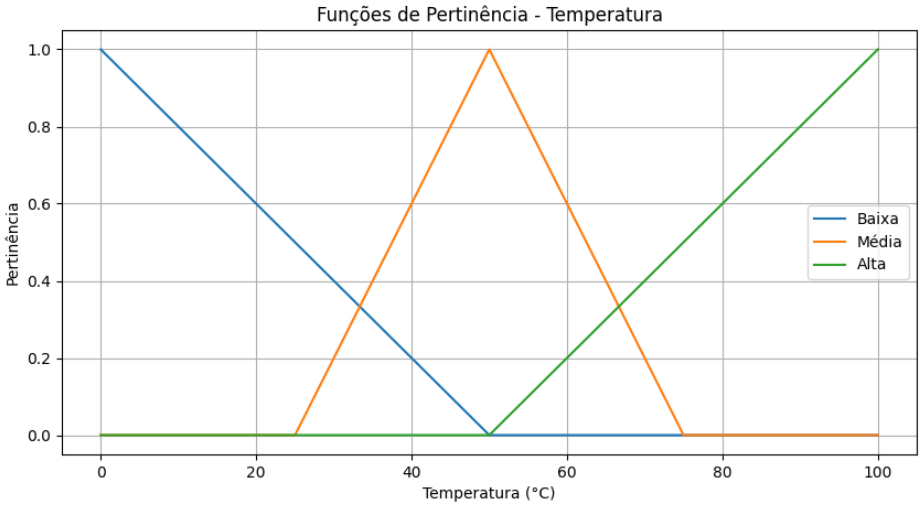

Figure 3: Função de inferência para temperatura.

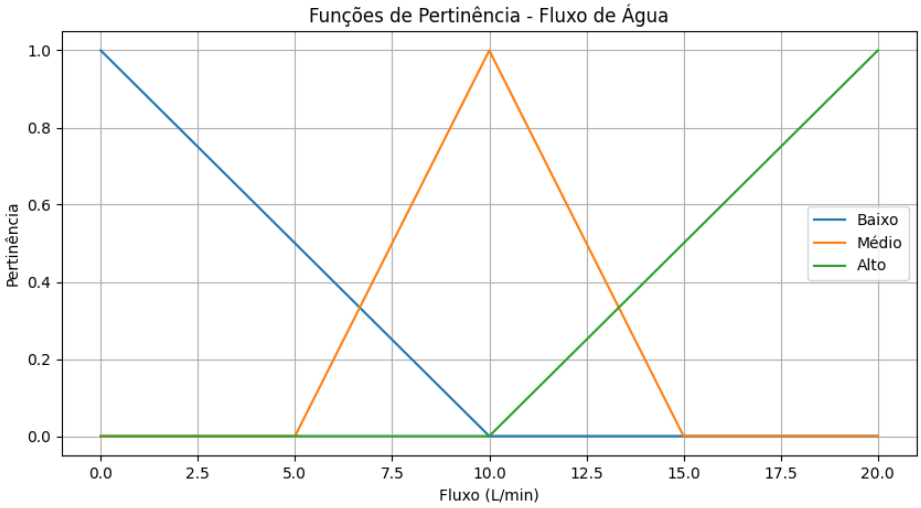

Figure 4: Função de pertinência para fluxo.

Inferência: Regras Fuzzy

A inferência é realizada utilizando o método **Mamdani**, aplicando a seguinte base de regras:

- **Regra 1**: SE temperatura é baixa E fluxo é alto, ENTÃO abertura é grande.
- **Regra 2**: SE temperatura é baixa E fluxo é médio, ENTÃO abertura é moderada.
- **Regra 3**: SE temperatura é média E fluxo é alto, ENTÃO abertura é moderada.
- **Regra 4**: SE temperatura é média E fluxo é baixo, ENTÃO abertura é pequena.
- **Regra 5**: SE temperatura é alta E fluxo é baixo, ENTÃO abertura é pequena.
- **Regra 6**: SE temperatura é alta E fluxo é alto, ENTÃO abertura é moderada.

no código em python ficou declarado da seguinte forma:

```
regra1 = ctrl.Rule(temperatura['baixa'] & fluxo['alto'], abertura['grande'])
regra2 = ctrl.Rule(temperatura['baixa'] & fluxo['medio'], abertura['moderada'])
regra3 = ctrl.Rule(temperatura['media'] & fluxo['alto'], abertura['moderada'])
regra4 = ctrl.Rule(temperatura['media'] & fluxo['baixo'], abertura['pequena'])
regra5 = ctrl.Rule(temperatura['alta'] & fluxo['baixo'], abertura['pequena'])
regra6 = ctrl.Rule(temperatura['alta'] & fluxo['alto'], abertura['moderada'])
```

Operadores lógicos:

- O operador **E** é implementado usando o **mínimo** (interseção).
- As saídas das regras são agregadas utilizando o **máximo**.

Defuzzificação: Conversão para Valor Real

Após aplicar as regras fuzzy e combinar os conjuntos de saída, é feita a **defuzzificação** para obter um valor único real da abertura da válvula.

No projeto, utilizamos o método **Centroide** (*Center of Gravity*), que calcula a média ponderada dos valores da saída fuzzy.

Esse método é o mais utilizado em controladores Mamdani por gerar uma resposta **suave e estável**.

Resultados obtidos:

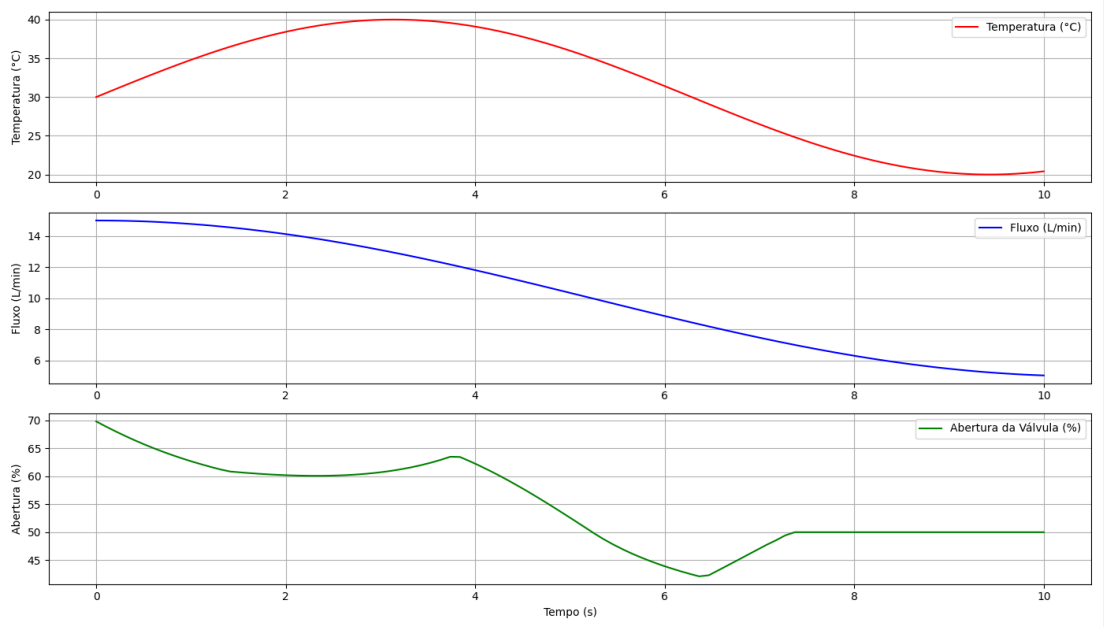

Figure 5: Função de inferência para temperatura.

3. Apresente e discuta um exemplo de aplicação de rede bayesiana para realizar um sistema de diagnóstico médico, apresentando modelagem das relações entre os sintomas e o diagóstico da doença. Apresente a solução de representação de conhecimento que você construiu e mostre o funcionamento da inferência, ilustrando com alguns exemplos. Utilize a ferramenta nética (ou alguma outra de sua livre escolha) para realizar essa aplicação.

A rede bayesiana foi modelada considerando Infecção Respiratória como a variável principal (raiz da rede), que influencia diretamente nos sintomas do paciente, como febre, tosse, dor de garganta, congestão nasal e fadiga. Cada sintoma foi representado por um nó binário (Sim/Não), e a probabilidade condicional de cada sintoma foi associada ao estado da doença. As probabilidades a priori indicam que a infecção respiratória ocorre em 10 por cento dos casos, enquanto os sintomas têm probabilidades variáveis de ocorrência dependendo da presença ou ausência da doença.

A rede foi construída utilizando a ferramenta Netica, que permite a criação de redes bayesianas com facilidade. Cada relação entre a doença e os sintomas foi representada por um arco, e as tabelas de probabilidade condicional (CPTs) foram definidas conforme dados clínicos ou estimativas:

```
Febre:
Sim         Nao         Gripe Pneumonia COVID19
0.99        0.01        Sim   Sim       Sim
0.94        0.06        Sim   Sim       Nao
0.95        0.05        Sim   Nao       Sim
0.8         0.2         Sim   Nao       Nao
0.93        0.07        Nao   Sim       Sim
0.7         0.3         Nao   Sim       Nao
0.9         0.1         Nao   Nao       Sim
0.05        0.95        Nao   Nao       Nao
Tosse:
Sim         Nao         Gripe Pneumonia COVID19
0.997       0.003       Sim   Sim       Sim
0.94        0.06        Sim   Sim       Nao
0.88        0.12        Sim   Nao       Sim
0.6         0.4         Sim   Nao       Nao
0.95        0.05        Nao   Sim       Sim
0.85        0.15        Nao   Sim       Nao
0.7         0.3         Nao   Nao       Sim
0.1         0.9         Nao   Nao       Nao
DorDeGarganta:
Sim         Nao         Gripe COVID19
0.82        0.18        Sim   Sim
0.68        0.32        Sim   Nao
0.3         0.7         Nao   Sim
0.07        0.93        Nao   Nao

Fadiga:
Sim         Nao         Gripe COVID19
0.95        0.05        Sim   Sim
0.75        0.25        Sim   Nao
0.8         0.2         Nao   Sim
0.05        0.95        Nao   Nao
```

```
Gripe:
Sim         Nao
0.05        0.95

Pneumonia:
Sim         Nao
0.02        0.98

COVID19:
Sim         Nao
0.03        0.97

FaltaDeAr:
Sim         Nao         Pneumonia COVID19
0.7         0.3         Sim       Sim
0.5         0.5         Sim       Nao
0.15        0.85        Nao       Sim
0.03        0.97        Nao       Nao
```

A inferência foi realizada para calcular a probabilidade da Infecção Respiratória dado os sintomas observados. Dois cenários de inferência foram simulados: um onde o paciente apresenta sintomas claros da doença, como febre, tosse e dor de garganta, e outro com sintomas mais sutis, como congestão nasal e fadiga. Os resultados da inferência mostraram uma alta probabilidade de infecção respiratória no primeiro cenário e uma probabilidade mais baixa no segundo, demonstrando como a rede bayesiana pode ser utilizada para inferir diagnósticos médicos a partir dos sintomas.

As imagens a seguir ilustram a construção da rede, as tabelas de probabilidade e o processo de inferência, que foram realizados de maneira eficiente com a ferramenta Netica.

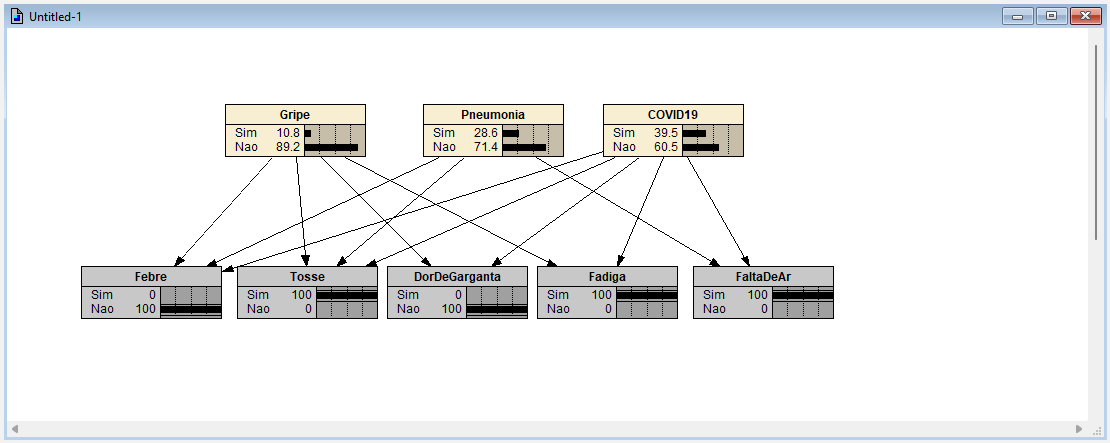

Figure 6: Resultado para fadiga, falta de ar e dor de garganta.

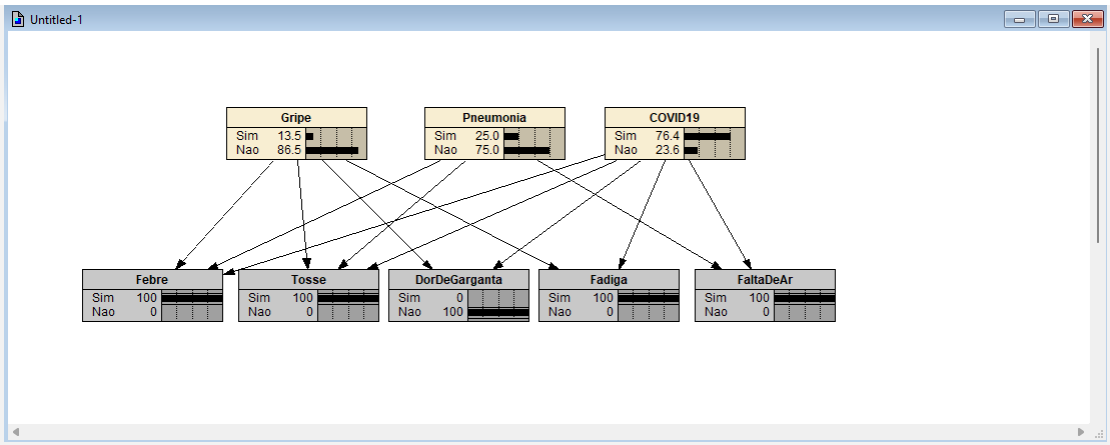

Figure 7: Resultado para fadiga, falta de ar, tosse e febre.

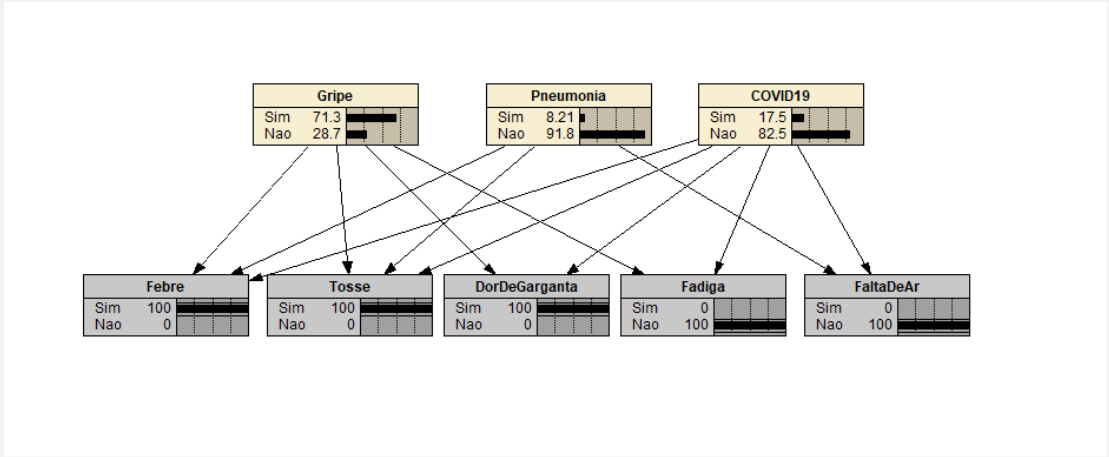

Figure 8: Resultado para febre, tosse e dor de garganta.

4. Desenvolva e discuta um sistema baseado em conhecimento, usando ontologia (OWL: Ontology Web Language) como solução para representão de conhecimento, tendo a finalidade de recomendar de vinho para acompanhar um determinado jantar, ou seja, sugerir boas combinações de vinhos e comidas, auxiliando nas decisões de clientes de um determinado restaurante (ou de uma loja).

A representação do conhecimento em nosso sistema é feita através de uma ontologia criada usando OWL (Ontology Web Language). Essa ontologia define conceitos, relações e propriedades relacionados a vinhos, comidas e suas combinações ideais.

Principais conceitos (classes) da ontologia:

Vinho: Representa diferentes tipos de vinhos.

Comida: Representa diferentes tipos de alimentos (massas, carnes, peixes, sobremesas).

Ocasião: Representa contextos como jantares românticos, festas, almoços casuais.

CombinaçãoIdeal: Relaciona vinhos e comidas que harmonizam bem.

Principais propriedades:

harmonizaCom (ObjectProperty): Relaciona um vinho a uma comida.

tipoDeVinho (DatatypeProperty): Indica se o vinho é tinto, branco, rosé ou espumante.

tipoDeComida (DatatypeProperty): Indica se a comida é carne vermelha, peixe, massa, sobremesa, etc.

adequadoParaOcasião (ObjectProperty): Relaciona vinho ou comida a uma ocasião.

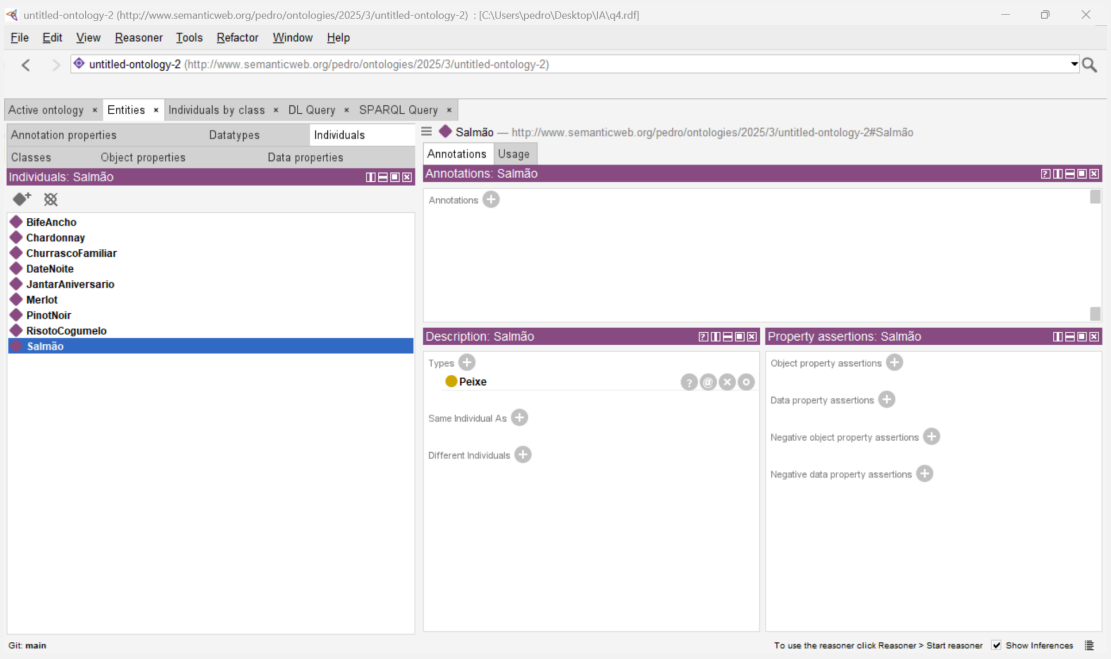

Figure 9: Entidades.

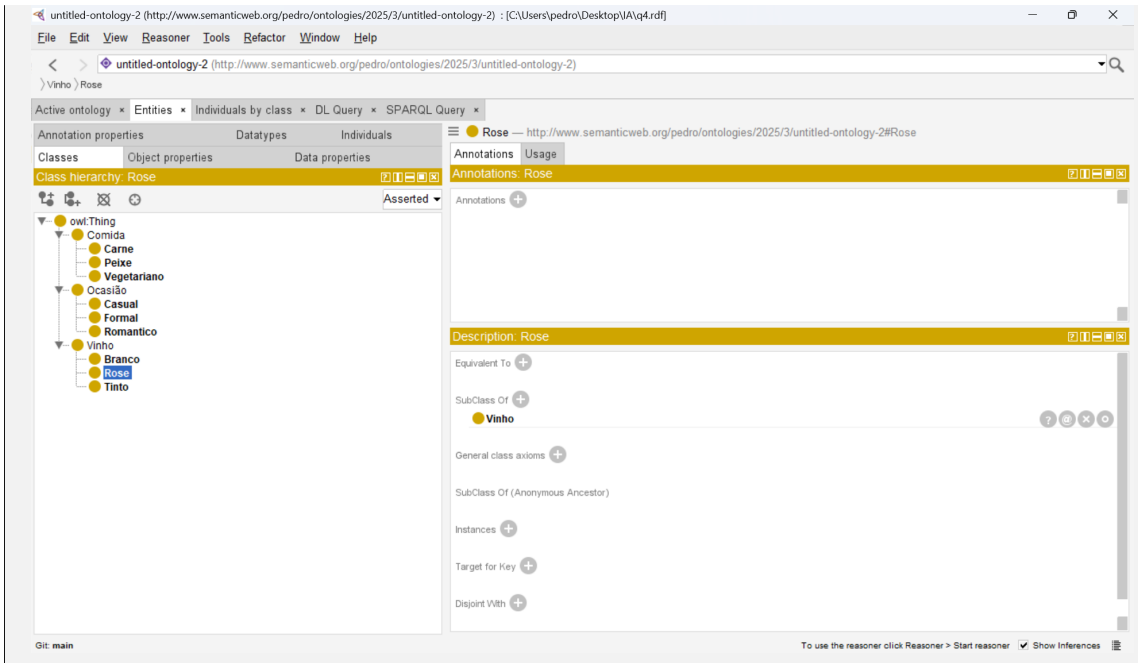

Figure 10: Classes.

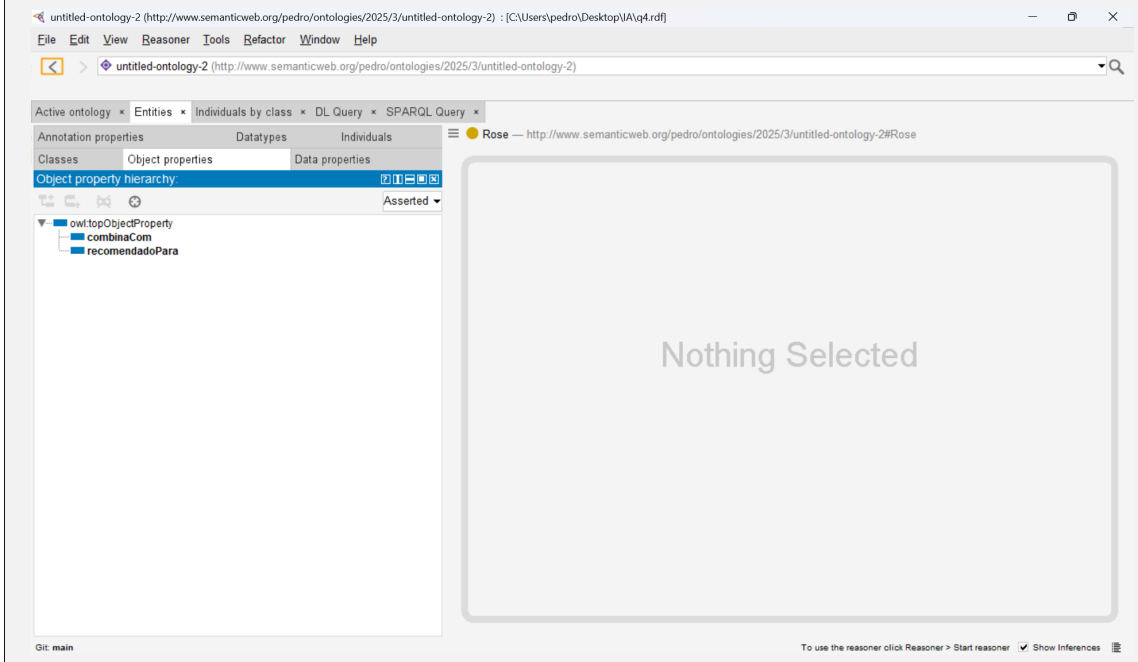

Figure 11: Propriedades dos objetos.

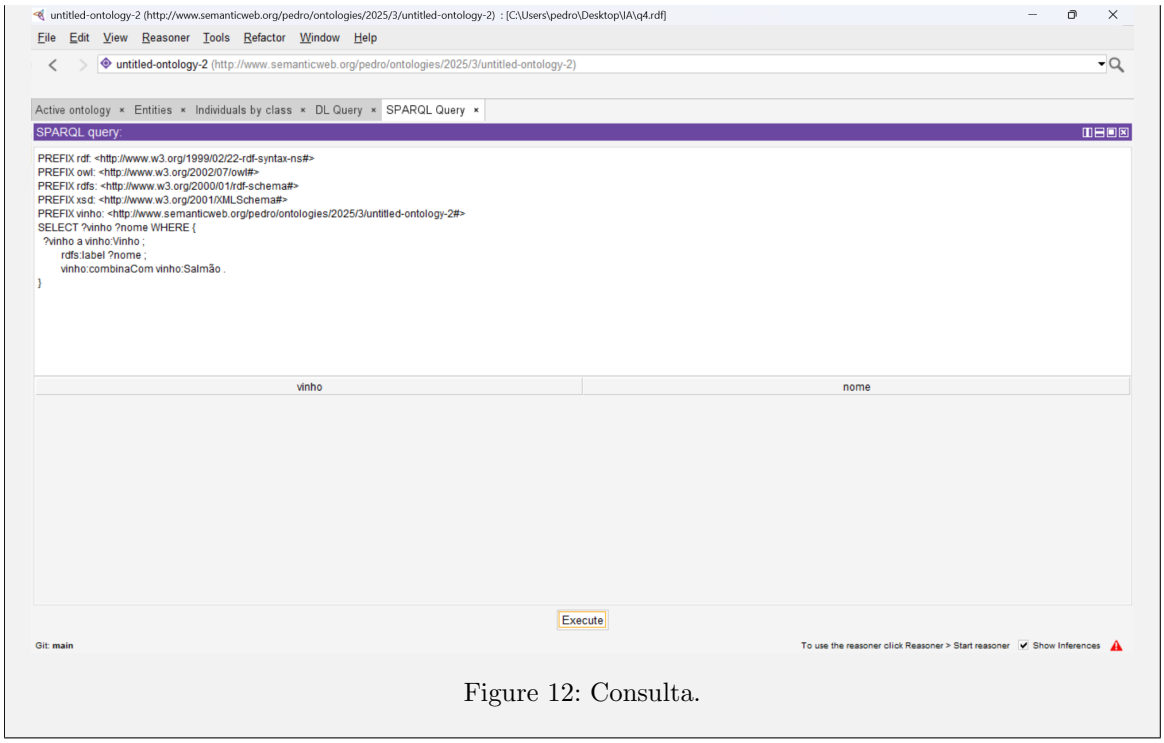

Figure 12: Consulta.

A base de casos é representada por uma lista de dicionários, onde cada dicionário contém:

- Lista de **sintomas** relatados.
- **Doença** diagnosticada.
- **Tratamento** recomendado.

Listing 1: Base de Casos

```
# Base de casos
base_de_casos = [
    {
        "sintomas": ["febre", "tosse", "fadiga"],
        "doenca": "Gripe",
        "tratamento": "Repouso, hidratacao
--------e medicamentos para febre."
    },
    {
        "sintomas": ["dor de garganta", "dificuldade para engolir",
        "febre"],
        "doenca": "Amigdalite",
        "tratamento": "Antibioticos e analgesicos
--------prescritos por medico."
    },
    {
        "sintomas": ["dor de cabeca", "nausea",
        "sensibilidade a luz"],
        "doenca": "Enxaqueca",
        "tratamento": "Analgesicos especificos e repouso
--------em ambiente escuro."
    },
    {
        "sintomas": ["tosse seca", "falta de ar", "dor no peito"],
        "doenca": "Bronquite",
        "tratamento": "Broncodilatadores e repouso."
    }
]
```

Implementou-se uma função que:

- Recebe os sintomas relatados pelo paciente.
- Compara os sintomas relatados com os casos armazenados.
- Encontra o caso mais semelhante.

Listing 2: Função de Recuperação de Casos

```
def recuperar_caso(sintomas_paciente, base_de_casos):
    melhor_caso = None
    maior_similaridade = 0

    for caso in base_de_casos:
        sintomas_em_comum = set(sintomas_paciente) &
        set(caso["sintomas"])
        similaridade = len(sintomas_em_comum)

        if similaridade > maior_similaridade:
            melhor_caso = caso
            maior_similaridade = similaridade

    return melhor_caso
```

Após a recuperação do caso, o sistema apresenta o diagnóstico provável e o tratamento recomendado.

Listing 3: Função de Diagnóstico

```
def diagnosticar(sintomas_paciente):
    caso_encontrado = recuperar_caso(sintomas_paciente, base_de_casos)

    if caso_encontrado:
        print(f"Possivel Doenca:
--------{caso_encontrado['doenca']}")
        print(f"Tratamento Recomendado:
--------{caso_encontrado['tratamento']}")
    else:
        print("Nenhum diagnostico encontrado.
--------Consulte um medico especializado.")
```