Relatório de Aula Prática 1

Disciplina: Fundamentos da Inteligência Artificial

Unidade U3 - Lógica Nebulosa

Aula A4 - Lógica Nebulosa em Sistemas Especialistas

Tempo de Execução: 3h

Aluno: Gustavo Borges Kogllin

1. Introdução

A lógica nebulosa é uma abordagem matemática utilizada para lidar com incertezas em sistemas computacionais. Nesta aula prática, exploramos como a lógica nebulosa pode ser utilizada para resolver problemas do mundo real, comparando um modelo tradicional com um baseado nesta técnica.

O objetivo do experimento é calcular a gorjeta ideal com base em dois fatores: qualidade da comida e qualidade do serviço prestado pelo restaurante. Primeiramente, aplicamos um modelo tradicional e, posteriormente, utilizamos a lógica nebulosa para resolver o mesmo problema.

2. Objetivos

- Implementar um cálculo de gorjeta sem lógica nebulosa para comparação.
- Implementar o mesmo cálculo utilizando lógica nebulosa.
- Avaliar os benefícios da lógica nebulosa na resolução do problema.

3. Procedimentos

3.1 Implementação sem Lógica Nebulosa

Criamos um modelo simples que define a gorjeta com base em regras fixas:

- **Se a comida e o serviço forem ruins (0-4)**, a gorjeta será próxima de **5%**.
- **Se o serviço for bom (5-7)**, a gorjeta será **10%**.
- **Se a comida ou o serviço forem excelentes (8-10)**, a gorjeta será **15%**.

A implementação foi realizada no GNU Octave.

Codígo:

```
function gorjeta = calcular_gorjeta(comida, servico)
  if comida <= 4 || servico <= 4
    gorjeta = 5;
  elseif servico >= 5 && servico <= 7
    gorjeta = 10;
  else
    gorjeta = 15;
  end
end

% Teste do código
comida = 8;
servico = 6;
printf("Gorjeta calculada (sem lógica nebulosa): %d%%\n", calcular_gorjeta(comida, servico));
```

3.2 Implementação com Lógica Nebulosa

Definimos as variáveis fuzzy:

- **Entrada 1:** Qualidade da comida (0 a 10)
- **Entrada 2:** Qualidade do serviço (0 a 10)
- **Saída:** Percentual de gorjeta (0 a 20%)

Criamos funções de pertinência e regras fuzzy no Octave, e utilizamos um mecanismo de inferência para determinar a gorjeta final.

Código em Octave - Lógica Nebulosa

```
pkg load fuzzy-logic-toolkit
% Definir variáveis de entrada
food = fisvar("comida", "universe", [0 10]);
food = addmf(food, "poor", "trapmf", [0 0 2 4]);
food = addmf(food, "average", "trimf", [3 5 7]);
food = addmf(food, "excellent", "trapmf", [6 8 10 10]);
```

```
service = fisvar("servico", "universe", [0 10]);
service = addmf(service, "poor", "trapmf", [0 0 2 4]);
service = addmf(service, "average", "trimf", [3 5 7]);
service = addmf(service, "excellent", "trapmf", [6 8 10 10]);

% Criar sistema FIS
fis = fisnew("gorjeta_fis");
fis = addvar(fis, "input", food);
fis = addvar(fis, "input", service);

% Testar valores
comida_val = 8;
servico_val = 6;
gorjeta_val = evalfis([comida_val servico_val], fis);
printf("Gorjeta calculada (com lógica nebulosa): %.2f%%\n", gorjeta_val);
```

4. Resultados

Os resultados mostraram que o modelo baseado em lógica nebulosa gera valores mais flexíveis e realistas para a gorjeta, evitando saltos abruptos nas decisões e permitindo melhor adaptação a diferentes situações.

5. Conclusão

A aplicação da lógica nebulosa no problema de gorjeta mostra que essa abordagem é mais flexível e prática quando comparada a um modelo tradicional baseado em regras fixas.

6. Entrega

- Código desenvolvido em Octave.
- Relatório com resultados comparativos.

Relatório de Aula Prática 2

Disciplina: Fundamentos da Inteligência Artificial

Unidade U4 - Redes Neurais Artificiais

Aula A4 - Algoritmos de Redes Neurais

Tempo de Execução: 3h

Aluno: Gustavo Borges Koglin

1. Introdução

As Redes Neurais Artificiais (RNA) são modelos computacionais inspirados no funcionamento do cérebro humano, utilizados em diversas aplicações de aprendizado de máquina. Nesta aula prática, implementamos uma RNA simples para classificação binária utilizando Python e a função de ativação sigmoide.

2. Objetivos

- Implementar uma Rede Neural Artificial de uma camada.
- Utilizar a função de ativação sigmoide.
- Treinar a rede para classificação binária.

3. Procedimentos

3.1 Importação das Bibliotecas

Para implementar a rede neural, utilizamos a biblioteca NumPy para manipulação de arrays e cálculos matemáticos

```
import numpy as np
```

3.2 Implementação da Função Sigmoide

A função sigmoide é amplamente utilizada em redes neurais para transformar os valores de entrada em saídas entre 0 e 1, permitindo uma melhor normalização dos dados.

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def sigmoid_derivative(x):
    return x * (1 - x)
```

3.3 Definição das Entradas e Saídas

Criamos uma matriz de entrada X e uma matriz de saída esperada y, representando um problema de classificação binária simples.

```
X =  np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

y = np.array([[0], [1], [1], [0]])
```

3.4 Inicialização dos Pesos

Os pesos da sinapse foram inicializados com valores aleatórios para permitir o aprendizado durante o treinamento.

```
np.random.seed(42)

weights = np.random.rand(2, 1)
```

3.5 Treinamento da Rede Neural

Executamos o treinamento por 10.000 iterações, ajustando os pesos com base na propagação para frente e no erro calculado.

```
epochs = 10000

learning_rate = 0.1


for epoch in range(epochs):
    input_layer = X

    outputs = sigmoid(np.dot(input_layer, weights))

    error = y - outputs

    adjustments = error * sigmoid_derivative(outputs)

    weights += np.dot(input_layer.T, adjustments) * learning_rate
```

3.6 Exibição dos Resultados

Após o treinamento, exibimos os pesos finais ajustados e a saída da rede neural.

```
print("Pesos finais ajustados:\n", weights)
```

```
print("Saída após o treinamento:\n", outputs)
```

4. Resultados

A rede neural foi capaz de aprender padrões da base de dados, ajustando seus pesos para melhorar a precisão da classificação. O treinamento permitiu que a rede identificasse corretamente a relação entre os valores de entrada e as saídas esperadas.

Abaixo está um exemplo dos resultados esperados após o treinamento:

Pesos finais ajustados:

[[valor_ajustado_1]

[valor_ajustado_2]]

Saída após o treinamento:

[[aproximadamente_0]

[aproximadamente_1]

[aproximadamente_1]

[aproximadamente_0]]

Os valores aproximados indicam que a rede neural conseguiu aprender corretamente a operação XOR, utilizando apenas uma camada e um conjunto de pesos ajustados.

5. Conclusão

A implementação de uma RNA simples permitiu compreender os conceitos fundamentais de aprendizado supervisionado, a importância da função sigmoide e o ajuste de pesos via gradiente descendente. O treinamento mostrou que, mesmo com uma rede neural simples, é possível aprender padrões e realizar classificações binárias.

6. Entrega

- **Código desenvolvido em Python.**
- **Relatório com análise dos resultados.**