# Ideia_GPT

April 20, 2025

## 1 Implementação com banco de dados

A seguir, devo puxar um banco de dados para treinamento. - Você carrega um CSV com uma coluna `Entrada`. - Converte os valores para float, limitando entre 1 e 6. - Aplica uma **lógica fuzzy** com `skfuzzy` para classificar essas entradas em: - `0.25` → Baixa confiança - `0.5` → Confiança média - `0.75` → Moderada-alta - `1.0` → Alta confiança

Isso gera um array `array_saidas` com valores exatamente nos níveis que você comentou antes: `{0.25, 0.5, 0.75, 1}`.

Agora você quer prever **o próximo valor da sequência fuzzy** (ou seja, prever `array_saidas[n+1]` com base nos anteriores).

[91]:
```
import skfuzzy as fuzz
import numpy as np
import pandas as pd
```

[90]:
```
def fuzzy_classification(odd):
    """
    Implementação da lógica fuzzy para classificar as odds no intervalo de 1 a␣
 ↪6.
    """
    odd_range = np.arange(1, 6.1, 0.1)

    # Conjuntos fuzzy ajustados para cobrir todo o intervalo de 1 a 6
    baixo = fuzz.trimf(odd_range, [1, 1, 2])
    medio = fuzz.trimf(odd_range, [1.5, 3, 4.5])
    alto = fuzz.trimf(odd_range, [3.5, 5, 6])
    muito_alto = fuzz.trimf(odd_range, [4.5, 6, 6])

    # Graus de pertinência
    pert_baixo = fuzz.interp_membership(odd_range, baixo, odd)
    pert_medio = fuzz.interp_membership(odd_range, medio, odd)
    pert_alto = fuzz.interp_membership(odd_range, alto, odd)
    pert_muito_alto = fuzz.interp_membership(odd_range, muito_alto, odd)

    # Classificação baseada nos graus de pertinência
    max_pert = max(pert_baixo, pert_medio, pert_alto, pert_muito_alto)
```

```
    if max_pert == 0:
        return 0  # Nenhuma confiança

    if max_pert == pert_muito_alto:
        return 1  # Alta confiança na subida
    elif max_pert == pert_alto:
        return 0.75  # Confiança moderada-alta
    elif max_pert == pert_medio:
        return 0.5  # Confiança média
    else:
        return 0.25  # Baixa confiança
```

[92]:
```
# Carregar os dados
path_inicial = '/home/darkcover/Documentos/Out/Documentos/dados/Saidas/FUNCOES/
 ↪DOUBLE - 17_09_s1.csv'
data = pd.read_csv(path_inicial)

array_entradas = data['Entrada'].iloc[:1500].reset_index(drop=True)
array_saidas = []
for i in range(len(array_entradas)):
    array_entradas[i] = float(array_entradas[i].replace(',', '.'))
    if array_entradas[i] <= 1:
        array_entradas[i] = 1
    if array_entradas[i] >= 6:
        array_entradas[i] = 6

    fuzzaqw = fuzzy_classification(array_entradas[i])
    array_saidas.append(fuzzaqw)
array_saidas = np.array(array_saidas)
```

[35]:
```
array_entradas.describe
```

[35]:
```
<bound method NDFrame.describe of 0        1.83
1         1,07
2        24,83
3        25,25
4         8,55
         …
1495      3,61
1496      1,18
1497      1,52
1498      1,42
1499      6,51
Name: Entrada, Length: 1500, dtype: object>
```

[94]:
```
print(array_saidas)
```

```
[0.5  0.25 1.   … 0.25 0.25 1.  ]
```

### 1.0.1 Plano para usar modelo de Machine Learning (complexo):

Vamos seguir com o plano do MLP (ou se quiser depois, podemos até jogar pra LSTM). Aqui vai um passo a passo com base no que você já tem:

---

[95]:
```
##  1. **Criar as janelas de entrada**
def criar_sequencias(array, janela=5):
    X, y = [], []
    for i in range(len(array) - janela):
        X.append(array[i:i+janela])
        y.append(array[i+janela])
    return np.array(X), np.array(y)

X, y = criar_sequencias(array_saidas, janela=5)
```

### 1.0.2 2. Treinar um MLP Classifier

Como os valores são discretos, tratamos como um problema de classificação:

[96]:
```
# Testando com diferentes sequências
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

# Encode as classes (0.25, 0.5, 0.75, 1.0)
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Treino/teste split
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.
 ↪2, random_state=42)

# Modelo
modelo = MLPClassifier(hidden_layer_sizes=(64, 32), max_iter=1000,␣
 ↪random_state=42)
modelo.fit(X_train, y_train)

# Acurácia
acc = modelo.score(X_test, y_test)
print(f"Acurácia: {acc:.2f}")
```

```
Acurácia: 0.42
```

### 1.0.3 3. Prever o próximo valor

[97]:
```
# Últimos 5 valores da série
entrada_atual = array_saidas[-5:].reshape(1, -1)
pred = modelo.predict(entrada_atual)
valor_previsto = le.inverse_transform(pred)
print("Valor previsto:", valor_previsto[0])
```

```
Valor previsto: 0.25
```