

LINK PARA MEUS CÓDIGOS:

<https://github.com/andrielmark/PUCMINAS/tree/main/QUARTO%20PER%C3%8DODO/IA/LISTA4>

Seção 1: Funções Utilitárias e Preparação dos Dados

Para garantir a robustez e a reprodutibilidade dos experimentos, foi desenvolvido um conjunto de funções de pré-processamento. Elas cuidam das etapas críticas de tratamento de dados antes do treinamento dos modelos.

Decisões de Projeto:

- **Tratamento de Dados Faltantes:** Optei por uma estratégia de imputação simples. Para colunas numéricas como 'Age', a mediana foi escolhida por ser menos sensível a outliers do que a média. Para colunas categóricas como 'Embarked', a moda (o valor mais frequente) foi utilizada para preencher os campos vazios.
- **Particionamento dos Dados:** Foi implementada uma função de divisão estratificada. Essa abordagem é crucial para datasets com classes desbalanceadas, como o do Titanic, pois assegura que a proporção de sobreviventes e não sobreviventes seja a mesma tanto no conjunto de treino quanto no de teste.
- **Discretização:** Como o algoritmo ID3 não lida com dados contínuos, criei funções para converter atributos como 'Age' e 'Fare' em categóricos. Implementei a discretização por frequência (quantis), que cria faixas com um número aproximadamente igual de amostras em cada uma.

Python

```
# Módulo: preparacao_dados.py
```

```
import pandas as pd
```

```
import numpy as np
```

```
from typing import Tuple
```

```
def preencher_valores_ausentes(dataframe: pd.DataFrame) -> pd.DataFrame:
```

```
    """
```

```
    Preenche dados faltantes usando mediana para colunas numéricas  
    e moda para colunas categóricas.
```

```
    """
```

```
    df_processado = dataframe.copy()
```

```

for coluna in df_processado.columns:
    if pd.api.types.is_numeric_dtype(df_processado[coluna]):
        if df_processado[coluna].isnull().any():
            mediana = df_processado[coluna].median()
            df_processado[coluna].fillna(mediana, inplace=True)
        else:
            if df_processado[coluna].isnull().any():
                moda = df_processado[coluna].mode()[0]
                df_processado[coluna].fillna(moda, inplace=True)
    return df_processado

def dividir_treino_teste_estratificado(X: pd.DataFrame, y: pd.Series,
percentual_teste=0.2, seed_aleatoria=42) -> Tuple:
    """
    Divide os dados em treino e teste de forma estratificada para manter
    a proporção das classes.
    """
    indices_treino, indices_teste = train_test_split_stratified(y,
test_size=percentual_teste, seed=seed_aleatoria)
    X_treino, X_teste = X.iloc[indices_treino], X.iloc[indices_teste]
    y_treino, y_teste = y.iloc[indices_treino], y.iloc[indices_teste]
    return X_treino, X_teste, y_treino, y_teste

def discretizar_por_quantis(coluna: pd.Series, num_faixas: int = 4) -> pd.Series:
    """
    Converte uma coluna contínua em categórica, dividindo-a em faixas
    com o mesmo número de amostras (quantis).
    """
    faixas = pd.qcut(coluna, q=num_faixas, labels=False, duplicates='drop')
    return faixas.astype(str)

```

Métricas de Avaliação:

Para avaliar a performance dos modelos, implementei as métricas de acurácia e a matriz de confusão.

Python

```

# Módulo: metrics.py
import numpy as np
import pandas as pd

```

```

def calcular_acuracia(y_real, y_previsto) -> float:
    """Calcula a porcentagem de previsões corretas."""

```

```
return np.mean(y_real == y_previsto)

def gerar_matriz_confusao(y_real, y_previsto):
    """Gera a matriz de confusão para análise de erros."""
    classes = sorted(np.unique(y_real))
    matriz = np.zeros((len(classes), len(classes)), dtype=int)
    mapeamento_classes = {classe: i for i, classe in enumerate(classes)}

    for real, previsto in zip(y_real, y_previsto):
        matriz[mapeamento_classes[real], mapeamento_classes[previsto]] += 1

    return classes, matriz
```

Seção 2: Implementação dos Algoritmos de Árvore de Decisão

Nesta seção, descrevo as decisões de projeto por trás da implementação de cada um dos três algoritmos de árvore de decisão: ID3, C4.5 e CART.

2.1) ID3

O ID3 foi o primeiro algoritmo implementado, servindo como base. Ele seleciona o melhor atributo para dividir os dados usando o

Ganho de Informação.

- **Justificativa:** A implementação focou em lidar puramente com dados categóricos. O critério de parada adotado foi um limite na profundidade da árvore (`max_depth`), uma técnica simples e eficaz para prevenir o superajuste (`overfitting`). Se um nó atingisse a profundidade máxima ou se todos os seus exemplos pertencessem a uma única classe, ele se tornaria uma folha.

2.2) C4.5

O C4.5 aprimora o ID3. A principal mudança é o uso da

Razão de Ganho como critério de divisão.

- **Justificativa:** A Razão de Ganho foi implementada para penalizar atributos com muitos valores únicos (como um ID), que tendem a ter um Ganho de Informação artificialmente alto. Outra implementação chave foi o tratamento

nativo de atributos contínuos. O código ordena os valores do atributo e testa cada ponto médio como um possível limiar de divisão, escolhendo aquele que maximiza a Razão de Ganho. Isso elimina a necessidade de discretização manual.

2.3) CART

O CART se diferencia por construir árvores estritamente binárias e usar o Índice de Gini como métrica de impureza.

- **Justificativa:** O maior desafio na implementação foi garantir que todas as divisões fossem binárias. Para atributos contínuos, a abordagem é similar à do C4.5 (encontrar o melhor limiar). Para atributos categóricos, a implementação testa todas as combinações possíveis para dividir as categorias em dois subconjuntos (ex: {A, B} vs. {C}), escolhendo a partição que resulta na maior redução da impureza de Gini.

Seção 3: Experimentos e Resultados

Os algoritmos foram aplicados ao dataset Titanic, seguindo uma configuração padrão para permitir uma comparação justa.

Configuração: Partição estratificada 80/20 (com seed=2024 para reprodutibilidade), profundidade máxima (max_depth) de 6.

3.1) Resultados com ID3

- **Acurácia (treino): 0.8750**
- **Acurácia (teste): 0.7978**
- **Matriz de Confusão (teste):**

	0	1
0	98	12
1	25	43

Árvore Gerada (resumo):

```
[Sex]
|-> 1: (male)
|   [Age_faixa]
|   |-> 0: (-inf, 21.0]
|   |   [SibSp_faixa]
|   |   |-> 0: Folha: 0
|   |   |-> 1: Folha: 0
|   |-> 3: (38.0, inf]
|   |   [Pclass]
|   |   |-> 3: Folha: 0
|   |   |-> 1: Folha: 0
```

3.2) Resultados com C4.5

- Acurácia (treino): 0.8347
- Acurácia (teste): 0.8258
- Matriz de Confusão (teste):

	0	1
0	102	8
1	23	45

```
[Sex]
|-> male:
|   [Age <= 6.5]
|   |-> True:
|   |   [SibSp <= 2.5]: Folha: 1
|   |-> False:
|   |   [Pclass <= 2.5]
|   |   |-> True: Folha: 0
|   |   |-> False: Folha: 0
|-> female:
|   [Pclass <= 2.5]
|   |-> True:
|   |   [Fare <= 27.5]: Folha: 1
|   |   [Fare > 27.5]: Folha: 1
|   |-> False:
|   |   [Fare <= 24.5]: Folha: 0
|   |   [Fare > 24.5]: Folha: 0
```

3.3) Resultados com CART

- Acurácia (treino): 0.8651
- Acurácia (teste): 0.8315
- Matriz de Confusão (teste):

	0	1
0	101	9
1	21	47

```
[Sex in {'male'}]
|-> True:
|   [Age <= 6.5]
|   |-> True:
|   |   [Parch <= 2.5]: Folha: 1
|   |-> False:
|   |   [Fare <= 24.5]: Folha: 0
```

