Códigos da lista: [Códigos no GitHub]

SEÇÃO 1

Código - utilitários (imputação, split, discretização)

```python
from __future__ import annotations
from typing import Tuple
import numpy as np
import pandas as pd

def split_stratified_indices(y: np.ndarray, test_ratio: float = 0.2, random_state: int = 42) -> Tuple[np.ndarray, np.
ndarray]:
    """
    Divide índices de forma estratificada entre treino e teste.

    Parâmetros:
        y (np.ndarray): vetor de rótulos.
        test_ratio (float): proporção de amostras para teste.
        random_state (int): semente para reprodução dos resultados.

    Retorna:
        (np.ndarray, np.ndarray): índices de treino e teste.
    """
    rng = np.random.default_rng(random_state)
    indices = np.arange(len(y))
    test_indices = []

    for classe in np.unique(y):
        classe_idx = indices[y == classe]
        rng.shuffle(classe_idx)
        qtd_teste = max(1, round(test_ratio * len(classe_idx)))
        test_indices.extend(classe_idx[:qtd_teste])

    test_indices = np.sort(np.array(test_indices))
    train_indices = np.setdiff1d(indices, test_indices)
    return train_indices, test_indices
```

```python
def discretize_quantile(series: pd.Series, n_bins: int = 4, as_str: bool = True) -> pd.Series:
    """
    Discretiza valores numéricos em faixas baseadas em quantis (frequências semelhantes).
    """
    quantiles = np.linspace(0, 1, n_bins + 1)
    limites = series.quantile(quantiles).unique()
    limites[0], limites[-1] = -np.inf, np.inf

    categorias = pd.cut(series, bins=limites, include_lowest=True)
    return categorias.astype(str) if as_str else categorias


def discretize_fixed_width(series: pd.Series, n_bins: int = 4, as_str: bool = True) -> pd.Series:
    """
    Discretiza valores numéricos em intervalos de largura fixa.
    """
    categorias = pd.cut(series, bins=n_bins, include_lowest=True)
    return categorias.astype(str) if as_str else categorias


def fill_missing_simple(df: pd.DataFrame) -> pd.DataFrame:
    """
    Preenche valores ausentes:
    - Numéricos: mediana.
    - Categóricos/objetos: moda.
    """
    result = df.copy()

    for col in result.columns:
        if result[col].isna().any():
            if pd.api.types.is_numeric_dtype(result[col]):
                result[col] = result[col].fillna(result[col].median())
            else:
                result[col] = result[col].fillna(result[col].mode().iloc[0])

    return result
```

Código - Métricas auxiliares

```python
# ======== metrics_utils.py ========
from __future__ import annotations
import numpy as np

def calc_accuracy(real, predicted) -> float:
    """
    Calcula a acurácia entre os rótulos reais e os previstos.

    Args:
        real (array-like): Valores verdadeiros.
        predicted (array-like): Valores previstos.

    Returns:
        float: Acurácia (proporção de acertos).
    """
    real = np.array(real)
    predicted = np.array(predicted)
    return float(np.mean(real == predicted))
```

```python
def build_confusion_matrix(real, predicted):
    """
    Gera a matriz de confusão com base nos rótulos reais e previstos.

    Args:
        real (array-like): Valores verdadeiros.
        predicted (array-like): Valores previstos.

    Returns:
        tuple: (lista de rótulos, matriz de confusão)
    """
    real = np.array(real)
    predicted = np.array(predicted)

    # Garante que todos os rótulos únicos de ambos estejam presentes
    classes = sorted(set(real).union(set(predicted)))
    n = len(classes)

    # Mapeamento rótulo → índice
    label_to_idx = {label: idx for idx, label in enumerate(classes)}

    # Inicializa a matriz de confusão
    matrix = np.zeros((n, n), dtype=int)

    # Conta as combinações (real x previsto)
    for r, p in zip(real, predicted):
        matrix[label_to_idx[r], label_to_idx[p]] += 1

    return classes, matrix
```

SEÇÃO 2

2.1 Utilidades Comuns

Implementações necessárias (ou já presentes nos códigos) para suportar todas as árvores:

- Cálculo de métricas:
    - Entropia
    - Ganho de informação
    - Razão de ganho (gain ratio)
    - Índice de Gini
- Procura da melhor divisão (split):
    - Atributos categóricos:
        - pelo valor (um ramo por valor) → usado em *ID3.py* e *C45.py*
        - binarização ótima → usado em *CART.py*
    - Atributos contínuos:
        - ordenar valores únicos
        - testar limiares em pontos médios entre valores adjacentes
- Critérios de empate: quando dois splits têm o mesmo valor de critério (ganho, gain ratio ou Gini), definir regra clara (por exemplo: escolher atributo com mais amostras; se empatar, ordem alfabética etc.).

2.2 ID3 (do zero) — Código: ID3.py

- Critério: ganho de informação
- Atributos categóricos (no código, a lógica de divisão trata categorias separadas)
- Dados faltantes: verificar se o código lida com missing — caso não, pode-se inserir imputação (moda para categóricos)

- O repositório ID3.py implementa recursão, criação de folha quando impureza zero ou nenhum atributo disponível. GitHub

---

2.3 C4.5 (do zero) — Código: C45.py

- Critério: razão de ganho = ganho de informação normalizado pelo split (informação do particionamento)
- Tratamento de atributos contínuos: o código executa busca de melhor limiar entre valores adjacentes. GitHub
- Para atributos categóricos: permite múltiplos ramos (um ramo por valor).
- Handling de missing: verificar se há parte no código que trata valores ausentes — se não, incluir média para numéricos e moda para categóricos.

2.4 CART (do zero) — Código: CART.py

- Critério: índice de Gini usado no CART.py para medir impureza. GitHub
- Divisões sempre binárias — mesmo para atributos categóricos, o código acerta uma partição em dois grupos.
- Comparação: validar com sklearn.tree.DecisionTreeClassifier(criterion="gini") nos mesmos dados para ver se os resultados (árvore ou métricas de acurácia / matriz de confusão) são semelhantes.

SEÇÃO 3

3.1) ID3

Acurácia (treino): 0.8808

Acurácia (teste): 0.8034

Matriz de confusão (teste):

labels: 0, 1

| | 0 | 1 |

| - | - | - |

| 0 | 99 | 11 |

| 1 | 24 | 44 |

Árvore:

[Sex]

├── female

│ ├── [Pclass]

│ │ ├── 1

│ │ │ ├── [Fare]

│ │ │ │ ├── (14.454, 30.5] → Folha: 1

│ │ │ │ └── (30.5, inf] → Folha: 1

│ │ ├── 2

│ │ │ ├── [Age]

│ │ │ │ ├── (-inf, 22.0] → Folha: 1

│ │ │ │ └── (22.0, 28.0] → Folha: 1

│ │ └── 3

│ │ ├── [Fare]

│ │ │ ├── (7.896, 14.454] → Folha: 1

│ │ │ └── (14.454, 30.5] → Folha: 1

├── male

│ ├── [SibSp]

│ │ ├── 0

│ │ │ ├── [Pclass]

│ │ │ │ ├── 1 → Folha: 0

│ │ │ │ ├── 2 → Folha: 0

│ │ │ │ └── 3 → Folha: 0

│ │ └── ≥1 → Folha: 0

3.2) C4.5

Acurácia (treino): 0.8219

Acurácia (teste): 0.8202

Matriz de confusão (teste):

labels: 0, 1

| | 0 | 1 |

|-| - | - |

|0|103| 7|

|1| 25|43|

Árvore:

[Sex]

├─ female

│ ├─ [SibSp < 6]

│ │ ├─ True

│ │ │ ├─ [Pclass < 2.5]

│ │ │ │ ├─ True

│ │ │ │ │ ├─ [Fare < 28.8562]

│ │ │ │ │ │ ├─ True

│ │ │ │ │ │ │ ├─ [Age < 53.5]

│ │ │ │ │ │ │ │ ├─ True → Folha: 1

│ │ │ │ │ │ │ │ ├─ False → Folha: 1

│ │ │ │ │ │ │ ├─ False → Folha: 1

│ │ │ │ │ │ ├─ False → Folha: 1

│ │ │ │ ├─ False → Folha: 1

├─ male

│ ├─ [Age < 1.5]

│ │ ├─ True → Folha: 1

│ │ ├─ False

│ │ │ ├─ [Fare < 387.665]

│ │ │ │ ├─ True

│ │ │ │ │ ├─ [SibSp < 4.5]

│ │ │ │ │ │ ├─ True

│ │ │ │ │ │ │ ├─ [Parch < 2.5]

│ │ │ │ │ │ │ │ ├─ True

│ │ │ │ │ │ │ │ │ ├─ [Pclass < 1.5] → Folha: 0

│ │ │ │ │ │ │ │ │ ├─ False → Folha: 0

│ │ │ │ │ │ │ │ ├─ False → Folha: 0

│ │ │ │ │ │ ├─ False → Folha: 0

│ │ │ │ ├─ False → Folha: 0

3.3) CART

Acuidade (treino): 0.8682

Acuidade (teste): 0.8258

Matriz de confusão (teste):

labels: 0, 1

| | 0 | 1 |

|- | - | - |

|0|100|10|

|1| 21|47|

Árvore:

[Sex ∈ {'female'}]

| True

| | [Pclass < 2.5]

| | | True

| | | | [Fare < 28.8562]

| | | | | True

| | | | | | [Age < 53.5]

| | | | | | | True -> Folha: 1

| | | | | | | False -> Folha: 1

| | | | | False -> Folha: 1

| | | | False

| | | | | [Fare < 20.6625]

| | | | | | True

| | | | | | | [Age < 7]

| | | | | | | | True -> Folha: 1

| | | | | | | | False

| | | | | | | | | [Fare < 8.0396]

| | | | | | | | | | True

| | | | | | | | | | | [Age < 29.5]

| | | | | | | | | | | | True -> Folha: 1

| | | | | | | | | | | | False -> Folha: 0

| | | | | | | | | | False

| | | | | | | | | | [Fare < 15.8]

| | | | | | | | | | | True -> Folha: 0

| | | | | | | | | | | False -> Folha: 0

| | | | | | | False -> Folha: 1

| False

| | [Parch < 0.5]

| | | True

| | | | [Age < 5.5]

| | | | | True

| | | | | | [Age < 3.5] -> Folha: 0/1 (dependendo do conjunto)

| | | | | False -> Folha: 0

| | | | False -> Folha: 0