Lista 3

Matéria: Aprendizado de Máquina I
Aluno: Ettore Motta Gazzinelli

17 de outubro de 2025

Sumário

Sumário

1 Implementação do Algoritmo ID3

Abaixo está o código-fonte completo da implementação da árvore de decisão ID3 em Python.

Listing 1: Implementação completa do algoritmo ID3 em Python.

```
import math
import pandas as pd

# Funcoes auxiliares (sem alteracao)
def calcular_entropia(coluna_alvo):
    if coluna_alvo.empty:
        return 0
    valores = coluna_alvo.value_counts(normalize=True)
    entropia = -sum(p * math.log2(p) for p in valores if p > 0)
    return entropia

def entropia_condicional(dataset, atributo, coluna_alvo):
    entropia_total = 0
    total = len(dataset)
    for valor in dataset[atributo].unique():
        subset = dataset[dataset[atributo] == valor]
        if not subset.empty:
            proporcao = len(subset) / total
            entropia_subset = calcular_entropia(subset[coluna_alvo])
            entropia_total += proporcao * entropia_subset
    return entropia_total

def calcular_ganho(dataset, atributo, coluna_alvo):
    return calcular_entropia(dataset[coluna_alvo]) - entropia_condicional(dataset, atr

# Classe para os nos da arvore (sem alteracao)
class NoArvoreID3:
    def __init__(self, atributo=None, filhos=None, classe=None):
        self.atributo = atributo
        self.filhos = filhos or {}
        self.classe = classe

# Classe principal da Arvore com a funcao de previsao adicionada
class ArvoreID3:
    def __init__(self):
        self.raiz = None
        self.classe_majoritaria_geral = None  # ADICIONADO: Para fallback na previsao

    def treinar(self, dataset, atributos, coluna_alvo):
        """Constroi a arvore e armazena a raiz e a classe majoritaria."""
        # ADICIONADO: Armazena a classe mais comum para usar em previsoes incertas
        self.classe_majoritaria_geral = dataset[coluna_alvo].mode()[0]
        self.raiz = self._construir_arvore(dataset, atributos, coluna_alvo)

    def _construir_arvore(self, dataset, atributos, coluna_alvo):
        # Caso base 1: todas as instancias tem a mesma classe
        if len(dataset[coluna_alvo].unique()) == 1:
            return NoArvoreID3(classe=dataset[coluna_alvo].iloc[0])

        # Caso base 2: nao ha mais atributos para dividir
        if not atributos:
            return NoArvoreID3(classe=dataset[coluna_alvo].mode()[0])

        # Melhor atributo pelo ganho
```

```
        ganhos = {atributo: calcular_ganho(dataset, atributo, coluna_alvo) for atribut
        melhor_atributo = max(ganhos, key=ganhos.get)

        no = NoArvoreID3(atributo=melhor_atributo)
        atributos_restantes = [a for a in atributos if a != melhor_atributo]

        # Itera sobre os valores unicos do melhor atributo para criar os ramos
        for valor in dataset[melhor_atributo].unique():
            subset = dataset[dataset[melhor_atributo] == valor]

            # Se um ramo nao tiver mais exemplos, cria uma folha com a classe majorit
            if subset.empty:
                no.filhos[valor] = NoArvoreID3(classe=dataset[coluna_alvo].mode()[0])
            else:
                # Chama a construcao da arvore recursivamente para o subconjunto
                no.filhos[valor] = self._construir_arvore(subset, atributos_restantes
        return no

    def predict(self, X_teste):
        """
        Recebe um DataFrame de teste e retorna uma lista de previsoes.
        """
        if self.raiz is None:
            raise ValueError("O modelo precisa ser treinado com o metodo .treinar() a

        # Usa o metodo .apply para percorrer cada linha do DataFrame e fazer a previs
        return X_teste.apply(self._prever_instancia, axis=1, args=(self.raiz,)).tolist

    def _prever_instancia(self, instancia, no_atual):
        """
        Navega na arvore recursivamente para classificar uma unica instancia (linha).
        """
        # Caso base: se chegamos a um no folha, retornamos sua classe
        if no_atual.classe is not None:
            return no_atual.classe

        # Obtem o valor do atributo de decisao para a instancia atual
        valor_da_instancia = instancia.get(no_atual.atributo)

        # Procura o ramo correspondente ao valor da instancia
        if valor_da_instancia in no_atual.filhos:
            # Continua a busca recursivamente no filho correspondente
            return self._prever_instancia(instancia, no_atual.filhos[valor_da_instanc
        else:
            # Fallback: se o valor nao foi visto no treino, retorna a classe majorita
            return self.classe_majoritaria_geral

    def imprimir_arvore(self):
        """Metodo publico para iniciar a impressao da arvore a partir da raiz."""
        if self.raiz is None:
            print("A arvore nao foi treinada.")
        else:
            # Inicia a chamada recursiva
            self._imprimir_no(self.raiz)

    def _imprimir_no(self, no: NoArvoreID3, indent: str = ""):
        """Imprime recursivamente a estrutura da arvore ID3 (multi-galhos)."""
```

```
        # CASO BASE: Se for um no folha, imprime a predicao e para a recursao.
        if no.classe is not None:
            predicao = "Sobreviveu" if no.classe == 1 else "Nao Sobreviveu"
            print(f"{indent}--> PREDIcaO: {predicao} (classe={no.classe})")
            return

        # NO DE DECISAO: Imprime o atributo que esta sendo usado para dividir.
        print(f"{indent}NO: Divisao por '{no.atributo}'")

        # Itera sobre cada possivel valor do atributo (as chaves do dicionario de filh
        # Esta e a principal diferenca em relacao ao CART.
        for valor, filho in no.filhos.items():
            # Imprime a regra para seguir este galho especifico.
            print(f"{indent}  - Se '{no.atributo}' == '{valor}':")

            # Faz a chamada recursiva para o filho, aumentando a indentacao.
            self._imprimir_no(filho, indent + "    ")
```

2 Implementação do Algoritmo C4.5

Abaixo, a implementação do algoritmo C4.5, que utiliza a razão de ganho para lidar com atributos categóricos e numéricos, além de incluir uma função para poda.

Listing 2: Implementação completa do algoritmo C4.5 em Python.

```
import math
import pandas as pd

def calcular_entropia(coluna_alvo):
    valores = coluna_alvo.value_counts(normalize=True)
    return -sum(p * math.log2(p) for p in valores if p > 0)

def entropia_condicional(dataset, atributo, coluna_alvo):
    entropia_total = 0
    total = len(dataset)

    for valor in dataset[atributo].unique():
        subset = dataset[dataset[atributo] == valor]
        proporcao = len(subset) / total
        entropia_total += proporcao * calcular_entropia(subset[coluna_alvo])

    return entropia_total

def calcular_ganho(dataset, atributo, coluna_alvo):
    entropia_inicial = calcular_entropia(dataset[coluna_alvo])
    entropia_attr = entropia_condicional(dataset, atributo, coluna_alvo)
    return entropia_inicial - entropia_attr

def calcular_razao_ganho(dataset, atributo, coluna_alvo):
    total = len(dataset)

    # - Calcula o ganho de informacao padrao
    ganho = calcular_ganho(dataset, atributo, coluna_alvo)

    # - Calcula a informacao intrinseca (split info)
    info_atributo = 0
    for valor in dataset[atributo].unique():
        proporcao = len(dataset[dataset[atributo] == valor]) / total
```

```
            if proporcao > 0:
                info_atributo -= proporcao * math.log2(proporcao)

        # - Evita divisao por zero
        if info_atributo == 0:
            return 0

        # - Razao de ganho
        return ganho / info_atributo

    def melhor_limite_numerico(dataset, atributo, coluna_alvo, usar_razao=False):
        """
    ␣␣␣␣Retorna␣o␣melhor␣ponto␣de␣corte␣para␣um␣atributo␣numerico
    ␣␣␣␣com␣base␣no␣ganho␣de␣informacao␣ou␣razao␣de␣ganho.
    ␣␣␣␣"""
        # Ordena os valores
        dataset_ordenado = dataset.sort_values(by=atributo)
        valores = dataset_ordenado[atributo].values
        classes = dataset_ordenado[coluna_alvo].values

        # Possiveis limiares entre valores consecutivos com classes diferentes
        candidatos = [(valores[i] + valores[i-1]) / 2
                      for i in range(1, len(valores))
                      if classes[i] != classes[i-1]]

        melhor_limite = None
        melhor_score = -float("inf")

        for limite in candidatos:
            # Cria coluna temporaria sem alterar a original
            dataset_temp = dataset.copy()
            dataset_temp['_tmp'] = dataset_temp[atributo].apply(
                lambda x: f"<{limite}" if x <= limite else f">{limite}"
            )

            # Calcula score
            score = calcular_razao_ganho(dataset_temp, '_tmp', coluna_alvo) if usar_razao

            if score > melhor_score:
                melhor_score = score
                melhor_limite = limite

        return melhor_limite, melhor_score

    # Classe para os nos
    class NoArvoreC45:
        def __init__(self, atributo=None, limite=None, filhos=None, classe=None):
            self.atributo = atributo
            self.limite = limite
            self.filhos = filhos or {}
            self.classe = classe

    class ArvoreC45:
        def __init__(self):
            self.raiz = None
            self.classe_majoritaria_geral = None  # ADICIONADO: Para fallback na previsao

        def treinar(self, dataset, atributos, coluna_alvo):
```

```
        """Constroi a arvore e armazena a raiz e a classe majoritaria."""
        # ADICIONADO: Armazena a classe mais comum para usar em previsoes incertas
        self.classe_majoritaria_geral = dataset[coluna_alvo].mode()[0]
        self.raiz = self._construir_arvore(dataset, atributos, coluna_alvo)

    def podar(self, dataset_poda, coluna_alvo):
        """Aplica a poda na arvore ja construida."""
        if self.raiz:
            self.raiz = self._poda_c45(self.raiz, dataset_poda, coluna_alvo)

    # (Seu metodo construir_arvore, com pequenas melhorias de robustez)
    def _construir_arvore(self, dataset, atributos, coluna_alvo):
        if dataset.empty:
            return NoArvoreC45(classe=self.classe_majoritaria_geral)
        if len(dataset[coluna_alvo].unique()) == 1:
            return NoArvoreC45(classe=dataset[coluna_alvo].iloc[0])
        if not atributos:
            return NoArvoreC45(classe=dataset[coluna_alvo].mode()[0])

        melhor_score = -float("inf")
        melhor_atributo, melhor_limite = None, None

        for atributo in atributos:
            if pd.api.types.is_numeric_dtype(dataset[atributo]):
                limite, score = melhor_limite_numerico(dataset, atributo, coluna_alvo
            else:
                score = calcular_razao_ganho(dataset, atributo, coluna_alvo)
                limite = None
            if score > melhor_score:
                melhor_score = score
                melhor_atributo = atributo
                melhor_limite = limite

        if melhor_atributo is None:
            return NoArvoreC45(classe=dataset[coluna_alvo].mode()[0])

        no = NoArvoreC45(atributo=melhor_atributo, limite=melhor_limite)
        atributos_restantes = [a for a in atributos if a != melhor_atributo]

        if melhor_limite is not None:
            subset_esq = dataset[dataset[melhor_atributo] <= melhor_limite]
            subset_dir = dataset[dataset[melhor_atributo] > melhor_limite]
            no.filhos[f"<= {melhor_limite:.3f}"] = self._construir_arvore(subset_esq,
            no.filhos[f"> {melhor_limite:.3f}"] = self._construir_arvore(subset_dir, a
        else:
            for valor in dataset[melhor_atributo].unique():
                subset = dataset[dataset[melhor_atributo] == valor]
                no.filhos[valor] = self._construir_arvore(subset, atributos_restantes
        return no

    def predict(self, X_teste):
        """
        Recebe um DataFrame de teste e retorna uma lista de previsoes.
        """
        if self.raiz is None:
            raise ValueError("O modelo precisa ser treinado com .treinar() antes de pr
        return X_teste.apply(self._prever_instancia, axis=1, args=(self.raiz,)).tolist
```

```
    def _prever_instancia(self, instancia, no_atual):
        """
        Navega na arvore recursivamente para classificar uma unica instancia.
        """
        # Caso base: se e um no folha, retorna a classe
        if no_atual.classe is not None:
            return no_atual.classe

        atributo_do_no = no_atual.atributo
        valor_da_instancia = instancia.get(atributo_do_no)

        # Se o valor for nulo na instancia de teste, nao podemos prosseguir
        if pd.isna(valor_da_instancia):
            return self.classe_majoritaria_geral

        # --- Logica Principal: Decide entre Divisao Numerica ou Categorica ---

        # CASO 1: NO DE DIVISAO NUMERICA (possui um 'limite')
        if no_atual.limite is not None:
            if valor_da_instancia <= no_atual.limite:
                # Procura a chave do filho que representa a condicao '<='
                for chave_filho in no_atual.filhos:
                    if chave_filho.startswith('<='):
                        return self._prever_instancia(instancia, no_atual.filhos[chav
            else:
                # Procura a chave do filho que representa a condicao '>'
                for chave_filho in no_atual.filhos:
                    if chave_filho.startswith('>'):
                        return self._prever_instancia(instancia, no_atual.filhos[chav

        # CASO 2: NO DE DIVISAO CATEGORICA
        else:
            # Procura o ramo correspondente ao valor da instancia
            if valor_da_instancia in no_atual.filhos:
                return self._prever_instancia(instancia, no_atual.filhos[valor_da_ins

        # FALLBACK: Se nenhum caminho for encontrado (ex: valor categorico nao visto
        return self.classe_majoritaria_geral

    def imprimir_arvore(self):
        """Metodo publico para iniciar a impressao da arvore a partir da raiz."""
        if self.raiz is None:
            print("A arvore nao foi treinada.")
        else:
            # Inicia a chamada recursiva
            self._imprimir_no(self.raiz)

    def _imprimir_no(self, no: NoArvoreC45, indent: str = ""):
        """Imprime recursivamente a estrutura da arvore C4.5 (hibrida)."""

        # CASO BASE: Se for um no folha, imprime a previsao e para.
        if no.classe is not None:
            predicao = "Sobreviveu" if no.classe == 1 else "Nao Sobreviveu"
            print(f"{indent}--> PREDIcaO: {predicao} (classe={no.classe})")
            return

        # --- NO DE DECISAO: LoGICA HIBRIDA ---
```

```
        # CASO 1: A divisao e NUMERICA (o no possui um 'limite')
        if no.limite is not None:
            print(f"{indent}NO: Divisao Numerica por '{no.atributo}'")

            # Itera sobre os dois filhos (<= e >)
            for condicao, filho in no.filhos.items():
                print(f"{indent}  - Se valor {condicao}:")
                self._imprimir_no(filho, indent + "    ")

        # CASO 2: A divisao e CATEGORICA (semelhante ao ID3)
        else:
            print(f"{indent}NO: Divisao Categorica por '{no.atributo}'")

            # Itera sobre todos os valores/galhos possiveis
            for valor, filho in no.filhos.items():
                print(f"{indent}  - Se '{no.atributo}' == '{valor}':")
                self._imprimir_no(filho, indent + "    ")

    @staticmethod
    def poda_c45(no, dataset, coluna_alvo):
        """
        Poda recursiva de uma arvore do tipo C4.5 (poda pessimista).
        """
        # Se for folha, nada a fazer
        if no.classe is not None:
            return no

        # Poda recursiva (bottom-up)
        for valor, filho in list(no.filhos.items()):
            if no.limite is not None:
                if valor.startswith("<="):
                    subset = dataset[dataset[no.atributo] <= no.limite]
                else:
                    subset = dataset[dataset[no.atributo] > no.limite]
            else:
                subset = dataset[dataset[no.atributo] == valor]

            no.filhos[valor] = ArvoreC45.poda_c45(filho, subset, coluna_alvo)

        # ----------------------------
        # Avaliar se deve podar o no atual
        # ----------------------------
        classe_mais_comum = dataset[coluna_alvo].mode()[0]

        # Erro se o no fosse folha
        erros_folha = sum(dataset[coluna_alvo] != classe_mais_comum)
        E_folha = (erros_folha + 0.5) / len(dataset)

        # Erro se mantivesse os filhos
        E_filhos = 0
        for valor, filho in no.filhos.items():
            if filho.classe is not None:
                if no.limite is not None:
                    if valor.startswith("<="):
                        subset = dataset[dataset[no.atributo] <= no.limite]
                    else:
                        subset = dataset[dataset[no.atributo] > no.limite]
                else:
```

```
                subset = dataset[dataset[no.atributo] == valor]
            if len(subset) > 0:
                erros = sum(subset[coluna_alvo] != filho.classe)
                E_filhos += erros / len(dataset)

        # Comparar e decidir podar
        if E_folha <= E_filhos:
            return NoArvoreC45(classe=classe_mais_comum)
        else:
            return no
```

3 Implementação do Algoritmo CART

Por fim, a implementação do algoritmo CART (Classification and Regression Trees), que constrói uma árvore estritamente binária utilizando o índice Gini como critério de divisão.

Listing 3: Implementação completa do algoritmo CART em Python.

```
import pandas as pd
import numpy as np
from typing import Dict, Any, Optional, Tuple, List

class NoArvoreCART:
    def __init__(self, atributo: Optional[str] = None, limite: Any = None,
                 filho_esq: Optional['NoArvoreCART'] = None, filho_dir: Optional['NoA
                 classe: Optional[Any] = None):
        self.atributo = atributo  # Atributo usado para a divisao
        self.limite = limite      # Limite (numerico) ou valor (categorico) para a di
        self.filho_esq = filho_esq
        self.filho_dir = filho_dir
        self.classe = classe      # Valor da previsao (apenas para nos folha)

class ArvoreCART:
    def __init__(self, max_depth: Optional[int] = 10, min_samples_leaf: int = 1):
        self.raiz = None
        self.max_depth = max_depth
        self.min_samples_leaf = min_samples_leaf

    # --- Metodos de Calculo do Gini (sem alteracoes) ---
    @staticmethod
    def _indice_gini(coluna_alvo: pd.Series) -> float:
        if coluna_alvo.empty: return 0
        proporcoes = coluna_alvo.value_counts(normalize=True)
        return 1 - sum(proporcoes ** 2)

    def _ganho_gini_ponderado(self, subsets: List[pd.Series], total_amostras: int) ->
        gini_ponderado = 0
        for subset in subsets:
            if not subset.empty:
                gini_ponderado += (len(subset) / total_amostras) * self._indice_gini(s
        return gini_ponderado

    # --- Metodos para Encontrar o Melhor Split ---
    def _melhor_split_numerico(self, dataset: pd.DataFrame, atributo: str, coluna_alvo
        melhor_limite, melhor_score = None, float('inf')

        valores_unicos = sorted(dataset[atributo].unique())
        candidatos = [(valores_unicos[i] + valores_unicos[i-1]) / 2 for i in range(1,
```

```
        for limite in candidatos:
            subset_esq = dataset[dataset[atributo] <= limite]
            subset_dir = dataset[dataset[atributo] > limite]

            if len(subset_esq) < self.min_samples_leaf or len(subset_dir) < self.min_
                continue

            gini_atual = self._ganho_gini_ponderado([subset_esq[coluna_alvo], subset_
            if gini_atual < melhor_score:
                melhor_score, melhor_limite = gini_atual, limite

        return melhor_limite, melhor_score

    # CORREcaO: Logica para split binario em atributos categoricos
    def _melhor_split_categorico(self, dataset: pd.DataFrame, atributo: str, coluna_a
        melhor_valor, melhor_score = None, float('inf')

        for valor in dataset[atributo].unique():
            subset_esq = dataset[dataset[atributo] == valor]
            subset_dir = dataset[dataset[atributo] != valor]

            if len(subset_esq) < self.min_samples_leaf or len(subset_dir) < self.min_
                continue

            gini_atual = self._ganho_gini_ponderado([subset_esq[coluna_alvo], subset_
            if gini_atual < melhor_score:
                melhor_score, melhor_valor = gini_atual, valor

        return melhor_valor, melhor_score

    # --- Metodo Principal de Treinamento ---
    def treinar(self, X: pd.DataFrame, y: pd.Series):
        dataset = pd.concat([X, y], axis=1)
        self.raiz = self._construir_arvore(dataset, y.name, 0)

    def _construir_arvore(self, dataset: pd.DataFrame, coluna_alvo: str, depth: int) -
        # --- Criterios de Parada (criacao de no folha) ---
        classe_majoritaria = dataset[coluna_alvo].mode()[0]
        if (len(dataset[coluna_alvo].unique()) == 1 or
            len(dataset) < self.min_samples_leaf * 2 or
            (self.max_depth is not None and depth >= self.max_depth)):
            return NoArvoreCART(classe=classe_majoritaria)

        # --- Encontrar o melhor split possivel em todos os atributos ---
        melhor_split = {'score': float('inf')}
        atributos = list(dataset.columns)
        atributos.remove(coluna_alvo)

        for atributo in atributos:
            if pd.api.types.is_numeric_dtype(dataset[atributo]):
                limite, score = self._melhor_split_numerico(dataset, atributo, coluna_
            else: # Categorico
                limite, score = self._melhor_split_categorico(dataset, atributo, colun

            if score < melhor_split['score']:
                melhor_split = {'atributo': atributo, 'limite': limite, 'score': score
```

```
    # Se nenhum split valido foi encontrado, cria uma folha
    if melhor_split['score'] == float('inf'):
        return NoArvoreCART(classe=classe_majoritaria)

    # --- Dividir o dataset com base no melhor split encontrado ---
    attr, limit = melhor_split['atributo'], melhor_split['limite']
    if pd.api.types.is_numeric_dtype(dataset[attr]):
        subset_esq = dataset[dataset[attr] <= limit]
        subset_dir = dataset[dataset[attr] > limit]
    else: # Categorico
        subset_esq = dataset[dataset[attr] == limit]
        subset_dir = dataset[dataset[attr] != limit]

    # --- Construir os filhos recursivamente ---
    filho_esq = self._construir_arvore(subset_esq, coluna_alvo, depth + 1)
    filho_dir = self._construir_arvore(subset_dir, coluna_alvo, depth + 1)

    return NoArvoreCART(atributo=attr, limite=limit, filho_esq=filho_esq, filho_d

def prever(self, X: pd.DataFrame) -> List[Any]:
    """Faz previsoes para um DataFrame de teste."""
    if self.raiz is None:
        raise ValueError("A arvore nao foi treinada. Chame o metodo .treinar() pri
    return [self._prever_amostra(self.raiz, amostra) for _, amostra in X.iterrows

def _prever_amostra(self, no: NoArvoreCART, amostra: pd.Series) -> Any:
    """Navega recursivamente na arvore para classificar uma unica amostra."""
    if no.classe is not None:
        return no.classe

    valor_amostra = amostra.get(no.atributo)

    if valor_amostra is None:
        return self._prever_amostra(no.filho_esq, amostra) # Decisao arbitraria

    # Verifica se o atributo e numerico ou categorico pelo tipo do limite
    if isinstance(no.limite, (int, float)): # Divisao Numerica
        if valor_amostra <= no.limite:
            return self._prever_amostra(no.filho_esq, amostra)
        else:
            return self._prever_amostra(no.filho_dir, amostra)
    else: # Divisao Categorica
        if valor_amostra == no.limite:
            return self._prever_amostra(no.filho_esq, amostra)
        else:
            return self._prever_amostra(no.filho_dir, amostra)

def poda(self, dataset, coluna_alvo, alpha=0.0):
    self.raiz = self._poda_subarvore(self.raiz, dataset, coluna_alvo, alpha)

def _poda_subarvore(self, no, dataset, coluna_alvo, alpha):
    if no.atributo is None:
        return no

    for valor, filho in list(no.filhos.items()):
        if no.limite is not None:
            subset = dataset[dataset[no.atributo] <= no.limite] if valor == "<=" e
        else:
```

```
                subset = dataset[dataset[no.atributo] == valor]
                no.filhos[valor] = self._poda_subarvore(filho, subset, coluna_alvo, alpha)

        erro_total = self._erro_subarvore(no, dataset, coluna_alvo)
        classe_mais_comum = dataset[coluna_alvo].mode()[0]
        erro_no = sum(dataset[coluna_alvo] != classe_mais_comum)
        n_folhas = self._conta_folhas(no)

        if n_folhas > 1 and (erro_no + alpha) <= (erro_total + alpha * n_folhas):
            return NoArvoreCART(classe=classe_mais_comum)

        no.classe = classe_mais_comum
        return no

    def imprimir_arvore(self):
        """Metodo publico para iniciar a impressao da arvore a partir da raiz."""
        if self.raiz is None:
            print("A arvore nao foi treinada.")
        else:
            self._imprimir_no(self.raiz)

    def _imprimir_no(self, no: NoArvoreCART, indent: str = ""):
        """Imprime recursivamente a estrutura da arvore."""
        if no.classe is not None:
            predicao = "Sobreviveu" if no.classe == 1 else "Nao Sobreviveu"
            print(f"{indent}-> PREDIcaO: {predicao} (classe={no.classe})")
            return

        if isinstance(no.limite, (int, float)): # Regra Numerica
            regra = f"Se '{no.atributo}' <= {no.limite:.3f}"
        else: # Regra Categorica
            regra = f"Se '{no.atributo}' == '{no.limite}'"

        print(f"{indent}{regra}:")

        self._imprimir_no(no.filho_esq, indent + "  |-- V: ")

        print(f"{indent}Senao:")
        self._imprimir_no(no.filho_dir, indent + "  |-- F: ")
```

4 Treinamento e Avaliação do Modelo ID3

Com a classe `ArvoreID3` implementada, o próximo passo é instanciar o modelo, treiná-lo com os dados de treino e, em seguida, avaliar seu desempenho no conjunto de teste. O script a seguir realiza essas etapas.

Listing 4: Script para treinar, visualizar e avaliar o modelo ID3.

```
from ID3 import ArvoreID3

# --- ETAPA 1: DEFINIR AS VARIAVEIS PARA O TREINO ---
# Criar o dataframe de treino, que o metodo .treinar() espera
df_treino = pd.concat([X_train, y_train], axis=1)

# Definir a lista de atributos (colunas de features)
atributos = list(X_train.columns)
# Definir o nome da coluna alvo
coluna_alvo = 'Survived'
```

```
# --- ETAPA 2: TREINAR O MODELO ---
# Instanciar a sua classe
modelo_id3 = ArvoreID3()

# Chamar o metodo de treino
modelo_id3.treinar(df_treino, atributos, coluna_alvo)
print("? Modelo ID3 treinado com sucesso!")


# --- ETAPA 3: VISUALIZAR A ARVORE ---
print("\n" + "="*60)
print("ESTRUTURA DA ARVORE DE DECISAO GERADA")
print("="*60)
modelo_id3.imprimir_arvore()
print("="*60 + "\n")


# --- ETAPA 4: FAZER PREVISOES E AVALIAR ---
# Usar o novo metodo .predict() diretamente no conjunto de teste
print("? Realizando previsoes no conjunto de teste...")
y_pred = modelo_id3.predict(X_test)
print("? Previsoes concluidas!")

# Importar as funcoes de metrica

# Calcular as metricas
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)

# Imprimir os resultados
print("\n" + "="*50)
print("METRICAS DE DESEMPENHO DO MODELO ID3")
print("="*50)
print(f"Acuracia (Accuracy): {accuracy:.4f}")
print(f"Precisao (Precision): {precision:.4f}")
print(f"Revocacao (Recall): {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
print("="*50)

print("\nRelatorio de Classificacao Completo:")
print(classification_report(y_test, y_pred, zero_division=0))
```

5 Resultados do Modelo ID3

A performance do modelo ID3 foi avaliada utilizando as métricas padrão de classificação. A Tabela 1 resume os principais indicadores de desempenho, enquanto a Tabela 2 apresenta o relatório de classificação detalhado por classe.

https://github.com/Ettorew/Aprendizado$_{d}e_{M}aquina_{I}$/blob/main/$saidas_{a}rvores/ID3.txtrvoreGeradaID3$

6 Treinamento e Avaliação do Modelo C4.5

De forma análoga ao ID3, o script a seguir é utilizado para treinar, visualizar a árvore gerada e avaliar a performance do modelo C4.5.

Tabela 1: Métricas de Desempenho do Modelo ID3.

Métrica	**Valor**
Acurácia (Accuracy)	0.6771
Precisão (Precision)	0.6346
Revocação (Recall)	0.3837
F1-Score	0.4783

Tabela 2: Relatório de Classificação Completo do Modelo ID3.

	Precision	**Recall**	**F1-Score**	**Support**
Classe 0 (Não Sobreviveu)	0.69	0.86	0.77	137
Classe 1 (Sobreviveu)	0.63	0.38	0.48	86
Accuracy			0.68	223
Macro Avg	0.66	0.62	0.62	223
Weighted Avg	0.67	0.68	0.66	223

Listing 5: Script para treinar, visualizar e avaliar o modelo C4.5.

```
from C45 import ArvoreC45

# --- ETAPA 1: DEFINIR AS VARIAVEIS PARA O TREINO ---
# Criar o dataframe de treino, que o metodo .treinar() espera
df_treino = pd.concat([X_train, y_train], axis=1)

# Definir a lista de atributos (colunas de features)
atributos = list(X_train.columns)
# Definir o nome da coluna alvo
coluna_alvo = 'Survived'


# --- ETAPA 2: TREINAR O MODELO ---
# Instanciar a sua classe
modelo_c45 = ArvoreC45()

# Chamar o metodo de treino
modelo_c45.treinar(df_treino, atributos, coluna_alvo)
print("? Modelo c45 treinado com sucesso!")


# --- ETAPA 3: VISUALIZAR A ARVORE ---
print("\n" + "="*60)
print("ESTRUTURA DA ARVORE DE DECISAO GERADA")
print("="*60)
modelo_c45.imprimir_arvore()
print("="*60 + "\n")


# --- ETAPA 4: FAZER PREVISOES E AVALIAR ---
# Usar o novo metodo .predict() diretamente no conjunto de teste
print("? Realizando previsoes no conjunto de teste...")
y_pred = modelo_c45.predict(X_test)
print("? Previsoes concluidas!")

# Importar as funcoes de metrica

# Calcular as metricas
```

```
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)

# Imprimir os resultados
print("\n" + "="*50)
print("METRICAS DE DESEMPENHO DO MODELO c45")
print("="*50)
print(f"Acuracia (Accuracy): {accuracy:.4f}")
print(f"Precisao (Precision): {precision:.4f}")
print(f"Revocacao (Recall): {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
print("="*50)

print("\nRelatorio de Classificacao Completo:")
print(classification_report(y_test, y_pred, zero_division=0))
```

7 Resultados do Modelo C4.5

O modelo C4.5, que utiliza a razão de ganho, apresentou os resultados de performance sumarizados na Tabela 3 e detalhados no relatório de classificação da Tabela 4.

Tabela 3: Métricas de Desempenho do Modelo C4.5.

Métrica	**Valor**
Acurácia (Accuracy)	0.8027
Precisão (Precision)	0.8387
Revocação (Recall)	0.6047
F1-Score	0.7027

Tabela 4: Relatório de Classificação Completo do Modelo C4.5.

	Precision	**Recall**	**F1-Score**	**Support**
Classe 0 (Não Sobreviveu)	0.79	0.93	0.85	137
Classe 1 (Sobreviveu)	0.84	0.60	0.70	86
Accuracy			0.80	223
Macro Avg	0.81	0.77	0.78	223
Weighted Avg	0.81	0.80	0.79	223

https://github.com/Ettorew/Aprendizado$_d$e$_M$aquina$_I$/blob/main/saidas$_a$rvores/C45.txtrvoreGeradaC45

8 Preparação de Dados para o CART: One-Hot Encoding

Diferente dos modelos ID3 e C4.5 que foram implementados para lidar com features categóricas diretamente, o modelo CART performa melhor com entradas puramente numéricas. Para isso, aplicamos a técnica de One-Hot Encoding, que transforma cada categoria de uma variável em uma nova coluna binária (0 ou 1).

Listing 6: Engenharia de features com One-Hot Encoding.

```
# Celula de Engenharia de Features (One-Hot Encoding)

print("--- Preparando dados com One-Hot Encoding para o CART ---")
```

```
# Usamos o df_modelo que ja tem as features tratadas ('Title', 'AgeGroup', etc.)
# A funcao get_dummies converte todas as colunas de tipo 'object' em colunas binarias
df_ohe = pd.get_dummies(df_modelo, drop_first=True)

# Separando X e y novamente com os novos dados
X_ohe = df_ohe.drop('Survived', axis=1)
y_ohe = df_ohe['Survived']

# E crucial fazer um novo split para garantir a consistencia
X_train_ohe, X_test_ohe, y_train_ohe, y_test_ohe = train_test_split(
    X_ohe, y_ohe, test_size=0.25, random_state=42, stratify=y_ohe
)

print(f"Formato original das features: {X_train.shape}")
print(f"Novo formato com One-Hot Encoding: {X_train_ohe.shape}")
print("\nNovas colunas criadas:", list(X_train_ohe.columns))
```

9 Treinamento e Avaliação do Modelo CART

Com os dados devidamente transformados, o modelo CART pode ser treinado. O script a seguir instancia a classe `ArvoreCART` com hiperparâmetros de controle (`max_depth` e `min_samples_leaf`), executa o treinamento e avalia a performance do modelo final.

Listing 7: Script para treinar, visualizar e avaliar o modelo CART com dados pré-processados.

```
from CART import ArvoreCART

# --- ETAPA 1: INSTANCIAR O MODELO ---
# Instanciamos o modelo com hiperparametros para controlar o crescimento da arvore
modelo_cart = ArvoreCART(max_depth=7, min_samples_leaf=5)


# --- ETAPA 2: TREINAR O MODELO ---
# O novo metodo .treinar() espera X e y separados, como no Scikit-learn.
# Nao precisamos mais criar o 'df_treino' ou passar os nomes das colunas.
modelo_cart.treinar(X_train_ohe, y_train_ohe)
print("? Modelo CART treinado com sucesso!")


# --- ETAPA 3: VISUALIZAR A ARVORE ---
# Esta chamada agora funcionara se voce adicionou o metodo ao seu arquivo .py
print("\n" + "="*60)
print("ESTRUTURA DA ARVORE DE DECISAO GERADA PELO CART")
print("="*60)
modelo_cart.imprimir_arvore()
print("="*60 + "\n")


# --- ETAPA 4: FAZER PREVISOES E AVALIAR ---
# Usamos o metodo .prever() que criamos
print("? Realizando previsoes no conjunto de teste...")
y_pred = modelo_cart.prever(X_test_ohe)
print("? Previsoes concluidas!")

# Calcular as metricas (com o erro de digitacao corrigido)
accuracy = accuracy_score(y_test_ohe, y_pred)
precision = precision_score(y_test_ohe, y_pred, zero_division=0)
recall = recall_score(y_test_ohe, y_pred, zero_division=0) # Corrigido de 'zeo_divisio
```

```
f1 = f1_score(y_test_ohe, y_pred, zero_division=0)

# Imprimir os resultados de forma clara
print("\n" + "="*50)
print("METRICAS DE DESEMPENHO DO MODELO CART")
print("="*50)
print(f"Acuracia (Accuracy): {accuracy:.4f}")
print(f"Precisao (Precision): {precision:.4f}")
print(f"Revocacao (Recall): {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
print("="*50)

# O classification_report e uma otima forma de ver tudo de uma vez
print("\nRelatorio de Classificacao Completo:")
print(classification_report(y_test_ohe, y_pred, zero_division=0))
```

10 Resultados do Modelo CART

A performance do modelo CART, que utiliza o índice Gini para construir uma árvore binária, é apresentada nas tabelas a seguir. A Tabela 5 mostra as métricas consolidadas e a Tabela 6 o detalhamento por classe.

Tabela 5: Métricas de Desempenho do Modelo CART.

Métrica	**Valor**
Acurácia (Accuracy)	0.7982
Precisão (Precision)	0.7971
Revocação (Recall)	0.6395
F1-Score	0.7097

Tabela 6: Relatório de Classificação Completo do Modelo CART.

	Precision	**Recall**	**F1-Score**	**Support**
Classe 0 (Não Sobreviveu)	0.80	0.90	0.85	137
Classe 1 (Sobreviveu)	0.80	0.64	0.71	86
Accuracy			0.80	223
Macro Avg	0.80	0.77	0.78	223
Weighted Avg	0.80	0.80	0.79	223

https://github.com/Ettorew/Aprendizado$_d$e$_M$aquina$_I$/blob/main/saidas$_a$rvores/CART.txtrvoreGeradaCART