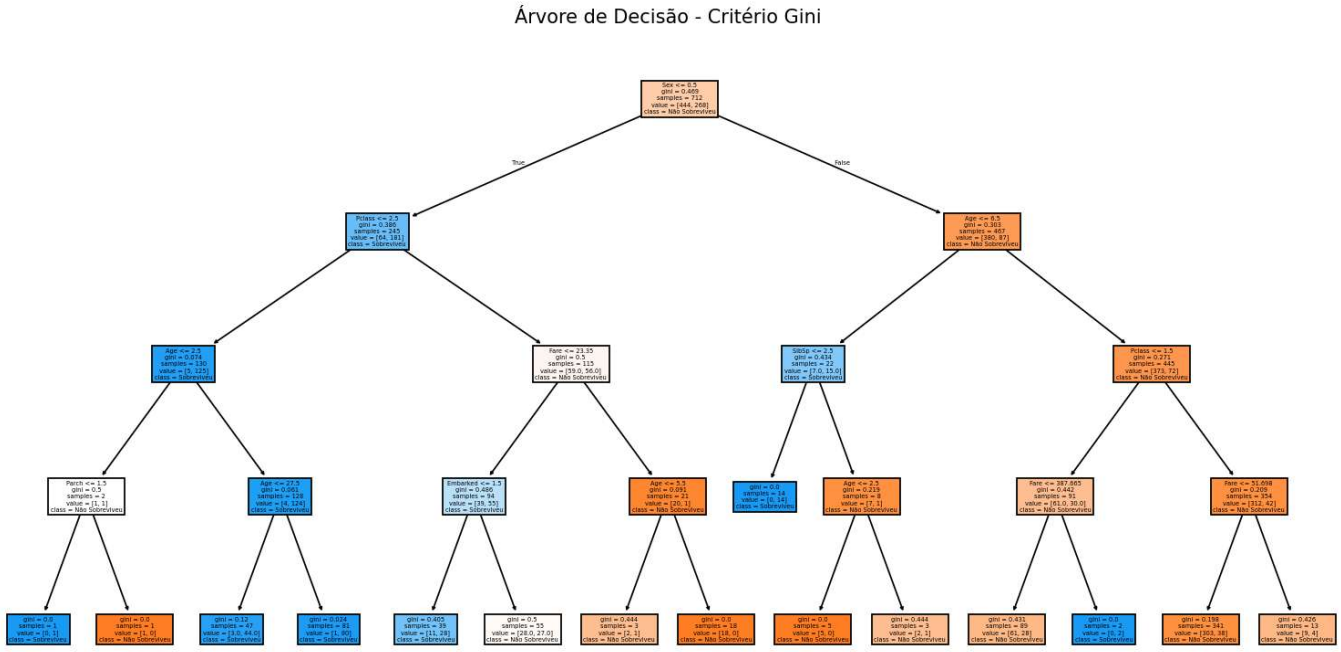
793605 - Caio Faria Diniz

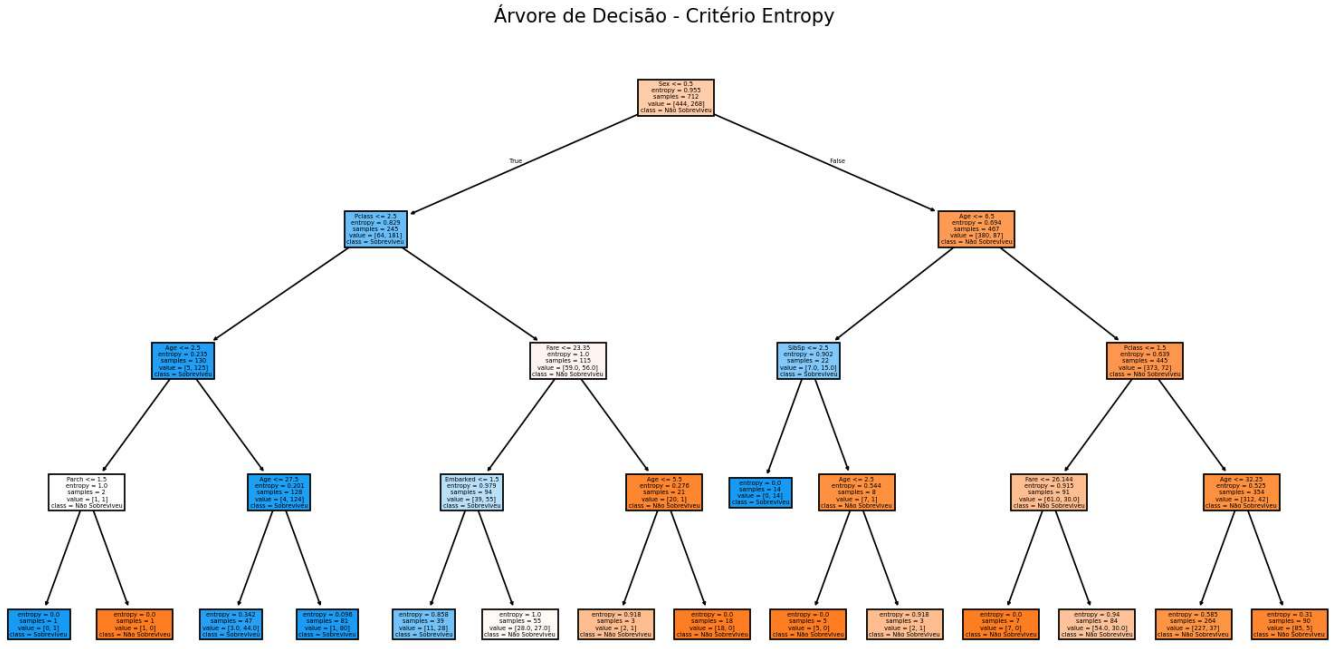
Lista 03

Questão 1

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Carregar dataset
arquivo = 'titanic.csv'
df = pd.read_csv(arquivo)

# Removendo colunas desnecessárias
df.drop(columns=['Cabin', 'Ticket', 'Name', 'PassengerId'], inplace=True)

# Preenchendo valores ausentes
df['Age'].fillna(df['Age'].median(), inplace=True)
df['Embarked'].fillna(df['Embarked'].mode()[0], inplace=True)

# Transformando dados categóricos
categoricas = ['Sex', 'Embarked']
label_encoder = LabelEncoder()
for col in categoricas:
    df[col] = label_encoder.fit_transform(df[col])

# Definir variáveis independentes e dependentes
X = df.drop(columns=['Survived'])
y = df['Survived']

# Dividir dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Criando os modelos
gini_tree = DecisionTreeClassifier(criterion='gini', max_depth=4, random_state=42)
entropy_tree = DecisionTreeClassifier(criterion='entropy', max_depth=4, random_state=42)

# Treinando os modelos
gini_tree.fit(X_train, y_train)
entropy_tree.fit(X_train, y_train)

# Fazendo previsões
gini_pred = gini_tree.predict(X_test)
entropy_pred = entropy_tree.predict(X_test)

# Avaliação do modelo
def avaliar_modelo(y_real, y_previsto, criterio):
    print(f"\n### Avaliação - {criterio} ###")
    print("Acurácia:", accuracy_score(y_real, y_previsto))
    print(classification_report(y_real, y_previsto))

avaliar_modelo(y_test, gini_pred, "Gini")
avaliar_modelo(y_test, entropy_pred, "Entropy")

# Função para exibir matriz de confusão
def exibir_matriz_confusao(y_real, y_previsto, titulo):
    cm = confusion_matrix(y_real, y_previsto)
    plt.figure(figsize=(5,4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Não Sobreviveu', 'Sobreviveu'], yticklabels=['Não Sobreviveu', 'Sob
    plt.xlabel('Previsto')
    plt.ylabel('Real')
    plt.title(titulo)
    plt.show()

exibir_matriz_confusao(y_test, gini_pred, "Matriz de Confusão - Gini")
exibir_matriz_confusao(y_test, entropy_pred, "Matriz de Confusão - Entropy")

# Função para visualizar árvore de decisão
def visualizar_arvore(modelo, titulo):
    plt.figure(figsize=(12, 8))
    plot_tree(modelo, feature_names=X.columns, class_names=['Não Sobreviveu', 'Sobreviveu'], filled=True)
```

```
    plt.title(titulo)
    plt.show()

visualizar_arvore(gini_tree, "Árvore de Decisão - Critério Gini")
visualizar_arvore(entropy_tree, "Árvore de Decisão - Critério Entropy")
```





Explicação dos Critérios Gini e Entropy:

1. **Critério Gini**: Utilizado para medir a impureza de um nó em uma árvore de decisão. A fórmula é:

   Gini = 1 - ∑ pi^2

   Onde p_i representa a proporção de amostras pertencentes a uma classe no nó.

2. **Critério Entropy**: Avalia o grau de desordem na distribuição das classes dentro de um nó. A fórmula utilizada é:

   Entropy = - ∑ pi*log(pi, 2)

Comparação entre os Critérios:

- O Gini geralmente resulta em divisões mais puras em cada nó, levando a decisões mais rápidas.
- A Entropy é mais sensível a pequenas variações nos dados, podendo gerar árvores ligeiramente diferentes.
- No geral, o desempenho entre ambos é semelhante, mas o critério Gini pode ser computacionalmente mais eficiente.

Questão 2

A Árvore de Decisão pode ser ajustada por diversos hiperparâmetros que influenciam diretamente sua complexidade e capacidade de generalização.

Principais Hiperparâmetros e seus Impactos

1. **max_depth (Profundidade Máxima)**
   - Define a profundidade máxima da árvore.
   - **Impacto:**
     - Muito alta → Overfitting (modelo aprende ruído dos dados).
     - Muito baixa → Underfitting (modelo perde padrões importantes).
2. **max_features (Número Máximo de Atributos por Divisão)**
   - Determina quantos atributos são considerados para cada divisão.
   - **Valores possíveis:**
     - "sqrt" → Raiz quadrada do número total de atributos.
     - "log2" → Logaritmo base 2 do número de atributos.
     - None → Usa todos os atributos.
     - Frações (0.2, 0.4, 0.6, 0.8) → Define a proporção de atributos usados.
   - **Impacto:**
     - Valor alto → Overfitting (pode capturar padrões irrelevantes).
     - Valor baixo → Mais generalização, melhora o desempenho.
3. **min_samples_split (Mínimo de Amostras para Divisão)**
   - Define o número mínimo de amostras necessário para dividir um nó.
   - **Impacto:**
     - Baixo (2, 5) → Gera muitas divisões, aumentando a complexidade.
     - Alto (20, 50) → Reduz overfitting, garantindo que os nós tenham dados suficientes.
4. **min_samples_leaf (Mínimo de Amostras por Folha)**
   - Determina o número mínimo de amostras que um nó folha pode ter.
   - **Impacto:**
     - Baixo (1, 5) → Modelo mais detalhado, mas pode sofrer overfitting.
     - Alto (10, 20) → Árvore mais generalizada, reduz overfitting.
5. **criterion (Critério de Impureza - Gini vs Entropy)**
   - Define a métrica usada para medir a impureza dos nós.
   - **Impacto:**
     - Gini → Mais eficiente computacionalmente.
     - Entropy → Pode gerar divisões mais informativas.
6. **max_leaf_nodes (Número Máximo de Nós Folha)**
   - Limita a quantidade de folhas na árvore.
   - **Impacto:**
     - Menos folhas → Árvore mais simples, generaliza melhor.
     - Mais folhas → Pode levar a overfitting.
7. **class_weight (Peso das Classes)**
   - Ajusta o peso das classes para lidar com desbalanceamento de dados.
   - **Impacto:**
     - Se a classe minoritária for menos representada, pode ajudar a equilibrar a decisão do modelo.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier

# Definição dos hiperparâmetros para GridSearch
parametros = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 7, None],
    'max_features': ['sqrt', 'log2', None],
    'min_samples_split': [2, 4, 8, 16],
    'min_samples_leaf': [1, 3, 5, 10],
```

```
}

dados = pd.read_csv('titanic.csv')
dados.drop(columns=['Cabin', 'Ticket', 'Name', 'PassengerId'], inplace=True)

dados['Age'].fillna(dados['Age'].median(), inplace=True)
dados['Embarked'].fillna(dados['Embarked'].mode()[0], inplace=True)

colunas_categoricas = ['Sex', 'Embarked']
dados[colunas_categoricas] = dados[colunas_categoricas].apply(LabelEncoder().fit_transform)

X = dados.drop(columns=['Survived'])
y = dados['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Aplicar GridSearchCV para encontrar a melhor configuração
modelo_otimizado = GridSearchCV(
    estimator=DecisionTreeClassifier(random_state=42),
    param_grid=parametros,
    cv=5,
    n_jobs=-1,
    verbose=2,
)

modelo_otimizado.fit(X_train, y_train)

print("Melhores hiperparâmetros:", modelo_otimizado.best_params_)
print("Melhor desempenho:", modelo_otimizado.best_score_)
```

Questão 3

1. GridSearchCV: Realiza uma busca exaustiva testando todas as combinações possíveis dos hiperparâmetros especificados. Garante encontrar a melhor configuração, mas pode ser computacionalmente caro.
2. RandomizedSearchCV: Amostra aleatoriamente um subconjunto de combinações possíveis, reduzindo o custo computacional, mas sem a garantia de encontrar a melhor configuração absoluta.
3. BayesSearchCV: Utiliza otimização Bayesiana para escolher inteligentemente os hiperparâmetros a serem testados com base nos resultados anteriores, balanceando precisão e eficiência.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
from sklearn.tree import DecisionTreeClassifier
from skopt import BayesSearchCV

# Carregar os dados
dados = pd.read_csv("titanic.csv")

dados.drop(columns=['Cabin', 'Ticket', 'Name', 'PassengerId'], inplace=True)

dados['Age'].fillna(dados['Age'].median(), inplace=True)
dados['Embarked'].fillna(dados['Embarked'].mode()[0], inplace=True)

colunas_categoricas = ['Sex', 'Embarked']
dados[colunas_categoricas] = dados[colunas_categoricas].apply(LabelEncoder().fit_transform)

X = dados.drop(columns=['Survived'])
y = dados['Survived']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)

# Definir hiperparâmetros
hiperparametros = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 3, 5, 7, 10],
    'min_samples_split': [2, 4, 8],
    'min_samples_leaf': [1, 3, 5],
    'max_features': ['sqrt', 'log2', None]
}

def avaliar_modelo(modelo, nome):
    modelo.fit(X_train, y_train)
    print(f"Melhores parâmetros ({nome}):", modelo.best_params_)
    print(f"Acurácia média ({nome}):", modelo.best_score_)
    print("-" * 50)

# GridSearchCV
```

```
grid_search = GridSearchCV(
    DecisionTreeClassifier(random_state=42),
    param_grid=hiperparametros,
    cv=5,
    n_jobs=-1,
    verbose=1
)
avaliar_modelo(grid_search, "GridSearchCV")

# RandomizedSearchCV
random_search = RandomizedSearchCV(
    DecisionTreeClassifier(random_state=42),
    param_distributions=hiperparametros,
    n_iter=10,
    cv=5,
    n_jobs=-1,
    verbose=1,
    random_state=42
)
avaliar_modelo(random_search, "RandomizedSearchCV")

# BayesSearchCV
bayes_search = BayesSearchCV(
    DecisionTreeClassifier(random_state=42),
    search_spaces=hiperparametros,
    cv=5,
    n_iter=10,
    n_jobs=-1,
    verbose=1,
    random_state=42
)
avaliar_modelo(bayes_search, "BayesSearchCV")
```

Questão 4

Considere um modelo de classificação binária que identifica fraudes em transações financeiras. Suponha que a base de dados tenha um número significativamente maior de transações legítimas do que fraudulentas. Com base nas métricas de avaliação precisão (precision), revocação (recall) e F1-score, analise as seguintes afirmações:

I. Se o modelo tem alta precisão, isso significa que a maioria das transações classificadas como fraudulentas realmente são fraudes, mas pode estar deixando muitas fraudes reais passarem despercebidas.

II. Se o modelo tem alta revocação, isso significa que ele consegue identificar quase todas as fraudes, mas pode incluir muitas transações legítimas como fraudulentas.

III. O F1-score é útil quando há um grande desequilíbrio entre classes, pois equilibra precisão e revocação, sendo sempre a média aritmética dessas métricas.

Qual das alternativas abaixo é correta?

A) Apenas I e II

B) Apenas II e III

C) Apenas I e III

D) I, II e III

> ***Resposta: Letra A***

Questão 5

Um modelo de diagnóstico de doenças raras foi desenvolvido para identificar pacientes infectados com uma condição grave. Com base nas métricas precisão (precision) e revocação (recall), analise as seguintes afirmações:

I. Se a revocação for aumentada, mais casos reais da doença serão detectados, mas isso pode aumentar os falsos positivos, reduzindo a precisão.

II. Se um modelo tem alta precisão, isso significa que a maioria dos pacientes diagnosticados como positivos realmente tem a doença, mas isso não garante que todos os doentes tenham sido identificados.

III. Para um diagnóstico de doenças altamente letais, um modelo com alta precisão sempre é preferível a um modelo com alta revocação, pois evita alarmes falsos e diagnósticos errados.

Qual das alternativas abaixo é correta?

A) Apenas I e II

B) Apenas I e III

C) Apenas II e III

D) I, II e III

> **Resposta: Letra A**

Questão 6

ID3

- Baseia-se exclusivamente na entropia e no ganho de informação para selecionar os atributos que farão a separação dos dados.
- Ignora exemplos com valores ausentes, reduzindo a quantidade de dados disponíveis para o treinamento.
- Funciona apenas com atributos categóricos.
- Não aplica poda após a construção da árvore, o que pode levar a sobreajuste.
- Gera árvores com múltiplos ramos por nó, ou seja, se um atributo possuir 10 valores distintos, o nó pode ter 10 ramos.

C4.5

- Introduz a razão de ganho (ganho de informação normalizado), reduzindo o viés do ID3 em favorecer atributos com muitos valores distintos.
- Lida com valores ausentes ao calcular probabilidades para cada possível valor do atributo ausente.
- Suporta atributos contínuos e determina automaticamente pontos de corte para dividi-los em faixas.
- Aplica poda pós-construção (post-pruning) para reduzir o tamanho da árvore e melhorar a generalização.
- Embora permita múltiplos ramos, pode converter a árvore para uma estrutura binária, simplificando sua representação.

> **Resumo**
>
> | Característica | ID3 | C4.5 |
> |---|---|---|
> | **Critério de Escolha** | Ganho de Informação | Razão de Ganho |
> | **Atributos Numéricos** | Não suporta | Suporta e define pontos de corte |
> | **Valores Ausentes** | Ignora os exemplos | Trabalha com probabilidades |
> | **Poda** | Não realiza | Aplica poda pós-construção |
> | **Estrutura da Árvore** | Multifurcada | Pode converter em binária |

Questão 7

A diferença essencial entre essas métricas está na forma como avaliam os atributos: o ganho de informação tende a favorecer atributos com um grande número de valores distintos, enquanto a razão de ganho corrige esse viés ao normalizar a métrica, penalizando divisões excessivamente complexas. Dessa forma, a razão de ganho proporciona uma seleção de atributos mais equilibrada e generalizável.