**Questão 01**

**Considerando o exercício da lista anterior sobre a base do TITANIC, gere a árvore de decisão com o hiperparâmetro criterion = Entropy e criterion=Gini Explique como é o funcionamento do critério Gini e mostre os cálculos são obtidos. Compare as árvores obtidas pelos dois critérios.**

O índice Gini mede a probabilidade de um elemento ser classificado incorretamente se for escolhido aleatoriamente de um conjunto de dados.
Ele varia de 0 a 1:

Gini = 0 → Puro (todos os elementos pertencem à mesma classe).

Gini = 1 → Máxima impureza (as classes estão igualmente distribuídas).

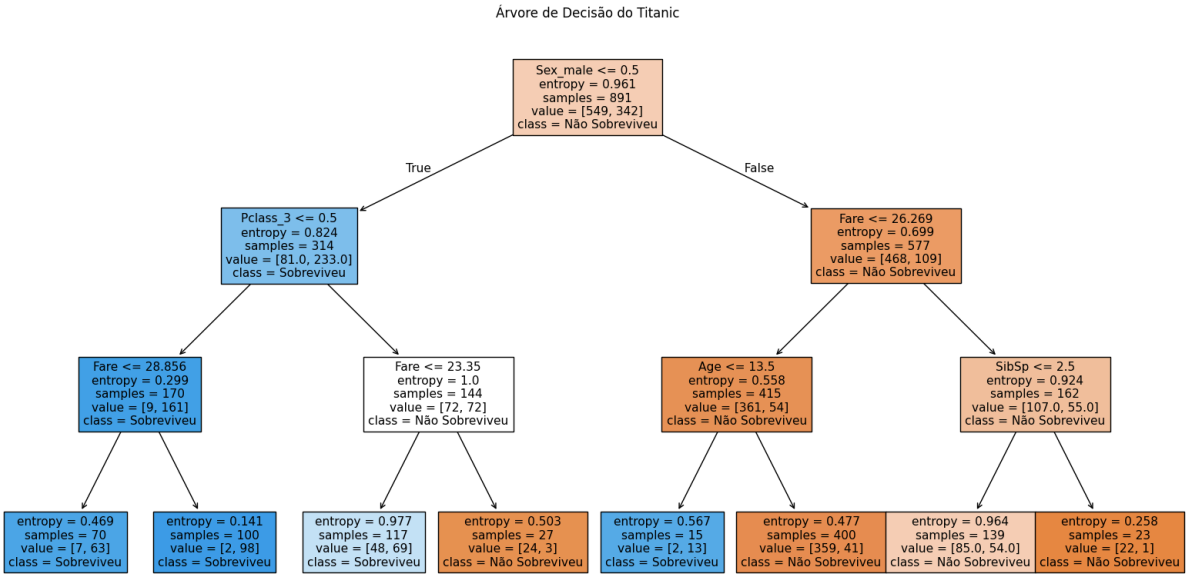
- Estruturas iguais? Falso, há diferenças internas (mesmo que visualmente parecidas).

Entropy

```
=== Relatório com Validação Cruzada (5 folds) ===
                precision    recall  f1-score   s

Não Sobreviveu       0.83      0.89      0.86
    Sobreviveu       0.79      0.70      0.75

      accuracy                           0.82
```



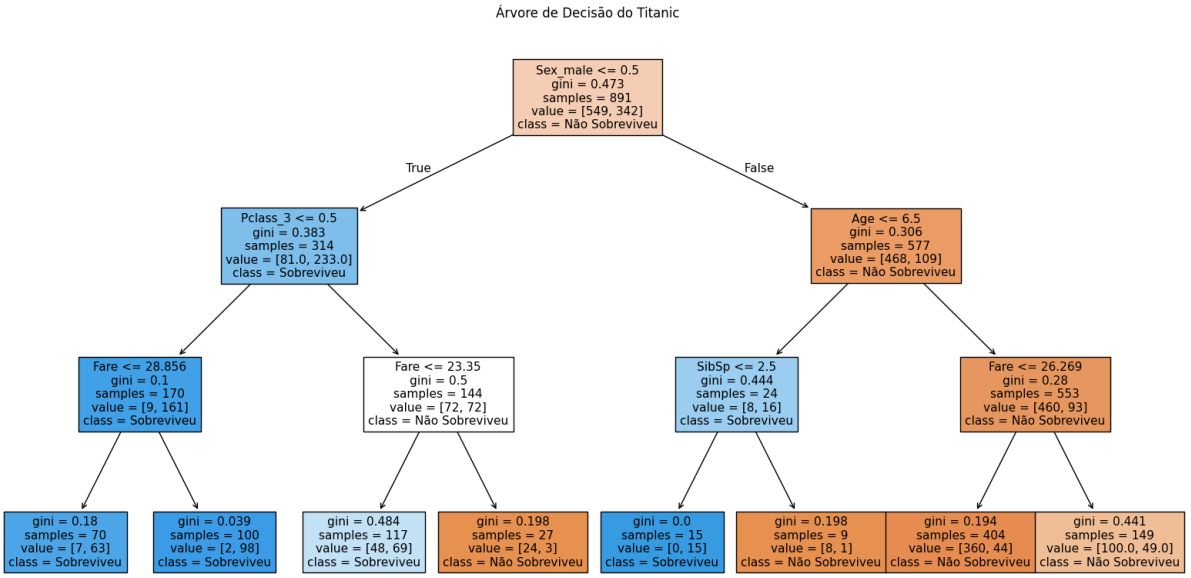

```python
clf = DecisionTreeClassifier(
    random_state=42,
    max_depth=3,  # Define uma altura máxima da árvore
    min_samples_split=15,  # Pelo menos 10 amostras para dividir um nó
    min_samples_leaf=5,  # Pelo menos 5 amostras em cada folha
    criterion="entropy"
)
```

Gini

```
=== Relatório com Validação Cruzada (5 folds) =
                precision    recall  f1-score

Não Sobreviveu       0.82      0.89      0.86
    Sobreviveu       0.80      0.69      0.74

      accuracy                           0.82
```




```python
clf = DecisionTreeClassifier(
    random_state=42,
    max_depth=3,  # Define uma altura máxima da árvore
    min_samples_split=15,  # Pelo menos 10 amostras para dividir um nó
    min_samples_leaf=5,  # Pelo menos 5 amostras em cada folha
    criterion="gini"
)
```

## Questão 02

**Considerando ainda a mesma base de dados do TITANIC, investigue o impacto de outros hiperparâmetros da árvore tais como: max_depth, max_features, min_samples_leaf, dentre outros. Discuta cada hiperparâmetro.**

**Overfitting** (modelo muito complexo que se ajusta demais aos dados de treino)
**Underfitting** (modelo muito simples que não captura os padrões dos dados).

- `max_depth`, Controla a profundidade máxima da árvore.
  Se `None`, a árvore cresce até que todas as folhas sejam puras ou contenham menos amostras que `min_samples_split`
  **Impacto**:
  **Valores pequenos** (ex: `max_depth=3`):
  - Árvore mais simples e interpretável.
  - Menos propensa a overfitting.
  - Pode levar a underfitting se os dados forem complexos.

  **Valores grandes** (ex: `max_depth=20`):
  - Árvore mais complexa.

- Pode capturar padrões específicos dos dados de treino, mas tende a overfitting.

- `min_samples_split`,

**O que é?**

Número mínimo de amostras necessárias para dividir um nó interno.

- Valor padrão: 2.

**Impacto**:

**Valores pequenos** (ex: `min_samples_split=2`):

- Permite que a árvore cresça profundamente, capturando padrões específicos.
- Pode levar a overfitting.

**Valores grandes** (ex: `min_samples_split=50`):

- Limita o crescimento da árvore, tornando-a mais generalista.
- Pode evitar overfitting, mas pode causar underfitting.

- `min_samples_leaf`:

**O que é?**

Número mínimo de amostras necessárias em uma folha (nó final).

Valor padrão: 1.

**Impacto**:

**Valores pequenos** (ex: `min_samples_leaf=1`):

- Permite folhas com poucas amostras, aumentando a complexidade da árvore.
- Pode levar a overfitting.

**Valores grandes** (ex: `min_samples_leaf=20`):

- Folhas mais generalistas, evitando overfitting.
- Pode causar underfitting se os dados forem complexos.

- `max_features`:

**O que é?**

Número máximo de features consideradas para dividir um nó.
Pode ser um valor inteiro (número de features) ou uma fração (ex: `0.5` para 50% das features).

**Impacto**:

**Valores pequenos** (ex: `max_features=3`):
- Reduz a complexidade da árvore.
- Pode evitar overfitting, mas pode perder informações importantes.

**Valores grandes** (ex: `max_features=None`, considera todas as features):
- Aumenta a complexidade da árvore.
- Pode capturar padrões específicos, mas tende a overfitting.

- `criterion`:

**O que é?**

Critério usado para medir a qualidade de uma divisão.
Opções: `gini` (índice de Gini) ou `entropy` (ganho de informação). **Impacto**:

**Gini**:
- Mais rápido computacionalmente.
- Funciona bem na maioria dos casos.

**Entropy**:
- Pode gerar árvores ligeiramente diferentes.
- Mais sensível a pequenas mudanças na distribuição das classes.

- `class_weight`:

**O que é?**

Ponderar as classes para lidar com desbalanceamento.
Opções: `None` (todas as classes têm peso 1) ou `balanced` (pesos inversamente proporcionais às frequências das classes).

**Impacto**:

**`None`**:
- Todas as classes têm o mesmo peso.
- Pode prejudicar a previsão da classe minoritária.

**`balanced`**:
- Aumenta o peso da classe minoritária.
- Melhora a sensibilidade do modelo para classes desbalanceadas.

- **`random_state`**:

**O que é?**

Controla a aleatoriedade do modelo (ex: ordem das features quando há empates).

**Impacto**:

**Valor fixo** (ex: `random_state=42`):
- Garante resultados reproduzíveis.

**`None`**:
- Resultados podem variar entre execuções.

- **`min_impurity_decrease`**

**O que é?**

Limiar mínimo de redução da impureza para considerar uma divisão.
Valor padrão: `0`.

**Impacto**:

**Valores pequenos** (ex: `min_impurity_decrease=0.01`):
- Permite divisões que melhoram pouco a pureza.
- Aumenta a complexidade da árvore.

**Valores grandes** (ex: `min_impurity_decrease=0.1`):
- Apenas divisões que melhoram significativamente a pureza são consideradas.

- Reduz a complexidade da árvore.

**Questão 03 -**

**Investigue o funcionamento dos seguintes otimizadores de hiperparâmetros dos algoritmos de aprendizado de máquina: GridSearchCV (from sklearn.model_selection import GridSearchCV) RandomizedSearchCV (rom sklearn.model_selection import RandomizedSearchCV) BayesSearchCV (from skopt import BayesSearchCV).**
**E investigue, implemente e discuta o funcionamento deles para otimização dos hiperparâmetros para a base do TITANIC.**

- **GridSearchCV** é ideal para espaços pequenos e quando a computação não é um gargalo.
- **RandomizedSearchCV** é uma boa escolha para espaços grandes e quando se quer equilibrar velocidade e performance.
- **BayesSearchCV** é superior em espaços complexos e de alta dimensão, onde a eficiência é crítica.

Para o dataset Titanic, que é relativamente pequeno, **GridSearchCV** ou **BayesSearchCV** são escolhas sólidas. Se o tempo de execução for uma preocupação, **RandomizedSearchCV** oferece um bom equilíbrio. • Melhores parâmetros (BayesSearch): OrderedDict({'criterion': 'gini',
'max_depth': 3, 'min_samples_leaf': 5, 'min_samples_split': 2}) • Acurácia (BayesSearch): 0.8244065793361568

```python
from skopt import BayesSearchCV
from skopt.space import Integer, Categorical

# Definir o espaço de busca
param_bayes = {
    'max_depth': Integer(3, 20),
    'min_samples_split': Integer(2, 20),
    'min_samples_leaf': Integer(1, 10),
    'criterion': Categorical(['gini', 'entropy'])
}

# Criar o modelo
model = DecisionTreeClassifier(random_state=42)

# BayesSearchCV
bayes_search = BayesSearchCV(
    estimator=model,
    search_spaces=param_bayes,
    n_iter=50,  # Número de iterações
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

# Treinar
bayes_search.fit(X_train, y_train)

# Melhores parâmetros
print("Melhores parâmetros (BayesSearch):", bayes_search.best_params_)
print("Acurácia (BayesSearch):", bayes_search.best_score_)
```

• Melhores parâmetros (RandomizedSearch): {'criterion': 'entropy', 'max_depth': 12, 'min_samples_leaf': 5, 'min_samples_split': 9} • Acurácia (RandomizedSearch): 0.8174529695656456

```python
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# Definir distribuições para amostragem
param_dist = {
    'max_depth': randint(3, 20),  # Valores entre 3 e 20
    'min_samples_split': randint(2, 20),
    'min_samples_leaf': randint(1, 10),
    'criterion': ['gini', 'entropy']
}

# Criar o modelo
model = DecisionTreeClassifier(random_state=42)

# RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_dist,
    n_iter=50,  # Número de amostras
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

# Treinar
random_search.fit(X_train, y_train)

# Melhores parâmetros
print("Melhores parâmetros (RandomizedSearch):", random_search.best_params_)
print("Acurácia (RandomizedSearch):", random_search.best_score_)
```

- Melhores parâmetros (GridSearch): {'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 2, 'min_samples_split': 2}
- Acurácia (GridSearch): 0.8230079779375554

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV

# Definir o espaço de hiperparâmetros
param_grid = {
    'max_depth': [3, 5, 7, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy']
}

# Criar o modelo
model = DecisionTreeClassifier(random_state=42)

# GridSearchCV
grid_search = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    cv=5,  # Validação cruzada com 5 folds
    scoring='accuracy',
    n_jobs=-1  # Usar todos os núcleos do processador
)

# Treinar
grid_search.fit(X_train, y_train)

# Melhores parâmetros
print("Melhores parâmetros (GridSearch):", grid_search.best_params_)
print("Acurácia (GridSearch):", grid_search.best_score_)
```

**Questão 04**

1- 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.

2- 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

Alternativa (A)

**Questão 05**

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

Alternativa (A)

## Questão 6

O algoritmo de árvore de decisão C45 possui algumas diferenças em relação ao ID3. Quais são estas diferenças? Explique.

**Tratamento de Atributos Contínuos**

**ID3**: Só trabalha com atributos categóricos. Se houver atributos numéricos, é necessário discretizá-los manualmente antes de usar o ID3.
**C4.5**: Aceita atributos contínuos diretamente. Ele automatiza a discretização, encontrando o melhor ponto de corte (*threshold*) para dividir os dados em intervalos.

**Critério de Seleção de Atributos**

**ID3**: Usa ganho de informação (*information gain*), que tende a favorecer atributos com muitos valores distintos (alta cardinalidade), levando a overfitting.
**C4.5**: Usa ganho de informação normalizado (*gain ratio*), que corrige o viés do ganho de informação ao dividir pelo valor intrínseco (*split information*) do atributo. Isso evita a preferência por atributos com muitas categorias.

**Poda**

**ID3**: Não realiza poda, gerando árvores complexas e propensas a overfitting.

**C4.5:** Aplica poda pós-construção (*post-pruning*), removendo ramos que não contribuem significativamente para a acurácia do modelo. Isso torna a árvore mais generalizável.

**Tratamento de Dados Ausentes**

**ID3**: Não lida com dados ausentes, exigindo pré-processamento (ex: remover

instâncias ou imputar valores).

**C4.5**: Ignora instâncias incompletas ao calcular o ganho de informação. Distribui probabilisticamente as instâncias com valores ausentes entre os nós filhos. Ignora instâncias incompletas ao calcular o ganho de informação. Distribui probabilisticamente as instâncias com valores ausentes entre os nós filhos.

**Eficiência e Escalabilidade**

**ID3**: Menos eficiente em grandes conjuntos de dados ou com atributos contínuos. **C4.5**: Mais otimizado, especialmente para lidar com dados numéricos e evitar splits irrelevantes.

## Questão 7

Para selecionar o melhor atributo na construção da árvore, o algoritmo de árvore de decisão C45 utiliza a Razão de Ganho (Gain Ratio) ao invés do ganho de informação utilizada pelo algoritmo ID3.

Explique as diferenças entre estas 2 medidas.

O **ganho de informação** mede a redução da entropia (desordem) após dividir os dados por um atributo.

**Problema:**

O ganho de informação tende a favorecer atributos com muitos valores distintos (alta cardinalidade), como um atributo "ID" único. Esses atributos criam muitas partições pequenas e puras, mas geram overfitting e árvores pouco generalizáveis.

A **razão de ganho** corrige o viés do ganho de informação ao normalizá-lo pela "informação intrínseca" (*split information*) do atributo.

**Objetivo**:

Penalizar atributos que geram muitas partições (alta cardinalidade), equilibrando a preferência por splits úteis e não apenas complexos

**Conclusão:**

Enquanto o ID3 prioriza atributos que maximizam a pureza imediata (mas geram overfitting), o C4.5 usa a razão de ganho para selecionar atributos que equilibram

pureza e simplicidade, resultando em árvores mais robustas e generalizáveis.