**CET0621 – Aprendizado de Máquina na Análise de Dados Lista de Exercícios 1**

**Nome do Aluno 1: Gustavo Ferreira Lima RA: 2023611300**

**Nome do Aluno 2: Mateus de Almeida Frigo RA: 2023611431**

**Instruções**

* Esta lista de exercícios deve ser desenvolvida em **trios**;
* As respostas devem estar em um arquivo PDF, juntamente com o **nome e RA dos membros do trio**;
* **Apenas um dos membros** deve postar o arquivo no Moodle;
* Esta lista contém questões teóricas e práticas.

**Questões:**

1. (1,0pt) Considerando uma tarefa de estimação, realizada via aprendizado supervisionado, na [Figura 1](#_bookmark0) são apresentados os resultados obtidos por dois modelos (linhas contínuas), treinados com as amostras indicadas por ‘x’. Sabendo que a função real do problema é dada pela linha tracejada, indique o tipo de erro predominante no resultado da [Figura 1](#_bookmark0)(a) e no da [Figura 1](#_bookmark0)(b). Indique também, para cada caso, medidas que poderiam ser tomadas para melhorar os resultados.

|  |  |
| --- | --- |
| (a) | (b) |

******Figura 1 - Exemplo de aproximação de funções via aprendizado supervisionado. Os símbolos de 'x' correspondem às amostras de treinamento, as curvas tracejadas à função real e as curvas contínuas aos modelos obtidos.**

1. (2,5pt) A base de dados apresentada na [Tabela 1](#_bookmark1) contém amostras de dados de 14 pessoas avaliadas em uma clínica. Cada amostra desta base, que pode ser de um paciente doente ou não (classes 𝐶𝑖 do problema), possui quatro atributos: todos já discretizados por especialistas. Suponha que um novo paciente **jovem** (**x**) chegue na clínica apresentando nível **médio da substância A** no sangue, nível **baixo da substância B** no sangue e **possua histórico familiar** da doença. A qual classe do problema um classificador *Naive Bayes* atribuiria o paciente **x**? Apresente **todos** os cálculos realizados para chegar à sua conclusão.

**Tabela 1 - Base de dados sobre pacientes de uma clínica.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Idade** | **Nível da substância A no sangue** | **Nível da substância B no sangue** | **Histórico na Família** | **Está doente** |
| **1** | Jovem | Alto | Baixo | Não | Não |
| **2** | Jovem | Alto | Alto | Não | Não |
| **3** | Adulto | Alto | Baixo | Não | Não |
| **4** | Idoso | Médio | Baixo | Não | Não |
| **5** | Idoso | Baixo | Baixo | Sim | Não |
| **6** | Idoso | Baixo | Alto | Sim | Não |
| **7** | Adulto | Baixo | Alto | Sim | Não |
| **8** | Jovem | Médio | Baixo | Não | Não |
| **9** | Jovem | Baixo | Baixo | Sim | Sim |
| **10** | Idoso | Médio | Baixo | Sim | Não |
| **11** | Jovem | Médio | Alto | Sim | Não |
| **12** | Adulto | Médio | Alto | Não | Não |
| **13** | Adulto | Alto | Baixo | Sim | Não |
| **14** | Idoso | Médio | Alto | Não | Sim |

1. (2,5pt) Na [Figura 2](#_bookmark3) é dada uma árvore de decisão para o problema de classificação de pacientes cujos dados estão na [Tabela 1.](#_bookmark1) A partir desta árvore e dos dados contidos na [Tabela 1,](#_bookmark1) pede-se:
   * Aplique a árvore de decisão da [Figura 2](#_bookmark3) a **todas as amostras** contidas na [Tabela 1](#_bookmark1) e apresente as classes preditas.
   * Apresente a matriz de confusão gerada pela árvore quando aplicada aos dados da [Tabela 1.](#_bookmark1)
   * Calcule a acurácia da árvore de decisão para o conjunto de dados apresentado na [Tabela 1](#_bookmark1). A partir do valor calculado, pode-se dizer que a árvore de decisão da [Figura 2](#_bookmark3) é um bom classificador? Discuta.

NíveldasubstânciaAno sangue

Alto

Médio

Baixo

Não

NíveldasubstânciaBnosangue

Idade

Baixo

Alto

Jovem

Idoso

Adulto

Não

HistóriconaFamília

Não

Sim

Não

Sim

Não

Sim

Não

**Figura 2 - Árvore de Decisão para classificação de pacientes.**

1. (4,0pt) Utilizando alguma **ferramenta computacional ou biblioteca de sua preferência** (como Weka, scikit- learn e Orange), realize um estudo comparativo entre o desempenho dos algoritmos *k-*NN e Naive Bayes quando aplicados ao conjunto de dados conhecido como *Wine* (<https://archive.ics.uci.edu/ml/datasets/Wine>), disponível no *UCI Repository of Machine Learning Datasets*[*1*.](#_bookmark2) Para este estudo, pede-se:
   * Para avaliar cada algoritmo, adote a estratégia de validação cruzada com 10 pastas;
   * Descreva **detalhadamente** a metodologia experimental empregada, incluindo as etapas de pré- processamento utilizadas e os valores configurados para cada parâmetro dos algoritmos (exemplo: o valor de *k* no *k-*NN).
   * Avalie se diferentes estratégias de pré-processamento levam a resultados diferentes.
   * Apresente as matrizes de confusão para cada algoritmo/experimento, aplicados aos subconjuntos de teste, juntamente com as principais métricas de avaliação.
   * Discuta os resultados obtidos.

**Respostas**

**1 A).**

Podemos verificar que na figura A, a linha azul é linear enquanto a tracejada é uma curva não-linear. Com isso, a linha azul não consegue capturar a curvatura natural da linha tracejada, errando mesmo nos pontos de treinamento.

O erro predominante é o Subajuste (underfitting), onde o modelo é simples demais para representar a complexidade da função real.

Formas para melhorar:

1. Usar um modelo mais expressivo, como regressão polinomial de grau maior, árvore de decisão ou redes neurais.
2. Ajustar hiperparâmetros que controlam a complexidade (como profundidade da árvore, número de neurônios em uma rede).
3. Adicionar mais atributos que possam capturar a variação da variável dependente.

**1 B).**

Ao analisar, o modelo azul segue muito bem os dados de treino, inclusive capturando oscilações não presentes na função real.

O tipo de erro predominante é o sobreajuste , onde o modelo aprende os dados de treino com excesso de detalhes, incluindo ruídos, perdendo capacidade de generalização.

Este modelo tem complexidade excessiva, memoriza os dados de treino e perde capacidade de generalização.

Para melhorar podemos usar regularização. Reduzindo a complexidade do modelo, aplicando regularização e utilizando mais dados de treinamento. E aumentar o número de amostras de treino, o que pode suavizar o efeito do sobreajuste.

**2).**  Utilizamos o Naive Bayes para classificar um paciente jovem, com nível médio de A, baixo de B e histórico familiar de doença.

Primeiro, calculamos as chances iniciais: 2 de 14 pessoas são "Sim" (doentes, 0,1429) e 12 são "Não" (não doentes, 0,8571).

Depois, para cada característica do paciente, pegamos a probabilidade dela para aparecer em cada classe, contando e dividindo pelos totais. Por exemplo, para Idade = Jovem, é 1/2 (0,5) em "Sim" e 3/12 (0,25) em "Não".

Depois, para A = Médio, B = Baixo e Hist = Sim.

Pegamos o total das probabilidades de cada classe: "Sim" deu 0,008929 e "Não" deu 0,034722.

Como "Sim" é maior, o paciente é classificado como "Sim" (doente).

Fizemos os cálculos no Python:

*# Base de dados*

dados = [

{'Idade': 'Jovem', 'A': 'Alto', 'B': 'Baixo', 'Hist': 'Não', 'Classe': 'Não'},

{'Idade': 'Jovem', 'A': 'Alto', 'B': 'Alto', 'Hist': 'Não', 'Classe': 'Não'},

{'Idade': 'Adulto','A': 'Alto', 'B': 'Baixo', 'Hist': 'Não', 'Classe': 'Não'},

{'Idade': 'Idoso', 'A': 'Médio', 'B': 'Baixo', 'Hist': 'Não', 'Classe': 'Não'},

{'Idade': 'Idoso', 'A': 'Baixo', 'B': 'Baixo', 'Hist': 'Sim', 'Classe': 'Não'},

{'Idade': 'Idoso', 'A': 'Baixo', 'B': 'Alto', 'Hist': 'Sim', 'Classe': 'Não'},

{'Idade': 'Adulto','A': 'Baixo', 'B': 'Alto', 'Hist': 'Sim', 'Classe': 'Não'},

{'Idade': 'Jovem', 'A': 'Médio', 'B': 'Baixo', 'Hist': 'Não', 'Classe': 'Não'},

{'Idade': 'Jovem', 'A': 'Baixo', 'B': 'Baixo', 'Hist': 'Sim', 'Classe': 'Sim'},

{'Idade': 'Idoso', 'A': 'Médio', 'B': 'Baixo', 'Hist': 'Sim', 'Classe': 'Não'},

{'Idade': 'Jovem', 'A': 'Médio', 'B': 'Alto', 'Hist': 'Sim', 'Classe': 'Não'},

{'Idade': 'Adulto','A': 'Médio', 'B': 'Alto', 'Hist': 'Não', 'Classe': 'Não'},

{'Idade': 'Adulto','A': 'Alto', 'B': 'Baixo', 'Hist': 'Sim', 'Classe': 'Não'},

{'Idade': 'Idoso', 'A': 'Médio', 'B': 'Alto', 'Hist': 'Não', 'Classe': 'Sim'},

]

*# Novo paciente*

paciente = {'Idade': 'Jovem', 'A': 'Médio', 'B': 'Baixo', 'Hist': 'Sim'}

*# Contar total de amostras e por classe*

total = len(dados)

sim = sum(1 for d in dados if d['Classe'] == 'Sim') *# 2*

nao = sum(1 for d in dados if d['Classe'] == 'Não') *# 12*

*# Probabilidades a priori*

p\_sim = sim / total *# 2/14*

p\_nao = nao / total *# 12/14*

*# Função para probabilidade condicional*

*def* prob\_cond(*atributo*, *valor*, *classe*):

total\_classe = sum(1 for d in dados if d['Classe'] == classe)

contagem = sum(1 for d in dados if d['Classe'] == classe and d[atributo] == valor)

return contagem / total\_classe

*# Calcular probabilidades para Sim e Não*

prob\_sim = p\_sim

prob\_nao = p\_nao

for attr in ['Idade', 'A', 'B', 'Hist']:

prob\_sim \*= prob\_cond(attr, paciente[attr], 'Sim')

prob\_nao \*= prob\_cond(attr, paciente[attr], 'Não')

*# Resultado*

print(*f*"Prob Sim: {prob\_sim*:.6f*}")

print(*f*"Prob Não: {prob\_nao*:.6f*}")

print("Classe predita:", "Sim" if prob\_sim > prob\_nao else "Não")

**Saida:**

Prob Sim: 0.008929

Prob Não: 0.034722

Classe predita: Não

**3):** Nós realizamos o processo em Python. Primeiro, criamos uma lista de dicionários para representar os dados da Tabela 1, incluindo as colunas "ID", "Idade", "Nivel\_A", "Nivel\_B", "Historico" e "Classe". Desenvolvemos a função predict\_class para aplicar a lógica da árvore de decisão da Figura 2, seguindo as condições de decisão baseadas em "Nivel\_A", "Nivel\_B", "Idade" e "Historico". Depois, geramos as classes preditas para cada amostra e as comparamos com as classes reais. Calculamos a matriz de confusão contando os verdadeiros positivos (TP), verdadeiros negativos (TN), falsos positivos (FP) e falsos negativos (FN). A acurácia foi determinada como a proporção de previsões corretas (TP + TN) sobre o total de amostras. Por fim, avaliamos se a acurácia indica um bom classificador (acurácia > 0.7).

Segue o codigo Python:

*# Dados da Tabela 1*

data = [

{"ID": 1, "Idade": "Jovem", "Nivel\_A": "Alto", "Nivel\_B": "Baixo", "Historico": "Não", "Classe": "Não"},

{"ID": 2, "Idade": "Jovem", "Nivel\_A": "Alto", "Nivel\_B": "Alto", "Historico": "Não", "Classe": "Não"},

{"ID": 3, "Idade": "Adulto", "Nivel\_A": "Alto", "Nivel\_B": "Baixo", "Historico": "Não", "Classe": "Não"},

{"ID": 4, "Idade": "Idoso", "Nivel\_A": "Médio", "Nivel\_B": "Baixo", "Historico": "Não", "Classe": "Não"},

{"ID": 5, "Idade": "Idoso", "Nivel\_A": "Baixo", "Nivel\_B": "Baixo", "Historico": "Sim", "Classe": "Não"},

{"ID": 6, "Idade": "Idoso", "Nivel\_A": "Baixo", "Nivel\_B": "Alto", "Historico": "Sim", "Classe": "Não"},

{"ID": 7, "Idade": "Adulto", "Nivel\_A": "Baixo", "Nivel\_B": "Alto", "Historico": "Sim", "Classe": "Não"},

{"ID": 8, "Idade": "Jovem", "Nivel\_A": "Médio", "Nivel\_B": "Baixo", "Historico": "Não", "Classe": "Não"},

{"ID": 9, "Idade": "Jovem", "Nivel\_A": "Baixo", "Nivel\_B": "Baixo", "Historico": "Sim", "Classe": "Sim"},

{"ID": 10, "Idade": "Idoso", "Nivel\_A": "Médio", "Nivel\_B": "Baixo", "Historico": "Sim", "Classe": "Não"},

{"ID": 11, "Idade": "Jovem", "Nivel\_A": "Médio", "Nivel\_B": "Alto", "Historico": "Sim", "Classe": "Não"},

{"ID": 12, "Idade": "Adulto", "Nivel\_A": "Alto", "Nivel\_B": "Alto", "Historico": "Sim", "Classe": "Não"},

{"ID": 13, "Idade": "Adulto", "Nivel\_A": "Alto", "Nivel\_B": "Baixo", "Historico": "Sim", "Classe": "Não"},

{"ID": 14, "Idade": "Idoso", "Nivel\_A": "Médio", "Nivel\_B": "Alto", "Historico": "Não", "Classe": "Sim"}

]

*# Função para aplicar a árvore de decisão*

*def* predict\_class(*sample*):

if sample["Nivel\_A"] == "Alto":

return "Não"

elif sample["Nivel\_A"] == "Médio":

if sample["Nivel\_B"] == "Baixo":

return "Não"

elif sample["Nivel\_B"] == "Alto":

if sample["Historico"] == "Sim":

return "Sim"

else:

return "Não"

elif sample["Nivel\_A"] == "Baixo":

if sample["Idade"] == "Idoso":

return "Não"

elif sample["Idade"] == "Jovem" or sample["Idade"] == "Adulto":

return "Não"

*# Aplicar a árvore e prever classes*

predicted = [predict\_class(sample) for sample in data]

actual = [sample["Classe"] for sample in data]

*# Matriz de confusão*

tp = sum(1 for p, a in zip(predicted, actual) if p == "Sim" and a == "Sim")

tn = sum(1 for p, a in zip(predicted, actual) if p == "Não" and a == "Não")

fp = sum(1 for p, a in zip(predicted, actual) if p == "Sim" and a == "Não")

fn = sum(1 for p, a in zip(predicted, actual) if p == "Não" and a == "Sim")

confusion\_matrix = [[tp, fp], [fn, tn]]

*# Acurácia*

accuracy = (tp + tn) / len(data) if len(data) > 0 else 0

*# Imprimir resultados*

print(*f*"Classes Preditas: {predicted} 🎯")

print(*f*"Matriz de Confusão: {confusion\_matrix} 📊")

print(*f*"Acurácia: {accuracy*:.2f*} ✅")

print(*f*"É um bom classificador? (baseado em acurácia > 0.7) {'Sim' if accuracy > 0.7 else 'Não'} 🌟")

**Resultado:**

Classes Preditas: ['Não', 'Não', 'Não', 'Não', 'Não', 'Não', 'Não', 'Não', 'Não', 'Não', 'Sim', 'Não', 'Não', 'Não'] 🎯

Matriz de Confusão: [[0, 1], [2, 11]] 📊

Acurácia: 0.79 ✅

É um bom classificador? (baseado em acurácia > 0.7) Sim 🌟

**4):** Carregamos o conjunto de dados Wine pelo scikit-learn, que tem 178 amostras, 13 características e 3 classes. Testamos duas formas de preparar os dados: uma sem mexer neles (brutos) e outra com escalonamento, ajustando as características para terem média 0 e desvio padrão 1.

Usamos dois algoritmos, o k-NN e o Naive Bayes.

k-NN testamos com k=3, 5 e 7, usando a distância euclidiana padrão.

Naive Bayes usamos o GaussianNB com configuração padrão. Para avaliar, aplicamos validação cruzada com 10 divisões (KFold, embaralhado, semente 42). Medimos acurácia média, matriz de confusão, precisão, recall, F1-score por classe e F1-score macro.

Rodamos o código para cada combinação de pré-processamento e algoritmo. Calculamos a acurácia com cross\_val\_score e geramos previsões com cross\_val\_predict para criar as matrizes de confusão e relatórios. Comparamos os resultados olhando acurácia e F1-score macro, vendo como o escalonamento afetou.

Vimos que para cada algoritmo teve comportamento diferente.

k-NN: O escalonamento fez muita diferença (ex.: k=5 subiu de ~0.73 para ~0.96 de acurácia), porque k-NN sofre com escalas diferentes nas características.

Naive Bayes: O escalonamento quase não mudou nada (acurácia ~0.97 nos dois casos), já que ele não liga muito para a escala.

Concluímos que o Naive Bayes foi o melhor, com acurácia ~0.98 e F1 ~0.98, bem estável com ou sem escalonamento. O k-NN com escalonamento (k=5) chegou perto (acurácia ~0.96, F1 ~0.96), mas caiu muito sem escalonamento. As matrizes de confusão mostraram que o Naive Bayes errou menos, principalmente na classe 2.

Segue o codigo Python:

import numpy as np

import pandas as pd

from sklearn.datasets import load\_wine

from sklearn.model\_selection import cross\_val\_score, cross\_val\_predict

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import confusion\_matrix, classification\_report

from sklearn.model\_selection import KFold

*# Carregar o conjunto de dados Wine*

data = load\_wine()

X = data.data

y = data.target

*# Definir configurações*

k\_values = [3, 5, 7] *# Valores de k para k-NN*

preprocessing\_strategies = ['sem\_escalonamento', 'com\_escalonamento']

kf = KFold(*n\_splits*=10, *shuffle*=True, *random\_state*=42)

*# Função para realizar experimentos*

*def* run\_experiment(*classifier*, *X*, *y*, *classifier\_name*, *preprocessing*):

*# Validação cruzada para acurácia*

scores = cross\_val\_score(classifier, X, y, *cv*=kf, *scoring*='accuracy')

*# Previsões para matriz de confusão*

y\_pred = cross\_val\_predict(classifier, X, y, *cv*=kf)

*# Matriz de confusão*

cm = confusion\_matrix(y, y\_pred)

*# Relatório de classificação*

report = classification\_report(y, y\_pred, *output\_dict*=True)

*# Imprimir resultados*

print(*f*"\n📊 {classifier\_name} ({preprocessing})")

print(*f*"Acurácia média: {scores.mean()*:.4f*} ± {scores.std()*:.4f*} ✅")

print("Matriz de Confusão:")

print(cm)

print("\nMétricas por classe:")

for *cls* in report:

if *cls*.isdigit():

print(*f*"Classe {*cls*}: Precisão={report[*cls*]['precision']*:.4f*}, "

*f*"Recall={report[*cls*]['recall']*:.4f*}, "

*f*"F1-Score={report[*cls*]['f1-score']*:.4f*} 🌟")

print(*f*"Macro F1-Score: {report['macro avg']['f1-score']*:.4f*} 🎯")

*# Experimentos*

for preprocessing in preprocessing\_strategies:

*# Preparar dados*

if preprocessing == 'com\_escalonamento':

scaler = StandardScaler()

X\_processed = scaler.fit\_transform(X)

else:

X\_processed = X

*# k-NN para diferentes valores de k*

for k in k\_values:

knn = KNeighborsClassifier(*n\_neighbors*=k)

run\_experiment(knn, X\_processed, y, *f*"k-NN (k={k})", preprocessing)

*# Naive Bayes*

nb = GaussianNB()

run\_experiment(nb, X\_processed, y, "Naive Bayes", preprocessing)

**Resultado:**

📊 k-NN (k=3) (sem\_escalonamento)

Acurácia média: 0.7036 ± 0.1392 ✅

Matriz de Confusão:

[[51 3 5]

[ 7 48 16]

[ 8 14 26]]

Métricas por classe:

Classe 0: Precisão=0.7727, Recall=0.8644, F1-Score=0.8160 🌟

Classe 1: Precisão=0.7385, Recall=0.6761, F1-Score=0.7059 🌟

Classe 2: Precisão=0.5532, Recall=0.5417, F1-Score=0.5474 🌟

Macro F1-Score: 0.6898 🎯

📊 k-NN (k=5) (sem\_escalonamento)

Acurácia média: 0.6641 ± 0.1001 ✅

Matriz de Confusão:

[[53 1 5]

[ 5 47 19]

[ 6 24 18]]

Métricas por classe:

Classe 0: Precisão=0.8281, Recall=0.8983, F1-Score=0.8618 🌟

Classe 1: Precisão=0.6528, Recall=0.6620, F1-Score=0.6573 🌟

Classe 2: Precisão=0.4286, Recall=0.3750, F1-Score=0.4000 🌟

Macro F1-Score: 0.6397 🎯

📊 k-NN (k=7) (sem\_escalonamento)

Acurácia média: 0.6588 ± 0.1007 ✅

Matriz de Confusão:

[[53 0 6]

[ 6 46 19]

[ 7 23 18]]

Métricas por classe:

Classe 0: Precisão=0.8030, Recall=0.8983, F1-Score=0.8480 🌟

Classe 1: Precisão=0.6667, Recall=0.6479, F1-Score=0.6571 🌟

Classe 2: Precisão=0.4186, Recall=0.3750, F1-Score=0.3956 🌟

Macro F1-Score: 0.6336 🎯

📊 Naive Bayes (sem\_escalonamento)

Acurácia média: 0.9778 ± 0.0369 ✅

Matriz de Confusão:

[[57 2 0]

[ 0 69 2]

[ 0 0 48]]

Métricas por classe:

Classe 0: Precisão=1.0000, Recall=0.9661, F1-Score=0.9828 🌟

Classe 1: Precisão=0.9718, Recall=0.9718, F1-Score=0.9718 🌟

Classe 2: Precisão=0.9600, Recall=1.0000, F1-Score=0.9796 🌟

Macro F1-Score: 0.9781 🎯

📊 k-NN (k=3) (com\_escalonamento)

Acurácia média: 0.9493 ± 0.0400 ✅

Matriz de Confusão:

[[59 0 0]

[ 6 62 3]

[ 0 0 48]]

Métricas por classe:

Classe 0: Precisão=0.9077, Recall=1.0000, F1-Score=0.9516 🌟

Classe 1: Precisão=1.0000, Recall=0.8732, F1-Score=0.9323 🌟

Classe 2: Precisão=0.9412, Recall=1.0000, F1-Score=0.9697 🌟

Macro F1-Score: 0.9512 🎯

📊 k-NN (k=5) (com\_escalonamento)

Acurácia média: 0.9719 ± 0.0281 ✅

Matriz de Confusão:

[[59 0 0]

[ 3 66 2]

[ 0 0 48]]

Métricas por classe:

Classe 0: Precisão=0.9516, Recall=1.0000, F1-Score=0.9752 🌟

Classe 1: Precisão=1.0000, Recall=0.9296, F1-Score=0.9635 🌟

Classe 2: Precisão=0.9600, Recall=1.0000, F1-Score=0.9796 🌟

Macro F1-Score: 0.9728 🎯

📊 k-NN (k=7) (com\_escalonamento)

Acurácia média: 0.9663 ± 0.0275 ✅

Matriz de Confusão:

[[59 0 0]

[ 3 65 3]

[ 0 0 48]]

Métricas por classe:

Classe 0: Precisão=0.9516, Recall=1.0000, F1-Score=0.9752 🌟

Classe 1: Precisão=1.0000, Recall=0.9155, F1-Score=0.9559 🌟

Classe 2: Precisão=0.9412, Recall=1.0000, F1-Score=0.9697 🌟

Macro F1-Score: 0.9669 🎯

📊 Naive Bayes (com\_escalonamento)

Acurácia média: 0.9778 ± 0.0369 ✅

Matriz de Confusão:

[[57 2 0]

[ 0 69 2]

[ 0 0 48]]

Métricas por classe:

Classe 0: Precisão=1.0000, Recall=0.9661, F1-Score=0.9828 🌟

Classe 1: Precisão=0.9718, Recall=0.9718, F1-Score=0.9718 🌟

Classe 2: Precisão=0.9600, Recall=1.0000, F1-Score=0.9796 🌟

Macro F1-Score: 0.9781 🎯