Aula 8: Avaliação Crítica, Métricas e Fechamento do Módulo

Aléssio Miranda Júnior

Agosto - 2026/2

Capítulo em Revisão

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

O que você verá neste capítulo

Neste fechamento do primeiro arco, consolidamos os dois últimos pilares da Engenharia de IA: a **Avaliação Crítica** de modelos e o **Deploy** (implantação em produção). Demonstraremos por que a Acurácia é uma métrica enganosa para dados desbalanceados, formalizaremos a **Matriz de Confusão** e as métricas derivadas (Precision, Recall e F1-Score), e introduziremos os conceitos fundamentais de serialização de modelos e implantação via API REST, preparando o terreno para o Trabalho Prático 1 (TP1).

1 A Ilusão da Acurácia

Neste arco, você já sabe construir, treinar, regularizar e realizar inferências com Redes Neurais. O último pilar fundamental é a **Avaliação Crítica**: a capacidade de diagnosticar se o seu modelo é genuinamente bom ou se está enganando a equipe com números superficiais.

Historicamente, olhamos para a *Acurácia* (taxa de acertos gerais) como a métrica definitiva. Mas imagine que você está construindo uma IA para detectar câncer, e em 100 pacientes, apenas 1 tem a doença. Se a sua IA for completamente inútil e chutar "Saudável" para absolutamente todo mundo, ela acertará 99 pacientes e errará 1. Ela terá uma **Acurácia de 99%**, mas será um **fracasso retumbante** na vida real — o único paciente doente não será tratado.

△ Importante: A Acurácia Mente em Dados Desbalanceados

Quando as classes do problema não são equilibradas (ex: 95% saudáveis, 5% doentes; ou 99% transações legítimas, 1% fraudes), a acurácia é uma **mentira estatística**. Um modelo que sempre prediz a classe majoritária terá acurácia altíssima sem nunca ter aprendido nada. Esse é um dos erros mais perigosos e recorrentes em projetos reais de Machine Learning.

2 A Matriz de Confusão

Para avaliar modelos em ambientes corporativos e acadêmicos com rigor, "quebramos" os acertos e erros em uma tabela chamada **Matriz de Confusão** (*Confusion Matrix*). Ela revela exatamente *como* o modelo está errando:

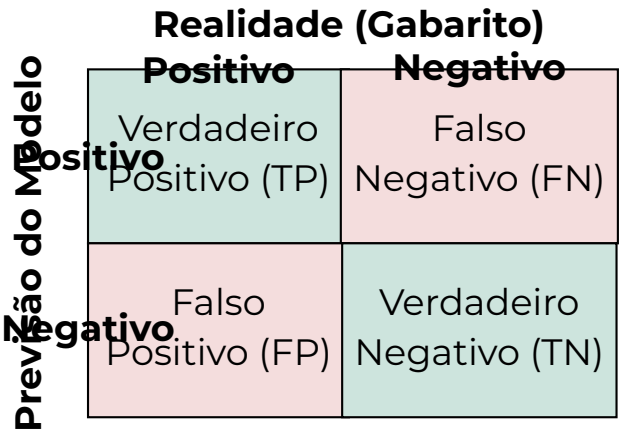


Figura 1: Estrutura de uma Matriz de Confusão Binária. As células verdes representam acertos e as vermelhas representam erros.

- **Verdadeiros Positivos (TP):** Tinha câncer e a IA acertou. **Acerto.**
- **Verdadeiros Negativos (TN):** Estava saudável e a IA acertou. **Acerto.**
- **Falsos Positivos (FP):** Estava saudável, mas a IA disse que tinha câncer. **Alarme Falso.**
- **Falsos Negativos (FN):** Tinha câncer, mas a IA disse que estava saudável. **Falha Fatal.**

2.1 Exemplo Numérico Completo

Considere uma IA de detecção de fraude que processou 1.000 transações bancárias:

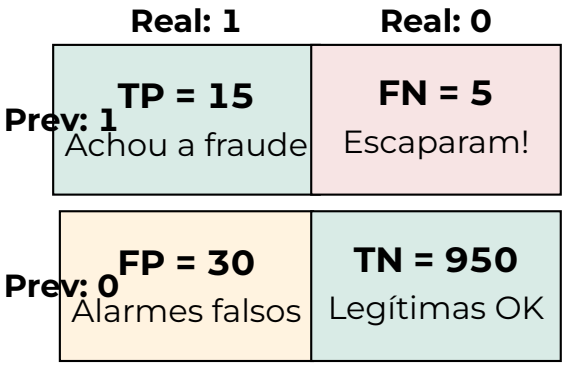


Figura 2: Matriz de Confusão preenchida com dados de detecção de fraude bancária. A Acurácia seria $\frac{15+950}{1000} = 96{,}5\%$, mas das 20 fraudes reais, 5 escaparam (FN=5).

A partir desses valores, calcularemos as três métricas que realmente importam.

3 Precision, Recall e F1-Score

A partir dos quatro quadrantes da Matriz de Confusão, extraímos as três métricas definitivas que substituem a acurácia em qualquer projeto sério:

3.1 Precision (Precisão)

$$\text{Precision} = \frac{TP}{TP + FP} \tag{1}$$

De todas as vezes que a IA **gritou "É Positivo!"**, quantas vezes ela estava certa? A Precision penaliza os **alarmes falsos** (FP). Uma Precision alta significa: "quando a IA diz que é câncer, ela tem razão".

3.2 Recall (Revocação / Sensibilidade)

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

De todos os positivos que **realmente existiam** na base, quantos a IA conseguiu encontrar? O Recall penaliza as **omissões cegas** (FN). Um Recall alto significa: "a IA não deixa escapar doentes".

3.3 F1-Score (Média Harmônica)

$$\text{F1} = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

É a média harmônica entre Precision e Recall. Se o F1 for alto, significa que a IA é **balanceada**: não é nem "medrosa" (que chuta tudo como positivo para não perder nenhum) nem "cautelosa demais" (que só diz positivo quando tem 100% de certeza, deixando muitos escaparem).

A tabela a seguir ilustra por que a média harmônica é superior à média aritmética simples:

Cenário	Precision	Recall	F1-Score	Diagnóstico
Balanceado	0.85	0.85	**0.85**	**Saudável**
Medroso (só acerta quando aposta)	1.00	0.20	**0.33**	**Omisso: deixa esc**
Agressivo (aposta em tudo)	0.10	1.00	**0.18**	**Caótico: alarmes**

Tabela 1: Comparação de cenários: o F1-Score (média harmônica) penaliza severamente desequilíbrios que a média aritmética mascararia.

3.4 O Trade-off Precision vs. Recall

Existe um dilema inerente entre Precision e Recall que pode ser visualizado como uma gangorra:

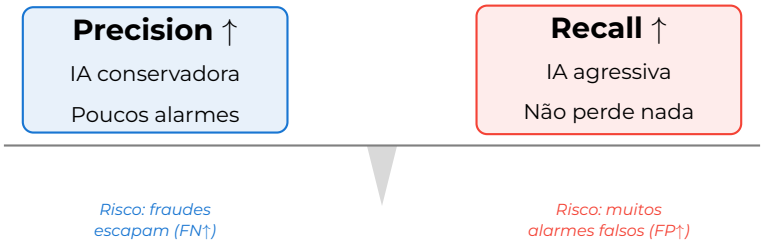


Figura 3: A gangorra Precision–Recall: melhorar uma métrica frequentemente deteriora a outra. O engenheiro escolhe o equilíbrio baseado no custo do erro no domínio.

A escolha de qual métrica priorizar depende inteiramente do **custo do erro** no domínio de aplicação:

Domínio	Prioridade	Justificativa
Diagnóstico Médico	**Recall**	É melhor gerar alarmes falsos (FP) do que deixar um paciente doente sem tratamento (FN).
Filtro de Spam	**Precision**	Um e-mail legítimo classificado como spam (FP) é inaceitável para o usuário.
Detecção de Fraude	**F1**	Equilibrar: não perder fraude nem bloquear clientes injustamente.
Carro Autônomo	**Recall**	Não identificar um pedestre (FN) pode causar um acidente fatal.

Tabela 2: Guia prático: quando priorizar Precision, Recall ou F1-Score, dependendo do custo do erro.

▶ Prática: O Classification Report do Scikit-Learn

O Scikit-Learn consolida todas as métricas em um relatório padronizado:

```
from sklearn.metrics import classification_report

# y_test: rotulos reais   |   y_pred: previsoes do modelo
y_pred = model.predict(X_test)
y_pred_classes = (y_pred > 0.5).astype(int)

print(classification_report(y_test, y_pred_classes))
```

A saída será uma tabela formatada:

```
              precision  recall  f1-score  support
           0       0.92    0.88      0.90      150
           1       0.85    0.90      0.87      100
    accuracy                         0.89      250
   macro avg       0.88    0.89      0.88      250
weighted avg       0.89    0.89      0.89      250
```

4 Serialização e Persistência de Modelos

Treinar uma rede neural pode levar horas ou dias. Não é viável retreinar o modelo toda vez que quisermos fazer uma previsão. O Keras permite **salvar** o modelo treinado (pesos + arquitetura) em um arquivo e **carregá-lo** posteriormente para inferência:

```
# Salvar o modelo completo
model.save('meu_modelo.keras')

# Em outro script, carregar e usar diretamente
from tensorflow.keras.models import load_model
modelo_carregado = load_model('meu_modelo.keras')

# Inferencia imediata
previsao = modelo_carregado.predict(nova_amostra)
```

O fluxo completo de serialização e uso em produção pode ser visualizado como um pipeline:

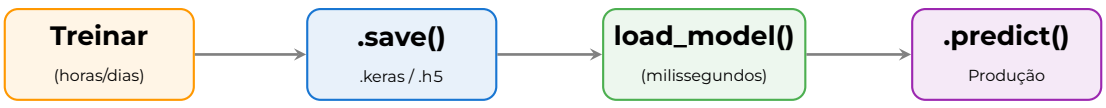


Figura 4: Pipeline de serialização: o treinamento (custoso) acontece uma única vez. O modelo salvo pode ser carregado e usado para inferência instantânea em qualquer ambiente de produção.

◇ **Informação**

O formato `.keras` (antigo `.h5`) armazena tanto a arquitetura quanto os pesos treinados. Isso permite que o modelo seja distribuído como um "executável" de IA — qualquer máquina com Keras instalado pode carregar e usar o modelo sem precisar dos dados de treinamento originais.

5 Deploy: Da Máquina Local para a Produção

O **Deploy** (implantação) é a etapa que separa o protótipo acadêmico do produto real. Um modelo que só funciona no Jupyter Notebook do criador não tem valor de mercado. A indústria exige que a IA seja acessível via rede, consumível por qualquer aplicação cliente.

A abordagem mais comum é encapsular o modelo em uma **API REST**:

```python
from flask import Flask, request, jsonify
from tensorflow.keras.models import load_model
import numpy as np

app = Flask(__name__)
model = load_model('meu_modelo.keras')

@app.route('/predict', methods=['POST'])
def predict():
    dados = request.json['features']
    X = np.array(dados).reshape(1, -1)
    previsao = model.predict(X).tolist()
    return jsonify({'prediction': previsao})

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

Com isso, qualquer sistema — seja um aplicativo mobile, um site web ou outro microserviço — pode enviar dados via HTTP e receber a previsão da IA em formato JSON.

6 O Trabalho Prático 1 (TP1)

A consolidação de todos os conhecimentos deste primeiro arco ocorrerá agora. No seu primeiro Trabalho Prático (TP1), você tem a liberdade de buscar um problema real na plataforma **Kaggle** (ex: *Detecção de fraudes financeiras*, *Previsão de evasão escolar*, *Diagnóstico de doenças cardíacas*).

Você deverá:

1. **Tratar os dados** com `Pandas` (limpeza, análise exploratória, tratamento de valores ausentes).

2 **Normalizar as features** com `StandardScaler` ou `MinMaxScaler`.

3 **Dividir em Treino/Validação/Teste** com `train_test_split`.

4 **Criar a arquitetura** com `Keras` (camadas Dense, Dropout, ativações).

5 **Impedir a memorização** com `Dropout` e `EarlyStopping`.

6 **Avaliar com rigor:** extrair o `classification_report` completo (Precision, Recall e F1).

7 **Serializar e servir:** salvar o modelo e disponibilizar uma API REST básica.

★ Checkpoint do Projeto: Entrega do TP1

O TP1 será entregue via repositório GitLab com CI/CD configurado. A pipeline deverá:

- Executar os testes unitários (PyTest).
- Treinar o modelo e gerar o `classification_report`.
- Salvar o modelo serializado como artefato da pipeline.

A nota será composta por: qualidade do código (30%), rigor da avaliação (40%) e apresentação dos resultados (30%).

7 Conclusão

Você concluiu o “Hello World” da Inteligência Artificial. Com as bases numéricas do Perceptron, o entendimento matemático da otimização via Descida do Gradiente, a poderosa abstração do Keras, as defesas contra o Overfitting, a visão computacional via CNNs e a avaliação crítica com métricas profissionais, você está equipado para transicionar para o próximo grande degrau.

No Módulo 2, pararemos de processar planilhas e números e começaremos a processar a **Linguagem Humana**. Exploraremos como transformar palavras, frases e conceitos abstratos em vetores matemáticos (*Embeddings*) que computadores conseguem processar, abrindo as portas para o universo dos *Large Language Models* (LLMs) e da IA Generativa.

✔ Takeaways

- A **Acurácia** é uma métrica enganosa em dados desbalanceados — um modelo que sempre prediz a classe majoritária pode ter 99% sem ter aprendido nada.
- A **Matriz de Confusão** decompõe os erros em quatro quadrantes: TP, TN, FP e FN.
- **Precision** ($\frac{TP}{TP+FP}$) mede a qualidade dos alarmes; **Recall** ($\frac{TP}{TP+FN}$) mede a cobertura dos positivos reais.
- O **F1-Score** (média harmônica) só é alto quando ambas as métricas são altas — é a métrica definitiva para dados desbalanceados.
- **Serializar** modelos com `model.save()` e `load_model()` é obrigatório para deploy em produção.
- O **classification_report** do Scikit-Learn consolida Precision, Recall e F1 num relatório padronizado.

✎ Questões: Autoavaliação

1 **[Direta]**Uma IA de diagnóstico médico apresenta: TP=80, FP=20, FN=10, TN=890. Calcule Precision, Recall e F1-Score. A IA é segura para uso clínico?

2 **[Direta]**Explique por que a média **harmônica** (F1) é preferida sobre a média aritmética simples para combinar Precision e Recall. Dê um exemplo numérico.

3 **[Direta]**Um modelo de filtro de spam tem Precision=0.99 e Recall=0.60. Interprete esses números em termos práticos para o usuário de e-mail.

4 **[Reflexiva]**Em um sistema de detecção de fraudes, é pior um **Falso Positivo** (bloquear uma transação legítima) ou um **Falso Negativo** (deixar uma fraude escapar)? A resposta muda se o sistema for de diagnóstico de câncer?

5 **[Reflexiva]**Você está entregando um modelo para um cliente. Ele olha a Acurácia de 97% e fica satisfeito. Que perguntas você faria antes de concordar que o modelo é bom?

Lab. 08: Métricas de Avaliação e Serialização

Capítulo em Revisão

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

⋆ Objetivo do Laboratório

Este é o **último laboratório do Módulo 1** — "Hello World das Redes Neurais". Aqui você enfrentará a situação mais traiçoeira do Machine Learning: um dataset **propositalmente desbalanceado** (95% vs 5%) onde a Acurácia **mente**. Você aprenderá a avaliar de verdade com **Precision, Recall e F1-Score**, plotará a **Matriz de Confusão** e **serializará** o modelo para uso em produção. Esta atividade consolida todas as técnicas do módulo e prepara o terreno para o **TP1**.

8 Parte 1: O Problema H — Estilo Maratona

Problema H: O Sentinela de Fraudes do NeoBank

Contexto: O *NeoBank*, um banco digital com 2 milhões de clientes, processa 100.000 transações por dia. Dessas, apenas ≈5% são fraudulentas — mas cada fraude não detectada custa em média R$ 15.000,00 ao banco. A equipe de ciência de dados treinou um modelo que atinge **99% de acurácia** e comemorou. Até que o CTO perguntou: *"das 5.000 fraudes de ontem, quantas vocês realmente pegaram?"* — silêncio total.
Sua missão é construir um classificador para este cenário desbalanceado e demonstrar que a **Acurácia é uma métrica enganosa**. O relatório final deve usar métricas que realmente importam.

Modelo de Entrada (X): Vetor com 15 features de transação:

- $x_1 \dots x_8$: Features informativas (valor, horário, localização, etc.)
- $x_9 \dots x_{12}$: Features redundantes (correlações internas)
- $x_{13} \dots x_{15}$: Features ruidosas (dados espúrios)
- **Desbalanceamento:** 95% legítimas (classe 0) vs 5% fraudes (classe 1)

Modelo de Saída: Três artefatos obrigatórios:

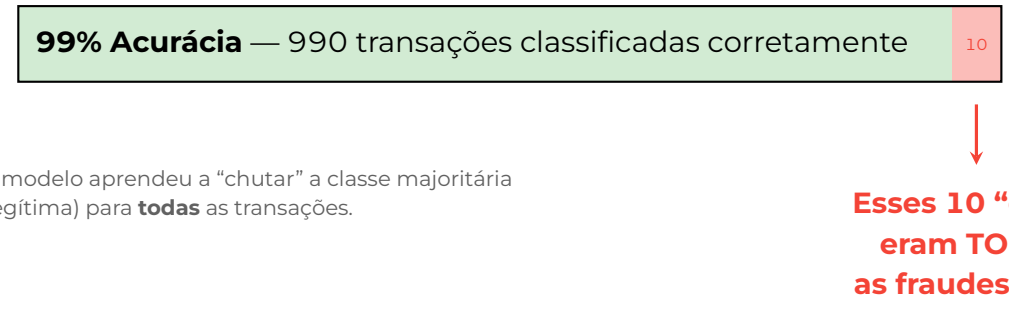
1. Relatório `classification_report` com Precision, Recall e F1 por classe
2. Gráfico `matriz_confusao.png` com mapa de calor
3. Modelo serializado `modelo_fraude.keras`

Critérios de Aprovação:

1. F1-Score da classe Fraude $> 30\%$ (acima de um classificador trivial)
2. Modelo serializado carrega e produz previsões idênticas
3. Previsões binárias válidas (0 ou 1)
4. `classification_report` contém ambas as classes

9 Parte 2: A Armadilha Visual da Acurácia

Antes de codar, entenda visualmente **por que a Acurácia engana**. Imagine 1.000 transações processadas pelo modelo:



Um modelo que **sempre diz “legítima”** atinge 95% de acurácia neste dataset — mas não detecta **nenhuma fraude**. A Acurácia é inútil aqui.

10 Parte 3: Dataset Desbalanceado (8 min)

Crie o arquivo `src/avaliacao_lab.py`:

```python
import numpy as np
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import (classification_report,
    confusion_matrix, ConfusionMatrixDisplay)
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# Dataset DESBALANCEADO: 95% classe 0, 5% classe 1
X, y = make_classification(
    n_samples=2000, n_features=15,
    n_informative=8, n_redundant=4,
    weights=[0.95, 0.05],  # 95% vs 5% !!
    random_state=42, flip_y=0.02
)

print(f"Classe 0 (legítima): {np.sum(y==0)}")
print(f"Classe 1 (fraude):   {np.sum(y==1)}")
print(f"Proporção: {np.sum(y==1)/len(y)*100:.1f}% fraudes")

# Separar com stratify para manter proporção
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

scaler = StandardScaler()
X_train_norm = scaler.fit_transform(X_train)
```

```
X_test_norm = scaler.transform(X_test)
```

• Teoria: Por que stratify=y?

O parâmetro `stratify=y` garante que a proporção 95%/5% seja mantida **tanto no treino quanto no teste**. Sem ele, em dados tão desbalanceados, é possível que o teste fique com zero exemplos da classe minoritária — tornando a avaliação impossível.

11 Parte 4: Modelo com Regularização (8 min)

```
def criar_modelo():
    """Modelo para classificação desbalanceada."""
    model = Sequential()
    model.add(Dense(64, activation='relu', input_shape=(15,)))
    model.add(Dropout(0.3))
    model.add(Dense(32, activation='relu'))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(
        optimizer='adam',
        loss='binary_crossentropy',
        metrics=['accuracy']
    )
    return model

modelo = criar_modelo()

vigilante = EarlyStopping(
    monitor='val_loss', patience=10,
    restore_best_weights=True)

historico = modelo.fit(
    X_train_norm, y_train,
    epochs=200, batch_size=32,
    validation_split=0.2,
    callbacks=[vigilante],
    verbose=0)

print(f"Parou na época {len(historico.epoch)}")
```

12 Parte 5: Desmascarando a Acurácia (8 min)

Agora, demonstre a armadilha e revele as métricas reais:

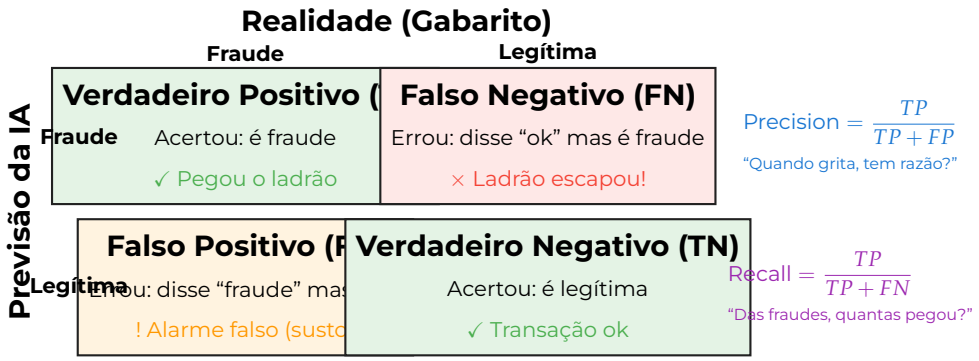
```python
# Avaliação com Acurácia (ENGANOSA)
_, acc = modelo.evaluate(X_test_norm, y_test, verbose=0)
print(f"\nAcurácia geral: {acc:.2%}")
print("Parece ótimo, certo? Mas espere...")

# Avaliação REAL com classification_report
y_pred = modelo.predict(X_test_norm, verbose=0)
y_pred_classes = (y_pred > 0.5).astype(int).flatten()

print("\n" + "="*50)
print("RELATÓRIO DE MÉTRICAS (A VERDADE)")
print("="*50)
report = classification_report(
    y_test, y_pred_classes,
    target_names=['Legítima (0)', 'Fraude (1)'])
print(report)
```

△ Importante: O que Observar no Relatório

A acurácia geral será >95%, mas observe o **Recall da classe 1 (Fraude)**. Se for baixo (<50%), significa que mais da metade das fraudes **escaparam** — exatamente o cenário catastrófico que o CTO temia. O F1-Score é o veredicto final que equilibra Precision e Recall.

A. Anatomia da Matriz de Confusão

O diagrama abaixo explica visualmente os 4 quadrantes que você vai plotar:



13 Parte 6: Matriz de Confusão Visual (8 min)

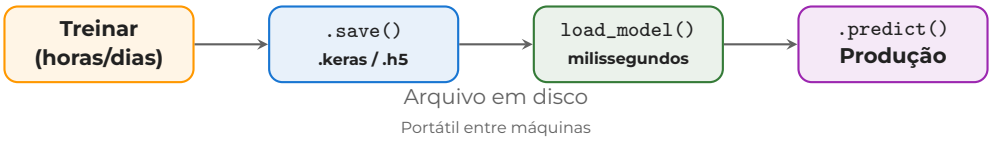
Plote a Matriz de Confusão como um mapa de calor:

```python
def plotar_matriz_confusao(y_test, y_pred_classes):
    """Gera a Matriz de Confusão visual."""
    cm = confusion_matrix(y_test, y_pred_classes)
    disp = ConfusionMatrixDisplay(
        confusion_matrix=cm,
```

```
        display_labels=['Legítima', 'Fraude'])

    fig, ax = plt.subplots(figsize=(6, 5))
    disp.plot(cmap='Blues', ax=ax, colorbar=False)
    ax.set_title('Matriz de Confusão')
    plt.tight_layout()
    plt.savefig('matriz_confusao.png', dpi=150)
    plt.close()
    print("Gráfico salvo: matriz_confusao.png")
    return cm
```

14 Parte 7: Serialização do Modelo (8 min)

O modelo treinado custou tempo e GPU — não queremos retreiná-lo toda vez. A serialização salva o modelo completo (arquitetura + pesos + otimizador) em disco:



Arquivo em disco

Portátil entre máquinas

```
# Salvar o modelo completo
modelo.save('modelo_fraude.keras')
print("Modelo salvo: modelo_fraude.keras")

# Recarregar em memória (simulando outro script)
modelo_carregado = load_model('modelo_fraude.keras')

# Verificar que as previsões são idênticas
preds_original = modelo.predict(X_test_norm[:5], verbose=0)
preds_carregado = modelo_carregado.predict(
    X_test_norm[:5], verbose=0)

assert np.allclose(preds_original, preds_carregado), \
    "Os modelos deveriam produzir previsões idênticas!"
print("Verificação OK: modelo salvo e carregado.")
```

• Teoria: .keras vs .h5 — Qual Formato Usar?

Formato	Vantagens	Uso
`.keras` (novo)	Padrão oficial, suporta tudo	**Recomendado** (TF 2.16+)
`.h5` (legado)	Compatível com código antigo	Projetos legados
`SavedModel/`	Diretório TF Serving	Deploy em produção

15 Parte 8: Bloco Principal e Execução (5 min)

Monte o bloco principal com o relatório completo:

```python
if __name__ == "__main__":
    cm = plotar_matriz_confusao(y_test, y_pred_classes)

    # Extrair os 4 quadrantes
    tn, fp, fn, tp = cm.ravel()
    precision = tp / (tp + fp) if (tp + fp) > 0 else 0
    recall = tp / (tp + fn) if (tp + fn) > 0 else 0
    f1 = (2*precision*recall/(precision+recall)
          if (precision+recall) > 0 else 0)

    print(f"\n{'='*50}")
    print(f"  MÉTRICAS REAIS (Classe Fraude)")
    print(f"  TP={tp}  FP={fp}  FN={fn}  TN={tn}")
    print(f"  Precision: {precision:.2%}")
    print(f"  Recall:    {recall:.2%}")
    print(f"  F1-Score:  {f1:.2%}")
    print(f"{'='*50}")
```

16 Parte 9: Garantia de Qualidade — Testes CI/CD (5 min)

Os testes já estão prontos no arquivo `tests/test_avaliacao.py`:

```python
import numpy as np
import os
from sklearn.metrics import f1_score, classification_report
from tensorflow.keras.models import load_model
from src.avaliacao_lab import (
    modelo, X_test_norm, y_test, y_pred_classes)

def test_f1_minimo():
    """F1-Score da fraude deve ser > 0.30."""
    f1 = f1_score(y_test, y_pred_classes, pos_label=1)
    assert f1 > 0.30, \
        f"F1 da fraude = {f1:.2f}, muito baixo"

def test_modelo_serializado():
    """Modelo deve ter sido salvo corretamente."""
    assert os.path.exists('modelo_fraude.keras'), \
        "Modelo não foi serializado!"
    m = load_model('modelo_fraude.keras')
    preds = m.predict(X_test_norm[:5], verbose=0)
    assert preds.shape == (5, 1)

def test_previsoes_validas():
```

```
    """Previsões devem ser 0 ou 1."""
    assert set(np.unique(y_pred_classes)).issubset({0, 1})

def test_report_existe():
    """O classification_report deve conter ambas as classes."""
    report = classification_report(
        y_test, y_pred_classes, output_dict=True)
    assert '0' in report and '1' in report
    assert 'precision' in report['1']
```

1 Rode os testes no terminal: `pytest tests/test_avaliacao.py -v`.

2 Se a tela exibir **4 passed** em verde, sua implementação está correta.

17 Parte 10: Commit e Push (5 min)

Com todos os testes passando localmente, efetue o envio para o GitLab:

```
git add src/avaliacao_lab.py modelo_fraude.keras matriz_confusao.png
git commit -m "Lab 08: Métricas reais + Serialização (Fechamento Mó
    dulo 1)"
git push origin main
```

O GitLab CI interceptará seu push e executará o `pytest` automaticamente. O **Selo Verde** confirma sua pontuação.

△ Importante: Se o Pipeline Falhar

Os erros mais comuns são: (1) `modelo_fraude.keras` não existir no diretório de trabalho do CI (verifique o path); (2) threshold de F1 > 0.30 não atingido — tente aumentar as épocas ou ajustar o `patience`; (3) importar de `avaliacao_lab` sem o prefixo `src.`.

Conclusão: A Jornada Completa do Módulo 1

Em 8 aulas e 8 labs, você percorreu a jornada completa de um engenheiro de ML:

✔ Takeaways

- Você treinou um classificador em dados **desbalanceados** (95% vs 5%) e demonstrou que a **Acurácia engana** — um modelo trivial atinge 95% sem detectar nenhuma fraude.
- Extraiu **Precision** ("quando grita, tem razão?"), **Recall** ("das fraudes, quantas pegou?") e **F1-Score** (veredicto final) com o `classification_report`.
- Plotou a **Matriz de Confusão** e analisou os 4 quadrantes: TP, TN, FP e FN — entendendo que o **custo do erro** depende do domínio.
- **Serializou** o modelo com `.save()` e comprovou que o modelo carregado produz previsões idênticas — pronto para deploy.
- Completou a **maratona de 8 problemas** (A–H), passando do Perceptron solitário até métricas profissionais de produção.

◇ Informação

Fechamento do Módulo 1 — "Hello World das Redes Neurais".
Em 8 aulas, você percorreu a jornada completa: do Perceptron solitário (Lab A) até a avaliação crítica com métricas profissionais e serialização (Lab H), passando por MLPs, Gradiente Descendente, Keras, Overfitting, Regularização, CNNs e Transfer Learning.

No Módulo 2 — "A Revolução da Linguagem e dos Vetores", abandonaremos números e pixels e entraremos no domínio da **linguagem humana**: como transformar palavras em vetores (*Embeddings*), como funcionam os *Transformers* que revolucionaram a IA, e como chegamos aos *Large Language Models* (LLMs) como GPT e Gemini. A pergunta que guiará o módulo: *"como o computador entende que 'rei' está para 'rainha' assim como 'homem' está para 'mulher'?"*