Lab. 08: Avaliação Crítica e Fechamento do Módulo

Aléssio Miranda Júnior

Agosto - 2026/2

Lab. 08: Métricas de Avaliação e Serialização

⋆ 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**.

1 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:

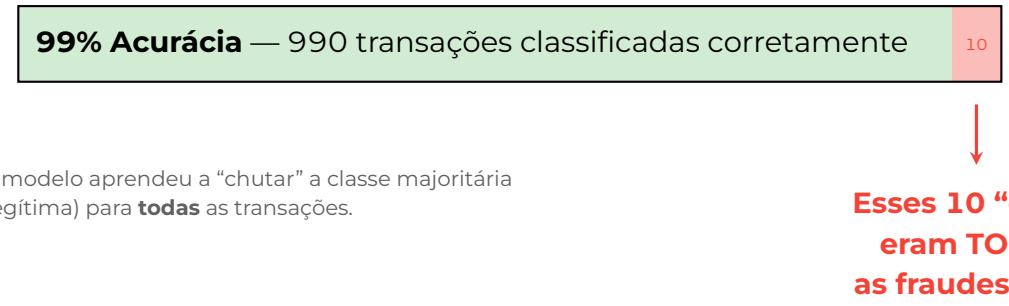
1. Relatório `classification_report` com Precision, Recall e F1 por classe
2. Gráfico `matriz_confusao.png` com mapa de calor
3. Modelo serializado `modelo_fraude.keras`

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

2 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.

3 Parte 3: Dataset Desbalanceado (8 min)

Crie o arquivo `src/avaliacao_lab.py`:

```
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.

4 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)}")
```

5 Parte 5: Desmascarando a Acurácia (8 min)

Agora, demonstre a armadilha e revele as métricas reais:

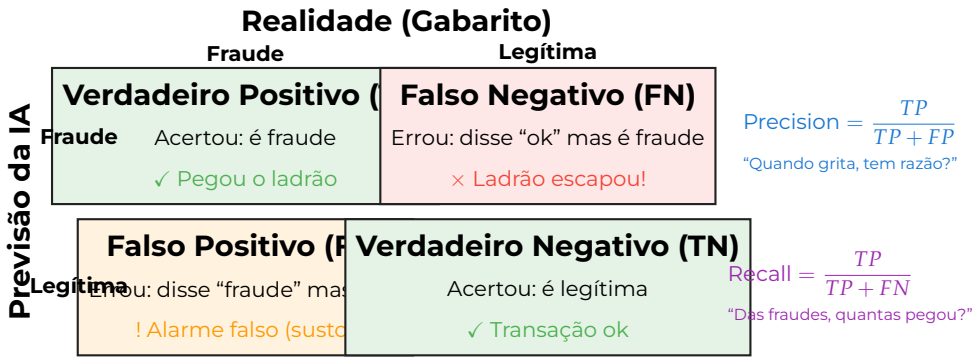
```
# Avaliação com Acurácia (ENGANOSA)
_, acc = modelo.evaluate(X_test_norm, y_test, verbose=0)
print(f"\nAcurácia geral: {acc:.2%}")
print("Parece ótimo, certo? Mas espere...")

# 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:



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

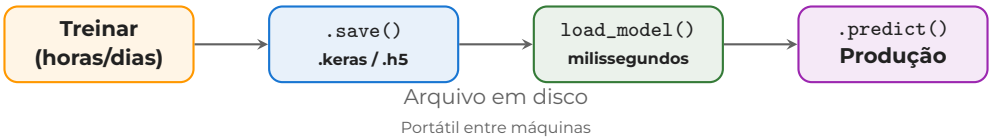
Plote a Matriz de Confusão como um mapa de calor:

```
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
```

7 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

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

Monte o bloco principal com o relatório completo:

```
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}")
```

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

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

```
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.

10 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 Módulo 1
git push origin main
```

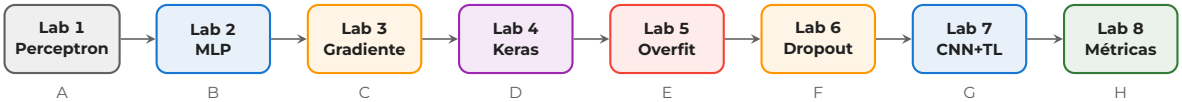
O GitLab CI interceptará seu push e executará o `pytest` automaticamente. O **Selo Verde** confirma sua pontuação.

△ 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'?"*