Lab. 08 (Parte 1): Avaliação Crítica e Serialização do Modelo

Aléssio Miranda Júnior

Agosto - 2026/2

Lab. 08 (Parte 1): Avaliação Crítica e Serialização

⋆ Objetivo da Sessão 1

Nesta primeira sessão do Lab 08, você enfrentará um dos cenários mais traiçoeiros do Machine Learning: um conjunto de dados **propositalmente desbalanceado** (95% de transações legítimas vs. 5% de fraudes), onde a taxa de acurácia global pode ser de 99% e, ainda assim, o modelo falhar nas situações críticas. Você aprenderá a desmascarar a acurácia calculando **Precision, Recall e F1-Score**, plotará a **Matriz de Confusão** visual e **serializará** o modelo treinado em formato `.keras` junto com seu escalonador de dados, homologando a entrega via testes no GitLab CI.

1 Parte 1: O Problema H — Sentinela de Fraudes do NeoBank

Problema H: O Sentinela de Fraudes do NeoBank

Contexto: O *NeoBank*, uma instituição financeira digital fictícia, processa centenas de milhares de transações diárias. Dessas operações, apenas $\approx 5\%$ correspondem a fraudes — contudo, cada fraude consumada gera prejuízo de milhares de reais e compromete a reputação da instituição. A equipe de tecnologia treinou um modelo inicial que atingiu 95% de acurácia global, mas a diretoria questionou: *"Das fraudes reais registradas, quantas o modelo detectou com sucesso?"*
Sua missão na Sessão 1 é construir um classificador para esse cenário desbalanceado, quantificar a falha da acurácia simples e entregar um relatório técnico baseado em métricas adequadas.

Modelo de Entrada (X): Vetor contendo 15 características numéricas contínuas:

- $x_1 \ldots x_8$: Variáveis informativas (valor da transação, horário relativo, score de geolocalização, etc.)
- $x_9 \ldots x_{12}$: Variáveis correlacionadas/redundantes
- $x_{13} \ldots x_{15}$: Ruídos amostrais adicionados propositalmente
- **Distribuição:** 95% classe 0 (legítimas) vs. 5% classe 1 (fraudes)

Artefatos Obrigatórios da Sessão 1:

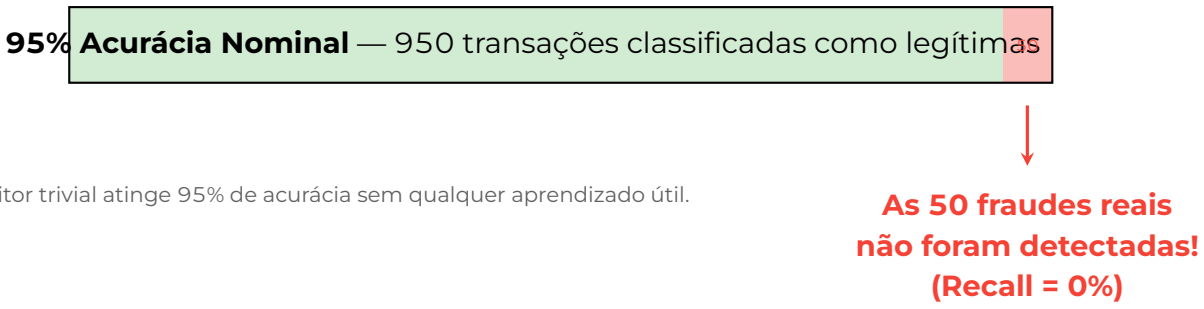
1. Relatório textual com `classification_report` (Precision, Recall e F1 da classe 1)
2. Gráfico em mapa de calor da Matriz de Confusão (`matriz_confusao.png`)
3. Modelo treinado serializado (`modelo_fraude.keras`) e escalonador salvo (`scaler_fraude.joblib`)

Critérios de Aprovação:

1. F1-Score da classe Fraude > 0.30 (superando significativamente o classificador trivial)
2. Modelo serializado em disco recarregável com predições idênticas
3. Execução com 100% de sucesso na suíte de testes `pytest tests/test_avaliacao.py`

2 Parte 2: A Armadilha Visual da Acurácia

Antes de programar, visualize por que a acurácia isolada pode induzir a conclusões errôneas. Considere 1.000 transações avaliadas por um modelo que simplesmente classifica todas as entradas como “legítimas”:



3 Parte 3: Criação do Dataset Desbalanceado (8 min)

Crie o arquivo `src/avaliacao_lab.py` no repositório local:

```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
import joblib
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

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

print(f"Classe 0 (legitimas): {np.sum(y == 0)}")
print(f"Classe 1 (fraudes):   {np.sum(y == 1)}")
```

```
print(f"Proporcao real: {np.sum(y == 1) / len(y) * 100:.1f}% fraudes")

# 2. Divisao estratificada para preservar a distribuicao
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# 3. Normalizacao Z-Score
scaler = StandardScaler()
X_train_norm = scaler.fit_transform(X_train)
X_test_norm = scaler.transform(X_test)

# Salva o scaler para reutilizacao no deploy (Sessao 2)
joblib.dump(scaler, 'scaler_fraude.joblib')
print("Scaler salvo em: scaler_fraude.joblib")
```

• Teoria: A Importância do Parâmetro `stratify=y`

A amostragem estratificada assegura que tanto o subconjunto de treino quanto o de teste contenham exatamente 5% de casos positivos da classe minoritária. Em partições aleatórias convencionais sem estratificação, corre-se o risco de o conjunto de teste não conter instâncias suficientes da classe 1 para mensurar o Recall.

4 Parte 4: Treinamento com Regularização (8 min)

Construa uma arquitetura MLP multicamadas contendo camadas de `Dropout` e monitoramento com `EarlyStopping`:

```
def criar_modelo():
    """Rede neural para classificacao binaria supervisionada."""
    model = Sequential([
        Dense(64, activation='relu', input_shape=(15,)),
        Dropout(0.3),
        Dense(32, activation='relu'),
        Dropout(0.2),
        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"Treinamento finalizado na epoca: {len(historico.epoch)}")
```

5 Parte 5: Avaliação Crítica das Métricas Reais (8 min)

Em vez de verificar apenas a acurácia escalar, extraia o relatório completo de métricas por classe:

```
# Acuracia isolada (pode mascarar o desempenho na classe minoritaria)
_, acc = modelo.evaluate(X_test_norm, y_test, verbose=0)
print(f"\nAcuracia Geral: {acc:.2%}")

# Predicoes probabilisticas convertidas com limiar padrao tau = 0.5
y_pred = modelo.predict(X_test_norm, verbose=0)
y_pred_classes = (y_pred > 0.5).astype(int).flatten()

print("\n" + "="*50)
print("RELATORIO DE METRICAS (CLASSIFICATION REPORT)")
print("="*50)
report = classification_report(
    y_test, y_pred_classes,
    target_names=['Legitima (0)', 'Fraude (1)']
)
print(report)
```

△ Importante: Análise dos Indicadores no Relatório

Observe atentamente a linha correspondente à **Classe 1 (Fraude)**:

- **Precision:** De todas as vezes que o modelo alertou fraude, quantas eram de fato fraudes?
- **Recall:** Do total de fraudes reais presentes no conjunto de teste, qual fração foi capturada pelo modelo?
- **F1-Score:** A média harmônica entre Precision e Recall, oferecendo um balanço mais fidedigno do que a acurácia global.

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

Adicione a rotina de geração da matriz de confusão gráfica:

```
def plotar_matriz_confusao(y_true, y_pred):
    """Gera e salva a Matriz de Confusao em formato PNG."""
    cm = confusion_matrix(y_true, y_pred)
    disp = ConfusionMatrixDisplay(
        confusion_matrix=cm,
        display_labels=['Legitima', 'Fraude']
    )

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

7 Parte 7: Serialização e Persistência do Modelo (8 min)

Para permitir que o modelo seja utilizado posteriormente em um microsserviço sem a necessidade de novo treinamento, persista a arquitetura completa e os pesos aprendidos:

```
# 1. Salvar o modelo em formato padrao Keras
modelo.save('modelo_fraude.keras')
print("Modelo serializado em: modelo_fraude.keras")

# 2. Verificacao de integridade: carregar e comparar saidas
modelo_carregado = load_model('modelo_fraude.keras')
```

```
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), \
    "Erro de integridade: saidas do modelo carregado diferem do original!"
print("Homologacao de serializacao concluida com sucesso.")
```

8 Parte 8: Testes Automatizados e Envio ao GitLab CI (5 min)

Execute a suíte de testes unitários localmente no terminal do projeto:

```
pytest tests/test_avaliacao.py -v
```

Com a confirmação de **4 passed**, envie os artefatos para o repositório remoto:

```
git add src/avaliacao_lab.py modelo_fraude.keras scaler_fraude.joblib matriz_confusao.png
git commit -m "Lab 08 (Parte 1): Metricas reais e serializacao do modelo"
git push origin main
```

✔ Takeaways

- Você comprovou empiricamente que a **Acurácia** pode ser enganosa em distribuições desbalanceadas (95%/5%).
- Avaliou o modelo através de métricas adequadas com o `classification_report`: **Precision**, **Recall** e **F1-Score**.
- Gerou e analisou a **Matriz de Confusão**, identificando Falsos Positivos e Falsos Negativos.
- Serializou o modelo (`.keras`) e o escalonador (`.joblib`), deixando os artefatos prontos para a **Sessão 2: Deploy com FastAPI e Docker**.