ESTUDO DE ESTRATÉGIAS DE PORTABILIDADE DE MODELOS DE IA PARA USO EM HARDWARES DE BAIXO PODER COMPUTACIONAL

ANTHONY ROBSON DA ROSA BARROS[1]; ULISSES GILIOLI FRANCESCATTO JÚNIOR[1]; LEOMAR SOARES DA ROSA JÚNIOR[1]; MARILTON SANCHOTENE DE AGUIAR[1]

[1]*Universidade Federal de Pelotas – {arrbarros, ugfjunior, leomarjr, marilton}@inf.ufpel.edu.br*

1. INTRODUÇÃO

Atualmente, observa-se a necessidade de processar dados diretamente na origem, conceito conhecido como Computação de Borda (*Edge Computing*). A execução local de modelos preditivos em hardwares de baixo poder computacional, como sistemas embarcados, é essencial para garantir respostas em tempo real, reduzir a dependência de conexão constante com a internet, minimizar a latência e aumentar a privacidade, evitando o envio contínuo de informações sensíveis para a nuvem (Shi et al., 2016). A detecção de fraudes em cartão de crédito reúne exatamente essas exigências: exige resposta em tempo real durante a transação e envolve dados financeiros sensíveis, o que torna esse domínio um caso de teste adequado para modelos executados na borda.

Para superar o gargalo computacional em dispositivos com recursos limitados, a literatura apresenta diversas estratégias de otimização. Entre as abordagens mais citadas estão a quantização, que consiste em reduzir a precisão numérica dos dados (por exemplo, converter de ponto flutuante de 32 bits para inteiro de 8 bits) com redução tolerável nas métricas de predição, e a poda (*pruning*), que elimina neurônios e pesos redundantes, reduzindo o consumo de recursos. Outra técnica que se destaca é a destilação de conhecimento (*knowledge distillation*), que consiste em treinar um modelo menor (aluno) para imitar o comportamento de um modelo maior e mais complexo (professor). No caso de modelos baseados em árvores, como o empregado neste trabalho, um conjunto diferente de estratégias mostra-se mais adequado.

Este trabalho tem como objetivo avaliar o impacto de técnicas de seleção de atributos, pré-poda (limitação da profundidade e do número de nós das árvores já na etapa de treinamento), ajuste no limiar de decisão e balanceamento de classes sobre o desempenho preditivo e o consumo de recursos de um modelo Random Forest aplicado à detecção de fraudes em cartão de crédito, executado em hardware embarcado. A principal contribuição reside na comprovação prática de que essas técnicas, combinadas, permitem reduzir de forma substancial o consumo de memória e a latência do modelo sem comprometer seu poder preditivo de forma significativa.

A estrutura do artigo organiza-se da seguinte forma: a Metodologia descreve a obtenção dos dados, os modelos comparados e os métodos de otimização empregados; em seguida, a seção de Resultados e Discussão analisa os dados da Tabela 1, que traz o comparativo das métricas de desempenho (F1-*Score*, latência, alocação de memória e *throughput*).

2. METODOLOGIA

Para treinamento e avaliação do modelo, utilizou-se o dataset público "*Credit Card Fraud Detection*", disponibilizado pelo Machine Learning Group - ULB (2018)

na plataforma Kaggle. Essa base de dados reúne um total de 284.807 instâncias, caracterizadas como transações, das quais apenas 0,172% (492 instâncias) são classificadas como fraude. Bases desse tipo, em que quase todas as instâncias pertencem a uma única classe, são chamadas de desbalanceadas (Kotsiantis, 2006). Devido a isso, não se faz uso da acurácia tradicional (Eq. 1).

$$Acurácia = \frac{(VP+VN)}{(VP+VN+FP+FN)} \quad (1)$$

onde VP, VN, FP e FN correspondem, respectivamente, a Verdadeiro Positivo, Verdadeiro Negativo, Falso Positivo e Falso Negativo. Mesmo que o modelo aprovasse todas as transações, a acurácia obtida seria de 99,8%, ainda que o modelo falhasse completamente em seu propósito. Por isso, a avaliação nesse cenário exige métricas diferentes, o que justifica a adoção do F1-*Score* (Eq. 2).

$$F_1 = \frac{2VP}{2VP+FN+FP} \quad (2)$$

Definidas as métricas de avaliação, o passo seguinte foi preparar os dados para o treinamento. O *dataset* foi dividido em dois conjuntos, o primeiro com 80% para treinamento e o segundo com 20% para testes de inferência. Como parâmetro de divisão dos dados, utilizou-se a estratificação, que preserva a proporção das classes nos dois conjuntos e permite testar a capacidade de generalização do modelo para dados não vistos no treino.

Com o objetivo de aumentar a eficiência computacional, aplicou-se a técnica de seleção de atributos (*feature selection*). A estrutura do *dataset* foi reduzida de 30 para 10 colunas, o que representa uma redução de dois terços das variáveis originais. Para essa redução, estabeleceu-se um ordenamento decrescente das features utilizando o ganho de informação como critério. A métrica F1-*Score* foi utilizada para analisar o ponto máximo de redução dos atributos, de modo a manter a degradação do desempenho preditivo em nível tolerável, compensada pelos ganhos em desempenho computacional.

Como essa base de dados se trata de um caso de classificação, optou-se pelo algoritmo Random Forest (Breiman, 2001), devido à sua versatilidade e alta precisão. Duas configurações do modelo foram utilizadas para análise de desempenho. O Modelo Baseline foi treinado com as configurações padrão da biblioteca Scikit-Learn (Pedregosa et al., 2011), com o intuito de avaliar como o modelo se comporta, em seu estado padrão, em hardware limitado. Já o Modelo Otimizado recebeu ajustes em seus hiperparâmetros, definidos por meio de ajuste manual: redução para 10 atributos, definição de 60 estimadores, limitação da profundidade máxima em 20, balanceamento de classes e ajuste no limiar de decisão. Os dois últimos ajustes foram aplicados com o intuito de preservar o desempenho preditivo e mitigar os efeitos do extremo desbalanceamento da base.

Para validar o desempenho do modelo em condições reais de uso, os testes finais foram executados em uma Raspberry Pi 2 Model B (*revision* 1.1), com o sistema operacional Raspberry Pi OS Lite, executado em modo nativo (*headless*), sem interface gráfica (GUI), o que minimiza o consumo de recursos computacionais (Patterson; Hennessy, 2014). Considerou-se também sua compatibilidade nativa com a linguagem Python (Raspberry Pi Foundation, 2026), o que facilita a portabilidade dos *scripts* para o dispositivo. A placa é equipada com o SoC Broadcom BCM2836, composto por um processador quad-core ARM Cortex-A7 a 900 MHz (arq. ARMv7-A de 32 bits) e 1 GB de memória RAM LPDDR2 compartilhada. Os resultados obtidos nesse ambiente são apresentados e discutidos na seção seguinte.

3. RESULTADOS E DISCUSSÃO

Foram executados experimentos de *benchmark* na Raspberry em *thread* única. A Tabela 1 evidencia a comparação entre o Modelo Baseline (30 atributos e limiar de 0,5) e o Modelo Otimizado (10 atributos e limiar ajustado para 0,39), apresentando métricas de desempenho preditivo e consumo de recursos. No geral, os dados da Tabela 1 evidenciam ganhos expressivos em eficiência computacional, obtidos a um custo limitado sobre o desempenho preditivo do modelo.

Tabela 1 - Comparativo de Desempenho na *Raspberry Pi 2*

Categoria	Métrica	Modelo Baseline (30 atributos)	Modelo Otimizado (10 atributos)	Variação / Ganho
Preditiva	Precisão	0,9412	0,8515	-9,53%
	Recall	0,8163	0,8776	+7,51%
	F1-Score	0,8743	0,8643	-1,14%
Footprint	RAM Carga (Modelo e *X_test*)	28,91 MB	8,01 MB	-72,29%
Execução (2000 amostras)	Latência Média (Unitária)	158,701 ms	99,113 ms	-37,55%
	Latência P95 (Unitária)	159,824 ms	100,020 ms	-37,42%
Execução (Lote)	Pico de RAM	8,49 MB	4,14 MB	-51,24%
	Throughput	6062,2 pred/s	16833,0 pred/s	+177,67%

Em relação às métricas preditivas, a Tabela 1 mostra que a redução de atributos da base de dados original para um terço (de 30 para 10) em conjunto com o reajuste do limiar de decisão (T = 0,39) resultou em variação tolerável na métrica F1-*Score*, com degradação de -1,14% (de 0,8743 para 0,8643). Esse resultado agregado esconde, porém, um *trade-off* estratégico entre as duas métricas que compõem o F1-*Score*: a Precisão recua -9,53% (de 0,9412 para 0,8515), enquanto o *Recall* apresenta elevação de +7,51% (de 0,8163 para 0,8776). Dada a natureza do problema de detecção de fraudes, prioriza-se o *Recall*, visando reduzir os falsos negativos (fraudes não detectadas), o que indica que os 20 atributos removidos apresentavam baixo ganho de informação relevante ao modelo.

O segundo conjunto de métricas relevantes diz respeito ao consumo de memória, restrição especialmente crítica na Raspberry Pi 2, que dispõe de apenas 1 GB de RAM. Nesse quesito, a Tabela 1 mostra que o modelo otimizado evidencia possibilidade de redução na alocação de memória em tempo de execução, registrando queda de -72,29% (de 28,91 MB para 8,01 MB) no consumo de memória RAM durante a etapa de carregamento e inicialização do modelo e da matriz de atributos do conjunto de teste (X_test). Somado a isso, o pico de consumo de memória durante o processamento dos dados em lote registrou queda de -51,24% (de 8,49 MB para 4,14 MB), mitigando de forma substancial possíveis problemas de esgotamento de memória.

Por fim, os ganhos de eficiência computacional se refletem também na velocidade de resposta do modelo. Os dados de latência unitária, na subamostragem de 2.000 amostras, evidenciaram ganho significativo de velocidade, com a média recuando -37,55% (de 158,701 ms para 99,113 ms) e o percentil P95, -37,42% (de 159,824 ms para 100,020 ms). Essa redução se tornou mais evidente no processamento dos dados em lote, gerando ganho de +177,67% no *throughput* (de 6.062,2 para 16.833,0 predições/s). A redução de dois terços das variáveis de entrada provocou queda na profundidade de cada árvore do modelo e, consequentemente, no número de nós, reduzindo os ciclos de processamento. Em conjunto, os resultados de memória e latência indicam que o Modelo Otimizado é substancialmente mais adequado ao hardware embarcado

utilizado, com uma perda de desempenho preditivo pequena frente aos ganhos obtidos em eficiência computacional.

4. CONCLUSÕES

Este trabalho teve como contribuição principal a comprovação prática de que técnicas de seleção de atributos, pré-poda, ajuste no limiar de decisão e balanceamento de classes permitem tornar um modelo Random Forest viável para execução em hardware de baixo poder computacional. A aplicação dessas técnicas em uma Raspberry Pi 2 resultou em reduções expressivas de consumo de recursos, com destaque para a queda de -72,29% na alocação de memória durante o carregamento do modelo e o ganho de +177,67% no *throughput* de predições em lote, ao custo de uma degradação de apenas -1,14% na métrica F1-Score.

Esses resultados refletem uma escolha deliberada de priorizar o *Recall* em detrimento da Precisão, adequada ao problema de detecção de fraudes, em que reduzir falsos negativos é mais relevante do que reduzir falsos positivos. O *trade-off* observado confirma que é possível obter ganhos substanciais em eficiência computacional sem comprometer de forma significativa o poder preditivo do modelo, desde que a otimização considere as particularidades do domínio de aplicação.

Como perspectiva de continuidade direta da linha de pesquisa, os resultados obtidos abrem caminho para a aplicação e o teste do modelo otimizado em sistemas embarcados reais, validando sua eficácia em cenários de operação. Como direções de expansão, sugere-se a aplicação de outros métodos de otimização, como quantização e poda, além de testes do mesmo conjunto de técnicas em diferentes datasets, avaliando sua generalização a outros domínios de detecção de anomalias.

5. REFERÊNCIAS BIBLIOGRÁFICAS

BREIMAN, L. Random Forests. **Machine Learning**, Holanda, v. 45, n. 1, p. 5–32, out. 2001. Disponível em: http://dx.doi.org/10.1023/A:1010933404324.

KOTSIANTIS, Sotiris et al. **Handling imbalanced datasets: A review**. GESTS international transactions on computer science and engineering, v. 30, n. 1, p. 25-36, 2006.

MACHINE LEARNING GROUP - ULB. **Credit Card Fraud Detection**. Kaggle, San Francisco, 2018. Datasets. Acessado em 8 set. 2026. Online. Disponível em: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

PATTERSON, David A.; HENNESSY, John L. **Organização e Projeto de Computadores**: a interface hardware/software. 5. ed. Rio de Janeiro: Elsevier, 2014.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. **Journal of Machine Learning Research**, USA, v.12, p.2825-2830, 2011.

RASPBERRY PI FOUNDATION. **Raspberry Pi Documentation**. Cambridge: Raspberry Pi Foundation, 2026. Online. Acessado em 8 set. 2026. Disponível em: https://www.raspberrypi.com/documentation/

SHI, Weisong; CAO, Jie; ZHANG, Quan; LI, Youhuay; XU, Liyu. **Edge Computing: Vision and Challenges**. IEEE Internet of Things Journal, Piscataway, v. 3, n. 5, p. 637-646, 2016.