Classificação de Objetos Astronômicos com Algoritmo *Random Forest*: um comparativo entre implementações *single* e *multi-threaded*

Pedro L. M. Garcia[1]

[1]ICET – Universidade Federal de Lavras (UFLA)

pedro.garcia2@estudante.ufla.br

Resumo. *Este trabalho apresenta uma análise comparativa do desempenho do algoritmo Random Forest na classificação de objetos astronômicos, utilizando a base Sloan Digital Sky Survey - DR18 e diferentes abordagens de implementação: Scikit-learn em Python, C++ single-threaded e C++ multi-threaded com OpenMP. Os resultados revelam que, embora o Scikit-learn apresente maior acurácia (99%) e f1-score (0.98), as versões em C++ se destacam pela velocidade de execução, especialmente a multi-threaded (34 segundos). No entanto, a queda na precisão e acurácia nas versões C++ sugere limitações na implementação, e não na paralelização em si. As métricas de paralelização, como Speedup (3.04) e Eficiência (0.38), indicam ganhos relevantes, mas também apontam oportunidade de otimização. Conclui-se que a exploração de paralelismo em algoritmos de classificação é promissora, desde que aliada a melhorias estruturais nas implementações.*

1. Introdução

Com o advento da tecnologia, a humanidade vêm produzindo dados e informações em um ritmo vertiginoso. Em 2020, o volume estimado de dados gerados foi de 64.2 *zettabytes (*ZB), em comparação com 2 ZB gerados no ano de 2010[1]. Esse aumento de volume pode ser observado em diversas áreas do conhecimento humano, incluindo a Astronomia. Dessa forma, a busca por meios ecoeficientes de processamento de dados torna-se imprescindível para que a ciência consiga acompanhar esse ritmo. Abordagens que utilizam aprendizado de máquina demonstram enorme potencial como ferramenta de apoio, destacando-se em tarefas como classificação de grande volumes de dados e reconhecimento de padrões.

Este trabalho busca experimentar a utilização do algoritmo *Random Forest* para classificação de objetos astronômicos[2] utilizando a base de dados de espectroscopia da *Sloan Digital Sky Survey,* bem como propor uma abordagem *multi-threaded* e comparar a eficácia e eficiência de ambas.

2. Referencial Teórico

Nesta seção serão apresentados os conceitos abordados neste trabalho, como o algoritmo de *Random Forest*, e as métricas utilizadas para a análise das implementações.

2.1. *Random Forest Algorithm*

O algoritmo *Random Forest* é uma técnica de aprendizado supervisionado baseada nos princípios de aprendizagem por conjunto (*ensemble learning*). Seu funcionamento se

baseia na construção de uma multiplicidade de árvores de decisão durante o treinamento, e combina as previsões de cada uma para aumentar a precisão e reduzir o risco de *overfitting*.

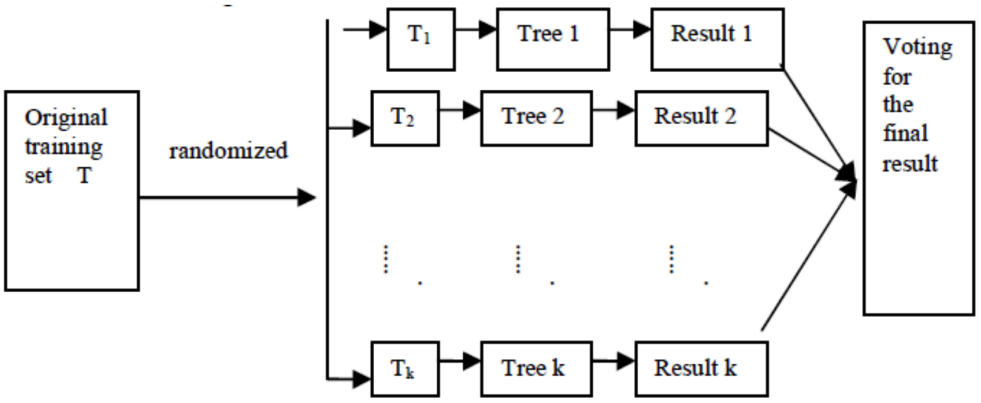

Figura 1. Esquemática do algoritmo de Random Forest[3]

2.2. *Accuracy, precision, recall* e *f1-score*

No contexto de algoritmos de aprendizagem de máquina, *Accuracy, precision, recall* e *f1-score* são métricas fundamentais para avaliar o quão bem os modelos estão se comportando. Abaixo serão enumeradas as métricas e as suas respectivas fórmulas.

2.2.1 *Accuracy*

Indica a proporção de previsões corretas em relação ao total de previsões.

$$Accuracy = \frac{VP + VN}{VP + VN + FP + FN}$$

Onde:

- VP = Verdadeiros Positivos
- VN = Verdadeiros Negativos
- FP = Falsos Positivos
- FN = Falsos Negativos

2.2.2 *Precision*

Indica quantas das previsões positivas feitas pelo modelo estavam corretas.

$$Precision = \frac{VP}{VP + FP}$$

2.2.3 *Recall*

Calcula a capacidade do modelo de identificar corretamente os casos positivos reais.

$$Recall = \frac{VP}{VP + FN}$$

2.2.4 *F1-Score*

Média harmônica entre *Precision* e *Recall*. Equilibra os dois quando ambos os valores são discrepantes.

$$f1\text{–}score = \frac{precision \times recall}{precision + recall}$$

2.3. *Speedup,* eficiência, fração paralelizável e métrica de *Karp-Flat*

Já no contexto de algoritmos paralelizados, algumas métricas nos auxiliam a avaliar o quão bem o algoritmo executa em comparação com um algoritmo sequencial. A seguir, serão descritas as métricas utilizadas neste trabalho.

2.3.1 *Speedup*

Avalia o quão mais rápido um algoritmo executa em paralelo em comparação ao sequencial.

$$Speedup = \frac{Ts}{Tp}$$

Onde:

- T_s = tempo de execução do algoritmo sequencial
- T_p = tempo de execução com múltiplas threads

2.2.2 Eficiência

Indica o aproveitamento das threads disponíveis.

$$Eficiência = \frac{Speedup}{N}$$

Onde N refere ao número de *threads* (unidades de processamento) utilizadas.

2.2.3 Fração Paralelizável

Indica a proporção do código que pode ser executada em paralelo.

$$fp = \frac{1}{(1-f) + \frac{f}{N}}$$

Onde:

- f = fração do código paralelizável
- N = número de threads

2.2.4 Métrica de *Karp-Flatt*

Usada para estimar o impacto de fatores de overhead e serialidade no speedup observado.

$$e = \frac{\frac{1}{S} - \frac{1}{N}}{1 - \frac{1}{N}}$$

Onde:

- S = speedup observado
- N = número de threads

3. Metodologia

Nesta seção, serão descritos os projetos utilizados como embasamento, a base de dados utilizada, os algoritmos implementados, bem como o hardware utilizado para os testes.

3.1. Referencial de projetos

Para a execução deste trabalho, foram utilizados para fins de embasamento projetos *open-source* que implementam classificação de objetos astronômicos, e algoritmos de categorização com abordagem de *Random Forest*.

O principal projeto referenciado foi o *stellar-classification-ML*[4], que provê uma base de dados de exemplo contendo dados de estrelas, como temperatura, luminosidade e magnitude, e implementa 4 algoritmos de classificação para fins de comparação: *Random Forest, K-means, Multi-layer Perceptron* e Redes Neurais Convolucionais.

Outros projetos *open-source* foram utilizados como base para a implementação do algoritmo de *Random Forest*. São eles: *random-forest-classifier*[5], *Random_Forests*[6] e *Weighted Random Forest*[7].

Também foi fundamental a documentação da biblioteca Boost.Python[8], que possibilita a comunicação entre algoritmos em *Python* e *C++*.

3.2. Descrição da base de dados

Para treinar e testar o modelo, foi utilizada a base de dados Sloan Digital Sky Survey - Data Release 18[9]. A base possui a representação de 10.000 objetos astronômicos, descritos em um arquivo *csv com* 43 *features*, dentre elas a classificação em três classes de objetos: STAR (estrela), GALAXY (galáxia) e QSO (Quasi-Stellar Object, ou Quasar).

80% dos dados foram utilizados para treinamento, enquanto os 20% restantes foram utilizados para testes.

3.3. Implementação

O projeto implementado conta com um algoritmo em Python3, responsável por pelo pré-processamento da entrada de dados. Além disso, o projeto utiliza a implementação do algoritmo de *Weighted Radom Forest* fornecido pela biblioteca Scikit-learn[12].

O projeto também conta com duas implementações em C++ do *Weighted Radom Forest:* uma implementação sequencial, e outra paralelizada por meio da biblioteca OpenMP[10]. Ambas as implementações são expostas ao projeto em Python por meio da biblioteca Boost.Python.

O projeto está disponível publicamente no GitHub[11], sob licença Creative-Commons.

3.4. Ambiente de Desenvolvimento e Teste

Para a execução do projeto, foi utilizado o Python 3.12 em um Single-Board Computer (SBC) com SoC Rockchip RK3588, com 8 núcleos de processamento (4 núcleos com velocidade de 1.8GHz, e 4 núcleos de 2.4Ghz), e 16GB de memória RAM DDR4.

Os algoritmos foram executados 10 vezes para a obtenção da média das métricas supracitadas.

4. Resultados

Abaixo serão apresentados os resultados após 10 execuções consecutivas das 3 implementação.

4.1. Accuracy, precision, recall, f1-score e tempo de execução

No que tange às métricas relacionadas ao desempenho do algoritmo de classificação, a implementação de referência do Scikit-learn levou vantagem, com *accuracy = 0.9907, precision = 0.99, recall = 0.98 e f1_score = 0.98.*

Já a implementação em C++ *single-threaded* obteve as pontuações de *accuracy = 0.84, precision = 0.40, recall = 0.98 e f1_score = 0.57.*

Por fim, a implementação em C++ *multi-threaded* obteve os resultados de *accuracy = 0.76, precision = 0.31, recall = 0.98 e f1_score = 0.47.*

A seguir, estará apresentado o gráfico comparativo entre as implementações, agrupado pelas métricas.

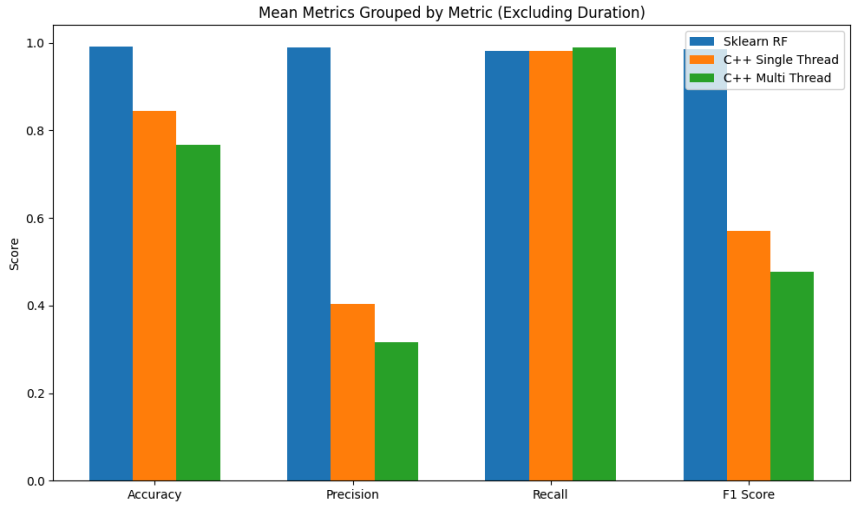

Figura 2. Comparativo de métricas de *accuracy, precision, recall e F1 Score*

Já quanto ao tempo de execução, como esperado a implementação em C++ *multi-threaded* teve vantagem, com execução média de 34 segundos, contra 103 segundos para a implementação em C++ *single-threaded*, e 470 segundos para a implementação do *Scikit-learn.*

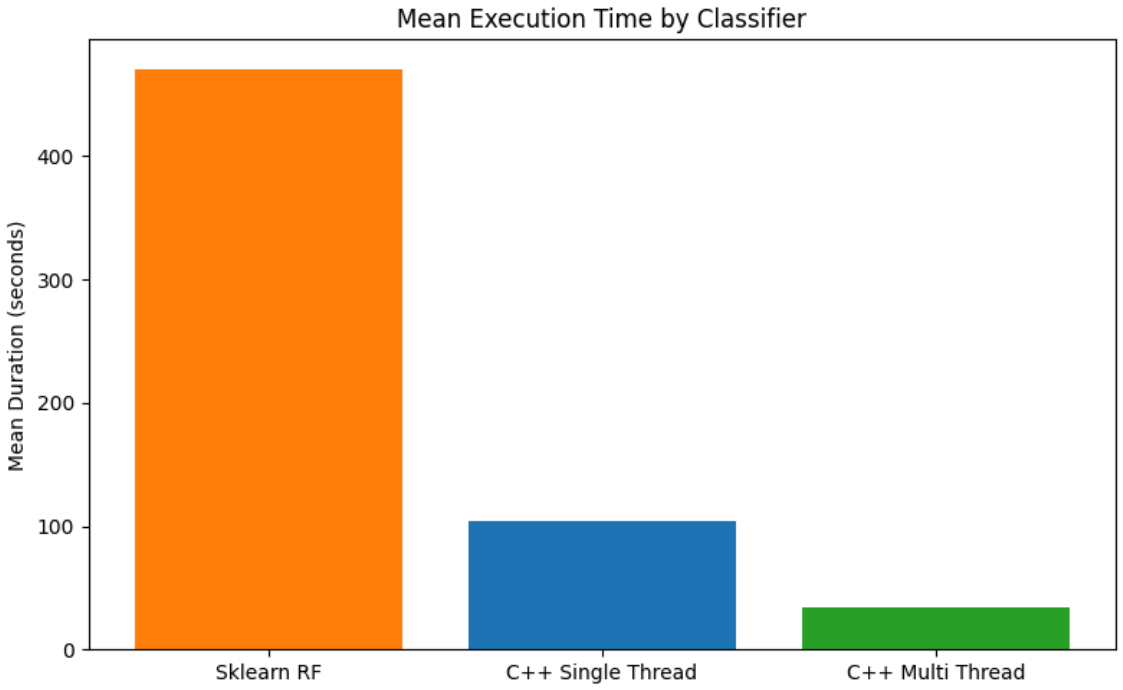

Figura 3. Comparativo de tempo de execução

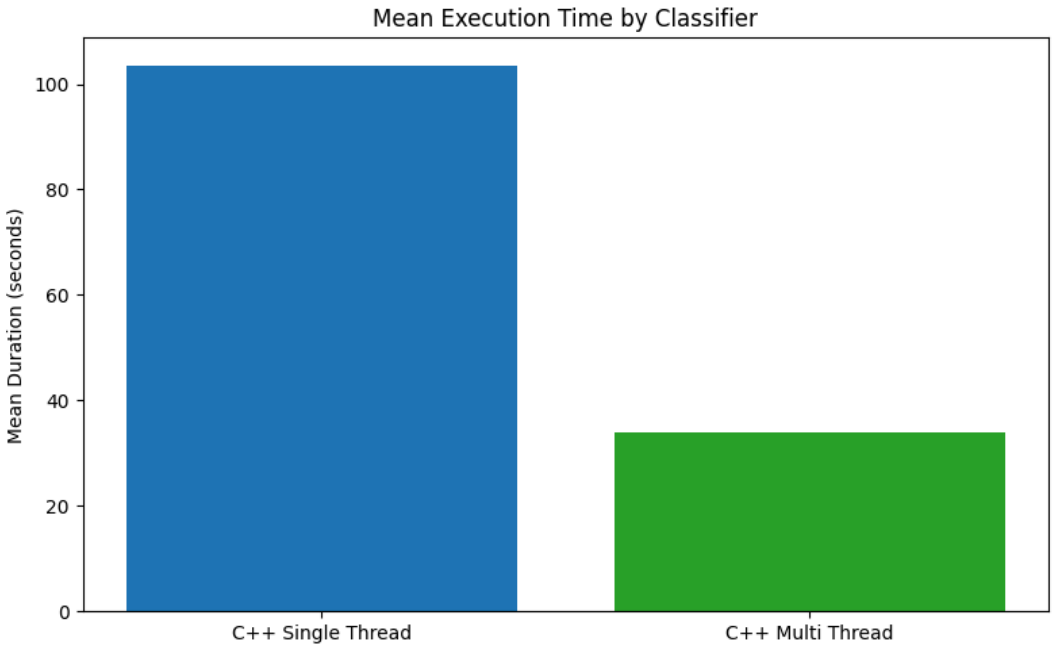

Figura 4. Comparativo de tempo de execução implementações C++

4.2. Speedup, eficiência, fração paralelizável e métrica de Karp-Flatt

Por fim, quanto às métricas de avaliação de algoritmos paralelos, a implementação paralela do *Random Forest* utilizando o OpenMP obteve um *Speedup* = 3.042, uma *eficiência* =0.38, fração P = 0.76, e a métrica de *Karp-Flatt* = 0.23

5. Considerações finais

Apesar de a implementação *multi-thread* em C++ ter trazido uma considerável melhora no tempo de execução, ambas as implementações em C++ tiveram impacto negativo na precisão e na acurácia para a classificação dos objetos astronômicos.

Tal resultado não indica, entretanto, que a paralelização é a causa desse comportamento. Características da implementação escolhida sugerem ser a causa mais provável. Dessa forma, alterações na implementação poderiam elevar a pontuação das métricas em questão, ao mesmo tempo em que manteriam o *speedup* alcançado.

Outras métricas que merecem destaque são a eficiência e a métrica de *Karp-Flatt,* cujos valores indicam que a paralelização do algoritmo foi realizada de maneira ineficiente, ocasionando em ociosidade das unidades de processamento em algum momento, e gargalos ocasionados por *overhead*.

Dessa forma, como conclusão deste trabalho, sugere-se que outros projetos relacionados explorem os recursos de *multithreading* das máquinas para aprimorar a eficiência desses e de outros algoritmos que auxiliem no processamento de grandes volumes de dados, e consequentemente na execução de estudos científicos, ou mesmo no cotidiano da humanidade.

Referências bibliográficas

[1] MICHALOWSKI, M. How Much Data Is Generated Every Day in 2024? Disponível em: <https://spacelift.io/blog/how-much-data-is-generated-every-day>. Acesso em: 7 jul. 2025.

[2] JOSÉ-LUIS SOLORIO-RAMÍREZ et al. Random forest Algorithm for the Classification of Spectral Data of Astronomical Objects. Algorithms, v. 16, n. 6, p. 293–293, 8 jun. 2023.

[3] LIU, Y.; WANG, Y.; ZHANG, J. New Machine Learning Algorithm: Random Forest. Information Computing and Applications, v. 7473, p. 246–252, 2012.

[4] AALAA-A. GitHub - Aalaa-A/stellar-classification-ML: A machine learning pipeline for classifying stars using Random Forest, MLP, K-means, and CNNs. Includes data preprocessing, hyperparameter tuning, and result visualization. Disponível em: <https://github.com/Aalaa-A/stellar-classification-ML/tree/main>. Acesso em: 7 jul. 2025.

[5] XINGYIZHOU. GitHub - xingyizhou/random-forest-classifier: my implementation of standard random forest classifier. Disponível em: <https://github.com/xingyizhou/random-forest-classifier>. Acesso em: 7 jul. 2025.

[6] EROGOL. GitHub - erogol/Random_Forests: This is a very efficient Random Forests implementation for MATLAB based on low level “cart” tree implementation. Disponível em: <https://github.com/erogol/Random_Forests>. Acesso em: 7 jul. 2025.

[7] JESSE-XLKUO. GitHub - jesse-xlkuo/WeightedRandomForest. Disponível em: <https://github.com/jesse-xlkuo/WeightedRandomForest>. Acesso em: 7 jul. 2025.

[8] Boost.Python. Disponível em: <https://www.boost.org/doc/libs/1_88_0/libs/python/doc/html/index.html>. Acesso em: 7 jul. 2025.

[9] FARID R. Sloan Digital Sky Survey - DR18. Disponível em: <https://www.kaggle.com/datasets/diraf0/sloan-digital-sky-survey-dr18>. Acesso em: 7 jul. 2025.

[10] TIM.LEWIS. Home. Disponível em: <https://www.openmp.org/>. Acesso em: 7 jul. 2025.

[11] PEDROMONTUANI. pedromontuani/Parallel-Random-Forest-Astronomic-Classification. Disponível em: <https://github.com/pedromontuani/Parallel-Random-Forest-Astronomic-Classification>. Acesso em: 7 jul. 2025.

[12] SCIKIT-LEARN. scikit-learn: Machine Learning in Python. Disponível em: <https://scikit-learn.org/stable/>. Acesso em: 7 jul. 2025.