[HPC] Final Project - Classificação de Asteroids usando Machine Learning e MPI

Athyrson M. Ribeiro
a203963@dac.unicamp.br

December, 2024

1 Introdução

A detecção e classificação de asteroides é uma área de crescente interesse devido ao seu potencial risco para a Terra. A tarefa de identificar asteroides potencialmente perigosos (PHA - Potentially Hazardous Asteroids) envolve a análise de grandes volumes de dados, como órbitas, dimensões e outras características físicas dos corpos celestes. A evolução das técnicas de aprendizado de máquina tem proporcionado novas abordagens para essa classificação, permitindo a automação e a melhoria na precisão dos modelos preditivos [1].

Este trabalho utiliza um conjunto de dados recente sobre asteroides, com o objetivo de avaliar modelos preditivos na identificação de se um asteroide representa uma ameaça. A partir deste conjunto, foram aplicadas diversas técnicas de aprendizado supervisionado, com ênfase na integração de técnicas de paralelização para otimizar o processo de treinamento dos classificadores. Especificamente, utilizamos as bibliotecas `Sklearn` e `Mpi4py` [2, 3] para explorar a execução paralela, reduzindo o tempo de processamento e permitindo a análise de grandes volumes de dados de maneira mais eficiente.

O estudo avalia a performance de diferentes classificadores, utilizando métricas como AUC (Área Sob a Curva), e foca na análise do desempenho em cenários com dados desbalanceados, comuns em problemas de classificação envolvendo eventos raros. O principal objetivo desse trabalho é fornecer uma análise de como as técnicas de paralelização podem ser aplicadas para melhorar a eficiência e a precisão dos modelos de classificação em problemas complexos como a detecção de asteroides perigosos.

2 Metodologia

2.1 Dados

Para este trabalho, foi selecionado o conjunto de dados Asteroide, disponível no repositório Kaggle[1]. Esse conjunto foi organizado pelo astrônomo e pesquisador em astrofísica Mir Sakhawat Hossain, que compilou os dados oficialmente mantidos pelo Jet Propulsion Laboratory (JPL) do California Institute of Technology (Caltech), uma organização vinculada à NASA.

O conjunto de dados reúne informações abrangentes relacionadas a asteroides e está publicamente acessível no site do JPL Small-Body Database Search Engine. Hossain já publicou um artigo utilizando este mesmo conjunto de dados, aplicando técnicas de aprendizado de máquina para identificar asteroides potencialmente perigosos. Nesse estudo, ele obteve resultados impressionantes, com altíssima acurácia nas tarefas de classificação e regressão [1].

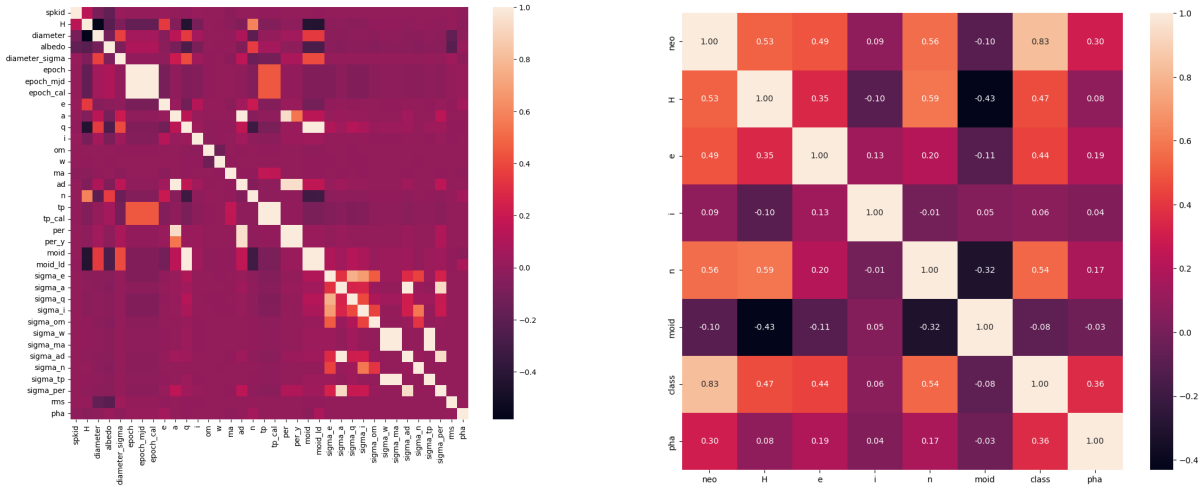
O conjunto de dados original contém **766.819 amostras** e **35 colunas**, sendo uma delas (*PHA - Potentially Hazardous Asteroid*) dedicada ao rótulo que indica se um determinado asteroide é classificado como *potencialmente perigoso* ou não. Como esperado, há um desbalanceamento significativo neste rótulo: apenas **0,2%** das amostras são identificadas como PHA.

2.2 Análise e pré-processamento dos dados

Foi realizada uma análise inicial no conjunto de dados para identificar valores faltantes e dados ruidosos. Colunas com uma quantidade muito grande de valores ausentes, bem como colunas de *IDs*, foram removidas. A Figura 1a apresenta a matriz de correlação das colunas restantes. Como pode ser observado

[1] `https://www.kaggle.com/datasets/sakhawat18/asteroid-dataset`

na figura, a maioria dessas colunas apresenta baixa correlação com a coluna *PHA*. Para identificar as colunas com informações relevantes para a tarefa de classificação, foi aplicada a técnica do coeficiente de correlação de Pearson, utilizando como critério $|r(c)| \geq 0.01$, onde c representa uma coluna do conjunto de dados. A Figura 2 apresenta as colunas selecionadas como resultado desta análise e a Figura 1b apresenta a nova matriz de correlação das colunas restantes.



(a) Correlação das colunas no conjunto de dados Asteroid.

(b) Correlação das colunas selecionadas após aplicação do critério de relevância.

Figure 1: Comparação entre as matrizes de correlação antes e depois da seleção de colunas.

```
r_value[(r_value.abs() >= 0.01)].sort_values(ascending=False)
✓ 0.0s
e           0.191074
n           0.167914
H           0.083232
i           0.033952
moid_ld    -0.029941
moid       -0.029941
q          -0.035188
dtype: float64
```

Figure 2: Colunas selecionadas como resultado da técnica coeficiente de correlação de Pearson.

Como observado na Figura 1, esse conjunto de dados traz a coluna *class*, que contém a classe de cada asteroide. Existem no total 11 classes de asteroides presentes no conjunto. Abaixo, são descritas as classes presentes no conjunto de dados e seus respectivos significados:

- **MBA**: *Main-Belt Asteroid* - Asteroide do cinturão principal.
- **APO**: *Apollo Asteroid* - Asteroide Apollo, com órbitas que cruzam a órbita da Terra e semieixos maiores que o da Terra.
- **IMB**: *Inner Main-Belt Asteroid* - Asteroide da região interna do cinturão principal.
- **OMB**: *Outer Main-Belt Asteroid* - Asteroide da região externa do cinturão principal.
- **AMO**: *Amor Asteroid* - Asteroide Amor, com órbitas próximas à da Terra, mas que não a cruzam.
- **TJN**: *Jupiter Trojan* - Asteroides troianos de Júpiter, localizados em torno dos pontos de Lagrange do planeta.
- **MCA**: *Mars-Crossing Asteroid* - Asteroide que cruza a órbita de Marte.
- **TNO**: *Trans-Neptunian Object* - Objeto transnetuniano, localizado além da órbita de Netuno.
- **ATE**: *Atira Asteroid* - Asteroide Atira, com órbitas inteiramente dentro da órbita da Terra.

- **AST**: *Asteroid* - Asteroide genérico que não pertence a nenhuma das subclasses específicas.
- **CEN**: *Centaur* - Centauro, objetos com características mistas de asteroides e cometas, localizados entre Júpiter e Netuno.
- **IEO**: *Inner-Earth Object* - Objeto com órbita interna à da Terra.

Porém, foi observado que algumas dessas classes não continham nenhum asteroide marcado como potencialmente perigoso, como pode ser visto na Figura 3. Isso pode enviesar a tarefa de classificação, pois o modelo pode simplesmente aprender que, se o asteroide pertencer a uma dessas classes, ele não pode ser perigoso. Por esse motivo, foram mantidas somente as amostras que pertenciam às classes *APO*, *ATE*, *AMO* e *IEO*. Essa decisão afetou consideravelmente a coluna *NEO*, que passou a ter apenas um tipo de valor, tornando-se não informativa. No final o tamanho total do dataset ficou em 22883 amostras e 7 colunas.

```
Class: MBA == 0.0 in 666210 samples
Class: MCA == 0.0 in 14287 samples
Class: OMB == 0.0 in 21729 samples
Class: APO == 0.139 in 10107 samples
Class: TJN == 0.0 in 6483 samples
Class: IMB == 0.0 in 15743 samples
Class: TNO == 0.0 in 2730 samples
Class: ATE == 0.104 in 1361 samples
Class: AMO == 0.014 in 6751 samples
Class: CEN == 0.0 in 409 samples
Class: IEO == 0.312 in 16 samples
Class: AST == 0.0 in 43 samples
```

Figure 3: Distribuição de asteroides potencialmente perigosos (*PHA*) em cada classe.

A Figura 4 apresenta a visualização em 3D da Análise de Componentes Principais (PCA) aplicada ao conjunto de dados de asteroides. Como pode ser observado, as diferentes classes tendem a se agrupar de forma semi-isolada, com uma leve sobreposição no centro da figura. Essa disposição sugere que, apesar das classes estarem relativamente separadas, existe alguma interdependência entre elas, o que pode indicar características compartilhadas entre os asteroides em certas regiões do espaço de características.

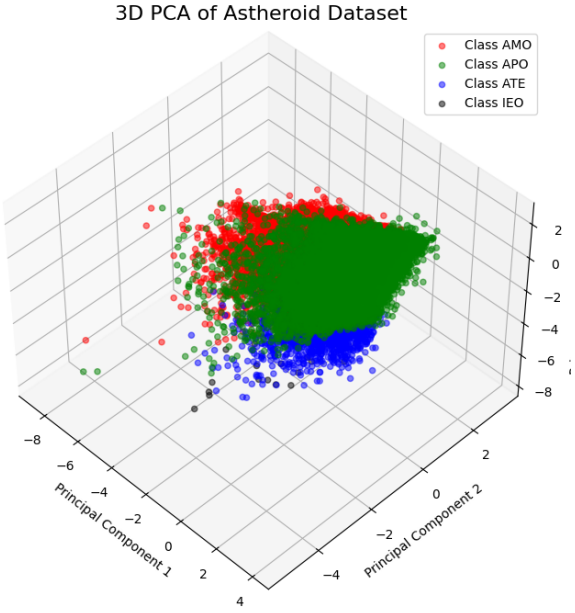


Figure 4: Representação em 3D da Análise de Componentes Principais do conjunto de dados de asteroides.

2.3 Ambiente de execução

Para a realização deste trabalho, foi utilizado o cluster de alto desempenho do Laboratório de Inteligência Artificial Recod.ai, vinculado ao Instituto de Computação da Universidade Estadual de Campinas (UNICAMP) [4]. O laboratório disponibiliza uma variedade de CPUs e GPUs para os seus membros

pesquisadores. Para a realização deste trabalho, foi utilizado o nó de GPU DL-17, equipado com processador Intel(R) Xeon(R) Gold 5220 CPU @ 2.20GHz, 72 núcleos e 512 GB de RAM. Porém, todo o processamento e a execução dos experimentos foram realizados exclusivamente com a CPU do nó. Para este trabalho foi utilizada a linguagem de programação Python.

2.4 Classificadores

Foram utilizados sete classificadores, extraídos da biblioteca scikit-learn [2], para o experimento:

- `LogisticRegression`: `LogisticRegression(random_state=args.seed, max_iter=1000)`,
- `RandomForest`: `RandomForestClassifier(random_state=args.seed)`,
- `GradientBoosting`: `GradientBoostingClassifier(random_state=args.seed)`,
- `AdaBoost`: `AdaBoostClassifier(algorithm='SAMME', random_state=args.seed)`,
- `Decision Tree`: `DecisionTreeClassifier(random_state=args.seed)`,
- `NeuralNetwork`: `MLPClassifier(random_state=args.seed)`,
- `SVM`: `SVC(random_state=args.seed, probability=True)`.

Foi optado por não fazer otimização dos hyper-parametros dos modelos, considerando que o foco deste trabalho não foi a acurácia dos modelos.

2.5 Execuções

Para os experimentos com MPI (*Message Passing Interface*), cada experimento consistiu em 20 execuções utilizando os mesmos parâmetros, mas com diferentes sementes de aleatoriedade. Considerando que o nó DL-17 possui 72 núcleos, foi definido que até 3 configurações poderiam ser executadas simultaneamente. Para cada configuração, as execuções foram distribuídas, utilizando funções da biblioteca Mpi4py, entre os *ranks* disponíveis, o que permitiu maior rapidez na realização dos experimentos. Ao término das execuções, a média dos resultados de todos os *ranks* foi calculada e armazenada como o resultado final. As configurações estudadas foram criadas a partir dos seguintes argumentos:

- **Seed (Sementes de Aleatoriedade):**
  - Define uma semente S para a geração de números aleatórios. Cada um dos 20 *ranks* (r_i, onde $i \in \{1, 2, \ldots, 20\}$) recebe uma semente individual $S_i = S + r_i$. Essa abordagem assegura a reprodutibilidade dos experimentos, atribuindo a cada *rank* uma sequência pseudoaleatória única, mas controlada. Ao término das execuções, a média dos resultados com todas essas sementes foi calculada e armazenada como o resultado final.
  - Valores possíveis:
    - 42
    - 123
    - 314
- **Frações:**
  - Controla a fração do conjunto de dados usada nos experimentos, simulando diferentes tamanhos de amostras.
  - Valores possíveis:
    - 1 (100% dos dados)
    - 2 (50% dos dados)
    - 4 (25% dos dados)
    - 8 (12.5% dos dados)*
    - 16 (6.25% dos dados)*
- **Modelos:**

  - Representam os classificadores utilizados nos experimentos. Nos experimentos focados em MPI, foram avaliados sete classificadores distintos.
  - Valores possíveis para os classificadores:
    * 0: Regressão Logística
    * 1: Random Forest
    * 2: Gradient Boosting
    * 3: AdaBoost
    * 4: Decision Tree
    * 5: Rede Neural (*Neural Network*)
    * 6: SVM (*Support Vector Machine*)

- **Drops:**
  - Indica quantas colunas foram removidas do conjunto de dados antes de treinar o modelo. A remoção foi realizada de acordo com a ordem de coeficiente de correlação de Pearson, removendo primeiro as colunas com menor valor informativo.
  - Valores possíveis:
    * 0: Nenhuma coluna removida.
    * 1: Uma coluna removida.
    * 2: Duas colunas removidas.
    * 3: Três colunas removidas.
    * 4: Quatro colunas removidas.

Portanto, para o experimento com MPI, foram executadas $3 \times 7 \times 4 = 84$ configurações distintas. Cada configuração foi executada $20 \times 3 = 60$ vezes com uma semente diferente. Foram realizadas 20 execuções paralelas, com até 3 configurações sendo executadas simultaneamente, aproveitando os 72 núcleos disponíveis no nó DL-17. Essa estratégia permitiu maior rapidez na execução dos experimentos e um uso otimizado dos recursos computacionais.

Já nos experimentos que investigaram o impacto do hiperparâmetro *número de trabalhos* (*n_jobs*) do modelo Random Forest, o hiperparâmetro *número de estimadores* (*n_estimators*) foi ajustado para 1000, enquanto o *número de trabalhos* foi variado entre os seguintes valores: 1, 2, 4, 8, 16, 32 e 64 e o parâmetro *fraction* foi variado entre os seguintes valores: 1, 2, 4, 8, e 16. O hiperparâmetro *número de trabalhos* controla a quantidade de núcleos ou threads utilizados pelo algoritmo, permitindo paralelizar o treinamento do modelo, com cada valor indicando o número de núcleos de processamento a ser usado [2]. Quanto maior o valor de *n_jobs*, maior a utilização de recursos computacionais, o que pode resultar em um treinamento mais rápido, dependendo da carga de trabalho e do hardware disponível. Para esse experimento com o hiperparâmetro *número de trabalhos*, foram executadas $7 \times 5 = 35$ configurações distintas, cada uma executada 20 vezes. Dessa vez, a paralelização foi realizada exclusivamente devido a esse hiperparâmetro, sem o uso de MPI ou execuções de script em paralelo, com o objetivo de analisar o speedup e a eficiência dessa forma de paralelização.

2.6 Métricas

2.6.1 AUC - Area Under the Curve

Quanto maior o valor da AUC, melhor o desempenho do classificador. A AUC também reflete o desempenho do modelo mesmo quando lidando com conjuntos de dados desbalanceados. A **Área Sob a Curva** é dada por:

$$AUC = P[p(y = 1|X_i) > p(y = 1|X_j)|y_i = 1, y_j = 0] \tag{1}$$

2.6.2 Comparação de Tempo de Execução

No contexto dos experimentos, três métricas principais foram analisadas: o **tempo de execução**, o **speedup** e a **eficiência**.

O **speedup** é uma métrica que indica o quanto o uso de paralelismo melhora o desempenho em relação à execução serial. Ele é calculado como a razão entre o tempo da execução serial e o tempo da execução paralela. Temos as seguintes fórmulas para o cálculo do **speedup**:

$$S_{mpi} = \frac{T_s}{T_{mpi}}$$

A **eficiência** é uma métrica que avalia o quão bem os recursos paralelos estão sendo utilizados. Ela é calculada como o speedup dividido pelo número de processos P utilizados, e indica se o ganho em desempenho é proporcional ao aumento no número de processos. A fórmula para a eficiência com MPI é dada por:

$$E_{mpi} = \frac{S_{mpi}}{P}$$

3 Resultados

3.1 Experimento I - Utilizando MPI para treinamento de diferentes classificadores

As Figuras 5, 6, 7 e 8 apresentam os resultados do experimento que analisou o impacto da remoção de amostras e colunas no tempo de execução e na AUC dos modelos. As execuções benchmark foram definidas como aquelas com os parâmetros *fraction* = 1.0 (100% dos dados de treino foram utilizados) e *drops* = 0 (nenhuma coluna do conjunto de treino foi descartada, cor azul nos gráficos). A Figura 5 demonstra que a diminuição do número de amostras do conjunto de dados resulta em uma execução mais rápida para alguns dos modelos testados, como SVM e NeuralNetwork. Porém, a remoção de certas colunas teve um impacto pouco significativo no tempo de execução na maioria dos modelos.

A Figura 6 mostra claramente o speedup obtido com a diminuição do número de amostras e colunas. Como se pode observar nos modelos Decision Tree, GradientBoosting e LogisticRegression com *fraction* = 1, o speedup nem sempre é positivo com a remoção de colunas. O speedup mais claro é obtido com a redução do número de amostras. Pode-se observar que, para o modelo SVM, é possível obter um speedup próximo a 16 vezes em relação ao modelo benchmark com uma redução de 4 vezes no número de amostras.

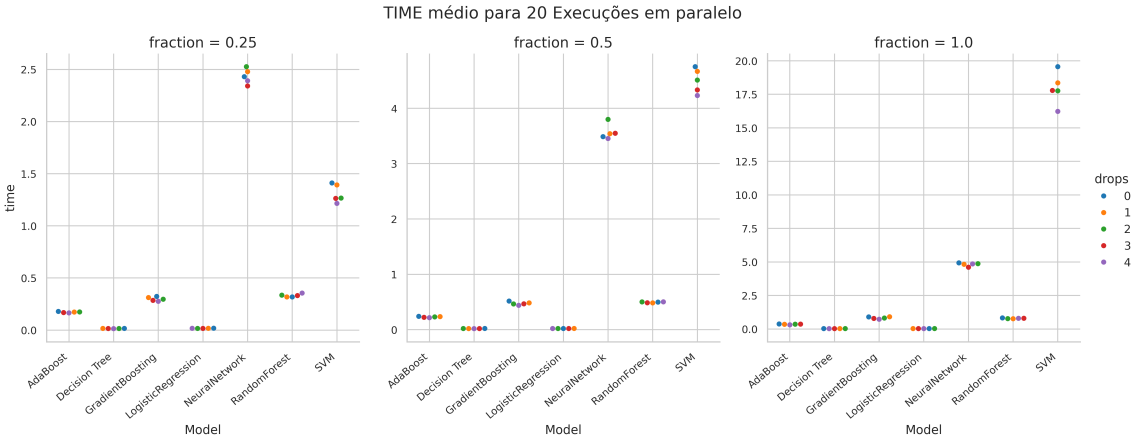


Figure 5: Tempo médio para 20 execuções em paralelo

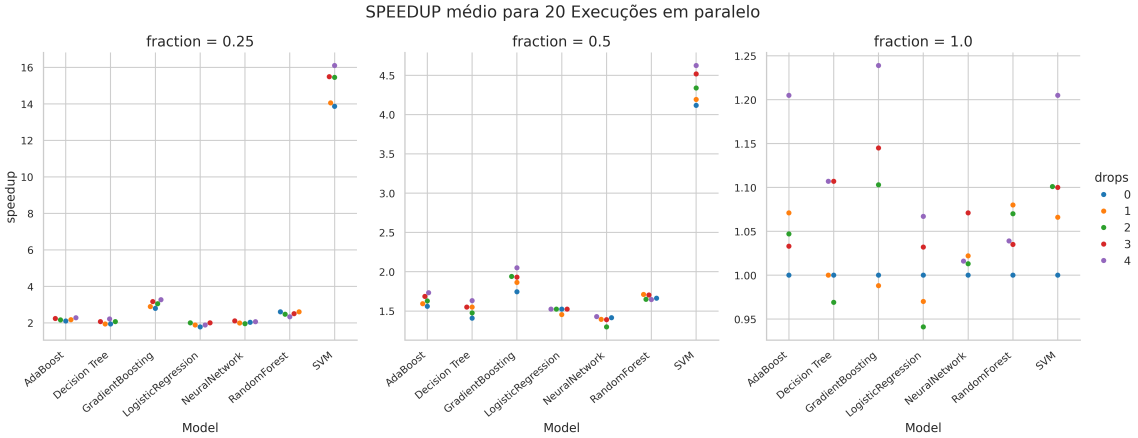


Figure 6: Speedup médio para 20 execuções em paralelo

A Figura 7 traz a visão da performance dos modelos ao serem submetidos a essas configurações, e a Figura 8 mostra o quanto cada modelo perdeu ou ganhou em AUC em relação ao modelo benchmark. É interessante notar que, para certos modelos, como Adaboost, GradientBoosting, LogisticRegression e NeuralNetwork, a redução do número de colunas ou amostras no treino teve impacto mínimo na performance. O modelo que mais sofreu com as reduções na dimensão do conjunto de treino foi o DecisionTree, e, curiosamente, o modelo de classificação SVM obteve um aumento de performance com a redução da dimensionalidade do conjunto de treinamento.

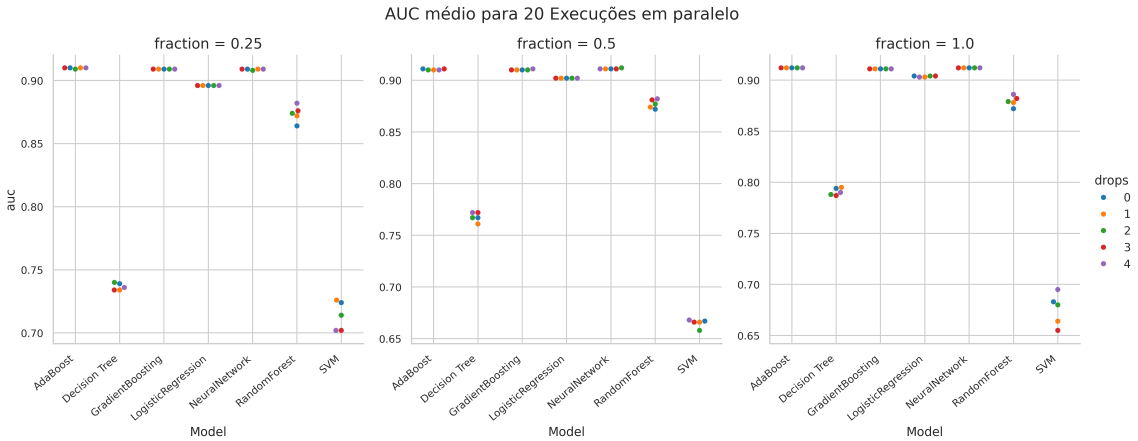


Figure 7: AUC médio para 20 execuções em paralelo

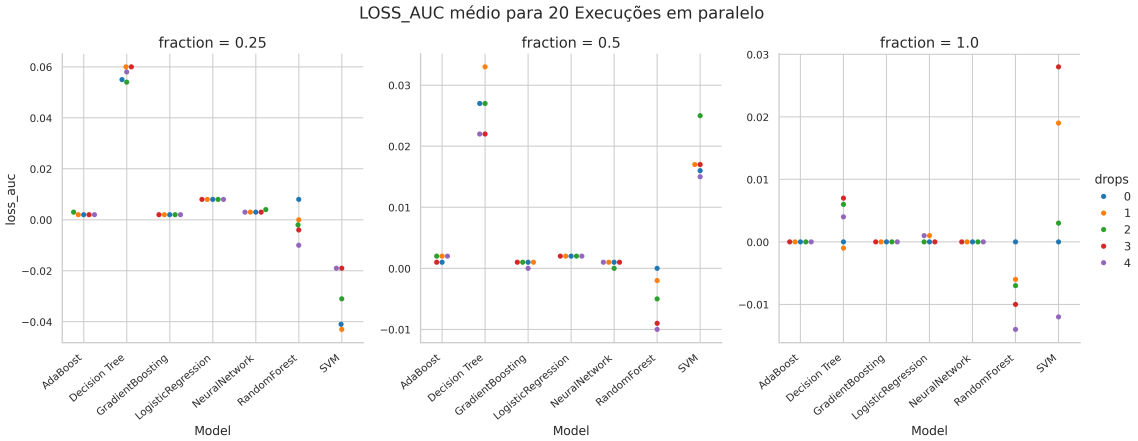


Figure 8: Loss AUC médio para 20 execuções em paralelo

Através deste experimento, foi possível concluir que, para tarefas específicas, como a identificação de asteroides potencialmente perigosos, a redução da dimensionalidade do conjunto de dados pode trazer benefícios para o desempenho de vários modelos. A partir desses resultados, é possível extrapolar um cenário em que as execuções foram realizadas de forma serial para todos os modelos, sem a utilização de paralelização. Ao empregar execuções paralelas com MPI, foi possível acelerar os experimentos em até 20 vezes, sem comprometer a robustez dos resultados. Um exemplo claro desse ganho de performance é observado no modelo SVM: ao utilizar apenas 0.25% dos dados iniciais, o speedup alcançado foi de 20 vezes em relação à execução serial e de aproximadamente 450 vezes em comparação com a execução serial de 20 modelos benchmark. Isso evidencia um ganho significativo na eficiência computacional e na redução do tempo de execução dos experimentos.

3.2 Experimento II - Analise da eficiência do parâmetro *número de trabalhos*

As Figuras **??**, 10 e 11 trazem respectivamente os resultados de tempo de execução, speedup e eficiência médios obtidos pelo modelo RandomForest através da utilização de diferentes valores do hyperparâmetro *número de trabalhos*. O objetivo desse experimento foi analisar o impacto de um ajuste simples e intuitivo na performance computacional de um modelo da biblioteca Sklearn. Analisamos dessa vez um número maior de opções de granularidade no número de amostras do conjunto de dados. Como se pode observar na Figura 9 o tempo de execução de cada configuração ficou entre 1.5 e 6 segundos.

A Figura 10 apresenta os valores de *speedup* obtidos para cada configuração testada, utilizando como referência a configuração *Benchmark* (com *fraction* = 1.0 e número de trabalhos = 1). Os resultados evidenciam uma tendência geral de aumento no *speedup* com a redução do tamanho do conjunto de treinamento (*fraction*) e o incremento no número de trabalhos em paralelo.

A configuração com o maior *speedup* foi aquela que combinou o menor tamanho de conjunto de dados (*fraction* mínima) com um número intermediário de trabalhos em paralelo, alcançando um *speedup* de aproximadamente 4 vezes em relação ao *Benchmark*. Entretanto, ao utilizar o conjunto completo de dados (*fraction* = 1), os melhores resultados de *speedup* foram obtidos com configurações intermediárias de paralelismo (número de trabalhos igual a 2, 4 ou 8). Esses resultados sugerem que, embora o paralelismo aumente a eficiência do processamento, um número excessivamente alto de trabalhos pode introduzir sobrecarga adicional, reduzindo os ganhos esperados.

A Figura 11 reforça os padrões observados anteriormente. Para as configurações que utilizaram o conjunto completo de dados (*fraction* = 1.0), a eficiência do uso de múltiplos processadores permaneceu consistentemente abaixo de 100%, indicando perdas devido à sobrecarga de paralelização e coordenação entre os processadores.

Por outro lado, a redução no número de amostras resultou em uma tendência clara de melhoria na eficiência, aproximando-se de um comportamento linear. Esse resultado sugere que a menor carga de dados por processador reduz significativamente a sobrecarga de comunicação e sincronização, permitindo que os recursos computacionais sejam utilizados de forma mais efetiva. Configurações com tamanhos menores de conjunto de dados (*fraction* menores) mostraram-se mais adequadas para explorar o potencial do paralelismo, especialmente em cenários com muitos processadores disponíveis.

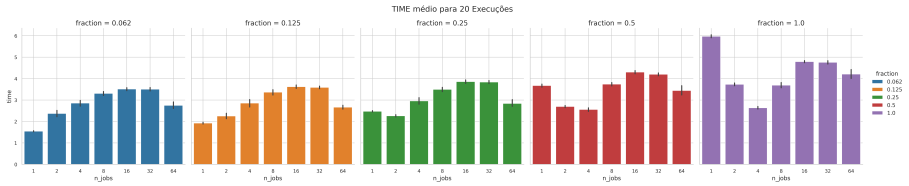


Figure 9: Tempo médio para 20 execuções

Figure 10: Speedup médio para 20 execuções

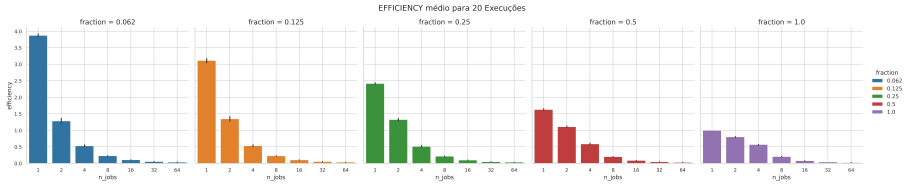


Figure 11: Eficiência média para 20 execuções

4 Conclusão

Neste trabalho, foi analisado o impacto de diferentes configurações de paralelização e redução de dimensionalidade em um problema de classificação voltado à identificação de asteroides potencialmente perigosos (PHAs). Foram realizados experimentos utilizando bibliotecas como *Scikit-learn* e *Mpi4py*, avaliando métricas de desempenho como tempo de execução, *speedup*, eficiência e *AUC*.

Os resultados demonstraram que a redução do tamanho do conjunto de dados, seja pela diminuição do número de amostras ou pela remoção de colunas, pode trazer benefícios significativos, como menor tempo de execução e, em alguns casos, melhorias no desempenho do modelo. Modelos como SVM apresentaram *speedup* e ganhos de desempenho notáveis com conjuntos de dados reduzidos, enquanto outros, como Decision Tree, mostraram maior sensibilidade à redução de dimensionalidade.

Adicionalmente, os experimentos com o hiperparâmetro *número de trabalhos* (*n_jobs*) do modelo Random Forest revelaram que o aumento excessivo de paralelismo nem sempre resulta em *speedups* proporcionais, devido à sobrecarga de coordenação entre os processadores. Configurações com *n_jobs* moderados demonstraram maior eficiência, especialmente quando combinadas com conjuntos de dados menores.

Por fim, este estudo evidencia a importância de considerar cuidadosamente a configuração de paralelização e a dimensionalidade do conjunto de dados em tarefas computacionalmente intensivas. O uso de paralelização com MPI mostrou-se eficaz na redução do tempo total de execução, garantindo reprodutibilidade e robustez nos experimentos. Os insights obtidos podem ser aplicados em problemas similares que exigem alta capacidade de processamento e otimização de recursos computacionais. Trabalhos futuros podem explorar outras abordagens de paralelização, diferentes estratégias de pré-processamento e análise do impacto em cenários com conjuntos de dados ainda mais desbalanceados ou heterogêneos.

References

[1] Mir Sakhawat Hossain and Md. Akib Zabed. Machine learning approaches for classification and diameter prediction of asteroids. In Mohiuddin Ahmad, Mohammad Shorif Uddin, and Yeong Min Jang, editors, *Proceedings of International Conference on Information and Communication Technology for Development*, pages 43–55, Singapore, 2023. Springer Nature Singapore.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[3] Lisandro Dalcin and Yao-Lung L. Fang. mpi4py: Status update after 12 years of development. *Computing in Science Engineering*, 23(4):47–54, 2021.

[4] NVIDIA. Recod.ai, laboratório de inteligência artificial da unicamp, se torna nvidia ai joint lab, 2023. Acesso em: 09 dez. 2024.