Em Busca de uma Inteligência Artificial Ecologicamente Viável: Um estudo de caso do Consumo Energético de Algoritmos de Árvore de Decisão

Felipe Bernardo, Mariza Ferro, Vitor Vieira, Gabrieli Silva, Bruno Schulze

[1]Laboratório Nacional de Computação Científica (LNCC)
Getúlio Vargas, 333, Quitandinha – Petrópolis – Rio de Janeiro

{felipeb,mariza,vvieira,gabrieli,schulze}@lncc.br

Abstract. *The use of Artificial Intelligence has been showing accelerated growth due to its use in solving problems in several application domains. This success is the result of the convergence of large amounts of data, high performance computing and precision of Machine Learning (ML) algorithms. Even with the relevance of ML algorithms, little is known about their computational requirements and power consumption, which has become an important task to achieve greener computing. The objective of this work is to evaluate the power consumption of the Decision Tree algorithms to identify their power consumption hotspots. Also, investigate the emission of CO_2 equivalent associated with the algorithms.*

Resumo. *O uso da Inteligência Artificial vem apresentando crescimento acelerado dado à sua utilização na solução de problemas em diversos domínios de aplicação. Este sucesso é resultado da convergência entre grande quantidade de dados, computação de alto desempenho e precisão dos algoritmos de Aprendizado de Máquina (AM). Mesmo com a relevância dos algoritmos de AM, pouco se sabe sobre seus requisitos computacionais e consumo energético, o que tornou-se tarefa importante para alcançar uma computação mais ecológica. O objetivo deste trabalho é avaliar o consumo de energia dos algoritmos de Árvore de Decisão, a fim de identificar os hotspots de energia dos mesmos. E ainda, investigar a emissão de CO_2 equivalente associada aos algoritmos.*

1. Introdução

A Inteligência Artificial (IA) está se tornando cada vez mais popular e vem sendo utilizada com muito sucesso devido ao desenvolvimento de soluções revolucionárias e com aplicações que vem sendo utilizadas no avanço científico em muitos domínios. Essas aplicações envolvem principalmente o uso de algoritmos da sua sub-área, o Aprendizado de Máquina (AM), que tem alcançado níveis de precisão muito altos em tarefas tais como, reconhecimento de imagens e voz, traduções automáticas e diagnóstico de patologias. Esses avanços se devem em grande parte ao grande volume de dados disponíveis, associado ao uso da Computação de Alto Desempenho (HPC), condições que permitem o treinamento desses algoritmos.

Porém, ambientes de HPC requerem muita energia para manter sua infraestrutura de nós computacionais e capacidade de refrigeração. Prevê-se que até 2040 metade do consumo mundial de energia elétrica será atribuído a instalações de

computação [Villani et al. 2018]. Por outro lado, estudos recentes apontam que o impacto para o desenvolvimento completo de um modelo de Processamento de Linguagem Natural (NLP), usando redes neurais profundas (Deep Learning), pode emitir tanto dióxido de carbono (CO_2) quanto cinco carros durante suas vidas útil [Strubell et al. 2019]. Isso levanta muitas preocupações sobre como fazer uma IA ecologicamente viável. A maioria dos algoritmos de AM foram desenvolvidos há décadas, quando ainda nem se pensava na convergência entre HPC e IA, por isso, não foram implementados com foco em eficiência energética, o que atualmente tornou-se uma tarefa essencial.

Mesmo com a relevância dos algoritmos de AM, pouco se sabe a respeito dos seus requisitos computacionais e consumo energético em diferentes arquiteturas computacionais. A maioria das pesquisas nesta área se concentram em melhorar a precisão dos algoritmos, sem avaliar seus requisitos computacionais e otimizar o seu desempenho [García-Martín et al. 2019]. Por exemplo, os requisitos de um algoritmo de Rede Neural Artificial (RNA) podem ser muito diferentes dos requisitos de um algoritmo de Árvore de Decisão (AD). Ou seja, as diferentes tarefas de AM, envolvem diferentes algoritmos, os quais podem ter requisitos computacionais e consumo energético bastante diversos. Além disso, com a crescente necessidade pela HPC para o treinamento dos algoritmos de AM, é necessário compreender se esses algoritmos possuem um desempenho satisfatório nessas arquiteturas e se podem ser otimizados para uma melhor eficiência energética.

Assim, o objetivo deste trabalho é estabelecer uma metodologia para: i) identificar as principais funções dos algoritmos de AM, ou seja, as funções que tem maior impacto sobre o desempenho e o consumo de energia dos algoritmos (o que denominamos de hotspots); ii) determinar as causas desse impacto e buscar soluções para uma melhor eficiência no desempenho e consumo energético, a fim de contribuir para uma IA ecologicamente viável. Para isso foi realizado um estudo de caso utilizando algoritmos de AD. Esses algoritmos, apesar de sua simplicidade, ainda estão entre os mais utilizados em AM devido a interpretabilidade dos seus resultados, lidar bem com atributos faltantes e ter uma boa precisão para muitas aplicações.

Este trabalho está organizado da seguinte forma: Na Seção 2 são apresentados alguns trabalhos relacionados. Na Seção 3 é apresentada a metodologia adotada para a execução dos experimentos. Na Seção 4 são descritos os algoritmos avaliados neste trabalho. Na Seção 5 são apresentados os experimentos e os resultados obtidos. Finalmente, na Seção 6 são apresentadas as considerações finais e os trabalhos futuros.

2. Trabalhos Relacionados

A maioria dos trabalhos encontrados na literatura tratam do desempenho de algoritmos de AM com foco na precisão dos modelos para resolver tarefas específicas em uma área de aplicação, tais como [Malakar et al. 2018, Olson et al. 2017, Serpa et al. 2018]. Ou ainda, trabalhos que usam algoritmos de AM para predizer o desempenho e o consumo de energia sobre a execução de uma aplicação científica [Siegmund et al. 2015, Wu et al. 2016, Ferreira et al. 2017, Klôh et al. 2019]. Porém, ainda existem poucos trabalhos que avaliam o consumo de energia dos algoritmos de AM [Li et al. 2016, Garcia-Martin et al. 2017, Yang et al. 2017, Abdelhafez et al. 2019, García-Martín et al. 2019].

Em [Garcia-Martin et al. 2017] são avaliados os hotspots de energia do algoritmo

Very Fast Decision Tree (VFDT). A implementação, o treinamento e teste do algoritmo são realizados no framework *Massive Online Analysis (MOA)* [Bifet et al. 2010] e a energia é medida com a ferramenta Jalen [Noureddine et al. 2015]. Os autores descrevem as 12 funções implementadas no código e as agrupa em quatro diferentes conjuntos, os quais representam as funções genéricas do algoritmo de AD *GrowTree* [Flach 2012]. Os experimentos foram realizados com 4 conjuntos de dados distintos, todos contendo 10 milhões de exemplos, mas com diferentes quantidades de atributos e sob diferentes configurações de parâmetros. Os autores demonstram que as diferentes configurações de parâmetros e conjuntos de dados possuem um impacto significativo no consumo de energia para o algoritmo VFDT. O trabalho apresenta resultados relevantes, apesar do framework MOA não ser comumente utilizado para execução de algoritmos de AM.

No trabalho de [Li et al. 2016] é realizado um estudo sobre a eficiência energética dos modelos de Redes Neurais Convolucionais (RNC). Os autores analisam os hotspots de energia e quais as influências das configurações de hardware (CPU e GPU) no consumo energético. São limitados o uso de núcleos dos processadores, a frequência de memória e outros parâmetros da placa de vídeo. Além disso, como existe um grande número de *frameworks* (Caffe, Tensorflow e Torch) para execução de algoritmos de AM, também é avaliado o desempenho desses *frameworks* de acordo com as configurações de hardware.

O trabalho de [García-Martín et al. 2019] é uma revisão da literatura que aborda as metodologias existentes para medir o consumo energético dos algoritmos de AM e quais softwares estão disponíveis para esta tarefa. Os autores fazem um levantamento das ferramentas que fornecem valores de estimativa de energia, identificando quais componentes podem ser monitorados e qual o impacto (*overhead*) causado pela ferramenta durante a execução do algoritmo. Para os experimentos foram avaliados os algoritmos *Hoeffding Tree* e *Very Fast Decision Tree* quando executados nas arquiteturas ARM, GPU e X86. Os trabalhos presentes na literatura que realizam a tarefa de monitorar a energia consumida sobre a execução de um algoritmo, são classificados de acordo com a metodologia utilizada para a coleta. Para os autores, uma das razões em não existir muitos trabalhos que avaliem o consumo de energia de algoritmos de AM está na falta de familiaridade com as ferramentas de coleta. Além disso, os autores demonstram a dificuldade de coletar informações relevantes sobre o consumo energético dos algoritmos, sendo as GPUs uma das arquiteturas mais difíceis de se obter informação. Porém, o estudo é focado nas diferentes metodologias e ferramentas utilizadas para medir energia, e não na avaliação detalhada dos algoritmos de AM a fim de identificar os hotspots de energia dos mesmos.

Em [Abdelhafez et al. 2019] são avaliados o consumo de energia dos Algoritmos Genéticos (AGs) com implementações sequencial e paralela. Os autores identificam os hotspots dos AGs e avaliam como as diferentes configurações de hardware podem interferir no consumo de energia e no tempo de execução. Para a coleta de energia foi utilizada a ferramenta RAPL e os autores apontam que não existe somente um fator que pode influenciar no consumo energético e no tempo de execução, mas um somatório de fatores, tais como a complexidade e o tamanho do problema, e a configuração do hardware.

Ainda, com exceção dos trabalhos de [Strubell et al. 2019] e [Bernardo et al. 2020], não foram encontrados trabalhos que avaliam a emissão de CO_2e (CO_2 equivalente, pois computadores não emitem CO_2) para a execução dos algoritmos de AM. Em [Strubell et al. 2019] foi realizada uma avaliação do ciclo de vida de

modelos de IA, no qual os autores mencionam que os custos computacionais e ambientais do treinamento crescem proporcionais ao tamanho do modelo, e este custo é ainda maior quando ajuste de parâmetros nos algoritmos são realizados para aumentar a precisão final dos modelos. Em [Bernardo et al. 2020] é avaliado o impacto do treinamento de diferentes algoritmos de AM (RNAs, AD, Kmeans e Floresta Randômica Regressiva) no consumo energético e na emissão de CO_2e, quando executados em diferentes arquiteturas computacionais (ARM, GPU e X86). Porém, ainda que os trabalhos sejam muito relevantes, não são identificados os hotspots de energia nos algoritmos de AM avaliados.

3. Metodologia de Experimentos

Nesta seção são descritos os algoritmos de AM avaliados, os conjuntos de dados, as arquiteturas e as ferramentas utilizadas nos experimentos. Além disso, são apresentadas as etapas para o cálculo da emissão de CO_2e dos algoritmos de AM.

A metodologia utilizada para os experimentos consiste em: i) identificar as principais funções e subfunções dos algoritmos; ii) realizar o treinamento dos algoritmos de AM com diferentes conjuntos experimentais em duas arquiteturas distintas; iii) medir o desempenho e o consumo de energia de cada parte do código, identificando as funções que tem maior impacto sobre o desempenho e consumo de energia dos algoritmos (o que denominamos de hotspots); iii) analisar e comparar os resultados de desempenho (tempo de execução e consumo de energia) dos dois algoritmos e arquiteturas, buscando identificar as causas e possíveis melhorias.

3.1. Algoritmos de AM e Conjuntos de Dados

Para os experimentos foram utilizados os algoritmos de AD para as tarefas de classificação (versão C4.5 [1]) e regressão (versão CART[2]), ambos implementados em Python.

Para o treinamento dos algoritmos foram utilizados conjuntos de dados sintéticos gerados com a ferramenta MOA [Bifet et al. 2010], pois, como já mencionado, o objetivo é a avaliação do desempenho computacional dos algoritmos e não a tarefa de aprendizado em si. Foram gerados 3 conjuntos de dados, detalhados na Tabela 1, os quais possuem, respectivamente, 5, 10 e 25 mil exemplos. Ainda, para cada conjunto de dados foram feitas 3 variações na quantidade de atributos: 5 atributos (4 numéricos e 1 nominal), 10 atributos (9 numéricos e 1 nominal) e 21 atributos (20 numéricos e 1 nominal).

Conjunto	Exemplos	Atributos		
		Total	Numérico	Nominal
1	5 mil	5	4	1
		10	9	1
		21	20	1
2	10 mil	5	4	1
		10	9	1
		21	20	1
3	25 mil	5	4	1
		10	9	1
		21	20	1

Tabela 1. Detalhamento dos conjuntos de dados utilizados nos experimentos.

[1]Disponível em: https://github.com/barisesmer/C4.5

[2]Disponível em: https://github.com/SebastianMantey/Decision-Tree-from-Scratch

3.2. Arquiteturas e Ferramentas

Para identificar as principais funções e subfunções dos algoritmos, detalhando cada parte do código, foram utilizadas as ferramentas *Cprofile* [3], a qual gera um arquivo com a extensão .cprof contendo toda a estrutura de funções do código, e a *KCacheGrind* [4], com a qual é possível a visualização do arquivo .cprof.

Após a identificação das principais funções e subfunções, os algoritmos foram treinados com os diferentes conjuntos de dados, utilizando as arquiteturas detalhadas na Tabela 2. A ARQ 1 refere-se a uma arquitetura x86 de alto desempenho com um processador Intel de Oitava Geração e a ARQ2 à arquitetura ARM de baixo consumo. Essas diferentes arquiteturas são utilizadas na avaliação do consumo de energia.

	ARQ1	**ARQ2**
CPU	Intel Core i7 8700 @ 3.2GHz	NVIDIA Carmel ARM v8.2 @2.3GHz
CPU Cores	6C 12T	8C
RAM	16GB	16GB
OS	Ubuntu 18.04	Ubuntu 18.04
Kernel	5.3	4.9

Tabela 2. Arquiteturas utilizadas para a execução dos experimentos.

Para coletar o consumo energético foi utilizada a ferramenta pyRAPL. Com esta ferramenta faz-se necessário a inserção de diretivas de instrumentação no código para que seja possível a coleta do consumo de energia dos trechos de interesse. Sabe-se que este tipo de abordagem normalmente ocasiona um *overhead* (sobrecarga que a ferramenta causa durante o monitoramento do código) significativo no resultado. Portanto, para avaliar o *overhead* da pyRAPL foram realizados experimentos analisando o tempo de execução do algoritmo de AD para a tarefa de classificação com e sem a utilização da ferramenta pyRAPL. Os resultados apresentados na Seção 5 são os valores totais com o overhead, que varia de 0,28% a aproximadamente 15%, de acordo com o tamanho do conjunto de dados e a quantidade de atributos utilizados para o treinamento do algoritmo.

3.3. Cálculo da emissão de CO_2e

Para o cálculo da emissão de CO_2e foram utilizadas as Equações 1 e 2, baseadas no trabalho de [Strubell et al. 2019]. Na Equação 1 é calculada a Energia Total (ET) em KWh, onde *PUE (Power Usage Effectiveness* [Avelar et al. 2012]) representa a eficiência energética, t, o tempo em horas, *P_CPU* , *P_GPU* e *P_ MEM*, respectivamente, potência em Watts da CPU, GPU e memória RAM. Essas métricas foram coletadas com a ferramenta perf [5]. Na Equação 2 é calculada a emissão de CO_2e, os parâmetros utilizados são: a constante de CO_2e do país, a qual de acordo com [Miranda 2012] é de 0,125 Kg/KWh para o Brasil, e a ET.

$$ET = \frac{PUE \times t \times (P\_CPU + P\_GPU + P\_MEM)}{1000} \quad (1)$$

$$CO2e = 0,125 \times ET \quad (2)$$

[3]https://docs.python.org/3/library/profile.html

[4]http://kcachegrind.sourceforge.net/html/Home.html

[5]Os detalhes de utilização da ferramenta perf estão disponíveis em [Klôh et al. 2019]

4. Estudo dos Algoritmos de Árvore de Decisão

Seguindo a metodologia descrita na Seção 3, o primeiro passo foi a identificação das principais funções dos algoritmos.

As AD são algoritmos de AM supervisionados, os quais podem ser utilizados para as tarefas de classificação ou regressão. No AM supervisionado o conjunto de dados utilizado para o treinamento do algoritmo tem a forma (x, y), onde x representa o vetor de atributos e y o rótulo ou a classe a ser previsto. O objetivo é encontrar uma função para predizer y, dado x ($y = f(x)$). Na AD clássica, como o C4.5 [Quinlan 1996] o objetivo é a classificação, ou seja, y representa um valor categórico. Para a árvore de regressão, como o CART [Breiman et al. 1984], o objetivo é predizer um y com valor numérico.

Nos algoritmos de AD f é representada no formato de uma estrutura do tipo árvore, onde a raiz fica no topo da árvore. Cada nó da árvore representa um teste sobre um atributo, e as ramificações desse nó, conhecidas como ramos, representam um teste com os valores do atributo. As folhas representam os rótulos y. Para construir o modelo de classificação o algoritmo usa uma abordagem do tipo dividir para conquistar onde o atributo com maior ganho de informação é usado na raiz da árvore (usando inicialmente todos os atributos e todo o conjunto de dados na avaliação) e o processo é repetido recursivamente para cada nó (com os atributos e o conjunto de dados correspondente) até que a informação tenha a menor entropia, ou seja, até que cada nó cubra o maior número de exemplos de uma classe. Assim o nó é uma folha e é rotulado com a classe apropriada. Para o algoritmo de regressão a principal diferença está na árvore ter uma saída numérica em suas folhas, além do cálculo para a escolha dos nós ser realizado através do Erro Quadrático Médio, e não sobre o ganho de informação como na árvore de classificação.

Como mencionado, foram avaliados dois algoritmos de Árvore de Decisão (AD), para classificação e regressão. Apesar dos algoritmos serem muito similares, o objetivo é analisar se as funções de avaliação de atributos que diferenciam os algoritmos e a necessidade de transformação dos valores categóricos no caso da regressão tem impacto no desempenho e consumo de energia dos algoritmos. Os algoritmos foram executados utilizando o conjunto experimental e as ferramentas descritas na Seção 3.

Os resultados obtidos são apresentados nas Figuras 1 (a) e 1 (b), onde são apresentadas as funções e subfunções dos algoritmos para classificação (C4.5) e regressão (CART), respectivamente.

Para a tarefa de classificação (Figura 1(a)) a função *C4.5* carrega os conjuntos de dados que são utilizados pela AD. Para que o algoritmo execute são necessários 2 arquivos: um contendo o nome dos atributos e o outro com os dados. A função *FetchData* recebe os arquivos da função *C4.5* e os agrupa. A principal função do algoritmo é a *GenerateTree*, na qual é realizada uma chamada à subfunção *RecursiveGenerateTree*, onde é responsável chamar as demais funções do algoritmo para a criação da árvore. A subfunção *RecursiveGenerateTree* recebe os resultados das subfunções *SplitAttribute* e *getMajClass*, as quais são responsáveis, respectivamente, por gerar ramos de decisão e a classificação dos exemplos (folhas). Para gerar os galhos e folhas da árvore, a subfunção *SplitAttribute* utiliza a subfunção *Gain*, na qual é realizado o cálculo do ganho de informação para fazer a escolha do atributo mais significativo. Para essa escolha é utilizada a subfunção *Entropy*, a qual realiza o cálculo da entropia, ou seja, o quanto um

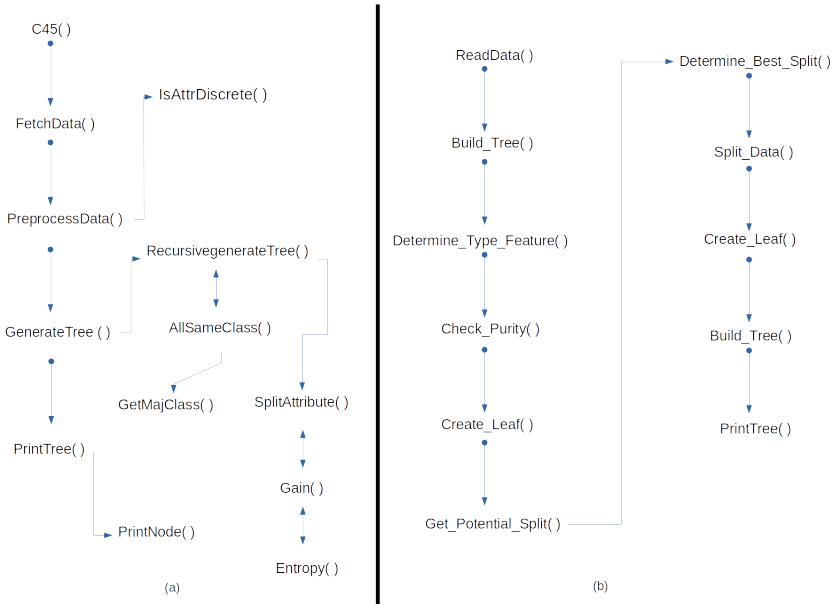

Figura 1. Funções e subfunções dos algoritmos de AD utilizados para as tarefas de classificação (a) e regressão(b).

exemplo consegue classificar os demais exemplos.

Para a tarefa de regressão (Figura 1(b)) o conjunto de dados é carregado através da função *ReadData*. Na função *build_tree* é realizada toda a criação da árvore. Para isso, essa função faz verificações à outras funções implementadas no algoritmo, como por exemplo, na *determine_type_of_Feature()*, a qual transforma atributos categóricos em numéricos, uma vez que atributos categóricos não são suportados pelo algoritmo de regressão. A função *check_purity* analisa se todos os exemplos são da mesma classe para avaliar se é necessário encontrar um novo nó para a árvore. Caso não seja necessário, gera a folha de decisão final. Caso contrário, verifica qual é o próximo nó da árvore através da chamada às subfunções *get_potential_split* e *split_data*. A função *print_tree* realiza a saída da algoritmo e mostra para o usuário a árvore resultante com suas respectivas divisões.

5. Experimentos e Resultados

Nesta seção são apresentados os experimentos e resultados obtidos na avaliação do consumo de energia dos algoritmos de AD, quando utilizados para as tarefas de classificação e regressão. Além disso, é apresentado a emissão de CO_2e atrelada à execução dos algoritmos em diferentes arquiteturas.

Com a metodologia adotada foram identificadas as funções com o maior impacto no desempenho (tempo de execução) e, consequentemente, no consumo energético dos algoritmos (hotspots). Na Tabela 3 são apresentadas as funções que caracterizam os hotspots de energia identificados para os algoritmos de classificação (C4.5) e regressão (CART). Para a tarefa de classificação as funções *preprocessData* e *generateTree* são as que tem maior impacto sobre o consumo de energia, sendo a *generateTree* a mais expressiva, consumindo 103562,09 Joules, quando utilizado o Conjunto 3 com 21 atributos (destacado em negrito na Tabela 3). Para a tarefa de regressão as funções *build_tree* e *determine_best_split* são as que mais consomem energia. Dentre essas duas funções a

determine_best_split é a mais expressiva, consumindo 1863,17 Joules, quando utilizado o Conjunto 3 com 21 atributos. Como pode ser observado o maior consumo energético está atrelado aos diferentes conjuntos de dados com 21 atributos. Para uma melhor análise sobre estes conjuntos com 21 atributos, nas Figuras 2 e 3 são apresentados o consumo de energia das principais funções dos algoritmos C4.5 e CART, respectivamente.

Funções	**Energia (Joules)**								
	5 atributos			**10 atributos**			**21 atributos**		
	Conj 1	**Conj 2**	**Conj 3**	**Conj 1**	**Conj 2**	**Conj 3**	**Conj 1**	**Conj 2**	**Conj 3**
Algoritmo de classificação (C4.5)									
generateTree	1906,2	7698,4	51436,2	5420,9	21880,9	145007,6	**13330,7**	**52996,8**	**103562,1**
preprocessData	0,3	0,6	1,4	0,6	1,3	3,2	**1,6**	**3**	**7,9**
Algoritmo de regressão (CART)									
build_tree	27,7	66,1	157,5	67,9	146,4	382,5	**247,9**	**613,0**	**1512,7**
determine_best_split	31	73,7	178,8	77,8	170,5	447,9	**295,2**	**729,9**	**1863,2**

Tabela 3. Consumo de energia das principais funções dos algoritmos C4.5 e CART quando executados na ARQ1 com os diferentes conjuntos de dados e quantidades de atributos.

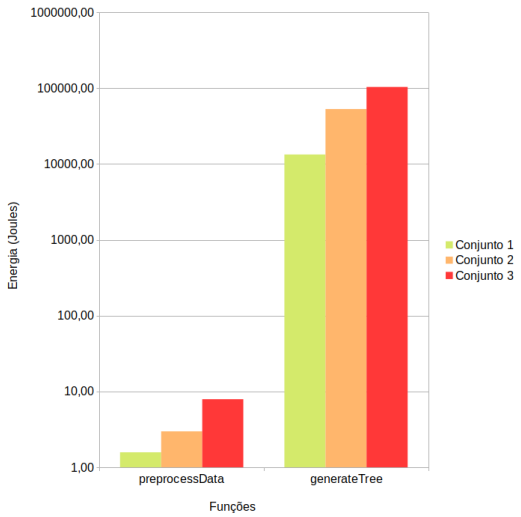

Figura 2. Energia das principais funções do algoritmo C4.5 quando executado com os diferentes conjuntos de dados de 21 atributos.

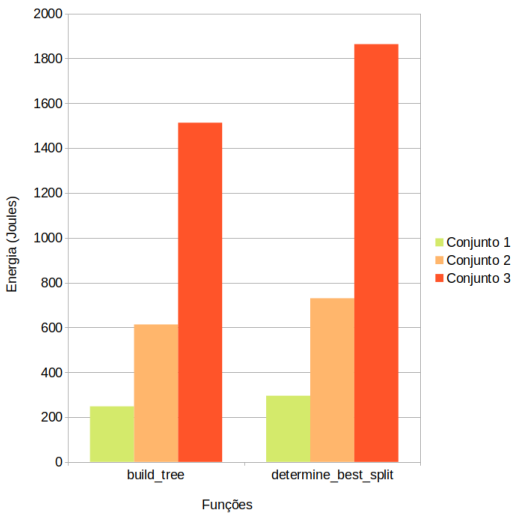

Figura 3. Energia das principais funções do algoritmo CART quando executado com os diferentes conjuntos de dados de 21 atributos.

Na Tabela 4 são apresentados o consumo de energia e tempo de execução para o treinamento e predição dos algoritmos C4.5 e CART, quando executados nas arquiteturas ARQ1 e ARQ2, com os diferentes conjuntos de dados detalhados na Tabela 1. Na ARQ1 ao duplicar a quantidade de atributos de 5 para 10 em todos os conjuntos, o consumo de energia aumenta em, aproximadamente, 4 vezes, enquanto na ARQ2 o aumento foi de aproximadamente, 5,5 vezes. Quando na ARQ1 a quantidade de atributos passa de 10 para 21, o aumento é ainda maior no consumo de energia. Os Conjuntos 1 e 2 com 21 atributos tiveram um aumento de, aproximadamente, 7 vezes em relação aos Conjuntos 1 e 2 com 10 atributos. Porém, o Conjunto 3 de 21 atributos consumiu, aproximadamente, o dobro de energia em relação ao Conjunto 3 de 10 atributos. Este comportamento pode

ser observado na Figura 4, na qual os Conjuntos 1, 2 e 3 possuem, respectivamente, 5, 10 e 25 mil exemplos para as diferentes quantidades de atributos (5, 10 e 21). Além disso, em todos os conjuntos de dados ao aumentar a quantidade de atributos, há um aumento no tempo de execução de, aproximadamente, 3 vezes (Tabela 4). Sobre o tempo de execução foram observados dois diferentes resultados na ARQ1: i) O conjunto 2 tem o dobro de exemplos do conjunto 1 e o tempo gasto para a execução do algoritmo com este conjunto foi, aproximadamente, 4 vezes maior que o Conjunto 1, para todos as quantidades de atributos (5, 10 e 21); ii) O Conjunto 3 possui um tempo de execução de, aproximadamente, 7 vezes maior que o Conjunto 2, para todas as diferentes quantidades de atributos. Para a ARQ2 o tempo de execução ficou em aproximadamente, 5,5 vezes. Vale salientar que na ARQ2 por limitação de armazenamento (apenas 32GB de armazenamento interno) não foi possível executar a coleta do conjunto 3.

Quantidade de atributos	Energia Total (Joules)			Tempo (Segundos)		
	Conjunto 1	Conjunto 2	Conjunto 3	Conjunto 1	Conjunto 2	Conjunto 3
Algoritmo de classificação (C4.5)						
ARQ1						
5	1906,55	5421,68	13332,62	73,82	298,57	2054,01
10	7699,13	21882,51	53000,53	208,97	861,87	5919
21	51438,06	145011,62	103571,75	516,38	2135,70	15412,67
ARQ2						
5	3631,62	18459	-	457	2315,09	-
10	17044,26	88528,61	-	2164,62	11168,67	-
21	87938,42	488821,67	-	11199	61895	-
Algoritmo de regressão (CART)						
ARQ1						
5	58,86	146	543,66	12,68	29,91	103,46
10	140,12	317,44	1343,98	29,26	62,41	242,09
21	336,79	831,39	3378	67,50	160,49	604,79
ARQ2						
5	110,38	276,06	1361,79	12,79	33,79	160,21
10	242,01	645,32	3704,40	28,88	78,22	435,30
21	596,74	1975,34	11599,79	71,81	238,28	1367,90

Tabela 4. Consumo de energia e tempo de execução para o treinamento dos algoritmos C4.5 e CART executados nas arquiteturas ARQ1 e ARQ2 com os diferentes conjuntos de dados e quantidades de atributos.

Na Tabela 5 é apresentada a quantidade de CO_2e (kg/KWh) emitido durante o processo de treinamento e predição do algoritmo C4.5, quando executado nas arquiteturas ARQ1 e ARQ2. Dentre as duas arquiteturas a ARQ1 é a que consome mais energia. Porém, este comportamento não foi observado no experimento, no qual o tempo de execução do algoritmo na ARQ2 varia de 5 à 29 vezes (quanto maior a quantidade de atributos e exemplos maior é o tempo) em relação à ARQ1. Para efeito comparativo, o carro popular mais vendido no Brasil, segundo o Inmetro, emite 0,100 Kg/km de CO_2 (motor 1.0 - 8 V, gasolina). Para que a arquitetura utilizada neste trabalho possa emitir 0,100 Kg de CO_2e são necessárias 57,3 horas de execução contínua do algoritmo AD, na ARQ2. Apesar dos valores encontrados neste experimento não serem tão expressivos quanto os valores de [Strubell et al. 2019], é importante observar que o algoritmo de AD foi apenas treinado, sem ajuste de parâmetros para se chegar a um modelo com precisão aceitável. Normalmente são necessários vários ajustes de parâmetros e muitas execuções de treinamento. Além disso, o conjunto de treinamento e predição utilizado neste trabalho para o algoritmo AD é pequeno (10000 exemplos), por isso o tempo de treinamento e predição é baixo (61895s), o que para um problema real (com conjuntos de dados bem maiores)

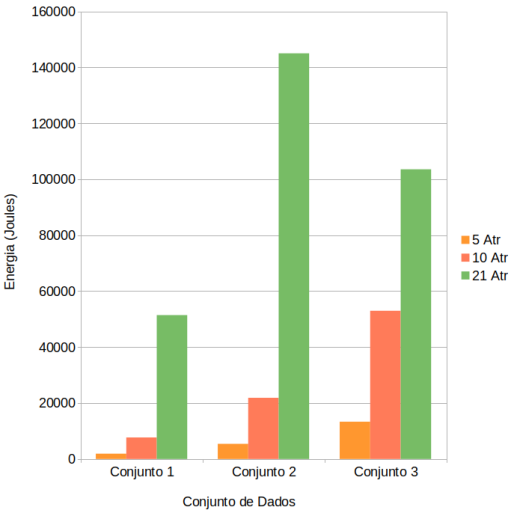

Figura 4. Consumo de energia do algoritmo C4.5 quando executados com diferentes conjuntos de dados e quantidades de atributos.

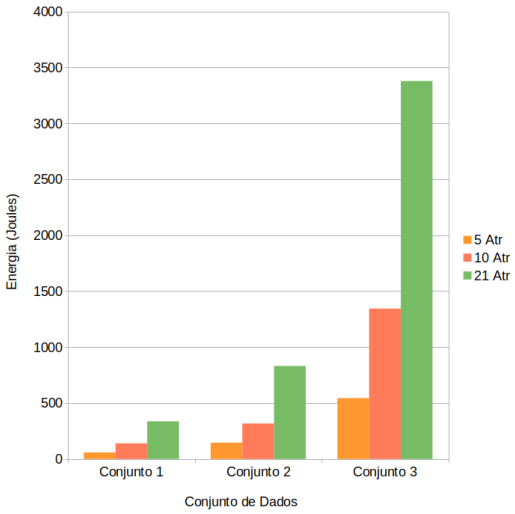

Figura 5. Consumo de energia do algoritmo CART quando executados com diferentes conjuntos de dados e quantidades de atributos.

pode facilmente chegar a mais de 100h. Outro fator que impacta neste resultado é a constante de CO2e utilizada no cálculo, a qual muda de acordo com o país onde é executado o treinamento. Neste trabalho é utiliza a constante para a matriz energética brasileira que é baseada em fontes mais limpas, como hidrelétricas, enquanto em [Strubell et al. 2019] é usada a constante para os EUA baseada em termoelétricas.

Qtd Atributos	CO2e (Kg/KWh) ARQ1			CO2e (Kg/KWh) ARQ2	
	Conjunto 1	Conjunto 2	Conjunto 3	Conjunto 1	Conjunto 2
5	0,0001	0,0003	0,0008	0 ,0002	0,0011
10	0,0004	0,0013	0,0031	0,0010	0,0051
21	0,0030	0,0084	0,0060	0,0051	0,0283

Tabela 5. Emissão de CO_2e para o algoritmo C4.5 quando executado nas arquiteturas ARQ1 e ARQ2 utilizando diferentes conjuntos de dados.

6. Considerações Finais e Trabalhos Futuros

Neste trabalho foi adotada uma metodologia para identificar as funções de algoritmos de AM que tenham maior impacto sobre o desempenho, e consequentemente, sobre o consumo de energia. Foram analisados dois algoritmos de AD, escolhidos justamente pela sua simplicidade. Os resultados obtidos, permitiram identificar os principais hotspots dos algoritmos C4.5 e CART, os quais são explicados pela característica, em comum para ambos os algoritmos, do processo de construção da árvore. As funções responsáveis pela escolha dos atributos mais significativos (nós de decisão) faz com que seja necessário testar todos os exemplos e todos os atributos para a construção da árvore. Assim, o consumo de energia e o tempo de execução aumentam com a adição de mais exemplos e também com uma maior dimensão de atributos. Além disso, a recursividade é um processo custoso para um algoritmo, sendo para a AD, a técnica que mais consome recurso computacional

e energia. Assim pensar numa maneira de substituir a recursividade por outra técnica provavelmente irá melhorar o desempenho e reduzir o consumo de energia.

Esse tipo de avaliação para os algoritmos de AM se mostrou satisfatória, permitindo a identificação das principais funções dos algoritmos, dos fatores que impactam no consumo de energia e onde seriam necessárias as otimizações. Além disso, é possível considerar o impacto das características das arquiteturas de acordo com os requisitos computacionais dos algoritmos. Esse tipo de abordagem para os algoritmos de AM é pouco utilizado, pois geralmente são avaliados e desenvolvidos somente em termos de desempenho preditivo. Porém, o impacto ecológico da IA e dos algoritmos de AM precisa ser considerado nas implementações para alcançarmos soluções mais viáveis ecologicamente.

Foi observado que ambos os códigos não são paralelizados para executarem em mais de um núcleo do processador, o que será estudado para trabalhos futuros. Além disso, serão avaliados outros algoritmos de AM e arquiteturas do tipo GPU.

Agradecimentos

Os autores agradecem ao LNCC, CNPq e a FAPERJ pelo apoio financeiro.

Referências

Abdelhafez, A., Alba, E., and Luque, G. (2019). A component-based study of energy consumption for sequential and parallel genetic algorithms. *The Journal of Supercomputing*, 75(10):6194–6219.

Avelar, V., Azevedo, D., French, A., and Power, E. N. (2012). Pue: a comprehensive examination of the metric.

Bernardo, F., Silva, G., Gritz, M., Ferro, M., and Schulze, B. (2020). Avaliação do consumo de energia e o impacto da emissão de co2 para algoritmos de inteligência artificial. In *Anais do XIV Brazilian e-Science Workshop*, pages 81–88, Porto Alegre, RS, Brasil. SBC.

Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. (2010). Moa: Massive online analysis http://sourceforge. net/projects/moa-datastream. *Journal of Machine Learning Research (JMLR)*.

Breiman, L., Friedman, J., Stone, C. J., and Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

Ferreira, A. R. et al. (2017). Um modelo analítico para estimar o consumo de energia de sistemas multi-camadas no nível de transação. Master's thesis, Universidade Federal de Goiás.

Flach, P. (2012). *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press.

Garcia-Martin, E., Lavesson, N., and Grahn, H. (2017). Identification of energy hotspots: A case study of the very fast decision tree. In *International Conference on Green, Pervasive, and Cloud Computing*, pages 267–281. Springer.

García-Martín, E., Rodrigues, C. F., Riley, G., and Grahn, H. (2019). Estimation of energy consumption in machine learning. *Parallel and Distributed Computing*, 134:75–88.

Klôh, V., Gritz, M., Schulze, B., and Ferro, M. (2019). Towards an autonomous framework for hpc optimization: Using machine learning for energy and performance modeling. In *Anais do XX Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 438–445, Porto Alegre, RS, Brasil. SBC.

Li, D., Chen, X., Becchi, M., and Zong, Z. (2016). Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus. In *2016 IEEE International Conferences on Big Data and Cloud Computing, Social Computing and Networking, Sustainable Computing and Communications*, pages 477–484. IEEE.

Malakar, P., Balaprakash, P., Vishwanath, V., Morozov, V., and Kumaran, K. (2018). Benchmarking machine learning methods for performance modeling of scientific applications. In *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 33–44.

Miranda, M. M. d. (2012). *Fator de emissão de gases de efeito estufa da geração de energia elétrica no Brasil: implicações da aplicação da Avaliação do Ciclo de Vida*. PhD thesis, Universidade de São Paulo.

Noureddine, A., Rouvoy, R., and Seinturier, L. (2015). Monitoring energy hotspots in software. *Automated Software Engineering*, 22(3):291–332.

Olson, R. S., La Cava, W., Orzechowski, P., Urbanowicz, R. J., and Moore, J. H. (2017). Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData mining*, 10(1):1–13.

Quinlan, J. R. (1996). Improved use of continuous attributes in c4.5. *Journal of artificial intelligence research*, 4:77–90.

Serpa, M. S., Krause, A. M., Cruz, E. H., Navaux, P. O. A., Pasin, M., and Felber, P. (2018). Optimizing machine learning algorithms on multi-core and many-core architectures using thread and data mapping. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 329–333. IEEE.

Siegmund, N., Grebhahn, A., Apel, S., and Kästner, C. (2015). Performance-influence models for highly configurable systems. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 284–294. ACM.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in nlp. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650.

Villani, C., Bonnet, Y., schoenauer, m., berthet, c., levin, f., cornut, a. c., and Rondepierre, B. (2018). *For a meaningful artificial intelligence: towards a french and european strategy*. Conseil national du numérique.

Wu, X., Taylor, V., Cook, J., and Mucci, P. J. (2016). Using performance-power modeling to improve energy efficiency of hpc applications. *Computer*, 49(10):20–29.

Yang, T.-J., Chen, Y.-H., and Sze, V. (2017). Designing energy-efficient convolutional neural networks using energy-aware pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5687–5695.