Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um estudo sobre algoritmos para redução de características no contexto de ataques baseados em botnets

Caroline Ferreira Pinto
Gustavo Macedo de Carvalho

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Marcelo Antonio Marotta

Brasília
2023

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Um estudo sobre algoritmos para redução de características no contexto de ataques baseados em botnets

Caroline Ferreira Pinto
Gustavo Macedo de Carvalho

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Marcelo Antonio Marotta (Orientador)
CIC/UnB

Prof. Dr. Marcos Fagundes Caetano
CIC/UnB

Prof. Dr. João Gondim
CIC/UnB

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 25 de janeiro de 2023

Agradecimentos

Em primeiro lugar, agradeço a Deus, que fez com que meus objetivos fossem alcançados e os obstáculos fossem ultrapassados durante todos os meus anos de estudos e permitiu que eu tivesse determinação para não desanimar durante a realização deste trabalho.

À minha família, por todo o apoio e ajuda nos momentos mais difíceis, tanto ao longo do curso como do trabalho, e por sempre estarem ao meu lado me dando incentivo e compreendendo minha ausência ao me dedicar à realização das atividades do curso e deste trabalho.

Ao professor Marotta, por ter sido meu orientador e ter desempenhado tal função com dedicação e paciência.

Aos professores que tive ao longo do curso, pelas correções e ensinamentos que me permitiram apresentar um melhor desempenho no meu processo de formação profissional ao longo do curso.

Aos meus colegas de curso, pela convivência durante os últimos anos, pelo companheirismo, pelo apoio e por compartilharem comigo momentos de aprendizado ao longo do curso.

Aos demais que contribuíram, de alguma forma, para a realização deste trabalho, direta ou indiretamente, auxiliando ao longo desse percurso e no desenvolvimento deste trabalho.

Por fim, à Universidade de Brasília (UnB), essencial no meu processo de formação profissional, pelo fornecimento de materiais e oportunidades e por tudo o que aprendi ao longo dos anos do curso, possibilitando assim meu desenvolvimento acadêmico e auxiliando no meu processo de aprendizado .

Dedicatória

Dedico este trabalho aos meus pais, por todos os ensinamentos, conselhos, por todo apoio, amor e paciência. A eles que me ensinaram o valor da educação, me mostraram o quanto sou capaz e sempre acreditaram em mim. Meus maiores incentivadores, meus maiores exemplos.

Resumo

O uso de dispositivos alimentados por Internet das Coisas se expandiu drasticamente durante os últimos anos, assim como a variedade de protocolos e recursos computacionais. Contudo, além das inúmeras vantagens geradas por esse avanço tecnológico, existem riscos inerentes que ameaçam principalmente a segurança desses aparelhos inteligentes. Uma dessas ameaças é a *botnet*, utilizando um dos ataques mais comuns, o DDoS, que muda constantemente seu formato de ataque. Dada a natureza mutante das botnets, algoritmos de otimização metaheurísticos, inspirados na natureza, são vistos como uma possível solução, visto que possibilitam uma maneira eficiente para encontrar soluções para problemas complexos do mundo real utilizando apenas informações de fluxo de rede e adaptando-se às mudanças sendo atualizados por treinamentos frequentes. Entretanto, o tempo de treinamento desses algoritmos é dependente do número de características utilizadas durante o treino, levando a tempos de processamento incompatíveis com seu uso em cenários de processamento em tempo real. Dessa forma, mecanismos de redução de características devem ser explorados para reduzir a complexidade dos treinamentos capacitando o uso desses algoritmos em cenários reais, ou seja, com processamento em tempo real. Nesse trabalho, propõe-se uma comparação entre três algoritmos de redução de características para treinamento de detectores metaheuríticos, que operam em tempo real. Alguns desses algoritmos podem ser inspirados no comportamento dos animais, como é o caso dos que serão aplicados nesse trabalho, em específico, o algoritmo de redução do lobo cinzento (*Gray Wolf Optimizer* - GWO), da baleia (*Whale Optimization Algorithm* - WOA) e do mocego (*Bat Algorithm* - (BAT)). Esses algoritmos foram implementados a partir do *framework* EvoloPy-FS, que consiste em uma estrutura de otimização de código aberto codificada em Python, inspirada na natureza para a seleção de recursos. Propõe-se analisar o desempenho do modelo de Árvore de Decisão (*Decision Tree*) na avaliação das diferenças obtidas nas métricas de *f1-score*, acurácia, precisão e tempo de treinamento, aplicando os três algoritmos de otimização utilizados para redução de características (*features*) no contexto de detecção de *botnets*.

Palavras-chave: Algoritmo de otimização, Árvore de Decisão, Botnet, Internet das Coisas.

Abstract

The use of devices powered by the Internet of Things has expanded dramatically over the last few years, as has the variety of protocols and computing resources. However, in addition to the numerous advantages generated by this technological advance, there are inherent risks that mainly threaten the security of these smart devices. One such threat is *botnet*, using one of the most common attacks, DDoS, which constantly changes its attack format. Given the changing nature of botnets, metaheuristic, nature-inspired optimization algorithms are seen as a possible solution, as they provide an efficient way to find solutions to complex real-world problems using only network flow information and adapting to changing conditions. changes being updated by frequent training. However, the training time of these algorithms is dependent on the number of features used during training, leading to processing times that are incompatible with their use in real-time processing scenarios. Thus, feature reduction mechanisms should be explored to reduce the complexity of training, enabling the use of these algorithms in real scenarios, that is, with real-time processing. In this work, we propose a comparison between three feature reduction algorithms for training metaheuritic detectors, which operate in real time. Some of these algorithms may be inspired by the behavior of animals, as is the case of those that will be applied in this work, specifically, the gray wolf reduction algorithm (*Gray Wolf Optimizer* - GWO), the whale (*Whale Optimization Algorithm* - WOA) and bat (*Bat Algorithm* - (BAT)). These algorithms were implemented from *framework* EvoloPy-FS, which is an open source optimization framework coded in Python, inspired by nature for feature selection. It is proposed to analyze the performance of the Decision Tree model (*Decision Tree*) in the evaluation of the differences obtained in the metrics of *f1-score*, accuracy, precision and training time, applying the three optimization algorithms used for feature reduction (*features*) in the context of detecting *botnets*.

Keywords: Botnet, Decision Tree, Internet of Things, Optimization algorithm.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Ao longo dos últimos anos, com o advento de novas tecnologias e velocidades de conexão cada vez maiores, o uso de dispositivos conectados à Internet se expandiu drasticamente, assim como suas aplicações e funcionalidades, impactando diretamente na vida das pessoas. A interação desses dispositivos juntamente à sensores avançados criados a fim de facilitar atividades diárias, define uma das tendências tecnológicas da atualidade, a Internet das Coisas (IoT, *Internet of Things*).

Segundo a pesquisa do portal IoT Analytics [4], espera-se que ainda em 2022 o mercado de IoT apresente um total de 14,4 bilhões de conexões, 18% a mais do que o ano anterior, com expectativa de alcançar o número de 27 bilhões de dispositivos conectados até o ano de 2025. Apesar de sua capacidade computacional e de gerarem diversas vantagens e possibilidades, a IoT apresenta características como a baixa potência, baixa memória computacional e restrições de bateria, que tornam a tecnologia mais suscetível à ameaças, principalmente no que se refere à segurança dos dispositivos inteligentes. Em síntese, esses dispositivos não dispões de recursos para implementar mecanismos elaborados de segurança, como um *Host Intrusion Detection System* (HIDS). Dessa forma, ataques maliciosos devem ser prevenidos nas redes antes de chegar até os dispositivos, utilizando-se *Network-Intrusion Detection Systems* (NIDS).

Considerando a complexidade da manutenção da segurança decorrente da diversidade de ataques existentes, os NIDS necessitam identificar e realizar ações de contramedidas contra vários tipos diferentes de ataques. Entre os ataques mais danosos a uma rede IoT, destaca-se a *botnet*, que consiste em uma rede de computadores e dispositivos IoT infectados por um *software* malicioso (*malware*) e usados para fins maliciosos, sendo um dos mais comuns, com ataques distribuídos de negação de serviço (DDoS, *Distributed Denial of Service*), que muda constantemente seu formato de ataque. Dada a natureza mutante das botnets, algoritmos de otimização metaheurísticos, inspirados na natureza, são vistos como uma possível solução, visto que possibilitam uma maneira eficiente para encontrar

soluções para problemas complexos do mundo real utilizando apenas informações de fluxo de rede e adaptando-se às mudanças sendo atualizados por treinamentos frequentes [5]. Entretanto, o tempo de treinamento desses algoritmos é dependente do número de características utilizadas durante o treino, levando a tempos de processamento incompatíveis com seu uso em cenários de processamento em tempo real. Dessa forma, mecanismos de redução de características devem ser explorados para reduzir a complexidade dos treinamentos capacitando o uso desses algoritmos em cenários reais, ou seja, com processamento em tempo real.

Nesse trabalho, propõe-se uma comparação entre três algoritmos de redução de características para treinamento de detectores metaheurísticos para NIDS que operam em tempo real. Em específico, todos os algoritmos de redução comparados nesse trabalho são bioinspirados, sendo eles: do lobo cinzento (*Gray Wolf Optimizer* - GWO) [6], da baleia (*Whale Optimization Algorithm* - WOA) [3] e do morcego (*Bat Algorithm* - (BAT)) [7]. Esses algoritmos foram implementados a partir do *framework* EvoloPy-FS, que consiste em uma estrutura de otimização de código aberto codificada em Python, inspirada na natureza para a seleção de recursos. Propõe-se analisar o desempenho do modelo de Árvore de Decisão (*Decision Tree*) na avaliação das diferenças obtidas nas métricas de *f1-score*, acurácia, precisão e tempo de treinamento, aplicando os três algoritmos de otimização utilizados para redução de características (*features*) no contexto de detecção de *botnets*.

1.1 Organização da monografia

Nesta seção será detalhada a organização desta monografia. O Capítulo 2 apresenta os principais trabalhos relacionados à questão de detecção de *botnets* utilizados como base para o trabalho. O Capítulo 3 descreve a fundamentação teórica, onde serão apresentados conceitos relevantes utilizados para o desenvolvimento do trabalho, tais como o conceito de *Botnets*, a definição dos algoritmos de otimização desenvolvidos, das métricas aplicadas e dos conjuntos de dados utilizados, entre outros conceitos cruciais. O Capítulo 4 descreve a metodologia utilizada para a elaboração do trabalho apresentado, sendo possível entender melhor cada conjunto de dados utilizado e como foram aplicados na avaliação das métricas dos algoritmos de otimização escolhidos. É apresentada também a ferramenta de manipulação de dados dos conjuntos de dados, *CICFlowMeter*. O Capítulo 5 apresenta os resultados obtidos considerando as métricas selecionadas e detalha os valores resultantes para cada um dos conjuntos de dados utilizados e ter um comparativo dos resultados gerados para cada um dos algoritmos de otimização definidos. Por fim, o Capítulo 6 conclui este trabalho com um resumo do trabalho apresentado e discussões sobre as contribuições realizadas.

Capítulo 2

Trabalhos Relacionados

Os dispositivos IoT são cada vez mais utilizados em diversos contextos, o que torna sua segurança um desafio importante. O surgimento de *botnets* é um dos maiores riscos para a segurança desses dispositivos. Diante disso, vários modelos de ML têm sido propostos para detectar essas ameaças. No artigo de Dong et al. [8], foi proposto o uso de um modelo de Máquina de Aprendizado Extremo (ELM, *Extreme Machine Learning*) chamado BotDetector para a detecção de *botnets* sem precisar processar os dados. Isso reduz o impacto na performance dos dispositivos IoT. O BotDetector identificou *botnets* com uma acurácia de 98,6% e reduziu o tempo de treinamento e o consumo de recursos.

Já no trabalho de Wazzan et al. [9], uma estrutura com foco na detecção de *botnets* IoT no estágio inicial foi desenvolvida. A proposta é detectar o recrutamento dos dispositivos para uma *botnet*, que apresenta características distintas. Para isso, desenvolveram o método eficaz de detecção Cross CNN-LSTM, que consiste em um modelo de fusão de aprendizagem profunda de uma rede neural convolucional (CNN, *Convolutional Neural Network*) e memória de longo prazo (LSTM, *Long Short-Term Memory*). O modelo atingiu 99,6% de acurácia em um conjunto de dados realista.

Nguyen et al. [10], propuseram um modelo de ML colaborativo para automatizar a detecção precoce de *botnets* em IoT com base em diversos recursos. Esse modelo de séries de tempo integral visa propor melhores soluções de resposta e mitigar os danos dos possíveis ataques. Outra solução de ML focada em detecção precoce de *botnet* foi apresentada por Malik et al. [11], baseada em classificador de uma classe de K-vizinhos mais próximos (KNN, *k-nearest neighbors*), que permite detectar *botnets* IoT no estágio inicial com alta acurácia. Esse modelo é considerado leve e eficiente já que seleciona os melhores recursos aproveitando métodos conhecidos. Essa solução foi capaz de detectar *botnet* com uma acurácia de aproximadamente 98.3% para diferentes conjuntos de dados de *botnet* IoT.

Um modelo de seleção de recursos foi desenvolvido por Khurma et al. [12]. Essa

solução utiliza a hibridização dos algoritmos *Salp Swarm Algorithm* (SSA) e *Ant Lion Optimization* (ALO), o SSA-ALO, para detectar ataques de *botnet*, aumentando assim a variedade de soluções e impedindo a convergência precoce. O modelo apresentou altas taxas de verdadeiros positivos na detecção de *botnets* no conjunto de dados N-BaIoT, atingindo 99.9%. Os mesmos autores propuseram o desenvolvimento de uma estrutura de otimização de código em Python, o EvoloPy-FS [13], que compreende diversos algoritmos de inteligência inspirados na natureza (SI, *Swarm Intelligence*). No cenário do processo de mineração de dados, onde a seleção de recursos (FS, *Feature Selection*) é um estágio crítico essencial, é necessário estar sempre em busca de construir bibliotecas e estruturas a fim de facilitar esse processo. Além disso, os autores afirmam que o desenvolvimento de ferramentas específicas com foco em FS tem despertado o interesse de diversos pesquisadores, pois além de acelerar e automatizar os processos de FS, também ameniza o esforço da implementação e minimiza as falhas de codificação, tornando as aplicações ainda mais descomplicadas para seus usuários. Com isso, concluem que o EvoloPy-FS é uma estrutura simples e acessível que pode auxiliar os pesquisadores a resolverem problemas e observar os resultados de maneira ágil com menos esforço, mesmo não possuindo altos níveis de conhecimento em SI.

Em relação à hibridização de algoritmos, no artigo de Elghamrawy et al. [6], é proposta uma estrutura baseada em sistemas de lógica fuzzy (FS, *Fuzzy Systems*) chamada Takagi–Sugeno–Kang (TSK-FS), que visa gerar as regras *fuzzy* necessárias e definir seus parâmetros mais relevantes. Pelo fato do desenvolvimento de um sistema *fuzzy* ser considerado um dos problemas de otimização mais complexos, foi aplicado o algoritmo otimização *Genetic-Grey Wolf* (GGWO), modelo híbrido inspirado na natureza, a fim de otimizar a estrutura TSK-FS. Esse algoritmo híbrido é composto pelo algoritmo genético (GA) combinado com a otimização *Grey Wolf* (GWO). A partir disso, os autores concluem que o algoritmo GGWO apresentou grandes vantagens em comparação a outros algoritmos de otimização inspirados na natureza em relação ao tempo computacional.

Outro trabalho que obteve vantagens em relação à hibridização de algoritmos de otimização, é o de Balasubramanian et al. [14], no qual é apresentado um sistema de inferência *neuro-fuzzy* melhorado baseado no algoritmo híbrido *Grey Wolf-Bat* (HWBO) com o intuito de prever com precisão a esquizofrenia a partir de sinais eletroencefalograma (EEG) multicanal. A esquizofrenia consiste em um transtorno mental crônico que debilita a capacidade mental de um indivíduo (pensamentos, sentimentos, comportamento, etc). Esse transtorno possui sintomas como alucinações, delírios, fala sem coesão, entre outros que impossibilitam a realização de atividades da rotina de uma pessoa, sendo consideravelmente necessário o diagnóstico assistido por computador. Com isso, os autores concluem que, a integração dos dados de EEG com a IoT torna possível a detecção de pacien-

tes com esquizofrenia com maior precisão, pois o modelo foi testado em dois conjuntos de dados EEG distintos utilizando validação cruzada (CV, *Cross-validation*), e obteve respectivamente, uma precisão de 97,8% e 98,5%.

O trabalho de Gharehchopogh et al. [15] propõe um sistema de detecção de *botnet* em IoT aplicando a técnica de seleção de recursos utilizando como base os algoritmos *Slime Mold* (SMA) e de *Salp Swarm* (SSA). Muitos sistemas de detecção de intrusão (IDS) não são capazes o suficiente para detectar ataques, apesar de fornecerem diversas soluções promissoras a fim de lidar com ataques de *botnet*. A partir disso, os autores apresentam um algoritmo prático com base na seleção de recursos, cujo desempenho é otimizado utilizando a Teoria do Caos. Com isso, concluíram que o algoritmo possui resultados promissores, com desempenho significativo e alta capacidade de detecção de *botnets* em IoT, atingindo uma baixa taxa de erro.

Por fim, no artigo de Abdulsattar et al. [16], é apresentada uma abordagem de aprendizado profundo que utiliza um algoritmo híbrido de otimização de *Shark Smell* e *Bear Smell* que permite pesquisar diferentes soluções de características nas regiões do espaço de busca a fim de garantir uma melhor solução. Recentemente, as redes *Flying Ad Hoc Network (FANET)*, também chamadas de tecnologia de drones, atraíram grande atenção por conta de suas implementações fundamentais, como nos sistemas presentes nos espaços aéreos civis. Com isso, os autores propuseram uma análise para detectar ameaças de *botnet* na FANET. Para isso, é utilizado um classificador de autoencoder convolucional dilatado para detectar e classificar as ameaças de segurança. A estrutura então proposta é chamada de *Hybrid Shark and Bear Smell Optimized Dilated Convolutional Autoencoder* (HSBSOpt _DCA). Os autores concluem que a abordagem proposta é promissora, já que, utilizando o conjunto de dados N-BaIoT foi possível atingir 97% de acurácia e 89% de precisão em relação aos modelos existentes.

Em resumo, esses trabalhos apresentam soluções para problemas relacionados à seleção de recursos em IoT, seja através da utilização de algoritmos híbridos, estruturas de otimização, sistemas de lógica *fuzzy*, ou ainda, sistemas de inferência *neuro-fuzzy*.

A tabela 2.1 mostra os trabalhos tomados como base para o desenvolvimento do projeto.

Tabela 2.1: Tabela de trabalhos relacionados.

Artigo	Tipo de solução	Dados de treinamento	Método de treinamento
[10]	Consiste em uma combinação de: *Sandbox component* (SC), *Preprocessing data component* (PPDC), *Data Normalization component* (DNC), *Feature Selection component* (FSC), *Machine learning Classifier component* (MLC), *Fusion component* (FC)	Usa combinação dados típicos de uma IoT *Botnet*, extraídos do V-Sandbox, incluindo fluxo de rede, chamada do sistema, informações de uso de recursos do sistema	Linear SVM
[6]	Estrutura baseada em sistemas de lógica *fuzzy* (TSK-FS), otimizada por meio da hibridização do algoritmo genético (GA) e da otimização *Grey Wolf* (GWO)	3 conjuntos de dados retirados da UC Irvine Machine Learning Repository: Abalone, Aerofólio e Power-Plant	Combinação das operações de *crossover* e mutação do AG com a exploração e fases de exploração do GWO
[8]	*Extreme Learning Machine*(ELM)	*IoT Gateway traffic*	Algoritmo de treinamento de rede neural que modifica a quantidade de nós e a estrutura do vetor de entrada a cada iteração
[16]	algoritmo de otimização Hybrid Shark and Bear Smell (HSBSOA) e classificador de *autoencoder* convolucional dilatado	conjunto de dados N-BaIoT	método de codificação *one-hot*
[9]	Modelo híbrido que consiste em diferentes camadas: uma camada de entrada, camada CNN, camada LSTM, camada plana, camada densa e camada de saída	MedBIoT *dataset*	KNN, *Decision Tree* e *Random Forest*
[15]	sistema de detecção de *botnet* em IoT utilizando como base os algoritmos de *Slime Mold* (SMA) e de *Salp Swarm* (SSA)	Conjuntos de dados padrão disponíveis na fonte UCI, criados com base no tráfego real de dispositivos infectados na *botnet* IoT	Método de seleção de recursos baseado em *wrapper*
[13]	Estrutura de otimização para seleção de recursos	conjuntos de dados de grande dimensionalidade	metodologia de treino e teste
[12]	modelo de seleção de recursos do *wrapper* (SSA–ALO) hibridizando o algoritmo de *Swarm Algorithm* (SSA) e *Ant Lion Optimization* (ALO)	Dados diferentes a depender do dispositivo sendo analisado (*webcam*, babá eletrônica, campainha, câmera de segurança, etc)	KNN no caso do artigo, mas podem ser usados outros métodos. Além disso, o método de treinamento por si só acaba sendo influenciado pelo algoritmo SSA-ALO
[11]	Seleção de recursos, aproveitando os métodos de filtro e *wrapper*	3 *datasets* fornecidos em formato PCAP	Solução de aprendizado de máquina baseada em KNN de uma classe
[14]	Sistema de inferência *neuro-fuzzy* melhorado baseado no algoritmo híbrido *Grey Wolf-Bat* (HWBO)	Dois conjuntos de dados separados de sinais de eletroencefalograma (EEG)	algoritmo de aprendizado ANFIS

Capítulo 3

Fundamentação Teórica

Nesta seção serão apresentados os conceitos fundamentais utilizados para a elaboração deste trabalho. Os conceitos estão divididos nas seguintes seções: 3.1 *Botnet*, onde é abordada sua definição e descrição dos passos realizados para a execução de um ataque; 3.2 Algoritmo de redução de recursos, que descreve sua definição, bem como a apresentação de exemplos de algoritmos de otimização inspirados na natureza; 3.3 Árvore de Decisão, onde é apresentado o conceito do modelo de aprendizado e seu funcionamento; 3.4 *CICFlowMeter*, que descreve a ferramenta de segurança, aplicação e características; 3.5 EvoloPy-FS, onde são descritos a motivação e aplicação da estrutura de otimização; 3.6 Métricas, que descreve as métricas utilizadas na avaliação de modelos; 3.7 Conjunto de dados, que aborda o conceito de *datasets* e descreve os conjuntos utilizados no trabalho.

3.1 *Botnet*

Uma *botnet* é um grupo de computadores que são infectados por um *malware* e usados para fins maliciosos, como interromper a operação de serviços ou extrair ganhos lucrativos de usuários não suspeitos, entre outras ameaças realizadas sob o comando de seu criador, conhecido como *botmaster* [17]. Segundo Chen et al. [18], a *botnet* consiste em um dos principais meios de transporte de *malware* pelas redes e é responsável por diversos ataques, como os DDoS. A execução de um ataque compreende os seguintes passos:

- **Infecção**: O *botmaster* libera o código do bot ou de seu controlador na rede, podendo ser malicioso como um todo ou parcialmente, como *emails*, URLs, anúncios, entre outros meios aparentemente inofensivos.
- **Recrutamento**: O código do bot ou seu controlador infiltrado na infraestrutura de rede é o responsável por executar o processo de recrutamento, que pode ser inicializado diretamente pelo *botmaster* que fornece o código para hospedeiros alvo

de interesse ou o código pode ser de autorecrutamento, ou seja, ele percorre a rede em busca de hospedeiros vulneráveis para infectar.

- **Sincronização**: Ocorre depois do recrutamento dos bots. Eles são reunidos de volta para a central de comando e controle que geralmente é administrada remotamente pelo *botmaster* ou por outros bots na *botnet*. A sincronização dos bots com a central é mantida a todo momento a fim de receber novos comandos, critérios de infiltração e definições da aquisição, que são executados prontamente [19].

3.2 Algoritmo de redução de recursos

Os algoritmos de redução de recursos, também conhecido como seleção de recursos (FS, *Feature Selection*), possuem como objetivo selecionar apenas os recursos mais significativos do conjunto de dados, removendo os recursos redundantes e irrelevantes. A ideia é que, ao manter as *features* importantes, seja possível evitar o sobreajuste (*overfitting*) causado pelo número excessivo de *features*, gerando assim inúmeras vantagens como a melhora da performance do modelo, a redução do tempo de treinamento e da dimensionalidade do conjunto de dados e o aumento da precisão do aprendizado, melhorando a compreensibilidade dos resultados [20].

Para a redução de recursos, algoritmos de otimização inspirados na natureza são amplamente utilizados, pois são capazes de lidar com problemas complexos e com uma grande quantidade de variáveis, sempre em busca da solução ótima. Alguns desses algoritmos foram desenvolvidos neste trabalho. São eles:

3.2.1 *Algoritmo Grey Wolf Optimization* (GWO)

O *Grey Wolf Optimization* (GWO) é um algoritmo de otimização inspirado na hierarquia e mecanismo de caça dos lobos cinzentos na natureza. Esses são considerados predadores de ponta e normalmente vivem em grupos (matilhas) [1]. Possuem uma hierarquia dominante e bem estruturada.

O GWO é composto por um grupo de lobos cinzentos simulados, onde cada lobo representa uma solução possível para o problema de otimização. O algoritmo trabalha com três tipos de lobos cinzentos: líderes ou lobos alfa, lobos beta e lobos delta [1].

- Os lobos alfa α são considerados os líderes da matilha e suas ordens devem ser seguidas pelos demais lobos. São eles os responsáveis por tomar decisões de caça e quando e onde dormir, por exemplo.

- Os lobos beta β são lobos subordinados, auxiliam o alfa nas decisões e são os melhores candidatos a suceder o alfa. Eles também reforçam os comandos do alfa e retornam *feedbacks*.
- Os delta δ são subordinados dos alfas e betas. Dentro desse grupo estão as diversas categorias, como sentinelas, caçadores, cuidadores, anciões, entre outros.

Além da hierarquia social, existem as principais fases de caça dos lobos cinzentos:

1. Rastrear, seguir e se aproximar da presa.
2. Perseguir, cercar e abordar a presa até que a mesma pare de se mover.
3. Atacar em direção à presa.

É importante notar que essas fases de caça podem ser adaptadas para diferentes problemas de otimização. Além disso, o GWO também usa outras técnicas para aumentar a eficiência do algoritmo, como a adaptação do passo de busca e a utilização de diferentes métricas de distância.

Na imagem 3.1 abaixo, é possível analisar o pseudocódigo do algoritmo do GWO.

```
Initialize the grey wolf population X_i (i = 1, 2, ..., n)
Initialize a, A, and C
Calculate the fitness of each search agent
X_α=the best search agent
X_β=the second best search agent
X_δ=the third best search agent
while (t < Max number of iterations)
    for each search agent
            Update the position of the current search agent by equation (3.7)
    end for
    Update a, A, and C
    Calculate the fitness of all search agents
    Update X_α, X_β, and X_δ
    t=t+1
end while
return X_α
```

Figura 3.1: Pseudocódigo do algoritmo GWO. [1]

O GWO é uma técnica de otimização de baixo custo computacional e tem sido aplicado com sucesso em diversos problemas de otimização, como otimização de redes neurais, otimização de sistemas de geração de energia e otimização de sistemas de controle. Além disso, esse algoritmo é uma técnica robusta e capaz de lidar com problemas de otimização não-lineares e com restrições [1].

3.2.2 *Algoritmo Bat* (BA)

O algoritmo *Bat* é uma técnica de otimização inspirada no comportamento de morcegos que utiliza conceitos de busca estocástica, como a busca randômica, mas adiciona algumas características inspiradas na biologia desses mamíferos. A ideia principal por trás do algoritmo é simular a busca por alimento por meio da emissão de ondas sonoras (chamadas de ecolocalização). Se a onda sonora encontra alimento, o morcego se move para essa direção. Se não encontra, o morcego muda sua frequência de emissão de ondas sonoras [2].

No algoritmo *Bat*, cada indivíduo (*bat*) é representado por sua posição no espaço de busca e sua frequência de emissão. A velocidade de cada *bat* é determinada pela sua frequência e pela intensidade do sinal (similar à intensidade do som emitido pelos morcegos). Cada iteração do algoritmo simula a emissão de ondas sonoras e a atualização da posição dos indivíduos de acordo com as regras estabelecidas [7].

O pseudocódigo do algoritmo Bat é apresentado na figura 3.2 abaixo.

Bat Algorithm

Objective function $f(\mathbf{x}), \quad \mathbf{x} = (x_1, ..., x_d)^T$
Initialize the bat population $\mathbf{x}_i$ $(i = 1, 2, ..., n)$ *and* $\mathbf{v}_i$
Define pulse frequency f_i *at* $\mathbf{x}_i$
Initialize pulse rates r_i *and the loudness* A_i
while *(t <Max number of iterations)*
Generate new solutions by adjusting frequency,
and updating velocities and locations/solutions [equations (2) to (4)]
    if *(rand* $> r_i$*)*
    Select a solution among the best solutions
    Generate a local solution around the selected best solution
    end if
    Generate a new solution by flying randomly
    if *(rand* $< A_i$ & $f(\mathbf{x}_i) < f(\mathbf{x}_*)$*)*
    Accept the new solutions
    Increase r_i *and reduce* A_i
    end if
Rank the bats and find the current best $\mathbf{x}_*$
end while
Postprocess results and visualization

Figura 3.2: Pseudocódigo do algoritmo Bat. [2]

O algoritmo *Bat* tem sido aplicado com sucesso em uma variedade de problemas de otimização [2], como otimização de funções de várias variáveis, otimização de sistemas dinâmicos e otimização de redes neurais [7]. Além disso, ele tem mostrado resultados promissores em termos de convergência e capacidade de encontrar soluções ótimas em problemas de otimização de alta dimensionalidade.

3.2.3 *Algoritmo Whale Optimization* (WOA)

Inspirado no comportamento de caça das baleias jubarte na natureza [3], o algoritmo *Whale Optimization Algorithm* (WOA) é uma técnica de otimização baseada na inteligência desses mamíferos marinhos, os quais são animais sociais e cooperativos, e usa a metaheurística da busca cooperativa dos indivíduos para encontrar a solução ótima. O algoritmo utiliza duas principais características das baleias: sua capacidade de se comunicar e a sua estratégia de busca de alimento, chamado de método de alimentação com rede de bolhas. As baleias preferem caçar peixes pequenos próximo à superfície, criando bolhas ao longo de um "caminho"no formato de um "9"[3].

As baleias jubarte são capazes de identificar onde suas presas estão e cercá-las. Como a posição do projeto ótimo no espaço de busca não é conhecida antecipadamente, o algoritmo WOA assume que a melhor solução atual é a presa alvo ou está próxima da posição do ótimo. Depois de identificar o melhor agente de busca, os outros agentes tentam atualizar suas posições em relação a ele. [3]

É possível encontrar o pseudocódigo na figura 3.3.

```
Initialize the whales population X_i (i = 1, 2, ..., n)
Calculate the fitness of each search agent
X*=the best search agent
while (t < maximum number of iterations)
    for each search agent
    Update a, A, C, l, and p
        if1 (p<0.5)
            if2 (|A| < 1)
                Update the position of the current search agent by the Eq. (2.1)
            else if2 (|A| ≥ 1)
                Select a random search agent (X_rand)
                Update the position of the current search agent by the Eq. (2.8)
            end if2
        else if1 (p≥0.5)
                Update the position of the current search by the Eq. (2.5)
        end if1
    end for
    Check if any search agent goes beyond the search space and amend it
    Calculate the fitness of each search agent
    Update X* if there is a better solution
    t=t+1
end while
return X*
```

Figura 3.3: Pseudocódigo do algoritmo WOA. [3]

Esse algoritmo tem sido aplicado com sucesso em vários problemas de otimização, incluindo otimização de funções, otimização de sistemas dinâmicos, e otimização de sistemas

multiobjetivo. A pesquisa recente tem mostrado que o WOA tem desempenho comparável ou superior a outros algoritmos de otimização populares.

3.3 Árvore de Decisão

A árvore de decisão (DT, *Decision Tree* é um modelo de aprendizado supervisionado utilizado para classificação e regressão. Ela funciona criando um diagrama hierárquico de decisões, onde cada nó representa uma decisão a ser tomada e cada ramo representa uma possível consequência dessa decisão. A árvore é construída a partir de um conjunto de dados de treinamento, onde cada exemplo é rotulado com uma classe.

A Decision Tree é construída recursivamente, começando com o nó raiz, que representa todo o conjunto de dados. O algoritmo divide o conjunto de dados em subconjuntos mais homogêneos a cada nível da árvore, utilizando uma medida de impureza, como entropia ou gini (índice de heterogeneidade dos dados), para avaliar a qualidade da divisão. A construção é interrompida quando os subconjuntos são suficientemente homogêneos ou quando atinge um limite pré-definido de profundidade.

Uma vez construída, a árvore pode ser usada para classificar novos exemplos, desde que esses sejam atribuídos à classe correspondente ao nó folha em que caem. As árvores de decisão são fáceis de interpretar e são amplamente utilizadas em diversas áreas, como negócios, ciência da computação e medicina.

3.4 *CICFlowMeter*

CICFlowMeter é uma ferramenta de segurança de rede que foi desenvolvida para detectar e prevenir ataques de negação de serviço distribuído (DDoS). Ele se concentra na detecção de fluxos de tráfego malicioso e na classificação dos fluxos de tráfego legítimo [21].

Essa ferramenta utiliza uma variedade de técnicas para detectar ataques DDoS, incluindo aprendizado de máquina e técnicas estatísticas. Ele coleta e analisa dados de fluxo de rede a partir de vários dispositivos de rede, como roteadores, e usa esses dados para identificar padrões anormais de tráfego.

Uma das principais características do *CICFlowMeter* é sua capacidade de classificar fluxos de tráfego legítimo e malicioso de forma precisa, permitindo que as medidas de mitigação sejam aplicadas somente aos fluxos maliciosos. Isso ajuda a evitar a interrupção do tráfego legítimo, o que é comum em outras soluções de mitigação de DDoS. Além disso, *CICFlowMeter* é capaz de lidar com ataques DDoS em larga escala, usando a análise distribuída para processar grandes volumes de dados de fluxo de rede e escalando automaticamente conforme necessário [21].

3.5 EvoloPy-FS

EvoloPy-FS é uma estrutura de otimização baseado em algoritmos genéticos (GA) e algoritmos evolutivos (EA) para seleção de *features* em problemas de ML [13]. Ele foi projetado para ser usado em problemas de classificação e regressão, onde a seleção de características é importante para melhorar a precisão do modelo e reduzir a complexidade. Além disso, é caracterizado por sua capacidade de lidar com problemas de alta dimensionalidade e selecionar recursos relevantes, diminuindo a complexidade e melhorando a precisão do modelo [13].

A estrutura utiliza os princípios de evolução biológica para encontrar a melhor combinação de características para um determinado problema. Inicia-se com uma população inicial de soluções (conjuntos de *features*) geradas aleatoriamente e, em seguida, são utilizadas técnicas de seleção, cruzamento e mutação para evoluir essas soluções ao longo de várias gerações. A avaliação dessas soluções é feita usando um classificador ou regressão, treinado com o conjunto de características atual. A solução final é aquele com melhor performance [13].

3.6 Métricas

As métricas são utilizadas para avaliar o desempenho de modelos, como uma rede neural ou um classificador baseado em árvore de decisão. Ao desenvolver um projeto, é crucial que sejam utilizadas medidas apropriadas para cada problema, pois seu valor influencia na qualidade do modelo.

A fim de calcular o resultado e medir a precisão de um modelo de classificação, são aplicados os conceitos de Verdadeiro Positivo (VP), Falso Positivo (FP), Verdadeiro Negativo (VN) e Falso Negativo (FN), descritos abaixo. Esses são baseados na comparação entre as previsões realizadas pelo modelo e os resultados verdadeiros.

- VP: Número de casos em que o modelo previu corretamente uma classe positiva;
- FP: Número de casos em que o modelo previu incorretamente uma classe positiva;
- VN: Número de casos em que o modelo previu corretamente uma classe negativa;
- FN: Número de casos em que o modelo previu incorretamente uma classe negativa.

Esses conceitos são frequentemente usados em conjunto para calcular métricas de desempenho de classificação, como acurácia, precisão, revocação (*recall*) e *f1-score*, que normalmente são usadas para avaliar modelos de aprendizagem e indicar eficiência de um algoritmo. Essas medidas são descritas abaixo:

- **Acurácia**: Representada pela equação 3.1, consiste na proporção de previsões corretas sobre o número total de previsões. Ela é uma métrica de desempenho comum em problemas de classificação e é fácil de entender e interpretar. No entanto, pode ser enganosa se a distribuição das classes no conjunto de dados não for equilibrada.

$$\textbf{Acurácia} = \frac{\text{VP} + \text{VN}}{\text{VP} + \text{FN} + \text{VN} + \text{FP}} \tag{3.1}$$

- **Precisão**: É a proporção de verdadeiros positivos sobre o número total de positivos, representada pela equação 3.2. Ela mede a capacidade do modelo de evitar previsões falsas positivas. É uma métrica importante em problemas onde as consequências de previsões falsas positivas são graves, como em diagnósticos médicos ou detecção de fraude.

$$\textbf{Precisão} = \frac{\text{VP}}{\text{VP} + \text{FP}} \tag{3.2}$$

- ***Recall***: Mede a capacidade de um modelo de identificar corretamente os casos positivos. Como é possível notar na equação 3.3, o *recall* consiste na proporção de casos positivos corretamente identificados em relação ao número total de casos positivos presentes. Essa métrica é usado para medir a sensibilidade, ou seja, a capacidade de um modelo de detectar casos positivos.

$$\textbf{\textit{Recall}} = \frac{\text{VP}}{\text{VP} + \text{FN}} \tag{3.3}$$

- ***f1-Score***: Consiste em uma métrica de desempenho comumente usada em problemas de classificação binária. Como demonstrada na equação 3.4, é a média harmônica entre precisão e *recall*, permitindo, assim, equilibrar tais métricas, sendo especialmente útil quando a distribuição de classe é desequilibrada.

$$\textbf{\textit{F1-Score}} = 2 * \frac{\text{Precisão} * \textit{Recall}}{\text{Precisão} + \textit{Recall}} \tag{3.4}$$

- **Tempo de treinamento**: É o tempo que o modelo leva para ser treinado com um conjunto de dados de treinamento. É uma métrica importante a ser considerada quando se está trabalhando com grandes conjuntos de dados ou quando o modelo precisa ser treinado em tempo real. Algoritmos de ML com tempos de treinamento curtos são mais desejáveis.

3.7 Conjunto de dados

Conjunto de dados (*dataset*), consiste em uma coleção de dados que é usada para treinar e avaliar modelos de ML. Eles são essenciais para a ciência de dados, pois permitem que os cientistas de dados explorem e analisem os dados, bem como treinem e testem modelos preditivos. Os *datasets* podem ser compostos por dados estruturados, como números e tabelas, ou dados não estruturados, como texto e imagens. Eles também podem ser rotulados ou não. Os conjuntos de dados rotulados contêm informações adicionais, como rótulos ou classificações, que são usadas para treinar modelos supervisionados.

Existem várias fontes diferentes de *datasets*, incluindo bases de dados governamentais, empresas privadas e comunidades de ciência de dados. É importante notar que a qualidade do *datasets* é crucial para o desempenho do modelo. Isso inclui a representatividade dos dados, a precisão dos dados e a ausência de viés. A seleção e preparação de *datasets* de qualidade é uma parte importante do processo de ciência de dados.

Abaixo, serão descritos os *datasets* utilizados para o desenvolvimento deste trabalho:

- **ISOT-2010**: Conjunto de dados de tráfego de rede que foi capturado pelo Laboratório de Tecnologia de Segurança da Informação (ISOT) do Instituto Nacional de Ciência e Tecnologia Industrial Avançada (AIST) no Japão. Foi criado em 2010 combinando tráfego de rede normal coletado pelo Traffic Lab da Ericsson Research na Hungria e pelo Lawrence Berkeley National Lab (LBNL) com tráfego malicioso obtido pelo capítulo francês do projeto *Honeynet* [22]. Ele foi utilizado para detectar *botnets* P2P [23] e está disponível publicamente, contendo 11 GB de dados baseados em pacotes no formato *PCAP*. o *dataset* é amplamente utilizado em pesquisas sobre segurança de rede e detecção de intrusão, e é especialmente útil para avaliar o desempenho de algoritmos de aprendizado de máquina para classificação de tráfego de rede e detecção de anomalias.

- **ISCX-2014**: Fornecido pelo Instituto Canadense de Segurança Cibernética, contém 16 tipos diferentes de *botnets*. Esse conjunto de dados consiste em 5,3 GB de pacotes capturados de tráfego de rede, incluindo tanto tráfego normal quanto malicioso. O *dataset* foi construído a partir da combinação de três coleções principais:

  - ***Dataset* ISOT** [24]: Inclui projetos variados, especialmente o projeto *Honeynet* [22], a pesquisa da Ericsson e a pesquisa do Lawrence Berkeley National Laboratory, cada um contendo traços maliciosos de *botnets* bem identificados como *Flow Storm* e Zeus. Para conexões não maliciosas, as fontes vêm principalmente de pacotes de jogos, tráfego HTTP e aplicativos P2P.

  - ***Dataset* ISCX 2012 IDS** [25]: Projetado para ser realista, incluindo tráfego benigno (HTTP, SMTP, SSH, IMAP, POP3 e FTP) e malicioso (*Botnet* ISCX IRC) produzido por dispositivos reais.
  - ***Malware Capture Facility Project*** [26]: Inclui oito tipos diferentes de *botnets* e possui como objetivos executar *malware* real por períodos prolongados de tempo, analisar o tráfego de *malware* manual e automaticamente, atribuir rótulos verdadeiros às etiquetas de tráfego, incluindo várias fases de *botnet*, ataques, normal e de fundo e publicar esses dados para a comunidade para ajudar a desenvolver melhores métodos de detecção.

- **CTU-13**: Conjunto de dados de tráfego de *botnet* que foi capturado pela Universidade Técnica Tcheca em 2011. Ele consiste em treze capturas diferentes de amostras de *botnet*, cada uma executada com *malware* específico usando diferentes protocolos e executando diferentes ações. Esse *dataset* inclui tráfego normal e malicioso e possui como objetivo ter uma grande captura de tráfego real de *botnet* misturado com tráfego normal e tráfego de fundo. A característica distintiva do conjunto de dados CTU-13 é que ele foi rotulado e analisado manualmente. O *dataset* contém arquivos PCAP, arquivos bidirecionais *NetFlow*[27] e arquivos executáveis originais. Considera-se que os *NetFlows* bidirecionais apresentam diversas vantagens sobre os direcionais, como resolver a questão da diferenciação entre o cliente e o servidor, incluindo mais informações e rótulos detalhados [28].

Capítulo 4

Métodos e Resultados

4.1 Metodologia

Para avaliar o efeito dos algoritmos de otimização *Bat*, *Gray Wolf* e *Whale* num contexto de detecção de *botnets*, foram escolhidos os seguintes *datasets*: ISOT-2010, ISCX-2014 e CTU-13. Esses *datasets* trazem dados de tráfego de rede com pacotes pertencentes a *botnets* e pacotes não maliciosos no formato *PCAP*. Com esses dados, foi utilizada a ferramenta *CICFlowMeter* para converter os dados dos *datasets* em outro *dataset* com dados de fluxo de rede padronizados e no formato *CSV*. As colunas presentes nos *datasets* gerados são definidas no Apêndice A.

Uma vez que estes dados estavam prontos, foi utilizado um *framework* de código aberto *open source* chamado *EvoloPy-FS*, que trás implementações para os algoritmos de otimização citados. Para a realização das análises deste artigo, foram realizadas modificações no código desse *framework* para que fosse possível combinar otimizações, isto é, dar como entrada para um dos algoritmos de otimização a saída de um dos outros algoritmos de forma automática.

Para isso, com um conjunto de instruções (*script*) em Python, os *datasets* que estavam divididos em arquivos *CSV* separados foram unidos em um único arquivo *CSV* para cada *dataset*. Após isso, as linhas com valores inválidos foram removidas de todos os arquivos. A partir desses *datasets* obtidos, foi extraída uma amostra aleatória com 1000 linhas de cada um, de modo a preservar a proporção entre fluxos maliciosos e não maliciosos do *dataset* original. A partir dessas amostras, foram realizadas otimizações para cada *dataset* e suas combinações. Além disso, esses dados foram usados para o treinamento e execução de um modelo de Decision Tree implementado com o uso da biblioteca *SciKitLearn* e com uma quantidade máxima de dez folhas (*leaf nodes*).

A integração entre os resultados obtidos com a otimização e os dados fornecidos para o algoritmo de Árvore de Decisão foi feita por meio de arquivos. Esse procedimento de

otimização e treinamento e predição do modelo foi repetido 30 vezes com o auxílio de um *script* executado na linguagem *Bash* para cada *dataset* e para cada combinação dois a dois dos algoritmos de otimização, assim como para cada algoritmo usado de forma isolada e uma das combinações dos três algoritmos usados em sequência.

Cada resultado obtido pelo modelo, após as predições para as métricas de *f1-score*, acurácia, precisão e tempo de treinamento e predição do modelo, foi armazenado em um arquivo CSV, de modo que há um arquivo para cada combinação de algoritmos de otimização e para cada *dataset*. Depois de coletados os resultados nos arquivos CSV, foi montada uma tabela de sinais para cada *dataset*, a qual é descrita no Apêndice B, e aplicada a técnica de alocação de variação para analisar a influência que cada um dos algoritmos de otimização utilizados, assim como suas combinações, possui sobre os valores de *f1-score*, acurácia, precisão e tempo.

Depois de coletados os resultados, foi montada uma tabela de sinais para cada *dataset*, conforme pode ser visto nas figuras B.1, B.2 e B.3, anexadas no Apêndice B.

4.2 Resultados

Para esta seção, serão considerados os valores médios das métricas de *f1-score*, precisão, acurácia e tempo necessário para treinar e realizar uma predição com o modelo de *Decision Tree* após 30 iterações.

A tabela 4.1 mostra os resultados obtidos para cada um dos *datasets* analisados, utilizando todas as colunas do conjunto de dados gerado pelo *CICFlowMeter*, exceto as seguintes: *Flow ID*, *Src IP*, *Dst IP*, *Dst IP*, *Dst Port* e *Timestamp*, que foram desconsideradas por não possuírem relevância ao determinar se um fluxo é ou não pertencente a uma *botnet*. Como é possível observar, cada coluna na tabela representa uma métrica, enquanto cada linha representa um *dataset*.

Tabela 4.1: Tabela dos resultados obtidos para os *datasets* sem otimização.

dataset	*f1-score*(%)	Acurácia(%)	Precisão(%)	Tempo(ms)
CTU-13	91.04	87.24	93.42	7.4
ISCX-2014	85.67	86.69	90.26	11
ISOT-2010	90.82	99.34	92.75	4

É possível notar que os resultados alcançados com o modelo mostram que ele já apresenta um bom desempenho mesmo sem nenhum tipo de otimização. Isso pode ser constatado pelos valores elevados das métricas *f1-score*, acurácia e precisão, as quais indicam

que o modelo possui boa capacidade de prever corretamente os resultados. Além disso, os valores baixos para o tempo de processamento do modelo indicam que ele é eficiente e não requer muito tempo para gerar os resultados. Em resumo, o modelo já apresenta uma boa performance sem qualquer tipo de melhoria.

Os resultados obtidos para os *datasets* analisados, CTU-13, ISCX-2014 e ISOT-2010, a partir das colunas selecionadas pelos algoritmos de otimização, estão presentes nas tabelas 4.2, 4.3 e 4.4, respectivamente. Para cada uma das linhas presentes é representado um dos algoritmos aplicados ou uma combinação entre eles, enquanto para cada coluna é representada uma métrica.

Tabela 4.2: Tabela dos resultados obtidos para o *dataset* CTU-13.

Algoritmo de otimização	*f1-score*(%)	Acurácia(%)	Precisão(%)	Tempo(ms)
Bat	91.32	87.21	92.62	3.5
Bat seguido de *Gray Wolf*	90.86	86.94	92.66	2.8
Bat seguido de *Whale*	91.37	87.4	93.5	2.6
Gray Wolf	91.35	87.42	92.91	3.7
Gray Wolf seguido de *Bat*	91.1	87.1	92.83	2.9
Gray Wolf seguido de *Whale*	91.17	87.09	92.42	2.8
Whale	91.38	87.54	93.28	3.3
Whale seguido de *Bat*	91.5	87.7	92.55	2.6
Whale seguido de *Gray Wolf*	91.07	86.98	92.19	2.7
Bat Wolf Whale	91.53	87.46	92.45	4.3

De acordo com a 4.2, em comparação com a 4.1, podemos perceber que a combinação de *Bat* seguido de *Gray Wolf* se mostrou a pior escolha de otimização para este *dataset*, pois, apesar de ter diminuído o tempo de forma considerável, diminuiu também o valor das outras três métricas, além de outras otimizações conseguirem diminuir mais o tempo. Por outro lado, podemos perceber que a combinação do algoritmo *Bat* com o algoritmo *Whale* se mostrou, de modo geral, a melhor escolha para otimizar este *dataset*, tendo em vista que resultou num tempo menor com um valor maior para cada uma das outras métricas analisadas, sem falar que foi a única combinação capaz de aumentar a precisão do modelo, além de ter conseguido o menor tempo atingido para este *dataset*.

Em situações onde deseja-se priorizar a acurácia ou o *f1-score*, a combinação *Whale* seguido de *Bat* pode ser uma boa escolha caso o tempo também seja importante, pois

essa combinação também atinge o menor valor de tempo encontrado para esse *dataset*, assim como a maior acurácia e o segundo maior *f1-score*, tendo como prejuízo apenas uma leve queda na precisão. Para este *dataset*, o algoritmo *Whale* se mostrou o que funciona melhor de forma isolada. A combinação dos três algoritmos atingiu o maior valor para *f1-score* neste *dataset*, mas com apenas uma pequena diferença para o segundo maior e com o maior tempo, o que tornou seu uso não muito promissor nesse teste. Os outros algoritmos e combinações não tiveram resultados muito notáveis.

Tabela 4.3: Tabela dos resultados obtidos para o *dataset* ISCX-2014.

Algoritmo de otimização	*f1-score*(%)	Acurácia(%)	Precisão(%)	Tempo(ms)
Bat	81.38	82.05	85.77	5.9
Bat seguido de *Gray Wolf*	70.4	71.54	70.7	4.1
Bat seguido de *Whale*	70.19	70.05	69.9	4.2
Gray Wolf	80.57	80.92	83	6.2
Gray Wolf seguido de *Bat*	72.17	71.72	71.08	4.3
Gray Wolf seguido de *Whale*	69.68	69.97	69.46	4.4
Whale	84.11	85	89.06	5.2
Whale seguido de *Bat*	70.75	70.6	71.05	4.4
Whale seguido de *Gray Wolf*	69.86	69.96	70.35	5.7
Bat Wolf Whale	69.55	70.14	69.9	6.6

Como é possível analisar na tabela 4.3, o algoritmo *Whale* e o algoritmo *Bat* obtiveram os melhores resultados em quase todas as métricas, alcançando valores significativamente maiores do que os outros algoritmos, o que indica que eles não são tão eficientes quanto o *Whale* e o *Bat*, executados de forma isolada. para prever corretamente os resultados. No entanto, houve uma exceção na métrica de tempo de treinamento, onde o algoritmo *Bat* seguido de *Gray Wolf* obteve vantagem.

Apesar do tempo ser uma métrica de grande importância para diversos projetos, o *Whale* pode ser útil em casos onde essa medida não seja tão relevante. No campo de perícia forense, por exemplo, cujas análises geralmente se concentram em recuperar informações de um dispositivo ou sistema para investigar uma violação de segurança ou outro incidente criminoso de forma rápida e eficiente, não é necessário um treinamento frequente para aperfeiçoar ou otimizar o modelo, já que normalmente são usados dados históricos, já coletados, o que torna o algoritmo em questão a melhor escolha para cenários desse tipo.

Quanto aos outros algoritmos e suas combinações, é possível notar que a combinação dos três algoritmos de otimização, *Bat Wolf Whale*, apresentou os piores resultados nas métricas *f1-score* e tempo de treinamento, não sendo diferente quanto às outras medidas, cujos valores estão dentre os menores, atrás apenas do algoritmo *Bat* seguido de *Whale* na acurácia e do algoritmo *Gray Wolf* seguido de *Whale* na precisão.

Tabela 4.4: Tabela dos resultados obtidos para o *dataset* ISOT-2010.

Algoritmo de otimização	*f1-score*(%)	Acurácia(%)	Precisão(%)	Tempo(ms)
Bat	89.19	99.29	92.33	2.6
Bat seguido de *Gray Wolf*	87.47	99.25	92.15	2
Bat seguido de *Whale*	86.26	99.16	90.1	2
Gray Wolf	88.42	99.26	90.36	2.3
Gray Wolf seguido de *Bat*	87.04	99.21	91.94	1.9
Gray Wolf seguido de *Whale*	85.98	99.21	89.69	1.8
Whale	91.23	99.4	93.7	2.1
Whale seguido de *Bat*	90.05	99.4	92.91	2.1
Whale seguido de *Gray Wolf*	89.46	99.32	95.2	2
Bat Wolf Whale	86.18	99.2	91.29	2.4

De acordo com a tabela 4.4, em comparação com a tabela 4.1, podemos notar que o algoritmo *Whale* se mostrou, de forma geral, como a melhor escolha para otimizar este *dataset* de modo que conseguiu diminuir o tempo e aumentar o valor das outras três métricas avaliadas, incluindo os maiores valores para *f1-score*, acurácia e o segundo maior valor para precisão. Por outro lado, podemos dizer que o algoritmo *Bat* parece ter sido aquele com pior resultado para este *dataset*, tendo em vista que a diminuição de tempo proporcionada pelo seu uso resultou no maior tempo dentre os resultados encontrados ao mesmo tempo em que diminuiu todas as outras três métricas analisadas. É possível afirmar também que não existe motivo para justificar o uso da combinação *Bat* e *Whale* nem da combinação *Bat* e *Wolf*, tendo em vista que estas combinações apresentam o mesmo tempo que a combinação *Whale* e *Wolf*, porém com resultados piores para as outras métricas.

A combinação dos três algoritmos de otimização também se encaixa nessa situação, exceto que possui um tempo ainda maior que o da combinação *Whale* e *Wolf*, que se mostra bastante promissora ao atingir a maior precisão encontrada para esse *dataset*,

assim como um leve aumento da acurácia, ao custo de uma pequena queda no *f1-score*. As combinação *Wolf* e *Bat*, assim como a combinação *Wolf* e *Whale* atingiram os menores tempos ao custo de pequenas quedas nos valores das outras três métricas, porém, podem ter seu valor em situações onde o tempo é crítico, como num sistema para detecção de *botnets* em tempo real.

Calculando a variação devida a cada algoritmo de otimização sobre as métricas analisadas por meio do método de alocação da variação, obtemos os resultados apresentados nas Tabelas 4.5, 4.6 e 4.7. Com essa tabela é possível calcular o quanto cada um dos algoritmos de otimização utilizados influencia na variação dos valores das métricas.

Tabela 4.5: Tabela de alocação da variação dos algoritmos sobre as métricas com o *dataset* CTU-13.

Algoritmo de otimização	*f1-score*(%)	Acurácia(%)	Precisão(%)	Tempo(%)
Gray Wolf	0.11	6.04	47.52	7.36
Bat	9.95	0.81	3.83	11.6
Whale	27.31	16.93	0.19	14.24
Gray Wolf Bat	1.59	2.82	2.69	21.79
Gray Wolf Whale	1.32	13.33	25.56	21.69
Bat Whale	5.86	20.52	2.45	12.17
Gray Wolf Bat Whale	53.82	39.51	8.02	0.58

Conforme a tabela 4.5, é possível verificar o potencial que cada combinação de algoritmo de otimização tem de variar os valores das métricas analisadas. É possível perceber que para o *dataset* CTU-13, a combinação de *Wolf* e *Bat* possui o maior potencial para influenciar o tempo. No entanto, essa combinação também se mostra bastante limitante para variar os valores das outras três métricas, o que é indesejável quando o objetivo é maximizar os valores de *f1-score*, acurácia e precisão ao mesmo tempo que se deseja minimizar o tempo.

É possível chegar mais perto de alcançar esse objetivo ao escolher a combinação *Wolf* e *Whale*, apesar do potencial de minimizar o tempo ser marginalmente menor que o da combinação *Wolf* e *Bat*, essa combinação oferece um maior potencial para aumentar os valores da acurácia e da precisão. Para situações onde o tempo não é uma prioridade, a combinação dos três algoritmos de otimização oferece um potecial considerável para aumentar o *f1-score* e a acurácia, com um potencial menor para variar a precisão. O algoritmo *Wolf* também pode ser valioso em situações onde o tempo não é prioritário ao

oferecer o maior potencial para variar a precisão. A combinação *Bat* e *Whale*, assim como o algoritmo *Whale* de forma isolada se apresentam um potencial considerável para variar a acurácia sem ter que sacrificar a diminuição do tempo, podendo ser úteis em situações onde a acurácia e o tempo são prioridade, como em detecções de *botnets* em tempo real.

Tabela 4.6: Tabela de alocação da variação dos algoritmos sobre as métricas com o *dataset* ISCX-2014.

ISCX-2014	*f1-score*(%)	Acurácia(%)	Precisão(%)	Tempo(%)
Gray Wolf	32.14	32.43	35.52	8.32
Bat	26.62	27.56	25.38	12.58
Whale	25.7	23.13	20.79	17.34
Gray Wolf Bat	3.49	3.75	3.04	14.89
Gray Wolf Whale	0.02	0.04	0.11	23.21
Bat Whale	0.06	0.02	0.11	23.63
Gray Wolf Bat Whale	11.94	13.03	15.02	0

De acordo com a tabela 4.6, o algoritmo *Wolf* é o que possui o maior potencial para maximizar os valores das métricas *f1-score*, acurácia e precisão, oferecendo um potencial relativamente pequeno para minimizar o tempo em comparação com os algoritmos *Bat* ou *Whale*, que oferecem, respectivamente, potenciais um pouco menores para maximizar *f1-score*, acurácia e precisão, porém, oferecem potenciais maiores para minimizar o tempo.

Em situações onde o tempo não é uma prioridade, o algoritmo *Wolf* se mostra como a melhor opção e em situações onde o tempo é importante, a melhor escolha é usar o algoritmo *Bat* ou o algoritmo *Whale*. Vale dizer que as combinações de algoritmos não se mostraram muito promissoras para este *dataet*, tendo em vista que limitam demasiadamente o potencial de variação de uma ou mais métricas sem, necessariamente, oferecer algo em troca.

Conforme a tabela 4.7, é possível afirmar que os algoritmos *Wolf* e *Whale* são boas alternativas quando o tempo é importante, por oferecerem um potencial considerável para diminuição do tempo ao mesmo tempo que oferecem um potencial considerável para aumento das outras métricas. Em situações onde o tempo não é tão importante, o algoritmo *Bat* e a combinação *Wolf* e *Bat* podem ser valiosos, pois oferecem grandes potenciais para variar a acurácia e a precisão, respectivamente. As outras combinações de algoritmos não parecem muito promissoras para este *dataset*.

Tabela 4.7: Tabela de alocação da variação dos algoritmos sobre as métricas com o *dataset* ISOT-2010.

ISOT-2010	*f1-score*(%)	Acurácia(%)	Precisão(%)	Tempo(%)
Gray Wolf	39.19	25.44	28.4	19.46
Bat	24.29	35.07	0.63	6.34
Whale	13.52	5.82	6.06	22.02
Gray Wolf Bat	11.78	13.9	46.65	12.4
Gray Wolf Whale	0.23	0.04	0	22.09
Bat Whale	1.24	5.8	9.05	17.37
Gray Wolf Bat Whale	9.7	13.9	9.19	0.3

Apesar do modelo de *Decision Tree* já possuir um bom desempenho sem qualquer tipo de otimização, é possível notar que alguns dos algoritmos de otimização utilizados e suas combinações otimizaram ainda mais o modelo, apresentando uma melhoria significativa nas métricas de *f1-score*, acurácia, precisão e tempo de treinamento. Pode-se observar que os valores obtidos para cada métrica a partir da aplicação de cada algoritmo varia de acordo com o *dataset* utilizado. A combinação dos três algoritmos, *Bat Wolf Whale*, por exemplo, não pareceu promissora para os *datasets* ISCX-2014 e ISOT-2010, porém, além de apresentar o pior resultado de tempo de treinamento para o *dataset* CTU-13, atingiu o maior resultado na métrica de *f1-score* e um dos melhores na de acurácia, mostrando que pode ser útil para cenários onde o tempo é uma métrica irrelevante. Outro algoritmo que chama a atenção é o *Whale*, pois apesar do valor do tempo de treinamento não estar entre os melhores, apresentou os melhores resultados nas demais métricas para todos os três *datasets*, tornando-se extremamente útil para o caso dos cenários citados.

Outra análise possível de se realizar a partir das tabelas, é que os algoritmos de otimização executados de forma isolada tendem a apresentar melhores resultados nas métricas *f1-score*, acurácia e precisão, enquanto as combinações, exceto a dos três algoritmos, apresentam um tempo de treinamento muito melhor, ainda que possam ter apresentado um comportamento inconsistente com relação às outras três métricas.

A partir dos resultados, é possível afirmar que, a fim de obter uma melhor eficiência em modelos onde a distribuição de classe é desbalanceada, cenários onde a métrica *f1-score* possui maior relevância, o algoritmo *Whale* apresenta-se como a melhor escolha. Já para garantir uma boa exatidão nos resultados, tendo a medida de *acurácia* como a mais

importante, os algoritmos mais favoráveis são o algoritmo *Whale* e o algoritmo *Whale* seguido de *Bat*. Com relação à precisão, o algoritmo *Whale* continua sendo a opção mais promissora. Por fim, considerando o tempo de treinamento, é possível afirmar que o algoritmo *Bat* seguido de *Whale* é a melhor escolha.

Assim, a partir dos resultados obtidos, é possível notar que o algoritmo *Whale* se apresenta como o algoritmo mais promissor considerando as métricas *f1-score*, acurácia e precisão, utilizadas no modelo de Decision Tree, apesar de não ser a melhor opção com relação ao tempo de treinamento. No entanto, o algoritmo *Bat* seguido de *Whale* apresenta-se como a escolha mais favorável em cenários onde o tempo é uma medida crucial, não sendo tão relevantes as demais métricas.

Capítulo 5

Conclusões

A exploração das vulnerabilidades identificadas nos dispositivos conectados à IoT e, por consequência, sua aplicação para a execução de ataques DDoS, tornou-se altamente comum nos últimos anos. Desta maneira, este trabalho teve como finalidade estudar algoritmos de otimização inspirados na natureza para otimizar a redução de recursos no contexto de detecção de ataques a dispositivos IoT com o uso de *botnets*.

O tema trabalhado permitiu uma melhor compreensão sobre uma das ameaças mais comuns enfrentadas atualmente, a *botnet*, e possibilitou também explorar os diversos algoritmos de otimização existentes, podendo assim estudar possibilidades de aprimorar a detecção de *botnets* com otimização de recursos.

Conforme os resultados obtidos, foi possível afirmar que o algoritmo *Whale* se mostrou bastante promissor de forma geral, por fornecer um aumento perceptível do *f1-score*, acurácia e precisão de forma consistente entre os *datasets*. A combinação *Bat* e *Whale* se mostrou bastante promissora com relação ao tempo, ainda que possa deixar um pouco a desejar com relação às outras métricas, essa pode ser uma troca válida em situações onde o tempo é crítico. Foi possível também notar que os algoritmos isolados tendem a possuir valores mais altos para as métricas *f1-score*, acurácia e precisão, ao custo de um tempo um pouco maior, o que pode ser útil em várias situações, quando o tempo não é uma prioridade. Foi perceptível também que a combinação dos três algoritmos não se mostrou muito promissora, proporcionando tempos mais elevados sem ganhos consideráveis nas métricas.

Ao longo da análise dos dados, é possível concluir que o *f1-score* é uma métrica de suma importância no contexto de detecção de *botnet* tendo em vista que os dados tendem a ser desbalanceados, o que faz sentido, tendo em vista o gigantesco volume de pacotes que trafega em uma rede e que a maioria desses pacotes não pertencem à *botnets*.

Com isso, é possível afirmar que esse estudo pode incentivar futuros trabalhos envolvendo combinações diferentes entre outros algoritmos de otimização inspirados na natu-

reza, maiores volumes de dados e *datasets* diferentes, em busca de resultados ainda mais precisos.

Referências

[1] Mirjalili, Seyedali, Seyed Mohammad Mirjalili e Andrew Lewis: *Grey wolf optimizer*. Advances in Engineering Software, 69:46–61, 2014, ISSN 0965-9978. https://www.sciencedirect.com/science/article/pii/S0965997813001853. ix, 8, 9

[2] Yang, Xin She: *A new metaheuristic bat-inspired algorithm*. 284, abril 2010. https://www.researchgate.net/publication/45913690_A_New_Metaheuristic_Bat-Inspired_Algorithm. ix, 10

[3] Mirjalili, Seyedali e Andrew Lewis: *The whale optimization algorithm*. Advances in Engineering Software, 95:51–67, 2016, ISSN 0965-9978. https://www.sciencedirect.com/science/article/pii/S0965997816300163. ix, 2, 11

[4] *State of iot 2022: Number of connected iot devices growing 18% to 14.4 billion globally*. Online disponível em: https://iot-analytics.com/number-connected-iot-devices/. Acesso em: 20 jan 2023. 1

[5] Pontes, Camila F. T., Manuela M. C. de Souza, João J. C. Gondim, Matt Bishop e Marcelo Antonio Marotta: *A new method for flow-based network intrusion detection using the inverse potts model*. IEEE Transactions on Network and Service Management, 18(2):1125–1136, 2021. 2

[6] Elghamrawy, Sally e Aboul Hassanien: *A hybrid genetic–grey wolf optimization algorithm for optimizing takagi–sugeno–kang fuzzy systems*. Neural Computing and Applications, 34, 2022. https://www.researchgate.net/publication/360954460_A_hybrid_Genetic-Grey_Wolf_Optimization_algorithm_for_optimizing_Takagi-Sugeno-Kang_fuzzy_systems. 2, 4, 6

[7] Xie, Jian, Liangliang Li e Mingzhi Ma: *Cloud model bat algorithm*. The Scientific World Journal, 2014:237102, maio 2014. https://www.researchgate.net/publication/263430240_Cloud_Model_Bat_Algorithm. 2, 10

[8] Dong, Xudong, Chen Dong, Zhenyi Chen, Ye Cheng e Bo Chen: *Botdetector: An extreme learning machine-based internet of things botnet detection model*. Transactions on Emerging Telecommunications Technologies, 32(5):e3999, 2021. https://onlinelibrary.wiley.com/doi/abs/10.1002/ett.3999. 3, 6

[9] Wazzan, Majda, Daniyal Algazzawi, Aiiad Albeshri, Syed Hasan, Osama Rabie e Muhammad Zubair Asghar: *Cross deep learning method for effectively detecting the propagation of iot botnet*. Sensors, 22(10), 2022, ISSN 1424-8220. https://www.mdpi.com/1424-8220/22/10/3895. 3, 6

[10] Nguyen, Giang L., Braulio Dumba, Quoc Dung Ngo, Hai Viet Le e Tu N. Nguyen: *A collaborative approach to early detection of iot botnet*. Computers Electrical Engineering, 97:107525, 2022, ISSN 0045-7906. https://www.sciencedirect.com/science/article/pii/S0045790621004717. 3, 6

[11] Malik, Kainat, Faisal Rehman, Tahir Maqsood, Saad Mustafa, Osman Khalid e Adnan Akhunzada: *Lightweight internet of things botnet detection using one-class classification*. Sensors, 22(10), 2022, ISSN 1424-8220. https://www.mdpi.com/1424-8220/22/10/3646. 3, 6

[12] Abu Khurma, Ruba, Iman Almomani e Ibrahim Aljarah: *Iot botnet detection using salp swarm and ant lion hybrid optimization model*. Symmetry, 13(8), 2021, ISSN 2073-8994. https://www.mdpi.com/2073-8994/13/8/1377. 3, 6

[13] Abu Khurma, Ruba, Ibrahim Aljarah, Ahmad Sharieh e Seyedali Mirjalili: *EvoloPy-FS: An Open-Source Nature-Inspired Optimization Framework in Python for Feature Selection*, páginas 131–173. janeiro 2020, ISBN 978-981-32-9989-4. https://www.researchgate.net/publication/337196982_EvoloPy-FS_An_Open-Source_Nature-Inspired_Optimization_Framework_in_Python_for_Feature_Selection. 4, 6, 13

[14] Balasubramanian, Kishore, K. Ramya e Gayathri Devi: *Optimized adaptive neuro-fuzzy inference system based on hybrid grey wolf-bat algorithm for schizophrenia recognition from eeg signals*. Cognitive Neurodynamics, 17:1–19, maio 2022. https://www.researchgate.net/publication/360927550_Optimized_adaptive_neuro-fuzzy_inference_system_based_on_hybrid_grey_wolf-bat_algorithm_for_schizophrenia_recognition_from_EEG_signals. 4, 6

[15] Hoseini, fatemeh, Farhad Soleimanian Gharehchopogh e Mohammad Masdari: *A botnet detection in iot using a hybrid multi-objective optimization algorithm*. New Generation Computing, 40, 2022. https://www.researchgate.net/publication/364329880_Botnet_Detection_Employing_a_Dilated_Convolutional_Autoencoder_Classifier_with_the_Aid_of_Hybrid_Shark_and_Bear_Smell_Optimization_Algorithm-Based_Feature_Selection_in_FANETs. 5, 6

[16] Ghanimi, Hayder e Ali Abosinnee: *Botnet detection employing a dilated convolutional autoencoder classifier with the aid of hybrid shark and bear smell optimization algorithm-based feature selection in fanets*. Big Data and Cognitive Computing, 6, outubro 2022. https://www.researchgate.net/publication/364329880_Botnet_Detection_Employing_a_Dilated_Convolutional_Autoencoder_Classifier_with_the_Aid_of_Hybrid_Shark_and_Bear_Smell_Optimization_Algorithm-Based_Feature_Selection_in_FANETs. 5, 6

[17] Nogueira, Michele: *Anticipating moves to prevent botnet generated ddos flooding attacks*. novembro 2016. https://www.researchgate.net/publication/311222392_Anticipating_Moves_to_Prevent_Botnet_Generated_DDoS_Flooding_Attacks. 7

[18] Chen, Shao Chien, Yi Ruei Chen e Wen Guey Tzeng: *Effective botnet detection through neural networks on convolutional features*. Em *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/ 12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, páginas 372–378, 2018. https://ieeexplore.ieee.org/document/8455930. 7

[19] Ogu, Emmanuel, Nikos Vrakas, Chiemela Ogu e Ajose Ismail B.M.: *On the internal workings of botnets: A review*. International Journal of Computer Applications, 138:975–8887, abril 2016. https://www.researchgate.net/publication/298788691_On_the_Internal_Workings_of_Botnets_A_Review. 8

[20] Kumar, Dr K Ramesh e Vanaja: *Analysis of feature selection algorithms on classification: A survey*. International Journal of Computer Applications, 96:28–35, junho 2014. https://www.researchgate.net/publication/266504650_Analysis_of_Feature_Selection_Algorithms_on_Classification_A_Survey. 8

[21] Habibi Lashkari, Arash: *Cicflowmeter-v4.0 (formerly known as iscxflowmeter) is a network traffic bi-flow generator and analyser for anomaly detection. https://github.com/iscx/cicflowmeter*. agosto 2018. https://www.researchgate.net/publication/326991554_CICFlowmeter-V40_formerly_known_as_ISCXFlowMeter_is_a_network_traffic_Bi-flow_generator_and_analyser_for_anomaly_detection_httpsgithubcomISCXCICFlowMeter. 12

[22] *The honeynet project*. Online disponível em: https://www.honeynet.org/. Acesso em: 20 jan 2023. 15

[23] Saad, Sherif, Issa Traore, Ali Ghorbani, Bassam Sayed, David Zhao, Wei Lu, John Felix e Payman Hakimian: *Detecting p2p botnets through network behavior analysis and machine learning*. julho 2011. https://www.researchgate.net/publication/215732543_Detecting_P2P_botnets_through_network_behavior_analysis_and_machine_learning. 15

[24] Zhao, David, Issa Traore, Bassam Sayed, Wei Lu, Sherif Saad, Ali Ghorbani e Dan Garant: *Botnet detection based on traffic behavior analysis and flow intervals*. Computers Security, 39:2–16, novembro 2013. https://www.researchgate.net/publication/259117704_Botnet_detection_based_on_traffic_behavior_analysis_and_flow_intervals. 15

[25] Shiravi, Ali, Hadi Shiravi, Mahbod Tavallaee e Ali Ghorbani: *Toward developing a systematic approach to generate benchmark datasets for intrusion detection*. Computers Security, 31:357–374, maio 2012. https://www.researchgate.net/publication/257006500_Toward_developing_a_systematic_approach_to_generate_benchmark_datasets_for_intrusion_detection. 16

[26] *Malware capture facility project*. Online disponível em: https://mcfp.weebly.com/. Acesso em: 20 jan 2023. 16

[27] *Netflow*. Online disponível em: https://www.cisco.com/c/pt_br/tech/quality-of-service-qos/netflow/index.html. Acesso em: 20 jan 2023. 16

[28] García, Sebastián, Martin Grill, Jan Stiborek e Alejandro Zunino: *An empirical comparison of botnet detection methods*. Computers Security, 45:100–123, setembro 2014. https://www.researchgate.net/publication/262921640_An_Empirical_Comparison_of_Botnet_Detection_Methods. 16

Apêndice A

Colunas dos datasets

Colunas presentes nos *datasets* gerados pelo *CICFlowMeter*:

- *Label classificação do fluxo*
- *Flow ID* Identificação do fluxo
- *Src IP* Endereço de IP de origem do fluxo
- *Src Port* Porta de origem do fluxo
- *Dst IP* Endereço de IP de destino do fluxo
- *Dst Port* Porta de destino do fluxo
- *Timestamp Data* e horário em que o fluxo foi detectado
- *Flow Duration* Duração do fluxo em microssegundos
- *total Fwd Packet* Total de pacotes sendo enviados
- *total Bwd packets* Total de paoctes sendo recebidos
- *total Length of Fwd Packet* Tamanho total dos pacotes enviados para frente
- *total Length of Bwd Packet* Tamanho total dos pacotes enviados para trás
- *Fwd Packet Length Min* Tamanho do menor pacote enviados para frente
- *Fwd Packet Length Max* Tamanho do maior pacote enviados para frente
- *Fwd Packet Length Mean* Tamanho médio dos pacotes enviados para frente
- *Fwd Packet Length Std* Desvio padrão no tamanho dos pacotes enviados para frente
- *Bwd Packet Length Min* Tamanho do menor pacote enviados para trás
- *Bwd Packet Length Max* Tamanho do maior pacote enviados para trás

- *Bwd Packet Length Mean* Tamanho médio dos pacotes enviados para trás
- *Bwd Packet Length Std* Desvio padrão no tamanho dos pacotes enviados para trás
- *Flow Bytes/s* Número de bytes por segundo
- *Flow Packets/s* Número de pacotes por segundo
- *Flow IAT Mean* Intervalo médio entre dois pacotes enviados para frente
- *Flow IAT Std* Desvio padrão do intervalo entre dois pacotes no fluxo
- *Flow IAT Max* Intervalo máximo entre dois pacotes no fluxo
- *Flow IAT Min* Intervalo mínimo entre dois pacotes no fluxo
- *Fwd IAT Min* Intervalo mínimo entre dois pacotes enviados no fluxo
- *Fwd IAT Max* Intervalo máximo entre dois pacotes enviados no fluxo
- *Fwd IAT Mean* Tempo médio entre dois pacotes enviados para frente
- *Fwd IAT Std* Desvio padrão entre dois pacotes enviados para frente
- *Fwd IAT Total* Tempo total entre dois pacotes enviados para frente
- *Bwd IAT Min* Tempo mínimo entre dois pacotes enviados na direção inversa
- *Bwd IAT Max* Tempo máximo entre dois pacotes enviados na direção inversa
- *Bwd IAT Mean* Tempo médio entre dois pacotes enviados na direção inversa
- *Bwd IAT Std* Tempo de desvio padrão entre dois pacotes enviados na direção inversa
- *Bwd IAT Total* Tempo total entre dois pacotes enviados na direção inversa
- *Fwd PSH flags* Número de vezes que o sinalizador PSH foi definido em pacotes viajando na direção direta (0 para UDP)
- *Bwd PSH Flags* Número de vezes que o sinalizador PSH foi definido em pacotes viajando na direção reversa (0 para UDP)
- *Fwd URG Flags* Número de vezes que o sinalizador URG foi definido em pacotes viajando na direção direta (0 para UDP)
- *Bwd URG Flags* Número de vezes que o sinalizador URG foi definido em pacotes viajando na direção reversa (0 para UDP)
- *Fwd Header Length* Total de bytes usados para cabeçalhos na direção direta
- *Bwd Header Length* Total de bytes usados para cabeçalhos na direção inversa

- *FWD Packets/s* Número de pacotes encaminhados por segundo
- *Bwd Packets/s* Número de pacotes para trás por segundo
- *Packet Length Min* Comprimento mínimo de um pacote
- *Packet Length Max* Comprimento máximo de um pacote
- *Packet Length Mean* Comprimento médio de um pacote
- *Packet Length Std* Comprimento do desvio padrão de um pacote
- *Packet Length Variance* Comprimento de variação de um pacote
- *FIN Flag Count* Número de pacotes com FIN
- *SYN Flag Count* Número de pacotes com SYN
- *RST Flag Count* Número de pacotes com RST
- *PSH Flag Count* Número de pacotes com PUSH
- *ACK Flag Count* Número de pacotes com ACK
- *URG Flag Count* Número de pacotes com URG
- *CWR Flag Count* Número de pacotes com CWR
- *ECE Flag Count* Número de pacotes com ECE
- *down/Up Ratio* Taxa de download e upload
- *Average Packet Size* Tamanho médio do pacote
- *Fwd Segment Size Avg* Tamanho médio observado na direção direta
- *Bwd Segment Size Avg* Tamanho médio observado na direção inversa
- *Fwd Bytes/Bulk Avg* Número médio de taxa em massa de bytes na direção direta
- *Fwd Packet/Bulk Avg* Número médio de taxa de volume de pacotes na direção direta
- *Fwd Bulk Rate Avg* Número médio de taxa em massa na direção direta
- *Bwd Bytes/Bulk Avg* Número médio de taxa em massa de bytes na direção inversa
- *Bwd Packet/Bulk Avg* Número médio de taxa de volume de pacotes na direção inversa
- *Bwd Bulk Rate Avg* Número médio de taxa em massa na direção para trás
- *Subflow Fwd Packets* Número médio de pacotes em um subfluxo na direção direta

- *Subflow Fwd Bytes* Número médio de bytes em um subfluxo na direção direta
- *Subflow Bwd Packets* Número médio de pacotes em um subfluxo na direção inversa
- *Subflow Bwd Bytes* Número médio de bytes em um subfluxo na direção inversa
- *Fwd Init Win bytes* Número total de bytes enviados na janela inicial na direção direta
- *Bwd Init Win bytes* Número total de bytes enviados na janela inicial na direção inversa
- *Fwd Act Data Pkts* Contagem de pacotes com pelo menos 1 byte de carga útil de dados TCP na direção direta
- *Fwd Seg Size Min* Tamanho mínimo do segmento observado na direção direta
- *Active Min* Tempo mínimo que um fluxo esteve ativo antes de ficar inativo
- *Active Mean* Tempo médio em que um fluxo esteve ativo antes de ficar ocioso
- *Active Max* Tempo máximo que um fluxo ficou ativo antes de ficar ocioso
- *Active Std* Tempo de desvio padrão em que um fluxo estava ativo antes de ficar ocioso
- *Idle Min* Tempo mínimo que um fluxo ficou ocioso antes de se tornar ativo
- *Idle Mean* Tempo médio que um fluxo ficou ocioso antes de se tornar ativo
- *Idle Max* Tempo máximo que um fluxo ficou ocioso antes de se tornar ativo
- *Idle Std* Tempo de desvio padrão em que um fluxo ficou ocioso antes de se tornar ativo

Apêndice B

Tabelas de sinais

Tabelas de sinais geradas a partir dos dados obtidos para cada *dataset*.

Figura B.1: Tabela de sinais montada com os dados obtidos para o *dataset* CTU-13.

I	WOLF	BAT	WHALE	WOLF BAT	WOLF WHALE	BAT WHALE	WBW	*f1 score*	Acurácia	Precisão	Tempo
1	-1	-1	-1	1	1	1	-1	0.910410	0.872400	0.934219	0.007485
1	-1	-1	1	1	-1	-1	1	0.913871	0.875467	0.932851	0.003397
1	-1	1	-1	-1	1	-1	1	0.913270	0.872133	0.926267	0.003502
1	-1	1	1	-1	-1	1	-1	0.913719	0.874069	0.935034	0.002653
1	1	-1	-1	-1	-1	1	1	0.913504	0.874267	0.830070	0.003736
1	1	-1	1	-1	1	-1	-1	0.911762	0.870933	0.694664	0.002824
1	1	1	-1	1	-1	-1	-1	0.911093	0.871034	0.710884	0.002935
1	1	1	1	1	1	1	1	0.915332	0.874667	0.699009	0.004364
								7.302961	6.984970	6.662998	0.030896

f1 score	7.302961	0.000421	0.003867	0.006407	-0.001549	-0.001413	0.002969	0.008993	Variâncias
Média f1	0.912870	0.000053	0.000483	0.000801	-0.000194	-0.000177	0.000371	0.001124	0.000019
Acurácia	6.984970	-0.003168	-0.001164	0.005302	0.002166	-0.004704	0.005836	0.008098	
Média acurácia	0.873121	-0.000396	-0.000145	0.000663	0.000271	-0.000588	0.000729	0.001012	0.000021
Precisão	6.662998	-0.793744	-0.120610	-0.139882	-0.109072	-0.154680	0.133666	0.113396	
Média precisão	0.832875	-0.099218	-0.015076	-0.017485	-0.013634	-0.019335	0.016708	0.014175	0.091336
Tempo	0.030896	-0.003178	-0.003988	-0.004420	0.005466	0.005454	0.005580	-0.000898	
Média tempo	0.003862	-0.000397	-0.000499	-0.000553	0.000683	0.000682	0.000698	-0.000112	0.000017

Fonte: Elaborado pelos autores.

A figura B.1 mostra a tabela de sinais obtida com as médias dos resultados dos experimentos com o modelo de Árvore de Decisão e com os algoritmos de otimização e suas combinações para o *dataset* CTU-13. Para cada uma das métricas analisadas, foram calculadas as médias e a variação. Os dados obtidos com essa tabela foram usados para a produção da tabela 4.5.

Já a figura B.2, demonstra a tabela de sinais obtida com as médias dos resultados dos experimentos com o modelo de Árvore de Decisão e com os algoritmos de otimização e suas combinações para o *dataset* ISCX-2014. Para cada uma das métricas analisadas, foram calculadas as médias e a variação. Os dados obtidos com essa tabela foram usados para a produção da tabela 4.6.

Figura B.2: Tabela de sinais montada com os dados obtidos para o *dataset* ISCX-2014.

I	WOLF	BAT	WHALE	WOLF BAT	WOLF WHALE	BAT WHALE	WBW	*f1 score*	Acurácia	Precisão	Tempo
1	-1	-1	-1	1	1	1	-1	0.856744	0.866933	0.902627	0.011021
1	-1	-1	1	1	-1	-1	1	0.841146	0.850000	0.890692	0.005286
1	-1	1	-1	-1	1	-1	1	0.813824	0.820533	0.857782	0.005938
1	-1	1	1	-1	-1	1	-1	0.701970	0.700533	0.699087	0.004243
1	1	-1	-1	-1	-1	1	1	0.805793	0.809200	0.830070	0.006230
1	1	-1	1	-1	1	-1	-1	0.696809	0.699733	0.694664	0.004499
1	1	1	-1	1	-1	-1	-1	0.721794	0.717200	0.710884	0.004358
1	1	1	1	1	1	1	1	0.695598	0.701467	0.699009	0.006629
								6.133678	6.165599	6.284815	0.048204

f1 score	6.133678	-0.293690	-0.267306	-0.262632	0.096886	-0.007728	-0.013468	0.179044	Variâncias
Média f1	0.766710	-0.036711	-0.033413	-0.032829	0.012111	-0.000966	-0.001684	0.022381	0.033546
Acurácia	6.165599	-0.310399	-0.286133	-0.262133	0.105601	0.011733	-0.009333	0.196801	
Média acurácia	0.770700	-0.038800	-0.035767	-0.032767	0.013200	0.001467	-0.001167	0.024600	0.037130
Precisão	6.284815	-0.415561	-0.351291	-0.317911	0.121609	0.023349	-0.023229	0.270291	
Média precisão	0.785602	-0.051945	-0.043911	-0.039739	0.015201	0.002919	-0.002904	0.033786	0.060762
Tempo	0.048204	-0.004772	-0.005868	-0.006890	0.006384	0.007970	0.008042	-0.000038	
Média tempo	0.006026	-0.000597	-0.000734	-0.000861	0.000798	0.000996	0.001005	-0.000005	0.000034

Fonte: Elaborado pelos autores.

Na figura B.3, é apresentada a tabela de sinais obtida com as médias dos resultados dos experimentos com o modelo de Árvore de Decisão e com os algoritmos de otimização e suas combinações para o *dataset* ISOT-2010. Para cada uma das métricas analisadas, foram calculadas as médias e a variação. Os dados obtidos com essa tabela foram usados para a produção da tabela 4.7.

Figura B.3: Tabela de sinais montada com os dados obtidos para o *dataset* ISOT-2010.

I	WOLF	BAT	WHALE	WOLF BAT	WOLF WHALE	BAT WHALE	WBW	*f1 score*	Acurácia	Precisão	Tempo
1	-1	-1	-1	1	1	1	-1	0.908288	0.993467	0.927565	0.004043
1	-1	-1	1	1	-1	-1	1	0.912394	0.994000	0.937076	0.002188
1	-1	1	-1	-1	1	-1	1	0.891974	0.992933	0.923394	0.002629
1	-1	1	1	-1	-1	1	-1	0.862641	0.991600	0.901085	0.002013
1	1	-1	-1	-1	-1	1	1	0.884216	0.992667	0.903615	0.002310
1	1	-1	1	-1	1	-1	-1	0.859880	0.992133	0.896983	0.001834
1	1	1	-1	1	-1	-1	-1	0.870411	0.992133	0.919446	0.001965
1	1	1	1	1	1	1	1	0.861868	0.992000	0.912939	0.002443
								7.051672	7.940933	7.322103	0.019425

f1 score	7.051672	-0.098922	-0.077884	-0.058106	0.054250	-0.007652	-0.017646	0.049232	**Variâncias**
Média f1	0.881459	-0.012365	-0.009736	-0.007263	0.006781	-0.000957	-0.002206	0.006154	0.003121
Acurácia	7.940933	-0.003067	-0.003601	-0.001467	0.002267	0.000133	-0.001465	0.002267	
Média acurácia	0.992617	-0.000383	-0.000450	-0.000183	0.000283	0.000017	-0.000183	0.000283	0.000005
Precisão	7.322103	-0.056137	-0.008375	-0.025937	0.071949	-0.000341	-0.031695	0.031945	
Média precisão	0.915263	-0.007017	-0.001047	-0.003242	0.008994	-0.000043	-0.003962	0.003993	0.001387
Tempo	0.019425	-0.002321	-0.001325	-0.002469	0.001853	0.002473	0.002193	-0.000285	
Média tempo	0.002428	-0.000290	-0.000166	-0.000309	0.000232	0.000309	0.000274	-0.000036	0.000003

Fonte: Elaborado pelos autores.