FACULDADE DE TECNOLOGIA DE AMERICANA
Curso de Tecnologia em Segurança da Informação

Tulio Cruvinel Gomes

APRENDIZADO DE MÁQUINA NA DETECÇÃO DE TRÁFEGO DE BOTNET

Americana, SP
2020

FACULDADE DE TECNOLOGIA DE AMERICANA
Curso de Tecnologia em Segurança da Informação

Tulio Cruvinel Gomes

APRENDIZADO DE MÁQUINA NA DETECÇÃO DE TRÁFEGO DE BOTNET

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso de Tecnologia em Segurança da Informação, sob orientação do(a) Prof. Me. Rossano Pablo Pinto.

Área de concentração: Segurança da Informação

Americana, SP
2020

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

G618a GOMES, Tulio Cruvinel

Aprendizado de máquina na detecção de tráfego de Botnet. / Tulio Cruvinel Gomes. – Americana, 2020.

74f.

Monografia (Curso Superior de Tecnologia em Segurança da Informação) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Rossano Pablo Pinto

1 Segurança em sistemas de informação 2. Inteligência artificial 3. Aprendizado de máquina I. PINTO, Rossano Pablo II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: *681.518.5*

007.52

Tulio Cruvinel Gomes

APRENDIZADO DE MÁQUINA NA DETECÇÃO DE TRÁFEGO DE BOTNET

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso de Tecnologia em Segurança da Informação, sob orientação do(a) Prof. Me. Rossano Pablo Pinto.

Área de concentração: Segurança da Informação

Americana, 2 de Dezembro de 2020.

Banca Examinadora:

Rossano Pablo Pinto
Mestre
Fatec American Ministro Ralph Biasi

Daniele Junqueira Frosoni
Especialista
Fatec American Ministro Ralph Biasi

Diógenes de Oliveira
Mestre
Fatec American Ministro Ralph Biasi

FACULDADE DE TECNOLOGIA DE AMERICANA
Curso de Tecnologia em Segurança da Informação

Tulio Cruvinel Gomes

APRENDIZADO DE MÁQUINA NA DETECÇÃO DE TRÁFEGO DE BOTNET

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso de Tecnologia em Segurança da Informação, sob orientação do(a) Prof. Me. Rossano Pablo Pinto.

Área de concentração: Segurança da Informação

Americana, SP
2020

AGRADECIMENTOS

Ao Prof. Me. Rossano Pablo Pinto, por aceitar conduzir o meu trabalho de monografia, pela paciência e principalmente por ter me auxiliado em desvendar os fundamentos da inteligência artificial.

À FATEC Piracicaba, instituição onde trabalho que me proporcionou suporte e flexibilidade durante a graduação.

À Alexandra Elbakyanpela criação do Sci-Hub, site que contribuiu para o acesso ao conhecimento científico no mundo.

Aos meus pais, familiares e amigos, que acompanharam o impacto positivo da graduação no meu desenvolvimento pessoal e sempre me apoiaram durante o percurso.

À Fernanda Guerra, pela compreensão, paciência e principalmente por estar presente nos momentos mais difíceis durante essa jornada.

EPÍGRAFE

"Ser é abdicar" - Fernando Pessoa.

RESUMO

Este trabalho apresenta o desenvolvimento de modelos de aprendizado de máquina para a detecção do tráfego de botnets, bem como a detecção de ataques do tipo port scanning de origem de botnets. A validação do modelo é explorada em um cenário para captura de tráfego de ataques oriundos de botnets utilizando a ferramenta tcpdump e posteriormente convertidos para tipo Netflow utilizando o software ARGUS. A aplicação utiliza o dataset CTU-13, sendo um conjunto de dados realístico. O pré-processamento dados aborda diversas combinações de atributos e datasets para comparação de resultados e comprovar a eficácia de cada um dos algoritmos de aprendizado de máquinas explorado dentre as diversas métricas apresentadas.

Palavras-chave: Aprendizado de Máquina. Botnet. Netflow. Port Scanning.

ABSTRACT

This work presents the development of machine learning models for the detection of botnet traffic, as well as the detection of botnet port scanning attacks. The model validation is explored in a scenario to capture traffic from attacks coming from botnets using the tcpdump tool and later converted to Netflow type using the ARGUS software. The application uses the CTU-13 dataset, being a realistic data set. The data preprocessing addresses several combinations of attributes and datasets for comparing results and proving the effectiveness of each of the machine learning algorithms explored among the various metrics presented.

Keywords: Machine Learning. Botnet. NetFlow. Port Scanning.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ARGUS	Audit Record Generation and Utilization System
ARP	Address Resolution Protocol
AUC	Area Under Curve
C&C	Command and Control
DDoS	Denial of Service
DT	Decision Tree
GPL	General Public License
ICMP	Internet Control Message Protocol
IoT	Internet of Things
IP	Internet Protocol
kNN	k-Nearest Neighbors
ML	Machine Learning
NB	Naive Bayes
NMAP	Network Mapper
PCAP	Packet Capture
RF	Random Forest
RLOG	Regressão Logística ou Logistic Regression
ROC	Receiver Operating Characteristics
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

SUMÁRIO

1 INTRODUÇÃO

Com o crescente aumento da rede de computares e a necessidade constante de diversos equipamentos ligados à ela, qualquer aplicação ou serviço desenvolvido tende a ser dependente de conexão com redes internas ou à internet (STONE-GROSS *et al.*, 2009). Dado isso, com mais equipamentos disponíveis em rede, maior o interesse de usuários mal intencionados para aproveitar das vulnerabilidades dos serviços e aplicações. O aumento das atividades maliciosas se deve também à ineficiência das atuais soluções em identificar, reduzir e interromper por completo esses tipos de atividades.

Para realizar muitas dessas ações maliciosas passaram a ser empregadas estruturas denominadas botnets, que são definidas como uma rede de máquinas infectadas por um software malicioso chamado de bot, que é controlado remotamente por um ou mais atacantes, denominados como botmasters (DAGON *et al.*, 2008; FEILY; SHAHRESTANI; RAMADASS, 2009). Essas estruturas tem aumentado em uma quantidade que os torna um grande desafio na área de segurança da informação. Como uma das maiores ameaças da internet, cita-se como exemplo a botnet *3ve*, que realizou fraudes publicitárias infectando 1,7 milhões de computadores e servidores, falsificou 5.000 sites para se passar por editores legítimos da web e utilizou-se 60.000 contas de empresas de publicidade digital para lucro com os anúncios recebidos, onde é estima-se um valor de $19 bilhões de dólares que foram roubados por esta prática em 2018 (BUZZFEED NEWS, 2018). As botnets tornaram-se uma das maiores ameaças na internet. A detecção dessas redes se tornou um desafio, já que as botnets são muito flexíveis e robustas, além de estarem em processo de aprimoramento contínuo.

Após um bot infectar a máquina, os antivírus nem sempre são capazes de identificar ou remover o bot, dado aos constantes aprimoramentos, tornando então a detecção de um botnet sendo uma tarefa complexa. Segundo Feily, Shahrestani e Ramadass (2009) as botnets possuem um ciclo de vida, então pode-se obter os padrões comportamentais a partir destes ciclos como o exemplo da fase de conexão com o C&C, sendo assim possível coletar o tráfego gerado por qualquer ciclo que compartilhe as informações via rede. Com isso, o uso do aprendizado de máquina para o problema para detecção de tráfego de botnet tem sido mais pesquisado e estudado (Biglar Beigi *et al.*, 2014a). Extraído então os padrões e atributos existentes no fluxo de pacotes, pode-se utilizar o aprendizado de máquina para a classificação do tráfego malicioso (FEILY; SHAHRESTANI; RAMADASS, 2009; SILVA; SILVA; SALLES, 2017; NEIRA; MEDEIRO; NOGUEIRA, 2020). É evidente a necessidade de uma aplicação que permita realizar a detecção, principalmente quando observa-se que de forma manual a análise de tráfego se torna complexa e exaustiva devida a grande quantidade de informações que trafegam em uma rede.

1.1 Motivação e Justificativa

Diversos incidentes de segurança tem ocorrido por origem de botnets, o que motiva a investigação de técnicas e estudos para a prevenção ou identificação de botnets na rede, porém a detecção de tráfego de botnets é um grande desafio para a área de segurança da informação devido aos desenvolvedores sempre aprimorarem o funcionamento dos bots e consequentemente as aplicações elaboradas falham após as novas implementações, reformulação de arquitetura ou desenvolvimento de novos botnets (EC-COUNCIL, 2019). Dadas estas características, o presente estudo busca desenvolver uma aplicação para auxiliar na detecção de botnets utilizando o tráfego de rede e classificando através dos algoritmos de aprendizado de máquina

supervisionado.

1.2 Objetivos

Este estudo tem como objetivo desenvolver uma aplicação de detecção de tráfego de origem de botnets, utilizando algoritmos de aprendizado de máquina e métodos de métricas para validar a classificação e comparar os diversos comportamentos dos algoritmos propostos. Esta ferramenta deverá ser capaz de ler informações de um arquivo tipo *PCAP* e aplicar um classificador para identificar se há tráfego de botnet ou não.

1.3 Objetivos Específicos

Dentre os objetivos específicos, esta aplicação aborda os seguintes tópicos:

- Explorar novos atributos para a composição do dataset.
- Aplicar diferentes composições de datasets.
- Explorar uma nova classe alvo, determinando se é tráfego normal, tráfego botnet ou ataque de origem botnet.
- Explorar a detecção de ataques do tipo *Port Scanning*.
- Explorar as métricas abordadas no estudo, comparando de forma profunda dentre os diversos subgrupos de datasets, algoritmos utilizados e a classificação final desenvolvida pela aplicação proposta.
- Determinar o melhor algoritmo de aprendizado de máquina supervisionado para detecção e para ataques do tipo *Port Scanning*.

1.4 Organização da Monografia

A estrutura da monografia é composta por cinco capítulos que são apresentados da seguinte forma:

- Capítulo 1 - Descreve uma introdução sobre o tema, bem como os objetivos gerais e específicos do estudo e a organização do decorrer desta monografia.
- Capítulo 2 - Aborda os fundamentos para o entendimento do estudo, como: Botnets, Netflow, Aprendizado de Máquina e Métricas de Validação.
- Capítulo 3 - Aborda os algoritmos de aprendizado de máquina mais utilizados para a detecção de fluxo de tráfego de botnets, descrevendo então seus conceitos básicos e modelos matemáticos.
- Capítulo 4 - Descreve o cenário virtual elaborado para a captura de fluxos de tráfego específico, bem como o dataset utilizado, tratamento de dados, seleção de atributos, pré-processamento e modelagem dos modelos de aprendizado de máquina.
- Capítulo 5 - Exibe os resultados obtidos para cada dataset utilizado, apresentando então as métricas abordadas no Capítulo 2, o comportamento dos modelos dado os tipos de objetivo de cada dataset e uma conclusão a respeito do modelo elaborado neste estudo.
- Capítulo 6 - Contém a conclusão, e considerações finais a respeito da monografia, bem como sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta a fundamentação teórica necessária para o desenvolvimento deste estudo. Os conceitos básicos abordados são: botnets, netflow, aprendizado de máquina, métricas estáticas de resultados. A contribuição deste capitulo é essencial para o entendimento do decorrer do estudo, como o exemplo do Capítulo 3 que é uma extensão da Seção 2.3, abordando então a composição e modelos matemáticos dos algoritmos utilizados.

2.1 Botnet

Botnet é um conjunto de bots (robôs), onde cada bot escraviza a máquina alvo e a torna em uma máquina infectada. Uma botnet pode ser controlada por um ou mais botmasters, sendo este o termo para definir quem realiza o ataque e envio de comandos para a botnet. Na literatura também é empregado outro termo, onde Santos *et al.* (2015) descreve as botnets como redes zumbis. A principal diferença de um *malware* para uma botnet é a existência do Comando & Controle (C&C), que permite coordenar de forma simultânea toda a infraestrutura e distribuir ataques de forma centralizada. A maioria dos sistemas operacionais infectados são de computadores, porém existe um crescente número de bots em plataformas mobile e IoT (ANTONAKAKIS *et al.*, 2017; DANGE, 2019). Esses *malwares* são distribuidos pela internet por auto-propagação com a intenção de aumentar a quantidade de bots a botnet a qual pertence. Com facil propagação e difícil detecção, é possível realizar *phishing*, ataques *DDoS*, reconhecimento de redes, ataques e crimes cibernéticos em geral.

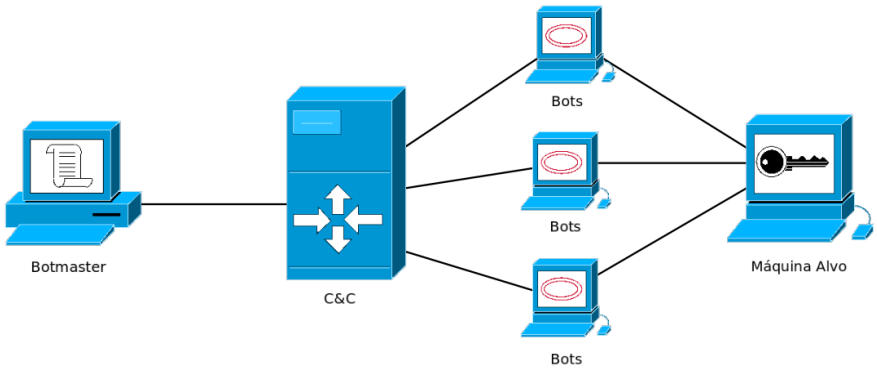
Apesar da arquitetura distinta, a caracterização de uma botnet possui elementos essenciais para a compreensão de seu funcionamento e entendimento da autonomia gerada pela botnet, que pode ser visualizado pela Figura 1 e possui os seguintes elementos:

- Botmaster: Usuário malicioso que controla a botnet, podendo ou não participar do desenvolvimento da mesma. Há diversos perfis deste usuário, usualmente possuem um conhecimento específico dentro da área de ciências da computação e segurança da informação. Há a possibilidade de pessoas sem qualquer orientação em computação controlar uma botnet, como os *ScriptKids*, termos para quem replica um ataque por um roteiro sem possuir qualquer conhecimento sobre a ferramenta.
- Bot: É um *malware* instalado em uma máquina vulnerável, capaz de executar uma série de ações coordenadas pelo botmaster por meio do *C&C*. Geralmente o *malware* é inicializado quando a máquina infectada é iniciada, dificultando então a detecção.
- Comando e Controle (*C&C*): É a entidade que realiza o controle dos bots, como intermédio de comunicação entre o botmaster e as máquinas infectadas. É considerado o componente mais crítico de uma botnet, já que seu projeto determina as caraterísticas de estabilidade e tempo de reação dos bots.
- Hosts: São as máquinas ou sistemas operacionais que são alvos para o botmaster.

2.1.1 Ciclo de Vida das Botnets

Em sua maioria, existe um ciclo de vida pré-definido para a botnet ser criada e mantida no ambiente das máquinas infectadas. A primeira fase é composta pela injeção inicial, onde o atacante realiza a etapa de reconhecimento de vulnerabilidades da máquina para posterior realizar a exploração e tornar o host em uma máquina infectada através do *malware*. Feito a etapa de reconhecimento, se utiliza as vulnerabilidades obtidas pelo *malware* para orientar

Figura 1 – Botnet - Elementos da Botnet



Fonte: Autor

uma conexão com os binários originais do bot e concluir a etapa de transformar a máquina alvo em um bot com todas as ferramentas necessárias (FEILY; SHAHRESTANI; RAMADASS, 2009). Após o bot estar estável na máquina, é efetuado a etapa de conexão com o *C&C* e que normalmente se repete sempre que a máquina infectada é iniciada, caracterizando-se então um padrão de conexão. Após realizado a conexão, o bot está apto para receber os comandos do botmaster, onde é possível efetuar os ataques, manutenção ou implementação de novos recursos para o bot (GONÇALVES, 2017).

2.2 Netflow

O *Netflow* é um padrão de sumarização de tráfego criado pela *CISCO Systems*[1], baseado no conceito de fluxo semelhante a uma sessão TCP, que coleta e mede os dados conforme as interfaces específicas de roteadores e switches (COLLINS, 2014). Os roteadores que executam o *Netflow* mantêm um "cache de fluxo" contendo registros que descrevem o tráfego encaminhado pelo roteador, que são exportados usando o protocolo UDP para um computador para coleta, armazena e analisa (FRANÇOIS *et al.*, 2011). O *Netflow* tem sido cada vez mais usado para análise de segurança desde a sua publicação devido a análise de tráfego de rede por meio de registros de fluxo ser mais eficiente do que por meio de pacotes individuais, pois fornece um resumo compacto das sessões de tráfego de rede dado a menor quantidade de dados para análise e contém as informações de maior valor de forma compacta, já que os registros de fluxo requerem apenas uma fração do espaço de armazenamento necessário para a captura total do pacote.

Todos os datagramas com o mesmo valor em tais campos são contados, agrupados e empacotados em um registro de fluxo. O *Netflow* caracteriza um fluxo como um conjunto de dados com os seguintes atributos e que aparecem com o mesmo valor, como exemplo: endereço IP de origem/destino, porta de origem/destino referente ao protocolo (TCP/UDP), o valor do campo Protocol do datagrama IP, a interface lógica de entrada do datagrama no roteador ou switch e dentre outros (BILGE *et al.*, 2012; COLLINS, 2014). Os campos *srcaddr, dstaddr, srcport, dstport, prot e tos* de um registro Netflow são copiados diretamente dos campos correspondentes nos pacotes IP enquanto os campos *packets, first, last e tcp_flags* resumem o tráfego de um ou mais pacotes. Registros do tipo *Netflow* podem ser obtidos de duas formas,

[1] Disponível em: https://www.cisco.com/ e Acessado em 20 de novembro de 2020

sendo: gerados diretamente pelo hardware de roteadores e switches ou usando softwares para converter os pacotes em fluxos de tipo *Netflow* (COLLINS, 2014).

2.3 Aprendizado de Máquina

O aprendizado de máquina, do termo em inglês *Machine Learning (ML)*, é um campo de estudo da Inteligencia Artificial onde a proposta é desenvolver um software que tem a capacidade de aprender. Esse aprendizado é feito a partir de um treinamento, onde são inseridos os dados que através de um processo os transforma em conhecimento para executar alguma tarefa ou ser aplicado a resolução de problema através do reconhecimento de padrões (MOHAMMED; KHAN; BASHIE, 2016). Em outros estudos, Nilsson (1998) complementa que podemos considerar o aprendizado como a partir do momento que sua estrutura é alterada, baseada nos dados inseridos e isso impacta nos resultados obtidos da aplicação. *ML* é sobre resolver problemas e garantir os resultados através de modelos matemáticos, fluxo de dados e modelos de problemas determinísticos (SMOLA; VISHWANATHAN, 2008).

Segundo Harrington (2012), *ML* é um encontro entre as áreas de Ciências da Computação, Engenharia e Estatística. Tem sido aplicada em diversas áreas do conhecimento humano e devido a sua eficácia, tem se popularizado pela mídia a partir de 2010 e aplicada por várias empresas como Amazon, Netflix, Google e Facebook. Suas aplicações finais mais comuns são motores de busca, reconhecimento facial, reconhecimento de voz, detecção de fraudes, sistemas de recomendações, reconhecimento de manuscrito, processamento de linguagem natural e dentre tantas outras aplicações. Qualquer situação onde se tenha um problema e que ele possua um padrão, então ele pode ser classificado e solucionado por *ML*.

No aprendizado de máquina é comum escrevermos um modelo a partir do problema, inserir os dados de entrada e os resultados desejados, porém este processo de aprendizagem pode se tornar difícil para exemplificar pela linguagem e através dos modelos matemáticos e probabilísticos podemos reconhecer os padrões usados e isolar as variáveis para trazer um maior entendimento sobre o modelo do problema. É esse o grande trunfo que o *ML* traz e que o torna valioso, já que ele chega aos resultados dos problemas do modelo porém ele pode também reconhecer outros padrões e problemas que não foram reconhecidos devido a barreira da linguagem humana (MOHAMMED; KHAN; BASHIE, 2016; HARRINGTON, 2012).

Por isso possuímos alguns modelos de aprendizado, que facilitam a automatizam da construção de modelos analíticos, que são: Aprendizado supervisionado, aprendizado não supervisionado, aprendizado semi-supervisionado e aprendizado por reforço.

2.3.1 Aprendizado Supervisionado

No aprendizado supervisionado prevemos onde uma instância de dados pode ser encaixar dentro de um conjunto de dados já rotulados. Este tipo de modelo torna claro para o algoritmo o que queremos prever. Neste tipo de aprendizado, (SHALEV-SHWARTZ; BEN-DAVID, 2013) diz que é semelhante ao processo de aprendizagem humano, onde acontece a interação de aluno e ambiente. Podemos considerar o ambiente como o professor que "Supervisiona" o aluno e fornece um conjunto de informações significativas (labels ou classes) e aplicando o processo de aprendizagem, podemos treinar o algoritmo aluno para "transformar a experiência em expertise". Frequentemente os supervisores são humanos, porém máquinas também podem ser usadas para essa rotulagem, apesar de que os dados rotulados manualmente tendem a ser um recurso mais preciso e confiável para o aprendizado de máquina, o que naturalmente torna os julgamentos humanos mais caros do que as máquinas Mohammed, Khan e Bashie (2016).

Em aprendizado supervisionado, temos duas categorias de algoritmos: Classificação e Regressão. Algoritmos de classificação são aplicados onde a variável de saída é uma categoria que já foi rotulada e consta como um atributo do dataset. Como exemplos de algoritmos de classificação, temos: kNN, Naive Bayes e Decision Trees. Os algoritmos de Regressão são quando precisamos fazer uma predição de um valor real ou numérico, como exemplos de algoritmos de regressão, temos: Linear Regression e SVM.

2.3.2 Aprendizado Não Supervisionado

No aprendizado não supervisionado é o treinamento da máquina usando informações que não são classificadas nem rotuladas, permitindo assim que o algoritmo processe as informações sem orientação ou supervisão. Neste momento, a tarefa da máquina é agrupar informações não classificadas de acordo com semelhanças, padrões e diferenças sem nenhum treinamento prévio de dados (MOHAMMED; KHAN; BASHIE, 2016).

Por não possuir supervisão, nenhum treinamento será dado à máquina, restringindo a ela encontrar os padrões ocultos nos dados não rotulados.

Em aprendizado não supervisionado, temos duas categorias de algoritmos: Clustering e Associação. Algoritmos de Clustering são aplicados quando precisamos encontrar grupos distintos dentro do nosso dataset. Algoritmos de Associação são quando precisamos encontrar uma relação ou dependência dentro do conjunto de dados do dataset. Regras de clustering e associação são essenciais para a mineração de dados para a extração de conhecimento.

2.4 Métricas de Resultados

A métrica mais utilizada na avaliação e seleção dos modelos é a *Acurácia* (taxa de erro), que pode ser descrita como o número de predições corretas dividido pelo número total de predições. Essa metodologia é um padrão utilizado em aprendizado de máquina, que minimiza a probabilidade do erro global, mas para problemas onde existe um desbalanceamento das instâncias, a acurácia não é o melhor tipo de métrica caso o seu grupo específico de interesse seja de uma classe minoritária (VISA *et al.*, 2011). Como exemplo, mesmo que o classificador possua uma alta taxa de acurácia, ele torna-se inefetivo para a classificação das classes minoritárias, exatamente por associa-las a uma classe majoritária. Para contornar este tipo de resultado desbalanceado, nessa seção será abordado as métricas de matriz de confusão e a validação cruzada para a composição dos resultados obtidos.

2.4.1 Matriz de Confusão

Uma matriz de confusão contém informações sobre as classificações reais e previstas feitas por um modelo de classificação. O desempenho é avaliado usando os dados da matriz e sendo possível visualizar cada classificação para cada classe. A matriz de confusão é composta por $n \times n$, onde *n* é a quantidade de classes existentes (STEHMAN, 1997; MACHART; RALAIVOLA, 2012). As linhas da matriz representam as instâncias em uma classe prevista, enquanto cada coluna representa a predição das classes. Assim, os elementos ao longo da diagonal representam as classificações corretas, sendo: verdadeiro negativo (TN) e verdadeiro positivo (TP). Qualquer elemento fora da diagonal representa os números de classificações realizadas de forma errônea, sendo: falsos positivos (FP) e falsos negativos (FN) (CASTRO; BRAGA, 2011; STEHMAN, 1997). A Tabela 1 representa uma matriz de confusão para um classificador de duas classes (True e False).

Tabela 1 – Matriz de Confusão Binária

	Predição - True	Predição - False
Valor Real - True	TN	FP
Valor Real - False	FN	TP

Fonte: Autor

A partir da Tabela 1, é possível extrair diversas métricas para avaliar o desempenho das classes positivas e negativas, Este estudo tem como objetivo elaborar uma aplicação de detecção de tráfego e ataques do tipo de *Port Scanning* de origem de botnets, utilizando aprendizado de máquina e métodos de métricas para validar a classificação e comparar os diversos comportamentos dos algoritmos propostos, como: Sensibilidade, Especificidade, Acurácia e Precisão (CASTRO; BRAGA, 2011). A *Acurácia* é a métrica mais utilizada como citada na Seção 2.4 e é representada pela Equação (3). A *Sensibilidade*, dada pela Equação (1), é a proporção de exemplos positivos que foram corretamente classificados, enquanto a *Especificidade*, dada pela Equação (2), é a proporção dos exemplos negativos corretamente classificados. A *Precisão*, dada pela Equação (4), refere-se à proporção de casos corretamente classificados como positivos em relação ao total de casos classificados como positivos, ou seja $TP + FP$.

Sensibilidade (True Positive Rate):

$$TPr = \frac{TP}{TP + FN} \tag{1}$$

Especificidade (True Negative Rate):

$$TNr = \frac{TN}{TN + FP} \tag{2}$$

Acurácia (Correct Prediction Rate):

$$CPr = \frac{TN + TP}{TN + TP + FN + FP} \tag{3}$$

Precisão (Predicted Positives Rate):

$$PPr = \frac{TP}{TP + FP} \tag{4}$$

2.4.2 Validação Cruzada - K-Fold

A validação cruzada é uma técnica para avaliar a capacidade de generalização de um modelo a partir de um conjunto de dados, onde sua finalidade é estimar a precisão do modelo e qual o seu desempenho a partir de novos conjuntos de dados (CUNHA, 2019). Particionando o conjunto de dados em subconjuntos de treinamento e validação, eles serão utilizados como estimativa dos parâmetros do modelo. Dentre os diversos métodos, os três mais utilizados são: Holdout, K-fold e o leave-one-out.

O método *K-fold* divide o dataset *d* dado o valor de *k*. O processo terá *K* iterações onde a amostra de validação será dada por d_k, sendo que $k = 1,2,3,...K$ e a amostra de treino para a criação da predição será o conjunto de outras $K - 1$ amostras. A cada K final, todos

os dados de treino e validação seriam utilizados pelo estimador. A estimativa de precisão da validação cruzada é o número total de classificações corretas dividido pelo número de instâncias no conjunto de dados (CUNHA, 2019; BENGIO; GRANDVALET, 2004). A métrica de *K-fold* é descrita pela Equação (5).

K-fold:

$$CV(D) = \frac{1}{K}\sum_{k=1}^{k}\frac{1}{m}\sum_{z_i \in T_k} L(A(D_k, Z_i)) \tag{5}$$

O valor de *K* é variado de acordo com os modelos utilizados, porém os valores elevados tendem a aumentar consideravelmente o custo computacional da técnica. Na literatura, os valores mais comuns citados pelos autores são a variação de *K*=2,3,5 ou 10 (Rodriguez; Perez; Lozano, 2010; BROWNE, 2000; BENGIO; GRANDVALET, 2004).

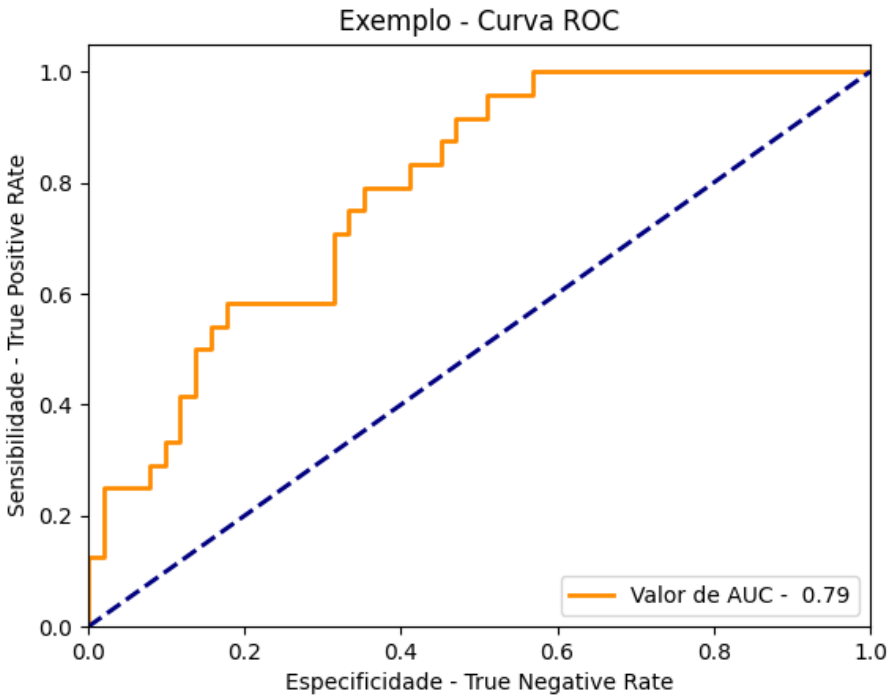
2.4.3 Receiver Operating Characteristics

A curva *ROC*, do termo em inglês *Receiver Operating Characteristics*, é um método gráfico que permite avaliar e comparar sistemas de predição a fim de favorecer a visualização do problema (FAWCETT, 2006). A curva *ROC* é uma representação bidimensional do desempenho de um classificador, dado o eixo y como a métrica Sensibilidade, representada pela Equação (1) como a taxa de verdadeiros positivos (TPr) e no eixo x a Especificidade, representado pela Equação (2), como a taxa de falsos positivos (TNr) (FAWCETT, 2006; BRAGA, 2001).

O desempenho do classificador é avaliado adotando a curva *ROC*, que sempre se apresenta convexa e crescente. Quando comparados os algoritmos classificadores, a curva que mais se aproxima do ponto (0,1) é a de melhor desempenho, dado como uma classificação perfeita, enquanto o ponto inferior esquerdo (0,0) representa que uma classificação nunca será positiva (BRAGA, 2001). Apesar do ponto superior esquerdo ser o ponto ideal citado, não é uma definição de carácter realista, então podemos extrair a área maior sobre a curva, do termo em inglês *Area Under Curve (AUC)* e comparar de forma mais objetiva a diferença de desempenho dos algoritmos classificadores. Também é importante observar-se a inclinação das curvas *ROC*, pois ela maximiza a taxa de verdadeiros positivos enquanto minimiza a taxa de falsos positivos (PRATI; BATISTA; MONARD, 2008).

A Figura 2 exibe um exemplo de desempenho de um classificador, visualizando então a inclinação e a curva *ROC*, bem como a métrica *AUC* com o valor de 0.79.

Figura 2 – Curva ROC - Exemplo



Fonte: Autor

3 MODELOS DE APRENDIZADO DE MÁQUINA PARA DETECÇÃO DE TRÁFEGO DE BOTNET

Neste trabalho, foram utilizados quatro algoritmos de *ML* a fim de comparar os resultados do aprendizado e escolher o melhor modelo para o problema. Esse capítulo apresenta uma introdução e os modelos matemáticos para cada algoritmo utilizado, complementando a Seção 2.3. A aplicação dos algoritmos é abordada na Seção 4.5 e na Apêndice A consta o código utilizado.

3.1 Naive Bayes

O Naive Bayes (NB) é um algoritmo do tipo probabilístico para classificação binária e de múltiplas classes, baseado no *Teorema de Bayes* que é representado pela Equação (6). Sua aplicação é mais utilizada em classificação de texto, como análise de sentimento, ou problemas onde existem múltiplas classes. É chamado Naive, ou Ingênuo, porque os cálculos das probabilidades para cada classe são simplificados e parte da suposição que todos os atributos são independentes dado a classe alvo, ignorando então as possíveis dependências, correlações e grupos de problemas. Em aplicações no mundo real, esta característica o torna muito efetivo devido ser recorrente que os atributos que compõem o dataset não tenham dependências entre si (MOHAMMED; KHAN; BASHIE, 2016; HARRINGTON, 2012).

Teorema de Bayes:

$$P(A/B) = \frac{P(B/A)P(A)}{P(B)} \tag{6}$$

Uma vantagem do *NB* é que ele precisa de um pequeno número de dados de treinamento para estimar a classificação, o que resulta em um menor consumo de memória de processamento. Há três variáveis que influenciam consideravelmente no poder de precisão do *NB*, sendo: ruido no dataset de treino, a tendência e a variação. O impacto pelo dataset de treino pode ser reduzido ao se escolher um bom dataset com a menor quantidade de ruido, a tendência é os erros atribuídos aos grupos de classe do próprio dataset de treino e erros de variança são atribuídos quando o dataset é muito pequeno ou que os grupos de classes não possuem instâncias o suficiente (MUKHERJEE; SHARMA, 2012; SHALEV-SHWARTZ; BEN-DAVID, 2013).

3.1.1 Classificação Bayesiana

Dado a Equação (7), a C_i representa as *i* classes alvos, $Data$ representa um vetor de dados e P representa a probabilidade condicional. Dado um conjunto de *i* classes $C = (c1, c2, c3, ..., c_i)$, queremos atribuir uma classe *C* para uma instância D. O NB calcula a probabilidade de $P(C|D)$ de uma instância pertencer a determinada classe a partir da probabilidade $P(C)$ da instância ser da mesma classe e das probabilidades condicionais de cada atributo ocorrer em uma instância. O objetivo do algoritmo é encontrar a melhor classe para a instância caso ela maximize sua probabilidade da instância *D* (MOHAMMED; KHAN; BASHIE, 2016; NILSSON, 1998).

Classificação Bayesiana:

$$P(C_i|D) = \frac{P(D|C_i)P(C_i)}{P(D)} \tag{7}$$

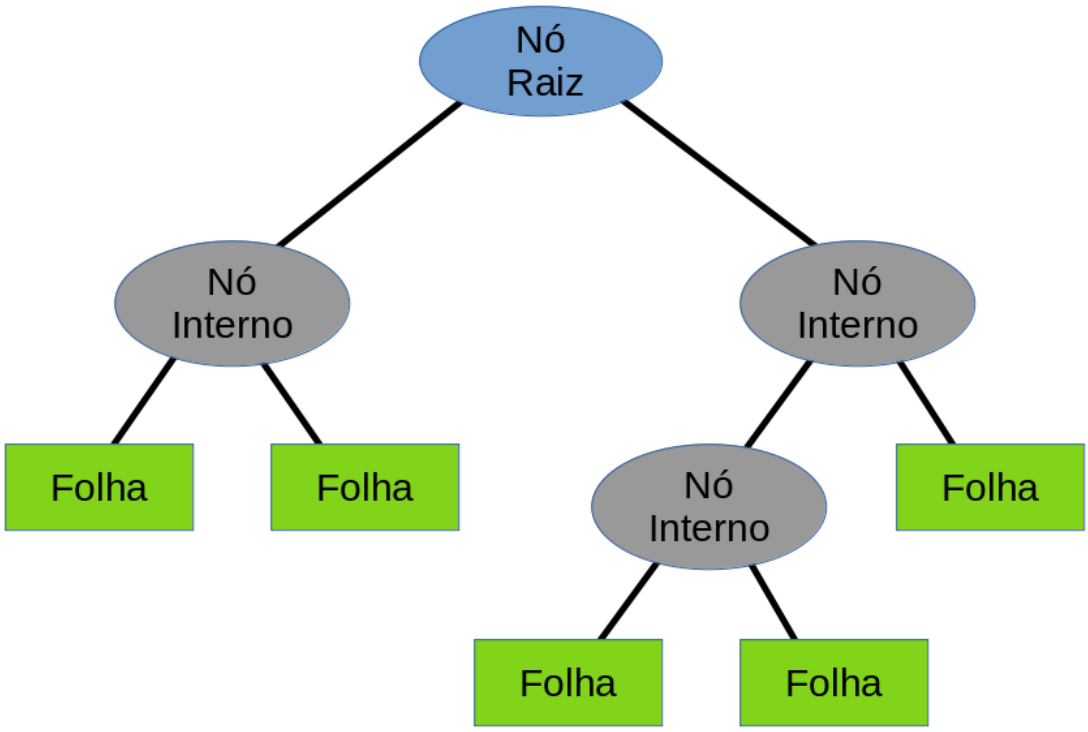
Para evitar o *Underflow*, um resultado de um cálculo onde o número de valor absoluto é menor do que o computador pode realmente representar na memória, o resultado das probabilidades é substituído pela soma dos logaritmos das probabilidades (HARRINGTON, 2012).

3.2 Decision Trees

O algoritmo de Decision Tree (DT) é definido como um modelo estatístico usado para classificação. Sua abordagem é classificar os dados em classes e representar os resultados em fluxogramas, semelhante a uma estrutura de uma árvore, percorrendo do nó raiz até as folhas. A raiz (root) representa o atributo principal da classificação, seus ramos (branches) representam a direção das decisões e suas folhas (leaf) representam a classe que será atribuída ao dado de entrada. Cada sucessor do nó raiz é definido pelas regras de divisão e percorrem caminho raiz $\rightarrow$ folha a fim de determinar uma característica específica (MOHAMMED; KHAN; BASHIE, 2016; SHALEV-SHWARTZ; BEN-DAVID, 2013).

A Figura 3 exibe um exemplo do fluxograma de *DT*. O nó raiz é a primeira característica que divide o dataset, o valor de entrada é comparado a ele e suas respostas são direcionadas pelos ramos até um próximo nodo interno. Este ciclo se repete pelos nós internos até encontrar uma folha, do qual é possível atribuir uma classe ao valor de entrada. Uma grande vantagem é que o fluxograma gerado pelo *DT* pode ser compreendido com facilidade por humanos, mesmo não tendo um prévio conhecimento estatístico ou estar familiarizado com o algoritmo (HARRINGTON, 2012).

Figura 3 – Fluxograma - Decision Tree



Fonte: Autor

Uma das características que compõem o *DT* é ele ser capaz de utilizar dados multivalo-

rados, que podem ser numéricos e não numéricos. Outra característica é que os resultados do aprendizado devem possuir duas ou mais saídas, sendo que se houver apenas duas saídas para classificação, podemos chamar este modelo de *Binary Decision Tree*. Apesar do algoritmo *DT* aceitar mais de duas classes, existe uma categoria específica chamada de *Boolean Decision Tree*, que se caracteriza quando possuímos duas classes binárias de entrada (NILSSON, 1998).

O maior problema de *DT* aplicado a atributos binários, é escolher qual é a melhor ordem para se iniciar os testes, diminuindo assim o tamanho da árvore e consequentemente o poder computacional necessário. Existem diversos métodos para selecionar a melhor ordem do atributo divisor ao se construir o *DT*, sendo que elas podem ser encontradas em áreas como: Teoria da informação, análise discriminante e técnicas de codificação. Os mais comuns são algoritmos *Interative Dichotomiser 3 (ID3)* e *C4.5*, sucessor do *ID3*. O *ID3* utiliza-se do conceito da *Entropia* para extrair do *Information Gain* de todos os atributos, assim como o *C4.5* utiliza-se do *Gain Ratio* (MOHAMMED; KHAN; BASHIE, 2016; RAILEANU; STOFFEL, 2004).

Podemos então observar que o *DT* percorre as seguintes etapas:

- Insere todos os exemplos de treino como raízes.
- Seleciona o atributo do nó raiz seguindo algum método de métrica.
- Seus ramos direcionam a um novo nó, o qual o particionamento recursivo continua até que nenhum exemplo de treinamento reste, ou até que nenhum atributo reste, ou os exemplos de treinamento restantes pertençam à mesma classe.

3.2.1 Entropia de Shannon

A entropia é usada na teoria da informação para determinar a pureza ou impureza de determinado conjunto. Se a amostra é completamente homogênea, a entropia é zero e se a amostra é igualmente dividida, possui uma entropia igual a um (MOHAMMED; KHAN; BASHIE, 2016; SHALEV-SHWARTZ; BEN-DAVID, 2013). Dado uma tabela e obtendo $p(X_i)$, que representa a probabilidade de escolher aquela classe e dado *n* como o número de classes, podemos utilizar a Equação (8) para calcular a entropia de Shannon.

Entropia de Shannon:

$$H(X) = -\sum_{i=1}^{n} p(X_i) \log_2 p(X_i) \tag{8}$$

Um método popular ao se utilizar os algoritmos ID3 e C4.5 é o *Information Gain* dado pela Equação (9), onde ele é a diferença entre a entropia da classe antes e depois da divisão, buscando encontrar atributos que retornem resultados mais homogêneos para a divisão do dataset.

Information Gain:

$$IG(X,Y) = H(X) - H(X|Y) \tag{9}$$

Como um exemplo prático, a entropia da Tabela 2 pode ser calculada da seguinte forma: $Entropia(Gostam\_de\_ML) = Entropia(4{,}11)$. Alterando então os valores pela probabilidade dos eventos, podemos dar continuidade a equação como: $Entropy(0.27{,}0.73) = -(0.27log_20.27) - (0.73log_20.73) = 0.8366$.

3.3 Regressão Logística

A Regressão Logística (*RLOG*) é um algoritmo de classificação usado de forma binária ou multilinear em um conjunto discreto de classes. A *RLOG* utiliza um método de otimização

Tabela 2 – DT - Exemplo Entropia - Gostam de ML

Sim	Não
11	4

Fonte: Autor

e sua saída será redirecionada para a função sigmoide, que retorna um valor de probabilidade entre o intervalo $(0,1)$. O objetivo é aprender os pesos ou parâmetros da equação que melhor descrevem a relação entre os preditores e a variável de resposta para utilizá-los em uma nova observação (KUHN; JOHNSON *et al.*, 2013). O conjunto de saídas ou classes retorna uma pontuação de probabilidade que deve estar limitada ao intervalo $(0,1)$ e apresentando então uma relação direta com os preditores mensurados para cada observação do dataset. Apesar de muitos comparativos, muitos se questionam entre a diferença entre regressão linear e *RLOG*, sendo que para a regressão linear, a constante Y sempre é contínua (BRZEZINSKI; KNAFL, 1999).

Em *RLOG* é necessário multiplicar cada atributo por um peso e então adicionar esta saída à *Função Sigmoide*. Com isso podemos compreender a *RLOG* como uma estimativa de probabilidades (HARRINGTON, 2012). Também é necessário definirmos o valor limite ou a fronteira de decisão, que define a fronteira entre as classes dado *f(x)*. Como exemplo podemos observar a Figura 4, temos o valor limite de 0,5 e as classes *c0* e *c1*. Se a função de previsão retornasse um valor de 0,8, então classificaríamos esta observação como *c1*. Se nossa previsão retornasse um valor de 0,4, então classificaríamos a observação como *c0*.

Podemos então observar a composição de *RLOG* em quatro partes:

- Aplicação da *Função Sigmoide* ou também referenciada como de *Função Logística*.
- Um método de otimização ou regressão de coeficientes.
- Um valor para a fronteira de decisão.
- Uma grande porção do dataset para a etapa de treinamento.

3.3.1 Função Sigmoide

O nome "Sigmoide" significa "em forma de S", referindo-se ao gráfico da função mostrado na Figura 4 (SHALEV-SHWARTZ; BEN-DAVID, 2013). O valor de retorno de uma função sigmoide, sendo o eixo *y*, se categoriza por estar no intervalo de $(0,1)$ ou de $(-1,1)$. A função sigmoide é dada pela Equação (10) , onde podemos entender a variável *z* como os pesos gerados por algum método de otimização e tornando então em nosso valor de entrada para a função sigmoide.

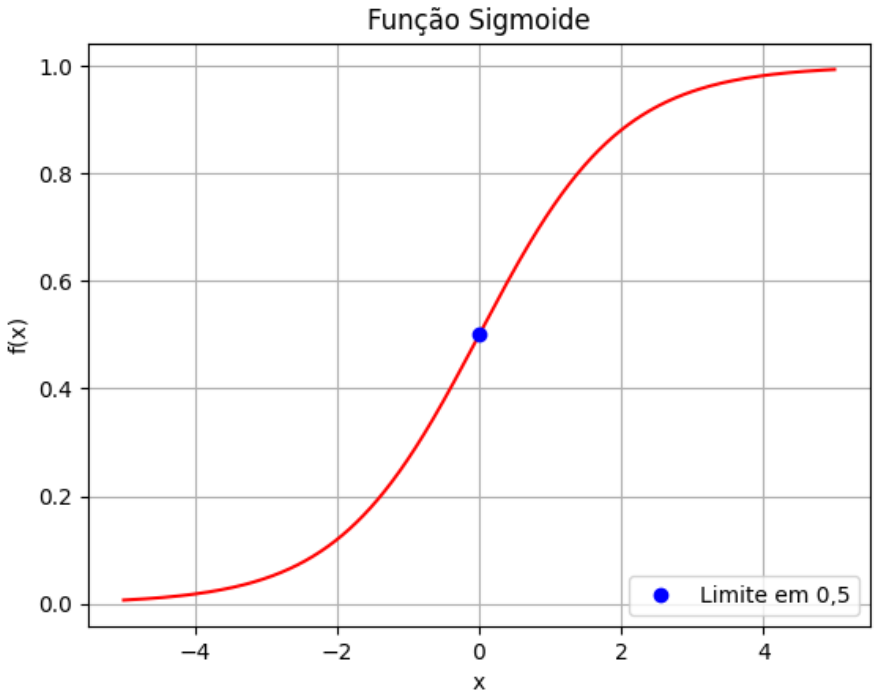
Função Sigmoide:

$$\alpha(z) = \frac{1}{1 + e^{-(z)}} \tag{10}$$

3.3.2 Método Gradiente Estocástico

O Gradiente Estocástico (GE) é um método de iterativo de otimização, onde o parâmetro selecionado é atualizado a cada iteração para cada instância do dataset, buscando o ponto máximo ou mínimo da função, se direcionando assim de uma melhor forma em direção ao gradiente (KUHN; JOHNSON *et al.*, 2013; HARRINGTON, 2012). Em *ML* é comum se utilizar o método *Descendente* em Redes Neurais e em nosso caso, utilizar o método *Ascendente*

Figura 4 – Função Sigmoide



Fonte: Autor

aplicado em *RLOG*. A diferença entre utilizar o *Método Gradiente* ou o *GE*, seja aplicado de forma *Ascendente* ou *Descendente*, é que o *Método Gradiente* utiliza o dataset completo à cada atualização de pesos, enquanto o *GE* utiliza apenas uma instância por vez para atualizar os pesos (HARRINGTON, 2012). Podemos visualizar as equações de ambos os métodos, onde fica visível que a diferença está no sinal aplicado em cada função, sendo assim, se estamos procurando o ponto mínimo da função utilizando o método *Descendente*, utilizamos a Equação (12) e para o ponto máximo da função utilizando o método *Ascendente*, utilizaremos a Equação (11).

Gradiente Estocástico Ascendente:

$$w := w + \alpha \nabla_w f(w) \tag{11}$$

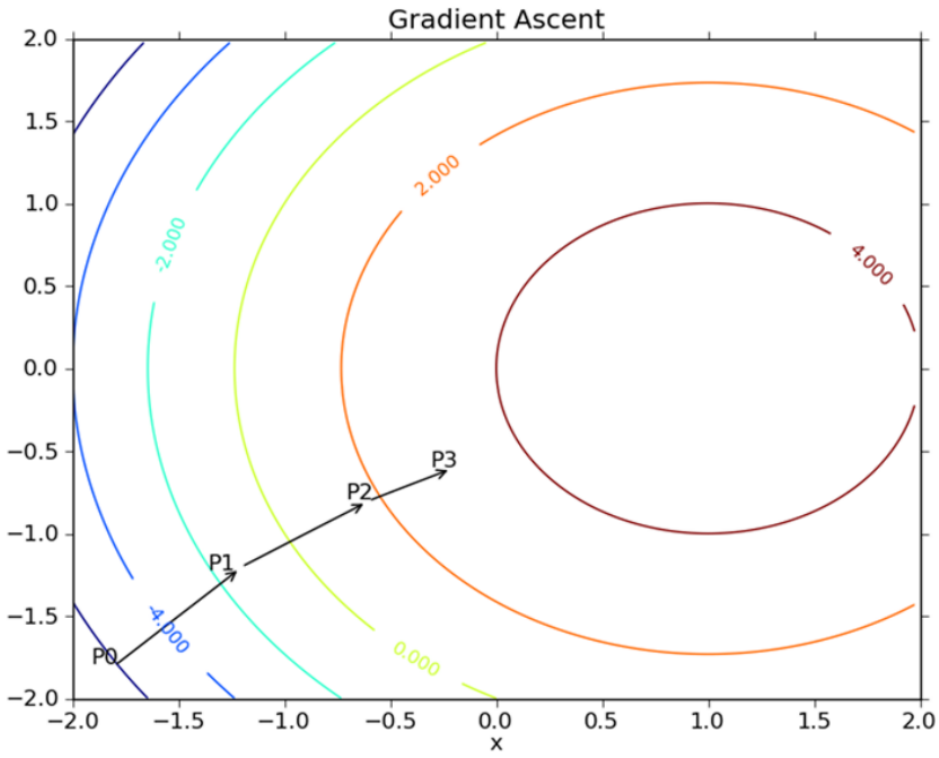
Gradiente Estocástico Descendente:

$$w := w - \alpha \nabla_w f(w) \tag{12}$$

Dado *z*, representado na função sigmoide como nosso valor de entrada, precisamos de um vetor dado por *x* e nosso dataset. É necessário que esses dois vetores de números sejam multiplicados para cada elemento e adicionados para gerar um número final, nosso *z*. Podemos simplificar por $z = w^T x$, ou de forma mais explícita, $z = w_0 x_0 + w_1 x_1 + \cdots + w_n x_m$, sendo então um incremento de pesos por iteração.

A Figura 5 exibe a direção dos pontos, sendo que a partir de cada atualização, os pesos são alterados e é possível visualizar a direção tomada a partir da otimização do método. Esse ciclo se repete até a condição ser atingida.

Figura 5 – Gradiente Ascendente



Fonte: (HARRINGTON, 2012)

3.4 Random Forest

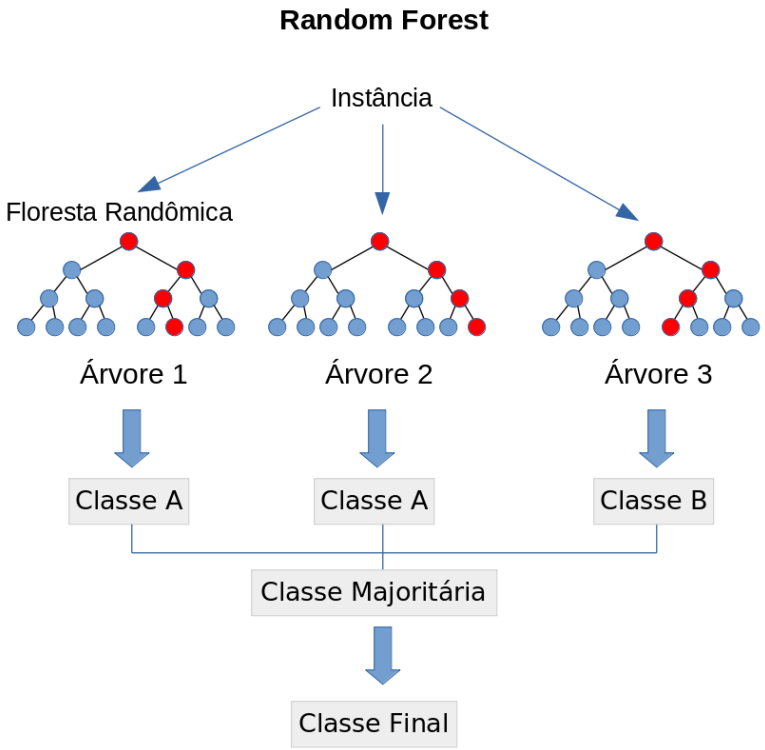
O *Ranfom Forest (RF)* é um algoritmo de classificação e regressão não paramétrico, robusto à ruídos e overfitting de dados de treinamento, que consiste de um conjunto de árvores de decisão binárias do tipo CART. Em etapa de treinamento, o *RF* produz uma grande quantidade de *Decision Tree* utilizando o método *Bagging*, que consiste na criação e aprendizado paralelo de preditores a partir da geração repetida de amostras de forma randômica e de mesmo tamanho do conjunto original de dados(Lin *et al.*, 2017). A utilização do método *Bagging* no treinamento tem como objetivo reduzir a complexidade dos modelos e reduzir a variância que interfere no desempenho da predição. O *RF* adiciona aleatoriedade ao modelo na criação das árvores, dado as melhores características para fazer a partição dos nós, a partir de subconjuntos aleatórios das variáveis (LIAW; WIENER *et al.*, 2002). Este procedimento gera diversidade, o que normalmente leva à formação de melhores preditores, também gerando a maior variedade possível a fim de evitar a construção de árvores correlacionadas entre si. A Figura 6 exibe um exemplo de fluxograma do *RF*.

O *RF* constrói a predição final dado por meio da contagem dos votos dos componentes preditores em cada classe, onde a classe é classificada dado o número de votos acumulados dentre todas as árvores da floresta. O *RF* presenta bom desempenho de precisão, principalmente se generalizado para outras amostras que não aquelas em que o classificador foi treinado e alto desempenho em pequenas amostras.

Podemos então observar as etapas do *RF* como:

- Constrói as amostras das arvores dado o conjunto de dados
- Para cada uma das amostras: Em cada nó escolhe a melhor divisão entre todos os

Figura 6 – Fluxograma - Random Forest



Fonte: Autor

preditores, realiza uma amostra aleatória dos preditores e escolhe a melhor divisão entre essas variáveis.

- Prevê novos dados adicionando as previsões dos votos da maioria das árvores para classificação ou média para regressão.

3.4.1 Modelo Matemático

Proposto por Breiman (2001), o *RF* é uma coleção de classificadores estruturados em árvores $\{h(X,\Theta_k),k=1,...\}$, onde Θ_k são vetores independentes e distribuídos de forma idêntica, onde cada árvore vota pela classe majoritária dado a entrada X. A partir de uma probabilidade fixa, os vetores de entrada são gerados de forma aleatória dado o vetor de entrada inicial. A precisão do *RF* é dado pela média probabilística dado o conjunto de classificadores $h1(X),h2(X),...,hk(X))$ e um conjunto de treinamento aleatório, onde a margem do classificador é dado pela equação Equação (13).

Margem do classificador:

$$mg(X,Y) = av_k I(hk(X) = Y) - max_{j \neq y} av_k I(hk(X) = j) \tag{13}$$

A medida que aumenta a correlação das árvores ou a força do conjunto diminui, que é representada pela Equação (14), o limite de erro de generalização aumenta. O limite de erro de generalização é dado pela equação $PE^* = P_{x,y}(mg(X,Y) < 0)$ quando o número de árvores for o suficientemente grande. Não ocorre overfitting no *RF* a cada adição de árvores, mas sim resulta em um valor de limite de erro de generalização (BREIMAN, 2001). O limite de erro é dado por dois parâmetros, sendo: a medida individual de precisão de cada classificador calculada pela equação da margem do classificador e a dependência entre eles.

Força do conjunto de classificadores $\{h(x,\Theta)\}$:

$$s = E_{x,y} mr(X,Y) \tag{14}$$

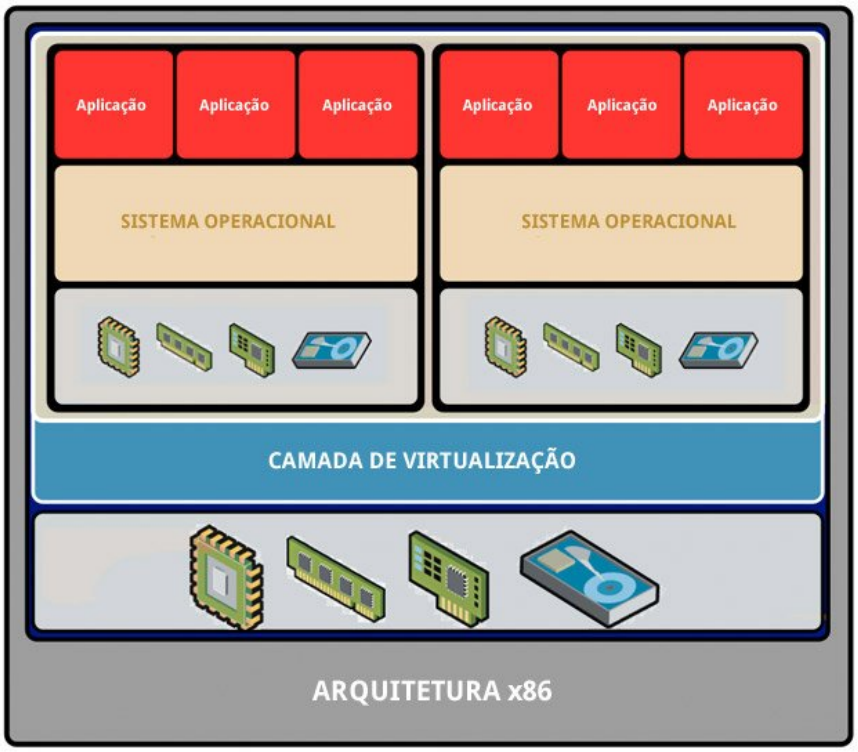
4 CENÁRIO

Esse capítulo apresenta o cenário utilizado no decorrer dessa monografia para a extração dos atributos necessários, a composição do dataset utilizado e a coleta de pacotes de uma varredura de porta. Sua estrutura é composta por: ambiente de virtualização, ferramentas de auxílio, datasets, tratamento dos dados e construção dos modelos de aprendizado de máquina.

4.1 Ambiente de Virtualização

Utilizando máquinas virtuais criadas com o *VirtualBox*, foi possível criar o ambiente para a captura dos pacotes de tráfego de botnet. O *VirtualBox*[1] é um software de virtualização x86 e AMD64/Intel64 de alto desempenho com suporte multiplataforma, como: Windows, Linux, Macintosh e entre diversos outros sistemas operacionais. O código fonte é aberto sobre os termos da Licença GNU GPL (General Public License) e possui uma comunidade ativa enquanto a *Oracle* garante que o produto atenda os critérios de qualidade.

Figura 7 – VirtualBox - Exemplo de virtualização e alocação de recursos



Fonte: (CANALTECH, 2012)

O *VirtualBox* permite virtualizar uma instalação e utilização de um sistema operacional dentro do seu sistema operacional, mas compartilhando fisicamente o mesmo hardware como a RAM e tamanho de disco, que devem ser alocados no momento da preparação da máquina virtual. A Figura 7 representa de forma visual como é a arquitetura básica de virtualização

[1] Disponível em: https://www.virtualbox.org e Acessado em 20 de novembro de 2020

que o *VirtualBox* utiliza. O *VirtualBox* foi instalado em um notebook com processador Core i7-6700HQ 2.6GHz com 6MB de cache, 4 núcleos e 8 threads, com 16GB RAM e Linux Pop!_OS 20.04 como sistema operacional.

4.2 Ferramentas de Auxílio

Nessa seção, serão abordadas as ferramentas utilizadas no ambiente virtualizado para realizar os ataques de *Port Scanning* com o *Netcat* e *Nmap*, captura do tráfego com o *tcpdump* e tratamento dos pacotes com *argus*, que converte os *PCAPS* em *Netflow* e posteriormente arquiva os dados em *CSV*, para se tornarem insumo de entrada para o algoritmo de classificação.

4.2.1 Tcpdump

O *tcpdump*[2] é utilizado para monitoração do tráfego de rede, exibido os cabeçalhos e a descrição dos pacotes de uma interface de rede. É possível salvar os dados do pacote em um arquivo para análise posterior em formato *PCAP* ou ler um arquivo de pacote salvo em vez de ler os pacotes de uma interface de rede. O *tcpdump* captura os pacotes até que seja interrompido ou até que o número especificado de pacotes tenha sido processado. O *tcpdump* está sob uma licença *BSD* de 3 cláusulas.

4.2.2 Argus

O projeto *Argus*[3] (Audit Record Generation and Utilization System) é um projeto de monitoramento de fluxo de redes em grande escala, possui trilhas de auditoria de rede, suporte a operações de rede, desempenho e gerenciamento de segurança. O *Argus* gera metadados de rede que podem ser usados para realizar um grande número de tarefas de gerenciamento e também foi projetado para ser um sistema de consciência situacional em tempo real, que rastreia e alerta sobre as condições da rede. Os dados também podem ser usados para estabelecer uma auditoria abrangente de todo o tráfego de rede, complementando a segurança de rede tradicional baseada no sistema de detecção de intrusão. A trilha de auditoria é tradicionalmente usada como dados históricos de medição de tráfego de rede para análise forense de rede e detecção de anomalias de comportamento. O Argus tem sido amplamente utilizado em segurança cibernética, análise de desempenho ponta a ponta e, mais recentemente, em pesquisa de redes definidas por software.

O *Argus* é composto por um gerador de dados de fluxo de rede abrangente que processa pacotes e gera relatórios detalhados de status de fluxo de tráfego de rede de todos os fluxos e monitora todo o tráfego de rede, plano de dados, plano de controle e plano de gerenciamento. O Argus captura grande parte da dinâmica e semântica de pacotes de cada fluxo, com grande redução de dados, para armazenar, processar, inspecionar e analisar grandes quantidades de dados de rede com eficiência. O *Argus* está sob uma licença GNU GPL, é uma das 100 principais ferramentas de segurança da Internet segundo a SecTools[4] e é gerenciado pela QoSient.

4.2.3 Netcat

O *Netcat*[5] é ferramenta de depuração e exploração de rede rica em recursos que consegue ler e gravar dados em conexões de rede utilizando o protocolo *TCP/IP*. Apesar de

[2]Disponível em: https://www.tcpdump.org/ e Acessado em 20 de novembro de 2020
[3]Disponível em: https://openargus.org/ e Acessado em 20 de novembro de 2020
[4]Disponível em: https://sectools.org/?page=4 e Acessado em 20 de novembro de 2020
[5]Disponível em: http://netcat.sourceforge.net e Acessado em 20 de novembro de 2020

poder criar a maioria dos tipos de conexões, ele possui vários recursos integrados como escutar uma conexão determinada por host, porta e interface, bem como é possível realizar um *Port Scanning* em um uma quantidade de portas determinadas e visualizar se as conexões estão abertas, fechadas ou filtradas. O *Netcat* está sob uma licença GNU GPL e é amplamente utilizado na área de segurança da informação.

4.2.4 Nmap

O *Network Mapper* (Nmap) é uma ferramenta open-source e de código aberto para descoberta de rede e auditoria de segurança, mas também é muito utilizado para tarefas como inventário de rede, gerenciamento de agendas de atualização de serviço e monitoramento de host ou tempo de atividade de serviço. O *Nmap*[6] usa pacotes IP brutos para determinar quais hosts estão disponíveis na rede, quais serviços e versões esses hosts estão oferecendo, quais sistemas operacionais eles estão executando, que tipo de filtros de pacotes e firewalls estão em uso e entre outras características. Ele foi projetado para verificar rapidamente grandes redes, mas funciona bem em hosts únicos. O *Nmap* é executado em todos os principais sistemas operacionais e os pacotes binários oficiais estão disponíveis para Linux, Windows e Mac OS X. O *Nmap* está sob a licença GNU GPL e foi nomeado "Produto de Segurança do Ano" pelo Linux Journal, Info World, LinuxQuestions.Org e Codetalker Digest.

4.3 Dataset

Um importante passo no aprendizado de máquina é utilizar dados adequados para o treinamento e teste do algoritmo. Neste projeto foram utilizados dois datasets, sendo o *Dataset CTU-13* publicado pelo Garcia *et al.* (2014) e um segundo dataset composto pela captura do fluxo de tráfego de um ataque de *Port Scanning* utilizando Nmap e o Netcat em um ambiente controlado e isolado. Nessa seção será apresentado os detalhes a respeito do *Dataset CTU-13*, a utilização de seus cenários e o método de captura do tráfego, que inclui sua criação conversão de fluxos de dados.

4.3.1 Dataset CTU-13

O *Dataset CTU-13* é um conjunto de dados de tráfego de botnet que foi capturado na Universidade CTU, localizada na República Tcheca, em 2011. O objetivo do conjunto de dados era capturar um grande tráfego de botnet, entendendo e classificando o tráfego normal e tráfego de fundo (ou background). O conjunto de dados CTU-13 consiste em treze cenários de diferentes amostras de botnets, onde em cada cenário foi executado um malware específico, que usava vários protocolos e realizava diferentes ações.

O *Dataset CTU-13* foi criado utilizando fluxos unidirecionais e posteriormente publicado no artigo "A Empirical Comparison of Botnet Detection Methods" de Garcia *et al.* (2014). A criação do dataset se deu por não existir naquele momento um conjunto de dados público e realístico para se explorar os métodos de detecção de botnets, então foi elaborado um cenário utilizando a rede de computadores da universidade para a captura de um tráfego real e a virtualização dos sistemas operacionais Windows XP e Debian para se comportarem como as máquinas infectadas pelos diversos botnets. A Figura 8 mostra as características dos ataques utilizados nos 13 cenários que compõem o dataset.

Para este trabalho foram utilizados apenas os datasets dos cenários 6 e 8, onde neles ocorrem apenas tráfegos de fundo e tráfegos de ataque realizando um *Port Scanning*. Cada

[6]Disponível em: https://nmap.org/ e Acessado em 20 de novembro de 2020

Figura 8 – CTU-13 - Características dos 13 cenários

Id	IRC	SPAM	CF	PS	DDoS	FF	P2P	US	HTTP	Note
1	√	√	√							
2	√	√	√							
3	√			√				√		
4	√				√			√		UDP and ICMP DDoS.
5		√		√					√	Scan web proxies.
6				√						Proprietary C&C. RDP.
7									√	Chinese hosts.
8				√						Proprietary C&C. Net-BIOS, STUN.
9	√	√	√	√						
10	√				√			√		UDP DDoS.
11	√				√			√		ICMP DDoS.
12							√			Synchronization.
13		√		√					√	Captcha. Web mail.

Fonte: (GARCIA *et al.*, 2014)

cenário gerou um arquivo PCAP e posteriormente foi convertido para *NetFlow*, contendo todas as informações de tráfego de cada botnet durante o período citado no artigo. Apesar de ser um grande tráfego coletado, a porcentagem de tráfego de botnets é miníma comparada ao tráfego total, sendo de 0,79% no *Cenário 6* e de 0,17% no *Cenário 8*.

O *Dataset CTU-13* possui 15 atributos, onde um deles é a classe alvo (Label) e o restante sendo: StartTime, Dur, Proto, SrcAddr, Sport, Dir, DstAddr, Dport, State, sTos, dTos, TotPkts, TotBytes e SrcBytes. Todos esses atributos são semelhantes ao padrão utilizado nas ferramentas como *NfDump* e no *Argus*, sendo este último utilizado e citado no artigo de Garcia *et al.* (2014).

4.3.2 Dataset de Fluxo de Ataques utilizando Nmap e Netcat

Nesta seção será abordado os processos de criação e captura dos fluxos de ataque utilizando o *Port Scanning*, conversão dos arquivos para o formato *Netflow* e os critérios para a composição dos fluxos de ataque e as ferramentas e parâmetros adotados.

4.3.2.1 Captura dos Fluxos de Ataque

Como forma de validação do ataque, foi feito a captura de ataques do tipo *Port Scanning* utilizando a ferramenta *tcpdump*. Três máquinas virtuais foram utilizadas, sendo uma máquina alvo (Debian), uma máquina infectada com o bot (Windows XP) e uma máquina para realizar o acesso e envio de comandos ao botnet (Kali). Com o ambiente isolado de tráfego externo, foi possível realizar a captura de um ataque direto partindo de uma máquina infectada com o botnet e salvando o fluxo de dados em arquivos do tipo *PCAP*. A captura durou apenas o intervalo de finalização de cada ferramenta e foram separados em arquivos distintos para uma análise individual.

Utilizando o *Netcat*, foi realizado um ataque com os parâmetros simples **$ nc -zv ip_alvo porta_alvo**, onde foi indicado para realizar o método de scan apenas na porta 22. Este tipo de ataque é curto e com menor ruído, já que foi selecionado apenas uma única porta alvo para ser escaneada.

Utilizando o *Nmap*, foi realizado dois tipos de ataques, onde um deles é o método com maior ruído e o outro o mais silencioso dentro da ferramenta. Para o método com o maior ruído, foi utilizado os parâmetros **$ nmap -T5 -A -v host_alvo**, onde é feito as requisições na maior velocidade da ferramenta (-T5) e realizando a varredura para detectar o sistema operacional, versão das aplicações, habilitar o *Script Scanning* e a tabela de rotas (Traceroute).

O método com menor ruído foi utilizado os parâmetros **$ nmap -sV -Pn -v host_alvo**, assim não é utilizado o protocolo *ICMP* (-Pn) e é possível visualizar os serviços de cada porta (-sV). Em ambas as execuções de ataque utilizando o *Nmap*, foi feita a varredura para as primeiras 1000 portas, sendo assim possível comparar a eficácia do modelo em detectar ataques de varreduras mais extensos, mesmo com os diferentes parâmetros utilizados a fim de diminuir a detecção por qualquer tipo de sistema.

4.3.2.2 Conversão dos Pacotes

Com os arquivos PCAP gerados pelo *tcpdump*, a próxima etapa é converter eles em tipo *Netflow*. Antes de realizar a conversão, é necessário remover alguns fluxos que não fazem parte do contexto do ataque, como por exemplo fluxos de protocolo ARP recebidos pela máquina alvo e o protocolo ICMP quando utilizado o *Nmap* em modo silencioso, já que não consta como parte do ataque. Para a conversão, é necessário primeiro utilizar *Argus* com o seguinte comando **argus -r arquivo.pcap -w nome_arquivo_saida.argus**, onde este processo foi repetido com todos os arquivos PCAP coletados.

Após a conversão, utilizando o comando *ra* (Read Argus), pode-se escolher quais atributos serão extraídos do arquivo em formato *Netflow*. Para a reprodução neste trabalho, não foram extraídos todos os atributos que constam no *Dataset CTU-13* da Seção 4.3.1. Essa decisão de baseia na decisão de explorar diferentes resultados, porém utilizando os melhores atributos para cada modelo de aprendizado de máquina explorado nesta monografia. Utilizando o comando **$ ra -r arquivo.argus -s dur,proto,dir,state,stos,dtos,pkts,bytes,sbytes** , podemos extrair os atributos necessários para dar continuidade a classificação do fluxo de rede. A Tabela 3 exibe o cabeçalho de saída de um PCAP contendo apenas os atributos selecionados.

Tabela 3 – Arquivo Argus - Exemplo de Cabeçalho

Dur	Proto	Dir	State	sTos	dTos	TotPkts	TotBytes	SrcBytes
1.023995	tcp	->	REQ	0	0	2	148	148
1.023831	tcp	->	REQ	0	0	2	148	148
1.023737	tcp	->	REQ	0	0	2	148	148
1.023650	tcp	->	REQ	0	0	2	148	148
0.002930	man	->	STA	0	0	0	0	0
0.002743	man	->	STA	0	0	0	0	0

Fonte: Autor

4.4 Tratamento dos Dados

Com os arquivos do tipo *Netflow* gerados na Seção 4.3.2 e o *Dataset CTU-13*, utilizando apenas os cenários 6 e 8, podemos dar continuidade a preparação dos dados para iniciar o treinamento do modelo de aprendizado de máquina. Todo o código utilizado no tratamento dos dados pode ser visualizado na Seção A.1.

4.4.1 Escolha dos Atributos

A primeira decisão é escolher quais atributos são relevantes para cada modelo, então foi utilizado a classe *feature_selection* do framework *sklearn*[7] para encontrar os melhores atributos para compor o dataset e consequentemente melhorar as métricas de precisão. Como resultado, os melhores atributos para compor o dataset são: Dur, Proto, Dir, State, sTos, dTos, TotPkts, TotBytes, SrcBytes e Label. A fim de explorar diferentes composições de atributos, foram adicionados 5 novos atributos a partir dos resultados apresentados do *sklearn.feature_selection*, sendo: *TotBytes_over_TotPkts* (TotBytes / TotPkts), *SrcBytes_over_TotPkts* (SrcBytes / TotPkts), *TotPkts_over_Dur* (TotPkts / Dur), *TotBytes_over_Dur* (TotBytes / Dur) e *SrcBytes_over_Dur* (SrcBytes / Dur).

4.4.2 Pré-processamento

Como forma de padronização e boas práticas para melhor desempenho dos modelos, foi realizado uma alteração nos atributos e classes que eram compostos por *strings*. Este tipo de pré-processamento facilita o desempenho dos modelos de aprendizado de máquina e em alguns algoritmos, como em *Random Forest* e *Decison Tree*, é um pré requisito os classes alvos serem de valor do tipo inteiro. Os seguintes atributos eram compostos por atributos do tipo *string*: Proto, Dir, State e a classe alvo Label.

O Quadro 1 realiza as associações de números inteiros para o atributo *Proto*.

Quadro 1 – Pré-processamento - Dicionário - Proto

Pré-processamento - Dicionário - Proto
dict_proto = 'tcp': 0, 'udp': 1, 'ipx/spx': 2, 'arp': 3, 'icmp': 4, 'pim': 5, 'rtcp': 6, 'igmp' : 7, 'ipv6-icmp': 8, 'esp': 9, 'ipv6': 10, 'udt': 11,'rtp':12, 'rarp':13, 'rsvp':14, 'unas':15

Fonte: Autor

O Quadro 2 realiza as associações de números inteiros para o atributo *Dir*.

Quadro 2 – Pré-processamento - Dicionário - Dir

Pré-processamento - Dicionário - Dir
dict_dir = '<?>': 0, '<->': 1, '?>': 2, '->': 3, 'who': 4, '<-': 5, '<?': 6

Fonte: Autor

O Quadro 3 realiza as associações de números inteiros para o atributo *State*.

Para a classe alvo *Label* foram elaborados em dois grupos, sendo: *Grupo Label - 1*, com o objetivo de classificar Tráfego Normal (0) e Tráfego de Botnet (1) e *Grupo Label - 2*, para classificar em Tráfego Normal (0), Tráfego Background de Botnet (1) e Tráfego de Botnet (2). O *Grupo Label - 1* possui duas classes alvo e o *Grupo Label - 2* possui três classes alvo. Com isso, podemos explorar diferentes comportamentos e resultados dos modelos utilizando o mesmo dataset, mas com o objetivo de problema diferente. Conforme citado no artigo de Garcia *et al.* (2014), dentro do atributo *Label*, os fluxos gerados pelo botnet começam com "From-Botnet" e os que possuem "To-Botnet" são fluxos enviados para o botnet

[7] https://scikit-learn.org/

Quadro 3 – Pré-processamento - Dicionário - State

Pré-processamento - Dicionário - State
dict_state = '': 1, 'FSR_SA': 30, '_FSA': 296, 'FSRPA_FSA': 77, 'SPA_SA': 31, 'FSA_SRA': 1181, 'FPA_R': 46, 'SPAC_SPA': 37, 'FPAC_FPA': 2, '_R': 1, 'FPA_FPA': 784, ... , 'NaN':99965,'_':99966

Fonte: Autor

por computadores desconhecidos e não devem ser considerados maliciosos. Também para os computadores de tráfego normal, o mesmo se equivale se começarem com "From-Normal". Os rótulos "To-Normal" são fluxos enviados para o botnet por computadores desconhecidos, então eles não devem ser considerados maliciosos.

Elaborado então uma lista única das *strings* que compõem o atributo *Label* e organizadas de acordo com as notas do artigo citado, que reduz a dimensionalidade de classes para cada cenário utilizado. O restante foi codificado com o numero 0, denominando então como tráfego normal. Mais detalhes sobre a implementação se encontram no Apêndice A.

Dataset CTU-13 - Cenário 6 - Background - Botnet
- flow=From-Normal-V47-Jist
- flow=From-Normal-V47-UDP-CVUT-DNS-Server
- flow=From-Normal-V47-Grill
- flow=From-Normal-V47-Stribrek
- flow=From-Normal-V47-MatLab-Server
- flow=From-Normal-V47-CVUT-WebServer
- flow=From-Normal-V47-HTTP-windowsupdate

Dataset CTU-13 - Cenário 6 - Botnet
- flow=From-Botnet-V47-UDP-DNS
- flow=From-Botnet-V47-TCP-Established-HTTP-Ad-4
- flow=From-Botnet-V47-TCP-CC73-Not-Encrypted
- flow=From-Botnet-V47-TCP-Attempt-SPAM
- flow=From-Botnet-V47-TCP-Established-HTTP-Ad-62
- flow=From-Botnet-V47-TCP-Attempt
- flow=From-Botnet-V47-UDP-Attempt

Dataset CTU-13 - Cenário 8 - Background - Botnet
- flow=From-Normal-V49-Stribrek
- flow=From-Normal-V49-Grill
- flow=From-Normal-V49-Jist
- flow=From-Normal-V49-CVUT-WebServer

Dataset CTU-13 - Cenário 8 - Botnet
- flow=From-Botnet-V49-UDP-DNS
- flow=From-Botnet-V49-TCP-Established-HTTP-Ad-40
- flow=From-Botnet-V49-TCP-HTTP-Google-Net-Established-6
- flow=From-Botnet-V49-UDP-Attempt
- flow=From-Botnet-V49-TCP-Attempt
- flow=From-Botnet-V49-TCP-WEB-Established

- flow=From-Botnet-V49-UDP-Established
- flow=From-Botnet-V49-TCP-Established
- flow=From-Botnet-V49-TCP-CC76-HTTP-Custom-PortNoEncryptedBinaryDownload
- flow=From-Botnet-V49-TCP-CC74-HTTP-Custom-Port-Not-Encrypted
- flow=From-Botnet-V49-TCP-CC75-HTTP-Custom-Port-Not-Encrypted-Non-Periodic
- flow=From-Botnet-V49-TCP-Established-HTTP-Binary-Download-11
- flow=From-Botnet-V49-TCP-Established-HTTP-Ad-62
- flow=From-Botnet-V49-UDP-Established-Custom-Encryption-2
- flow=From-Botnet-V49-UDP-Established-Custom-Encryption-1

4.4.3 Subgrupos do Dataset

Com todo o cenário descrito, as situações de escolha de atributos e classes alvo abordadas durante este capitulo, é evidente que existam variações do dataset a serem utilizadas durante a preparação dos modelos. Durante a fase de preparação dos dados, foram elaborados quatro grupos de datasets para serem explorados, onde alguns incluem os cinco atributos elaborados na Seção 4.4.1, sendo:

- Dataset 1 - O Datasset possui 2 classes (Tráfego Normal e Tráfego Botnet)
- Dataset 2- O Datasset possui 2 classes (Tráfego Normal e Tráfego Botnet) + 5 atributos
- Dataset 3 - O Datasset possui 3 classes (Tráfego Normal e Tráfego Botnet e Tráfego Background Botnet)
- Dataset 4 - O Datasset possui 3 classes (Tráfego Normal e Tráfego Botnet e Tráfego Background Botnet) + 5 atributos

A intenção é explorar em diversos estados o comportamento dos ataques para cada tipo de dataset, assim pode-se compreender se existe um diferente comportamento influenciado por cada modelo, atributo ou tipo de ferramenta utilizada. Como finalização da etapa de particionamento dos datasets, os mesmos foram salvos em formato *CSV*.

4.5 Treinamento dos Modelos

Após a coleta, pré-processamento dos dados e os quatro grupos de datasets preparados conforme a Seção 4.4.3, a próxima etapa é elaborar um modelo para cada grupo de dataset, contendo então todos os algoritmos apresentados no Capítulo 3. O código fonte está no Seção A.2. Para gerar cada modelo de aprendizado de máquina, foram utilizadas as bibliotecas: Pandas, NumPy, MatPlotLib, Pickle e Sklearn. Todas as métricas utilizadas para a coleta de resultados foram descritas na Seção 2.4 e os resultados serão abordados no Capítulo 5.

Para a composição de treinamento e validação, do *CTU-13* foram utilizadas 25 mil instâncias de tráfego normal e todo o tráfego de botnet do *Cenário 6*, além de todas as instâncias de tráfego normal de botnet do *Cenário 8*, equilibrando então a proporção de tráfego normal e tráfego botnet, se totalizando em: *Cenário 6* - 25 mil instâncias de tráfego normal , 7433 instâncias de tráfego background de botnet, 230 instâncias de tráfego normal de botnet e *Cenário 8* - 397 instâncias de tráfego normal de botnet. Cada classe alvo foi determinada por um número inteiro, sendo: 0 - Tráfego Normal, 1 - Tráfego Background de Botnet e 2 - Tráfego Normal de Botnet.

Cada grupo de dataset foi particionado com 75% de instâncias para treinamento e 25% das instâncias para validação, utilizando os algoritmos: *Decision Tree*, *Naive Bayes*, *Random Forest* e *Logistic Regression* para a classificação. Todos os modelos já treinados foram salvos

utilizando a biblioteca *Pickle*, sendo assim possível utilizar os modelos em outros projetos e ou cenários experimentais. Os resultados coletados para análise são: Acurácia, Matriz de Confusão, Validação Cruzada e curva ROC.

5 RESULTADOS

Nessa seção é abordado os resultados obtidos na elaboração dos modelos para cada dataset proposto no Seção 4.4.3 e seu comportamento para detecção de tráfego e ataques do tipo *Port Scanning*, sendo então determinados por valores inteiros as predições: 0 - Tráfego e 1 - Ataque. O código para a detecção de tráfego e ataques de botnet se encontra na Seção A.3.

5.1 Dataset 1 - Detecção de Tráfego de Botnet

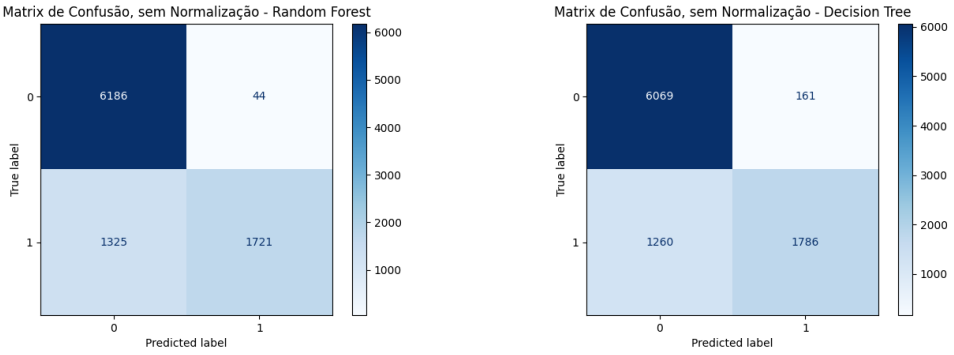
Com o modelo gerado, a Tabela 4 descreve os percentuais de acurácia obtidos. A maior acurácia é do algoritmo *Random Forest* com 85,2% e o *Decision Tree* com 84,6% acompanha com um percentual próximo. Apesar da alta acurácia em ambos os algoritmos, podemos visualizar pela matriz de confusão que a tendência de ambos é uma classificação majoritária para a classe 0 (tráfego normal) e caso não se enquadre, é classificado como classe 1 (tráfego botnet). Este tipo de situação é mais claro ao observar a Figura 9, onde ambos possuem um alto percentual de predições incorretas para a classe 1, sendo então 1260 instancias (41%) *Decision Tree* e 1325 instâncias (43%) pelo *Random Forest*.

Tabela 4 – Dataset 1 - Acurácia dos Algoritmos

Algoritmo	Acurácia
Decision Tree	84,6%
Naive Bayes	47,3%
Random Forest	85,2%
Logistic Regression	77,7%

Fonte: Autor

Figura 9 – Dataset 1 - Matriz de Confusão - Random Forest e Decision Tree

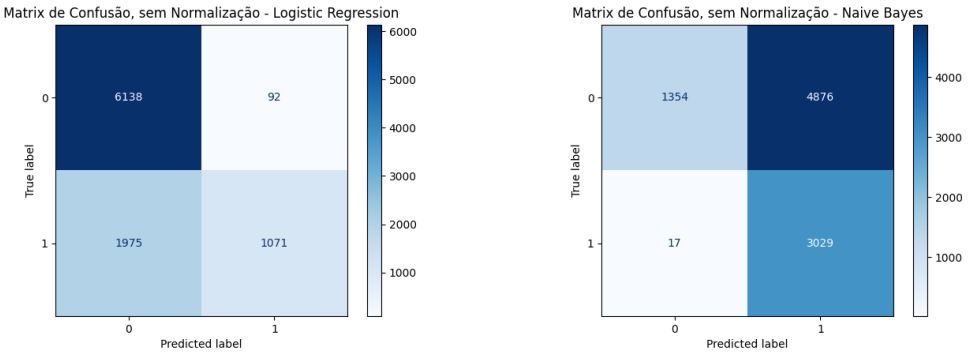


Fonte: Autor

O *Logistic Regression* possui uma taxa de acurácia valida para aplicação em mundo real, porém seus resultados da matriz de confusão demonstram que ele possui dificuldade em

classificar corretamente a classe 1. Apesar da baixa taxa de acurácia do *Naive Bayes*, o mesmo apresenta uma maior taxa de predições corretas comparados aos outros algoritmos, onde era esperado devido à natureza do algoritmo tratar de forma independente os atributos, porém ele possui uma alta de taxa predições incorretas para a classe 0. Os resultados são de 1975 instâncias (65%) para o *Logistic Regression* em comparação as 17 instâncias (0,05%) do *Naive Bayes* errôneamente classificados como de classe 1, porém o *Naive Bayes* obteve uma taxa de falsos positivos de 4876 instâncias (78%) para a classe 0. Os resultados podem ser visualizados na Figura 10.

Figura 10 – Dataset 1 - Matriz de Confusão - Logistic Regression e Naive Bayes



Fonte: Autor

A Tabela 5 exibe os resultados obtidos das validações cruzadas, onde foi efetuado 5 validações para cada algoritmo, concluindo então que existe uma taxa mínima de variação de acurácia para os algoritmos, exceto o *Decision Tree* que possui o mesmo valor em todas as validações efetuadas.

Tabela 5 – Dataset 1 - Validação Cruzada

Algoritmos	Resultados
Decision Tree	[0.85 0.85 0.85 0.85 0.85]
Naive Bayes	[0.46 0.47 0.48 0.48 0.48]
Random Forest	[0.86 0.86 0.86 0.85 0.86]
Logistic Regression	[0.78 0.79 0.79 0.79 0.78]

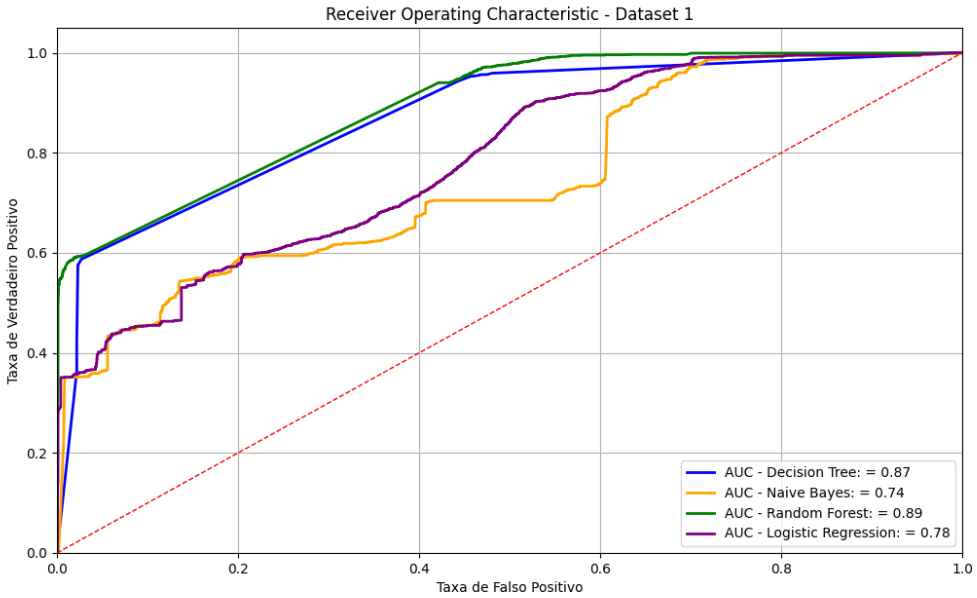
Fonte: Autor

A Figura 11 exibe a curva da métrica *ROC* e os valores de *AUC* para cada algoritmo, sendo: *Decision Tree* - 0.87, *Naive Bayes* - 0.74, *Random Forest* - 0.89 e *Logistic Regression* - 0.78.

5.1.1 Dataset 1 - Comportamento no Cenário

A Tabela 6 exibe como os algoritmos classificaram cada tipo de tráfego botnet, onde observa-se que mesmo utilizando o *netcat* para uma varredura de porta única, o mesmo foi facilmente detectado por todos os modelos. O *Nmap* com maior ruído foi detectado pelos

Figura 11 – Dataset 1 - ROC e AUC



Fonte: Autor

algoritmos *Random Forest* e *Logistic Regression*, porém é evidente que o *Nmap* com os parâmetros mais silenciosos só foi detectado pelo *Random Forest*, o que conclui que apesar da alta precisão dos modelos, a matriz de confusão nos revela que a maioria dos algoritmos não está apta a detectar o tráfego de botnet nesta porção de dataset.

Tabela 6 – Dataset 1 - Comportamento de Classificação

Ferramenta	DT	NB	RF	RLOG
Netcat	1	1	1	1
Nmap (Silencioso)	0	0	1	0
Nmap (Maior Ruído)	0	0	1	1

Fonte: Autor

5.2 Dataset 2

Com o modelo gerado utilizando a adição dos atributos extras citados no Seção 4.4.1, a Tabela 7 descreve os percentuais de acurácia obtidos. Comparado aos resultados da Tabela 4, os algoritmos *Random Forest* e *Decision Tree* possuem a maior taxa de acurácia entre os modelos, porém houve um aumento significativo, sendo respectivamente de 91,5% e 89,8%. Os demais algoritmos obtiveram um aumento da acurácia, sendo de 49,4% para o *Naive Bayes* e 79,9% para o *Logistic Regression*, porém como já observado nos resultados da Seção 5.1, é necessário avaliar o comportamento da matriz de confusão para entender além da acurácia.

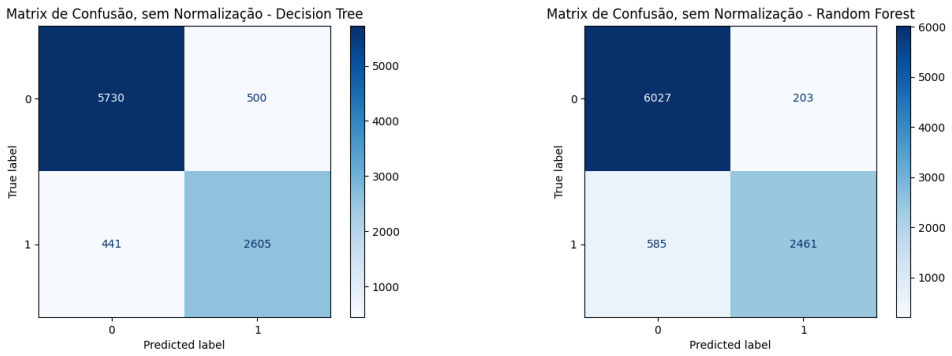
Tabela 7 – Dataset 2 - Acurácia dos Algoritmos

Algoritmo	Acurácia
Decision Tree	89,8%
Naive Bayes	49,4%
Random Forest	91,5%
Logistic Regression	79,9%

Fonte: Autor

Com a adição dos atributos extras, os algoritmos *Random Forest* e *Decision Tree* obtiveram uma menor taxa de *Falsos Positivos* e *Falsos Negativos* comparados aos resultados da Figura 9. O percentual de predições incorretas para a classe 1 é de 441 instâncias (14%) para o *Decision Tree* e 585 instâncias (19%) para o *Random Forest*, onde podemos concluir que a adição dos atributos extras contribuiu diretamente para a eficiência dos algoritmos citados. Os resultados podem ser visualizados na Figura 12.

Figura 12 – Dataset 2 - Matriz de Confusão - Random Forest e Decision Tree



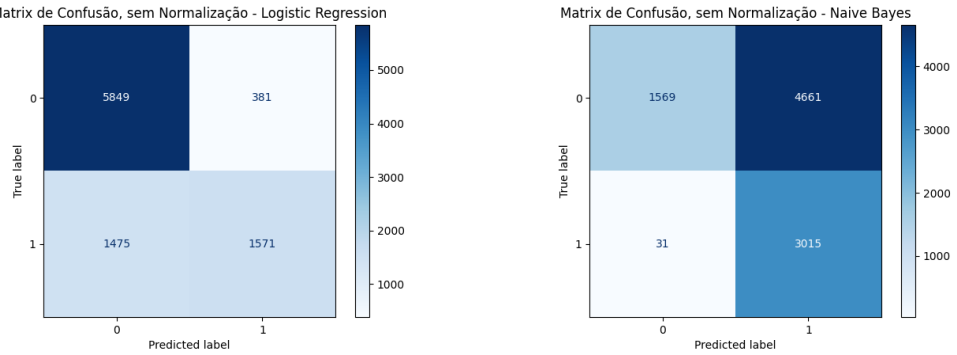
Fonte: Autor

A Figura 13 nos apresenta uma melhoria mínima na taxa de predições incorretas feitas pelos algoritmos *Logistic Regression* e *Naive Bayes* comparado aos resultados da Figura 10, respectivamente 1475 instâncias (48%) de classe 1 e 4661 instâncias (75%) de classe 0. É evidente que os algoritmos possuem uma boa taxa de acurácia, mas mesmo com as melhoras apresentadas com os atributos adicionais, não possuem um bom resultado quando observado pela matriz de confusão.

A Tabela 8 exibe os resultados obtidos das validações cruzadas, onde foi efetuado 5 validações para cada algoritmo e observa-se que existe uma variação miníma comparada a acurácia final apresentada na Tabela 7.

A Figura 14 exibe a curva métrica ROC e os valores de AUC para cada algorítmo, sendo: *Decision Tree* - 0.88, *Naive Bayes* - 0.81, *Random Forest* - 0.97 e *Logistic Regression* - 0.82. Comparado a Figura 11, houve um aumento significativo em todos os algoritmos com a adição dos atributos extras, onde o algoritmo *Random Forest* chega próximo ao valor ideal de AUC de 1.0.

Figura 13 – Dataset 2 - Matriz de Confusão - Logistic Regression e Naive Bayes



Fonte: Autor

Tabela 8 – Dataset 2 - Validação Cruzada

Algoritmos	Resultados
Decision Tree	[0.9 0.9 0.91 0.9 0.91]
Naive Bayes	[0.48 0.49 0.5 0.5 0.5]
Random Forest	[0.92 0.92 0.92 0.91 0.92]
Logistic Regression	[0.8 0.8 0.81 0.8 0.81]

Fonte: Autor

5.2.1 Dataset 2 - Comportamento no Cenário

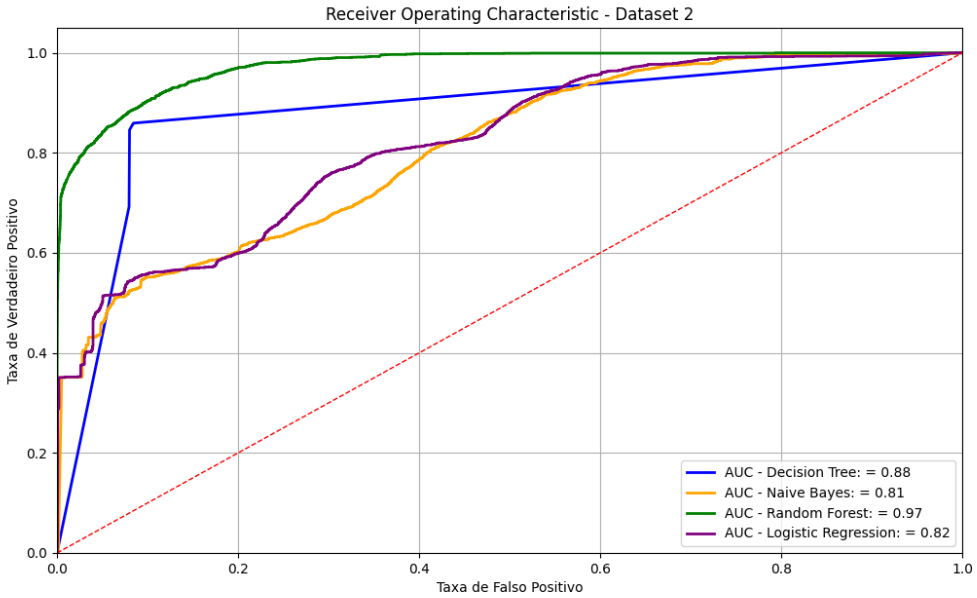
A Tabela 9 exibe como os algoritmos classificaram cada tipo de tráfego botnet e em comparação aos resultados apresentados na Tabela 6, o *netcat* permanece sendo detectado por todos os modelos. O *Nmap* com maior ruído foi novamente detectado pelos algoritmos *Random Forest* e *Logistic Regression*, porém o *Nmap* com os parâmetros mais silenciosos foi detectado pelo *Random Forest* e *Logistec Regression*. Novamente, apesar da alta precisão dos modelos e melhoras significativas com os resultados apresentados na Seção 5.1 dado a adição dos novos atributos, a matriz de confusão nos revela que alguns modelos não estão aptos para serem utilizados em ambiente real de detecção de tráfego de botnet.

Tabela 9 – Dataset 2 - Comportamento de Classificação

Ferramenta	DT	NB	RF	RLOG
Netcat	1	1	1	1
Nmap (Silencioso)	0	0	1	1
Nmap (Maior Ruído)	0	0	1	1

Fonte: Autor

Figura 14 – Dataset 2 - ROC e AUC



Fonte: Autor

5.3 Dataset 3

Com o modelo gerado para a detecção de três classes alvo, sendo: 0 - Tráfego normal, 1 - Tráfego Background de Botnet e 2 - Tráfego Normal de Botnet, a Tabela 10 apresenta a acurácia de cada algoritmo. O *Random Forest* e *Decision Tree* apresentando a maior taxa, respectivamente de 87,3% e 86,6%, enquanto o *Naive Bayes* possui uma baixa taxa de acurácia de 48,4% de todos os algoritmos. Apesar de três classes alvo, são percentuais semelhantes ao proposto pelo Seção 5.1, o que é evidente a necessidade de observar as matrizes de confusão e entender como ocorre a predição para cada classe.

Tabela 10 – Dataset 3 - Acurácia dos Algoritmos

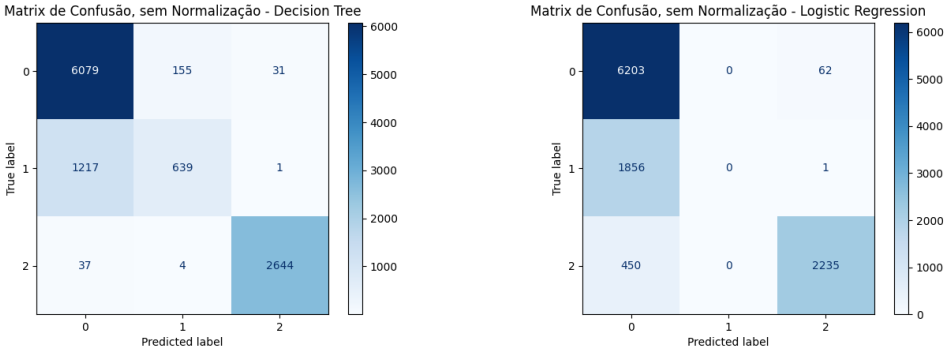
Algoritmo	Acurácia
Decision Tree	86,6%
Naive Bayes	48,4%
Random Forest	87,3%
Logistic Regression	78%

Fonte: Autor

Conforme a Figura 15, os algoritmos *Decision Tree* e *Logistic Regression* possuem uma boa taxa de predições corretas para a classe 2 que representa a detecção de ataques de *Port Scanning*, sendo respectivamente 2644 instâncias (98%) e 2235 instâncias (83%), porém existe uma grande dificuldade dos modelos classificarem corretamente as classes restantes exatamente por ser difícil em um ambiente real conseguir distinguir o que é um fluxo de tráfego normal e

um fluxo de tráfego de background de botnet. Em compensação, a alta eficácia da taxa de predição evidencia que ambos os algoritmos conseguem detectar um ataque de varredura de portas.

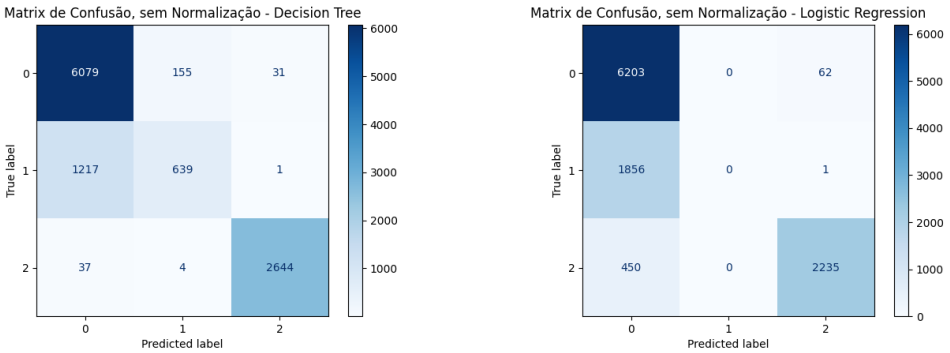
Figura 15 – Dataset 3 - Matriz de Confusão - Decision Tree e Logistic Regression



Fonte: Autor

Para os algoritmos *Naive Bayes* e *Random Forest*, a Figura 16 descreve os resultados obtidos com a matriz de confusão. Apesar da alta taxa de acurácia do *Random Forest*, ele possui 99% de predições corretas para as classes 0 e 2, porém possui a mesma dificuldade dos outros algoritmos para distinguir entre um tráfego normal e um tráfego background de botnet. O *Naive Bayes* possui a mais baixa taxa de acurácia apresentada, porém ele tem alta eficiência ao analisarmos a taxa de *Verdadeiros Positivos* para a classe 2, sendo de 2588 instâncias (96%) com a predição correta. Apesar dos bons resultados, o *Naive Bayes* claramente não é o melhor modelo devido a inconsistência das predições para as classes 0 e 1.

Figura 16 – Dataset 3 - Matriz de Confusão - Decision Tree e Logistic Regression



Fonte: Autor

A Tabela 11 exibe os resultados das validações cruzadas, onde também foi efetuado 5 validações para cada algoritmo. Todos os resultados apresentam uma estabilidade na acurácia obtida e um baixo percentual de variação para todos os quatro algoritmos.

Tabela 11 – Dataset 3 - Validação Cruzada

Algoritmos	Resultados
Decision Tree	[0.87 0.87 0.87 0.87 0.86]
Naive Bayes	[0.48 0.48 0.48 0.49 0.5]
Random Forest	[0.88 0.87 0.87 0.87 0.87]
Logistic Regression	[0.78 0.78 0.79 0.78 0.79]

Fonte: Autor

5.3.1 Dataset 3 - Comportamento no Cenário

A Tabela 12 exibe como os algoritmos classificaram cada tipo das três classes apresentadas, onde observa-se que o ataque utilizando o *Nmap* com maior ruído e o *netcat* apenas não foram detectados pelo algoritmo *Naive Bayes*, que tem coerência aos resultados de acurácia e de matriz de confusão obtidos durante o estudo. O *Nmap* com os parâmetros silenciosos apenas foi classificado pelo algoritmo *Random Forest*, o que evidencia a alta taxa de predições para a classe 2 dado as matrizes de confusão e a própria taxa de acurácia apresentada na Tabela 10.

Tabela 12 – Dataset 3 - Comportamento de Classificação

Ferramenta	DT	NB	RF	RLOG
Netcat	1	0	1	1
Nmap (Silencioso)	0	0	1	0
Nmap (Maior Ruído)	1	0	1	1

Fonte: Autor

5.4 Dataset 4

Com o modelo gerado utilizando a adição dos atributos extras citados na Seção 4.4.1, a Tabela 13 descreve os percentuais de acurácia obtidos dado o modelo para as três classes. Comparando os resultados com a Tabela 10, os algoritmos *Decision Tree* e *Random Forest* obtiveram melhoras perceptíveis de acurácia, respectivamente de 91,3% e 92,4%, enquanto os algoritmos *Naive Bayes* com 50,4% e *Logistic Regression* com 79,4%, apresentam melhoras pouco expressivas. É evidente que existe uma melhora nos percentuais de acurácia, mas é necessário comparar com as predições realizadas utilizando as matrizes de confusão para cada algoritmo.

A Figura 17 exibe os resultados obtidos com a matriz de confusão para os algoritmos *Decision Tree* e *Random Forest*. Ambos os resultados são semelhantes, principalmente quando observa-se a predição da classe 2, sendo respectivamente 2660 instâncias (99%) e 2653 instâncias (99%), mas o mesmo comportamento de repete na dificuldade dos algoritmos classificarem corretamente as classes 0 e 1. Essa dificuldade já foi abordada na Seção 5.3 e há uma melhora significativa nas predições para ambos os algoritmos citados em comparações

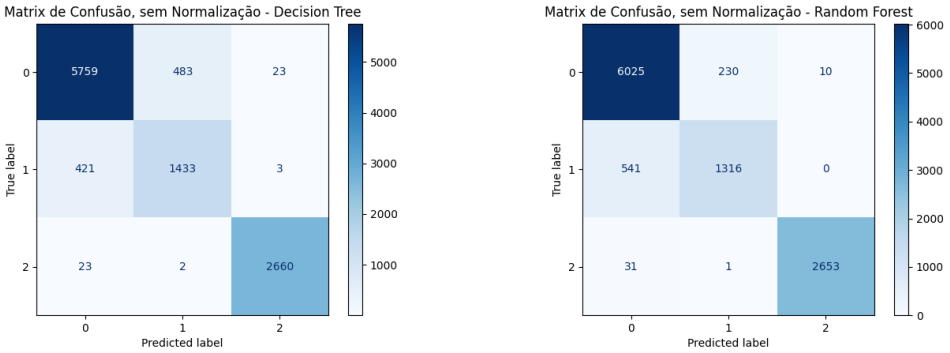
Tabela 13 – Dataset 4 - Acurácia dos Algoritmos

Algoritmo	Acurácia
Decision Tree	91,3%
Naive Bayes	50,4%
Random Forest	92,4%
Logistic Regression	79,4%

Fonte: Autor

com os resultados apresentados da Figura 17, mesmo que ainda ocorram *Falsos Positivos* e *Falsos Negativos* expressivos nas predições.

Figura 17 – Dataset 4 - Matriz de Confusão - Decision Tree e Random Forest



Fonte: Autor

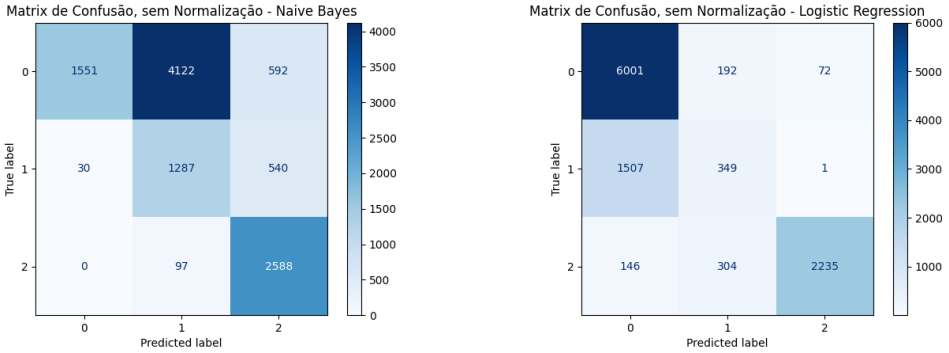
A Figura 18 exibe os resultados obtidos com os algoritmos *Naive Bayes* e *Logistic Regression*, onde ambos possuem uma alta taxa de predições corretas para a classe 2, respectivamente 2588 instâncias (96%) e 2235 instâncias (83%), porém apenas o *Logistic Regression* possui consistência na classificação da classe 0 e uma grande dificuldade em classificar a classe 1 - Tráfego Background de Botnet. O *Naive Bayes* demonstra a mesma inconsistência nas classes 0 e 1, porém em comparação com os dados apresentados na Seção 5.3, houve um aumento de predições incorretas para a classe 0, enquanto o *Logistic Regression* apresentou um melhor percentual em comparação aos resultados citados, mas não de forma expressiva para ser um algoritmo utilizado em ambiente real. A adição dos atributos extras citados na Seção 4.4.1 não contribuiu de forma expressiva nos aspectos de classificação entre classe 0 - Tráfego Normal e classe 1 - Tráfego Background de Botnet.

A Tabela 14 os resultados das validações cruzadas e foi efetuado 5 validações para cada algoritmo. Todos os resultados apresentam uma estabilidade na acurácia e com baixo percentual de variação, exceto o *Logistic Regression* que apresenta um valor constante.

5.4.1 Dataset 4 - Comportamento no Cenário

A Tabela 15 exibe como os algoritmos classificaram cada tipo de tráfego de ataque de botnet para cada ferramenta apresentada. O *netcat* e o *Nmap* com maior ruído são detectados com os algoritmos *Decision Tree*, *Random Forest* e *Logistic Regression*, mas realizando a

Figura 18 – Dataset 4 - Matriz de Confusão - Naive Bayes e Logistic Regression



Fonte: Autor

Tabela 14 – Dataset 4 - Validação Cruzada

Algoritmos	Resultados
Decision Tree	[0.91 0.91 0.91 0.92 0.91]
Naive Bayes	[0.5 0.5 0.5 0.51 0.52]
Random Forest	[0.93 0.92 0.93 0.93 0.92]
Logistic Regression	[0.8 0.8 0.8 0.8 0.8]

Fonte: Autor

comparação com a Tabela 12, os resultados são idênticos e não houve diferença com a adição dos atributos extras. Para o *Nmap* com os parâmetros silenciosos, houve a detecção pelo *Random Forest* e *Logistic Regression*, o que traz uma melhora de detecção comparada a Tabela 12 onde o *Logistic Regression* não foi capaz de detectar o ataque de *Port Scanning*.

Tabela 15 – Dataset 4 - Comportamento de Classificação

Ferramenta	DT	NB	RF	RLOG
Netcat	1	0	1	1
Nmap (Silencioso)	0	0	1	1
Nmap (Maior Ruído)	1	0	1	1

Fonte: Autor

5.5 Considerações Finais

De forma geral, os resultados obtidos neste estudo demonstram que com a adição dos atributos da Seção 4.4.1 e os métodos empregados para a detecção de tráfego botnet apresentam resultados positivos de acurácia com os algoritmos de *Decision Tree*, *Random Forest* e *Logistc Regression*, utilizando as matrizes de confusão e análise dos resultados *ROC* e

AUC para comparação com o *Dataset 1*. Os resultados obtidos para a detecção dos ataques *Port Scanning* de origem de botnets foram superiores em resultados de acurácia e matrizes de confusão dado o *Dataset 4*, que utilizou da adição dos atributos adicionais já citados, mas é importante ressaltar a eficácia do algoritmo *Random Forest* que conseguiu realizar a detecção dos tráfegos e ataques em todos os grupos de datasets citados neste estudo. Com isso, podemos concluir que é viável a aplicação dos modelos em ambiente real e que o mesmo deve ser monitorado já que existem diversas ferramentas e botnets em desenvolvimento atualmente e também em quesito de desempenho e disponibilidade de serviço, que foram pontos não abordados neste estudo e que devem ser levados em consideração ao se utilizar em um ambiente real.

6 CONCLUSÃO

Neste estudo foi abordado como a área de segurança da informação possui um deficit de acompanhar as diversas variações de implementações de botnets a fim de realizar a detecção de tráfego. No Capítulo 2 foi possível explorar os conceitos de botnets, a funcionalidade do padrão de tráfego *Netflow*, os principais conceitos abordados sobre aprendizado de máquina e bem como as principais métricas utilizadas para certificar de que a validação de classificadores é consistente e precisa. A fim explorar o aprendizado de máquina, no Capítulo 3 foi abordado os algoritmos mais utilizados para classificação de tráfego botnet e seus modelos matemáticos.

Dado o contexto, este estudo apresentou um modelo de aprendizado de máquina capaz de realizar tanto a detecção do tráfego de botnets quanto de ataques do tipo *Port Scanning* originados desses botnets. Utilizando então a classificação supervisionada, foi possível analisar os comportamento dos fluxos, determinar os melhores algoritmos para o problema e realizar a analise dos resultados utilizando diversas métricas para uma validação concreta e eficiente. O estudo proposto foi além dos trabalhos semelhantes que utilizaram o *Dataset CTU-13*, do qual autores como Silva, Silva e Salles (2017) e Neira, Medeiro e Nogueira (2020) apenas utilizaram o dataset e validaram a relação da acurácia dos modelos, mas não aplicaram uma melhora na preparação dos dados apresentados para o treinamento ou a aplicação real dos cenários de tráfego e/ou ataque para validar os modelos em um cenário real. O modelo proposto adota técnicas sofisticadas de aprendizado de máquina via classificação supervisionada, técnicas de estatística para a fundamentação das métricas utilizadas durante o estudo e a aplicação prática da ciência de dados ao se utilizar uma adição de atributos para a construção do dataset de treinamento do modelo.

A avaliação do modelo se deu por datasets realísticos, cujos fluxos foram categorizados de forma manual e já é amplamente utilizado em ambienteis reais e acadêmicos conforme o artigo de Garcia *et al.* (2014). O fluxos de ataque de origem de botnet ocorreu em um cenário experimental elaborado para a captura dos mesmo, isolado de eventos e tráfegos externos, proporcionando então integridade dos fluxos.

Com os resultados obtidos no Capítulo 5, foi possível constatar uma melhor eficiência entre determinados datasets e algoritmos para cada objetivo proposto no estudo, assim como é evidente que o modelo é funcional para a detecção de tráfego e ataques de origem de botnets e pode ser replicado e utilizado em ambientes reais ou como base para outros estudos.

6.1 Trabalhos Futuros

Como sugestão para a melhoria desse estudo, adicionar e/ou explorar outros datasets de caráter realísticos como o proposto do *CTU-13*, tal como o *Botnet Dataset 2014* que consta no artigo "Towards effective feature selection in machine learning-based botnet detection approaches" elaborado por Biglar Beigi *et al.* (2014b) ou utilizar diferentes softwares para análise de tráfego, a fim de explorar diferentes atributos e suas composições para a aplicação. Como exemplo, *Tranalyzer2* é um analisador de pacotes projetado para processamento de dumps de pacotes em larga escala e possui integração com o *Cisco NetFlow*, seu artigo "Tranalyzer: Versatile High Performance NetworkTraffic Analyser" de Burschka e Dupasquier (2016) detém mais detalhes sobre o projeto. Um estudo recente com título de "nPrint: Standard Packet-level Network Traffic Analysis" de Holland *et al.* (2020) é outro analisador de tráfego de rede com grande potencial para agregar neste estudo, onde o *nPrint* é uma representação de dados para tráfego de rede destinada a ser utilizado para algoritmos de aprendizado de máquina, criando

então seu próprio padrão de atributos. As três sugestões citadas foram avaliadas durante a fase deste estudo do tema, porém o aumento significativo de tempo para implementação aliado ao curto prazo proposto inviabilizaram um estudo mais aprofundado sobre os artigos citados.

Referências

ANTONAKAKIS, M. *et al.* Understanding the mirai botnet. In: **26th USENIX security symposium (USENIX Security 17)**. [S.l.: s.n.], 2017. p. 1093–1110.

BENGIO, Y.; GRANDVALET, Y. No unbiased estimator of the variance of k-fold cross-validation. **Journal of machine learning research**, v. 5, n. Sep, p. 1089–1105, 2004.

Biglar Beigi, E. *et al.* Towards effective feature selection in machine learning-based botnet detection approaches. In: **2014 IEEE Conference on Communications and Network Security**. [S.l.: s.n.], 2014. p. 247–255.

Biglar Beigi, E. *et al.* Towards effective feature selection in machine learning-based botnet detection approaches. In: **2014 IEEE Conference on Communications and Network Security**. [S.l.: s.n.], 2014. p. 247–255.

BILGE, L. *et al.* Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In: **Proceedings of the 28th Annual Computer Security Applications Conference**. [S.l.: s.n.], 2012. p. 129–138.

BRAGA, A. Curvas roc: aspectos funcionais e aplicações. **Universidade do Minho**, 2001.

BREIMAN, L. Random forests. **Machine learning**, Springer, v. 45, n. 1, p. 5–32, 2001.

BROWNE, M. W. Cross-Validation Methods. **Journal of Mathematical Psychology**, v. 44, n. 1, p. 108–132, 2000. ISSN 0022-2496. Disponível em: http://www.sciencedirect.com/science/article/pii/S0022249699912798.

BRZEZINSKI, J. R.; KNAFL, G. J. Logistic regression modeling for context-based classification. **IEEE**, p. 755–759, 1999.

BURSCHKA, S.; DUPASQUIER, B. Tranalyzer: Versatile high performance network traffic analyser. In: IEEE. **2016 IEEE Symposium Series on Computational Intelligence (SSCI)**. [S.l.], 2016. p. 1–8.

BUZZFEED NEWS. **3ve botnet ad-fraud FBI takedown**. 2018. Acesso em: 20 de novembro 2020.

CANALTECH. **O que é virtualização**. 2012. Disponível em: https://canaltech.com.br/internet/O-que-e-virtualizacao/. Acesso em: 20 de novembro 2020.

CASTRO, C. L. de; BRAGA, A. P. Aprendizado supervisionado com conjuntos de dados desbalanceados. **Sba: Controle & Automação Sociedade Brasileira de Automatica**, scielo, v. 22, p. 441–466, 2011. ISSN 0103-1759. Disponível em: http://www.scielo.br/scielo.php?script=sci_arttext&pid=S0103-17592011000500002&nrm=iso.

COLLINS, M. S. **Network Security Through Data Analysis: Building Situational Awareness**. [S.l.: s.n.], 2014. 325 p. ISSN 1449357881. ISBN 1449357881.

CUNHA, J. P. Z. Um estudo comparativo das técnicas de validação cruzada aplicadas a modelos mistos. **Instituto de Matemática e Estatística, University of São Paulo**, 2019.

DAGON, D. *et al.* A taxonomy of botnet structures. In: . [S.l.: s.n.], 2008. v. 36, p. 325–339. ISBN 978-0-7695-3060-4.

DANGE, S. Iot botnet: The largest threat to the iot network. In: . [S.l.: s.n.], 2019.

EC-COUNCIL. **Nine of the biggest botnet attacks of the 21st century**. 2019. Disponível em: https://blog.eccouncil.org/9-of-the-biggest-botnet-attacks-of-the-21st-century/. Acesso em: 20 de novembro 2020.

FAWCETT, T. Introduction to roc analysis. **Pattern Recognition Letters**, v. 27, p. 861–874, 06 2006.

FEILY, M.; SHAHRESTANI, A.; RAMADASS, S. A survey of botnet and botnet detection. In: IEEE. **2009 Third International Conference on Emerging Security Information, Systems and Technologies**. [S.l.], 2009. p. 268–273.

FRANÇOIS, J. *et al.* Bottrack: tracking botnets using netflow and pagerank. In: SPRINGER. **International Conference on Research in Networking**. [S.l.], 2011. p. 1–14.

GARCIA, S. *et al.* An empirical comparison of botnet detection methods. **computers & security**, Elsevier, v. 45, p. 100–123, 2014.

GONÇALVES, J. W. G. Desenvolvimento de uma botnet com foco em ataques de negaÇÃo de serviÇo. **Universidade Regional de Blumenau**, 2017.

HARRINGTON, P. **Machine Learning in Action**. [S.l.]: Manning Publications Co., 2012. ISBN 9781617290183.

HOLLAND, J. *et al.* **nPrint: Standard Packet-level Network Traffic Analysis**. 2020.

KUHN, M.; JOHNSON, K. *et al.* **Applied Predictive Modeling**. [S.l.]: Springer, 2013. 615 p. ISBN 9781461468486.

LIAW, A.; WIENER, M. *et al.* Classification and regression by randomforest. **R news**, v. 2, n. 3, p. 18–22, 2002.

Lin, W. *et al.* An ensemble random forest algorithm for insurance big data analysis. **IEEE Access**, v. 5, p. 16568–16575, 2017.

MACHART, P.; RALAIVOLA, L. **Confusion Matrix Stability Bounds for Multiclass Classification**. 2012.

MOHAMMED, M.; KHAN, M. B.; BASHIE, E. B. M. **Machine learning: Algorithms and applications**. [S.l.]: Crc Press, 2016. v. 112. 112 p. ISBN 9781498705387.

MUKHERJEE, S.; SHARMA, N. Intrusion Detection using Naive Bayes Classifier with Feature Reduction. **Procedia Technology**, v. 4, p. 119–128, 2012. ISSN 22120173.

NEIRA, A. B.; MEDEIRO, A.; NOGUEIRA, M. Identificação antecipada de botnets por aprendizagem de máquina. In: SBC. **Anais do XXXVIII Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**. [S.l.], 2020. p. 896–909.

NILSSON, N. J. **Introduction to Machine Learning An Early Draft of A Proposed**. [S.l.]: Robotics Laboratory Department of Computer Science Stanford University, 1998.

PRATI, R.; BATISTA, G.; MONARD, M. Curvas roc para avaliação de classificadores. **Revista IEEE América Latina**, v. 6, n. 2, p. 215–222, 2008.

RAILEANU, L. E.; STOFFEL, K. Theoretical comparison between the Gini Index and Information Gain criteria. **Annals of Mathematics and Artificial Intelligence**, v. 41, n. 1, p. 77–93, 2004. ISSN 10122443.

Rodriguez, J. D.; Perez, A.; Lozano, J. A. Sensitivity analysis of k-fold cross validation in prediction error estimation. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 32, n. 3, p. 569–575, 2010.

SANTOS, M. *et al.* Uma arquitetura baseada em botnet para aplicações voltadas a avaliação de aspectos de segurança da informações. In: SBC. **Anais do XLII Seminário Integrado de Software e Hardware**. [S.l.], 2015. p. 59–70.

SHALEV-SHWARTZ, S.; BEN-DAVID, S. **Understanding Machine Learning: From Theory to Algorithms**. [S.l.]: University Press, Cambridge, 2013. 1–397 p. ISBN 9781107057135.

SILVA, D.; SILVA, S.; SALLES, R. M. Metodologia de detecção de botnets utilizando aprendizado de máquina. **XXXV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais**, p. 97–101, 2017.

SMOLA, A.; VISHWANATHAN, S. **Introduction to Machine Learning**. [S.l.]: University Press, Cambridge, 2008. 213 p. ISBN 0521825830.

STEHMAN, S. V. Selecting and interpreting measures of thematic classification accuracy. **Remote Sensing of Environment**, v. 62, n. 1, p. 77–89, 1997. ISSN 0034-4257. Disponível em: http://www.sciencedirect.com/science/article/pii/S0034425797000837.

STONE-GROSS, B. *et al.* Your botnet is my botnet: analysis of a botnet takeover. In: **Proceedings of the 16th ACM conference on Computer and communications security**. [S.l.: s.n.], 2009. p. 635–647.

VISA, S. *et al.* Confusion matrix-based feature selection. In: . [S.l.: s.n.], 2011. v. 710, p. 120–127.

Apêndices

APÊNDICE A – Códigos Utilizados

Neste apêndice consta todos os códigos elaborados para o tratamento dos dados, treinamento dos modelos e aplicação da detecção de botnet abordados no Capítulo 4.

A.1 Preparar Dataset - Duas ou Três classes alvos

preparar_data.py

```
import pandas as pd
import numpy as np

# Dataset - https://www.stratosphereips.org/datasets-ctu13

#Carregando os datasets --- usar o delim_whitespace=True para os pcaps
                                    convertidos

df_s6 = pd.read_csv("./data/pcaps_ataque/nmap_sV.flow",delim_whitespace=
                                    True,low_memory=False)
df_s6 = df_s6.fillna(0)
#df_s6 = pd.read_csv("./data/scenario_6_capture20110817.bitnetflow",
                                    header=0)
#df_s8 = pd.read_csv("./data/scenario_8_capture20110816-3.binetflow",
                                    header=0)


# Drop features
#df_s6 = df_s6.drop(['StartTime', 'SrcAddr', 'Sport', 'DstAddr', 'Dport
                                    '], axis=1)
#df_s8 = df_s8.drop(['StartTime', 'SrcAddr', 'Sport', 'DstAddr', 'Dport
                                    '], axis=1)

# Fill all NaN with -1.
df_s6['dTos'] = df_s6['dTos'].fillna(-1)
df_s6['sTos'] = df_s6['sTos'].fillna(-1)
df_s6['State'] = df_s6['State'].fillna(-1)

# df_s8['dTos'] = df_s8['dTos'].fillna(-1)
# df_s8['sTos'] = df_s8['sTos'].fillna(-1)
# df_s8['State'] = df_s8['State'].fillna(-1)

# Criando novas features
df_s6['TotBytes_over_TotPkts'] = df_s6['TotBytes'] / df_s6['TotPkts']
df_s6['SrcBytes_over_TotPkts'] = df_s6['SrcBytes'] / df_s6['TotPkts']
df_s6['TotPkts_over_Dur'] =  df_s6['TotPkts'] / df_s6['Dur']
df_s6['TotBytes_over_Dur'] =  df_s6['TotBytes'] / df_s6['Dur']
df_s6['SrcBytes_over_Dur'] =  df_s6['SrcBytes'] / df_s6['Dur']

# df_s8['TotBytes_over_TotPkts'] = df_s8['TotBytes'] / df_s8['TotPkts']
# df_s8['SrcBytes_over_TotPkts'] = df_s8['SrcBytes'] / df_s8['TotPkts']
# df_s8['TotPkts_over_Dur'] = df_s8['TotPkts'] / df_s8['Dur']
# df_s8['TotBytes_over_Dur'] =  df_s8['TotBytes'] / df_s8['Dur']
# df_s8['SrcBytes_over_Dur'] =  df_s8['SrcBytes'] / df_s8['Dur']

# Existem Quatro grupos de dataset explorados:
```

```
        # Grupo 1 - Datasset possui 2 labels (Trafego Normal e Trafego
                                              Botnet)
        # Grupo 2 - Datasset possui 2 labels (Trafego Normal e Trafego
                                              Botnet) + features extras
        # Grupo 3 - Datasset possui 3 labels (Trafego Normal e Trafego
                                              Botnet e Trafego background
                                              Botnet)
        # Grupo 3 - Datasset possui 3 labels (Trafego Normal e Trafego
                                              Botnet e Trafego background
                                              Botnet) + features extras

# Denominar as tuples para facilitar a hora da criacao dos datasets.


# Apenas os labels 'from-bonet' e o nosso label alvo do atacante.
# Entao vamos transformar todos os que possuem "from-botnet" em um unico
                                      label alvo.

# ############################################################
# Alterar os valores de Proto, State, Dir e Label para valores inteiros
# ############################################################

# 2 Labels --- Label 0 (Trafego Normal) e Label 1 (Trafego Botnet)
dict_s6_2_labels = {"flow=From-Normal-V47-Jist": 1, "flow=From-Normal-
                                    V47-UDP-CVUT-DNS-Server": 1,"flow=
                                    From-Normal-V47-Grill": 1, "flow=
                                    From-Normal-V47-Stribrek": 1,"flow=
                                    From-Normal-V47-MatLab-Server": 1,"
                                    flow=From-Normal-V47-CVUT-WebServer"
                                    : 1,"flow=Normal-V47-HTTP-
                                    windowsupdate": 1, "flow=From-Botnet
                                    -V47-UDP-DNS": 1,"flow=From-Botnet-
                                    V47-TCP-Established-HTTP-Ad-4": 1,"
                                    flow=From-Botnet-V47-TCP-CC73-Not-
                                    Encrypted": 1,"flow=From-Botnet-V47-
                                    TCP-Attempt-SPAM": 1,"flow=From-
                                    Botnet-V47-TCP-Established-HTTP-Ad-
                                    62": 1,"flow=From-Botnet-V47-TCP-
                                    Attempt": 1,"flow=From-Botnet-V47-
                                    UDP-Attempt": 1}

dict_s8_2_labels = {"flow=From-Normal-V49-Stribrek": 1,"flow=From-Normal
                                    -V49-Grill": 1,"flow=From-Normal-V49
                                    -Jist": 1,"flow=From-Normal-V49-CVUT
                                    -WebServer": 1, "flow=From-Botnet-
                                    V49-UDP-DNS": 1,"low=From-Botnet-V49
                                    -TCP-Established-HTTP-Ad-40": 1,"
                                    flow=From-Botnet-V49-TCP-HTTP-Google
                                    -Net-Established-6": 1,"flow=From-
                                    Botnet-V49-UDP-Attempt": 1,"flow=
                                    From-Botnet-V49-TCP-Attempt": 1,"
                                    flow=From-Botnet-V49-TCP-WEB-
                                    Established": 1,"flow=From-Botnet-
                                    V49-UDP-Established": 1,"flow=From-
                                    Botnet-V49-TCP-Established": 1,"flow
                                    =From-Botnet-V49-TCP-CC76-HTTP-
                                    Custom-Port-Not-Encrypted-Binary-
                                    Download": 1,"flow=From-Botnet-V49-
                                    TCP-CC74-HTTP-Custom-Port-Not-
```

```
                                    Encrypted": 1,"flow=From-Botnet-V49-
                                    TCP-CC75-HTTP-Custom-Port-Not-
                                    Encrypted-Non-Periodic": 1,"flow=
                                    From-Botnet-V49-TCP-Established-HTTP
                                    -Binary-Download-11": 1,"flow=From-
                                    Botnet-V49-TCP-Established-HTTP-Ad-
                                    62": 1,"flow=From-Botnet-V49-UDP-
                                    Established-Custom-Encryption-2": 1,
                                    "flow=From-Botnet-V49-UDP-
                                    Established-Custom-Encryption-1": 1}

# 3 lalbels --- Label 0 (Trafego Normal) e Label 1 (Trafego Background
                                    Botnet) e Label 2 (Trafego Botnet)
dict_s6_3_labels = {"flow=From-Normal-V47-Jist": 1, "flow=From-Normal-
                                    V47-UDP-CVUT-DNS-Server": 1,"flow=
                                    From-Normal-V47-Grill": 1, "flow=
                                    From-Normal-V47-Stribrek": 1,"flow=
                                    From-Normal-V47-MatLab-Server": 1,"
                                    flow=From-Normal-V47-CVUT-WebServer"
                                    : 1,"flow=Normal-V47-HTTP-
                                    windowsupdate": 1,"flow=From-Botnet-
                                    V47-UDP-DNS": 2,"flow=From-Botnet-
                                    V47-TCP-Established-HTTP-Ad-4": 2,"
                                    flow=From-Botnet-V47-TCP-CC73-Not-
                                    Encrypted": 2,"flow=From-Botnet-V47-
                                    TCP-Attempt-SPAM": 2,"flow=From-
                                    Botnet-V47-TCP-Established-HTTP-Ad-
                                    62": 2,"flow=From-Botnet-V47-TCP-
                                    Attempt": 2,"flow=From-Botnet-V47-
                                    UDP-Attempt": 2}

dict_s8_3_labels ={"flow=From-Normal-V49-Stribrek": 1,"flow=From-Normal-
                                    V49-Grill": 1,"flow=From-Normal-V49-
                                    Jist": 1,"flow=From-Normal-V49-CVUT-
                                    WebServer": 1, "flow=From-Botnet-V49
                                    -UDP-DNS": 2,"low=From-Botnet-V49-
                                    TCP-Established-HTTP-Ad-40": 2,"flow
                                    =From-Botnet-V49-TCP-HTTP-Google-Net
                                    -Established-6": 2,"flow=From-Botnet
                                    -V49-UDP-Attempt": 2,"flow=From-
                                    Botnet-V49-TCP-Attempt": 2,"flow=
                                    From-Botnet-V49-TCP-WEB-Established"
                                    : 2,"flow=From-Botnet-V49-UDP-
                                    Established": 2,"flow=From-Botnet-
                                    V49-TCP-Established": 2,"flow=From-
                                    Botnet-V49-TCP-CC76-HTTP-Custom-Port
                                    -Not-Encrypted-Binary-Download": 2,"
                                    flow=From-Botnet-V49-TCP-CC74-HTTP-
                                    Custom-Port-Not-Encrypted": 2,"flow=
                                    From-Botnet-V49-TCP-CC75-HTTP-Custom
                                    -Port-Not-Encrypted-Non-Periodic": 2
                                    ,"flow=From-Botnet-V49-TCP-
                                    Established-HTTP-Binary-Download-11"
                                    : 2,"flow=From-Botnet-V49-TCP-
                                    Established-HTTP-Ad-62": 2,"flow=
                                    From-Botnet-V49-UDP-Established-
                                    Custom-Encryption-2": 2,"flow=From-
                                    Botnet-V49-UDP-Established-Custom-
                                    Encryption-1": 2}
```

```
# Associando numeros integer a cada protocolo
dict_proto = {'tcp': 0, 'udp': 1, 'ipx/spx': 2, 'arp': 3, 'icmp': 4, '
                                    pim': 5, 'rtcp': 6, 'igmp' : 7, '
                                    ipv6-icmp': 8, 'esp': 9, 'ipv6': 10,
                                     'udt': 11,'rtp':12, 'rarp':13, '
                                    rsvp':14, 'unas':15}

# Associando numeros integer a cada direcco
dict_dir = {'  <?>':0,'<?>':0, '  <->':1,'<->':1, '   ?>':2,'?>':2, '
                                    ->':3,'->':3, '  who':4,'who':4, '
                                    <-':5, '<-':5,'  <?':6,'<?':6}

# Associando numeros integer a cada state
dict_state = {'': 1, 'FSR_SA': 30, '_FSA': 296, 'FSRPA_FSA': 77, 'SPA_SA
                                    ': 31, 'FSA_SRA': 1181, 'FPA_R': 46,
                                     'SPAC_SPA': 37, 'FPAC_FPA': 2, '_R'
                                    : 1, 'FPA_FPA': 784, 'FPA_FA': 66, '
                                    _FSRPA': 1, 'URFIL': 431, 'FRPA_PA':
                                     5, '_RA': 2, 'SA_A': 2, 'SA_RA':
                                    125, 'FA_FPA': 17, 'FA_RA': 14, '
                                    PA_FPA': 48, 'URHPRO': 380, '
                                    FSRPA_SRA': 8, 'R_':541, 'DCE': 5, '
                                    SA_R': 1674, 'SA_': 4295, 'RPA_FSPA'
                                    : 4, 'FA_A': 17, 'FSPA_FSPAC': 7, '
                                    RA_': 2230, 'FSRPA_SA': 255, 'NNS':
                                    47, 'SRPA_FSPAC': 1, 'RPA_FPA': 42,
                                    'FRA_R': 10, 'FSPAC_FSPA': 86, '
                                    RPA_R': 3, '_FPA': 5, 'SREC_SA': 1,
                                    'URN': 339, 'URO': 6, 'URH': 3593, '
                                    MRQ': 4, 'SR_FSA': 1, 'SPA_SRPAC': 1
                                    , 'URP': 23598, 'RPA_A': 1, 'FRA_':
                                    351, 'FSPA_SRA': 91, 'FSA_FSA':
                                    26138, 'PA_': 149, 'FSRA_FSPA': 798,
                                     'FSPAC_FSA': 11, 'SRPA_SRPA': 176,
                                    'SA_SA': 33, 'FSPAC_SPA': 1, 'SRA_RA
                                    ': 78, 'RPAC_PA': 1, 'FRPA_R': 1, '
                                    SPA_SPA': 2989, 'PA_RA': 3, '
                                    SPA_SRPA': 4185, 'RA_FA': 8, '
                                    FSPAC_SRPA': 1, 'SPA_FSA': 1, '
                                    FPA_FSRPA': 3, 'SRPA_FSA': 379, '
                                    FPA_FRA': 7, 'S_SRA': 81, 'FSA_SA':
                                    6, 'State': 1, 'SRA_SRA': 38, 'S_FA'
                                    : 2, 'FSRPAC_SPA': 7, 'SRPA_FSPA':
                                    35460, 'FPA_A': 1, 'FSA_FPA': 3, '
                                    FRPA_RA': 1, 'FSAU_SA': 1, '
                                    FSPA_FSRPA': 10560, 'SA_FSA': 358, '
                                    FA_FRA': 8, 'FSRPA_SPA': 2807, '
                                    FSRPA_FSRA': 32, 'FRA_FPA': 6, '
                                    FSRA_FSRA': 3, 'SPAC_FSRPA': 1, 'FS_
                                    ': 40, 'FSPA_FSRA': 798, 'FSAU_FSA':
                                     13, 'A_R': 36, 'FSRPAE_FSPA': 1, '
                                    SA_FSRA': 4, 'PA_PAC': 3, 'FSA_FSRA'
                                    : 279, 'A_A': 68, 'REQ': 892, 'FA_R'
                                    : 124, 'FSRPA_SRPA': 97, 'FSPAC_FSRA
                                    ':20, 'FRPA_RPA': 7, 'FSRA_SPA': 8,
                                    'INT': 85813, 'FRPA_FRPA': 6, '
                                    SRPAC_FSPA': 4, 'SPA_SRA': 808, '
                                    SA_SRPA': 1, 'SPA_FSPA': 2118, '
```

```
                                    FSRAU_FSA': 2, 'RPA_PA': 171,'_SPA':
                                     268, 'A_PA': 47, 'SPA_FSRA': 416, '
                                    FSPA_FSRPAC': 2, 'PAC_PA': 5, '
                                    SRPA_SPA': 9646, 'SRPA_FSRA': 13, '
                                    FPA_FRPA': 49, 'SRA_SPA': 10, '
                                    SA_SRA': 838, 'PA_PA': 5979, '
                                    FPA_RPA': 27, 'SR_RA': 10, 'RED':
                                    4579, 'CON': 2190507, 'FSRPA_FSPA':
                                    13547, 'FSPA_FPA': 4, 'FAU_R': 2, '
                                    ECO': 2877, 'FRPA_FPA': 72, '
                                    FSAU_SRA': 1, 'FRA_FA': 8, '
                                    FSPA_FSPA': 216341, 'SEC_RA': 19, '
                                    ECR': 3316, 'SPAC_FSPA': 12, 'SR_A':
                                     34, 'SEC_': 5, 'FSAU_FSRA': 3, '
                                    FSRA_FSRPA': 11, 'SRC': 13, 'A_RPA':
                                     1, 'FRA_PA': 3, 'A_RPE': 1, '
                                    RPA_FRPA': 20, '_SRA': 74, 'SRA_FSPA
                                    ': 293, 'FPA_': 118, 'FSRPAC_FSRPA':
                                     2,
'_FA': 1, 'DNP': 1, 'FSRPA_FSRPA': 379, 'FSRA_SRA': 14, '_FRPA': 1, 'SR_
                                    ': 59, 'FSPA_SPA': 517, 'FRPA_FSPA':
                                     1, 'PA_A': 159, 'PA_SRA': 1, '
                                    FPA_RA': 5, 'S_': 68710, 'SA_FSRPA':
                                     4, 'FSA_FSRPA': 1, 'SA_SPA': 4, '
                                    RA_A': 5, '_SRPA': 9, 'S_FRA': 156,
                                    'FA_FRPA': 1, 'PA_R': 72, '
                                    FSRPAEC_FSPA': 1, '_PA': 7, 'RA_S':
                                    1, 'SA_FR': 2, 'RA_FPA': 6, 'RPA_':
                                    5, '_FSPA': 2395, 'FSA_FSPA': 230, '
                                    UNK': 2, 'A_RA': 9, 'FRPA_': 6, 'URF
                                    ': 10, 'FS_SA': 97, 'SPAC_SRPA': 8,
                                    'S_RPA': 32, 'SRPA_SRA': 69, 'SA_RPA
                                    ': 30, 'PA_FRA': 4, 'FSRA_SA': 49, '
                                    FSRA_FSA': 206, 'PAC_RPA': 1, 'SRA_'
                                    : 18, 'FA_': 451, 'S_SA': 6917, '
                                    FSPA_SRPA': 427, 'TXD': 542,'SRA_SA'
                                    : 1514, 'FSPA_FA': 1, 'FPA_FSPA': 10
                                    , 'RA_PA': 3, 'SRA_FSA': 709, '
                                    SRPA_SPAC': 3, 'FSPAC_FSRPA': 10, '
                                    A_': 191, 'URNPRO': 2, 'PA_RPA': 81,
                                     'FSPAC_SRA':1, 'SRPA_FSRPA': 3054,
                                    'SPA_': 1, 'FA_FA': 259, 'FSPA_SA':
                                    75, 'SR_SRA': 1, 'FSA_': 2, 'SRPA_SA
                                    ': 406, 'SR_SA': 3119, 'FRPA_FA': 1,
                                     'PA_FRPA': 13, 'S_R': 34, '
                                    FSPAEC_FSPAE': 3, 'S_RA': 61105, '
                                    FSPA_FSA': 5326, '_SA': 20, 'SA_FSPA
                                    ': 15, 'SRPAC_SPA': 8, 'FPA_PA': 19,
                                     'FSRPAE_FSA': 1, 'S_A': 1, 'RPA_RPA
                                    ': 3, 'NRS': 6, 'RSP': 115, '
                                    SPA_FSRPA': 1144, 'FSRPAC_FSPA': 139
                                    ,'FRPA_FRA':99901,'FRPA_A':99902,'
                                    FRA_A':99903,'SRA_FPA':99904,'
                                    FSRA_FA':99905,'FRA_RA':99906,'
                                    FA_FSA':99907,'IRQ':99908,'PA_SA':
                                    99909,'SRPA_FPA':99910,'A_FA':99911,
                                    'SA_FA':99912,'FSA_FA':99913,'
                                    SRA_FSRA':99914,'FSRAEC_SPA':99915,'
                                    FSPAEC_FSPA':99916,'FSRAC_SPA':99917
```

```
                                        ,'PA_FSPA':99918,'_RPA':99919,'
                                        SRAEC_FSA':99920,'S_SPA':99921,'
                                        FRPA_FPAC':99922,'SRPA_PA':99923,'
                                        FSRPA_FSPAC':99924,'FA_FSPA':99925,'
                                        SRA_FA':99926,'FSRPA_FPA':99927,'
                                        FSPA_A':99928,'S_FSPA':99929,'
                                        RPA_FSA':99930,'FSPA_FRPA':99931,'
                                        RPA_SPA':99932,'SEC_SA':99933,'
                                        RPA_FSRPA':99934,'RA_R':99935,'R_FA'
                                        :99936,'SRC_SA':99937,'SRA_R':99938,
                                        'FRPA_SPA':99939,'RA_RPA':99940,'
                                        SPA_PA':99941,'FSA_A':99942,'SPA_FA'
                                        :99943,'SRAE_RA':99944,'FPA_FRPAC':
                                        99945,'PA_FA':99946,'A_FRA':99947,'
                                        FA_PA':99948,'FRPAC_FPA':99949, '
                                        RPA_RA':99950,'FPAC_FRPA' :99951,'
                                        R_PA':99952,'A_FPA' :99953,'R_FPA' :
                                        99954,'SRE_SA':99955,'SRPAC_SRPA':
                                        99956,'SPAEC_SPA' :99957,'FRPAC_PA':
                                        99958,'F_':99959,'A_FSPE':99960,'
                                        FSAEC_FSPAE':99961,'SRPAEC_FSPA':
                                        99962,'SREC_':99963,'SPA_R':99964,'
                                        NaN':99965,'_':99966}

# Cenario 6 e 8 - Setando os labels e preenchendo o restante com zero
# df_s6['Label'].replace(dict_s6_3_labels, inplace=True)
# df_s6['Label'] = df_s6['Label'].replace(regex='([a-zA-Z])', value=0)

# df_s8['Label'].replace(dict_s8_3_labels, inplace=True)
# df_s8['Label'] = df_s8['Label'].replace(regex='([a-zA-Z])', value=0)

# Cenario 6
df_s6["Proto"].replace(dict_proto, inplace=True)
df_s6["State"].replace(dict_state, inplace=True)

# Cenario 8
# df_s8["Proto"].replace(dict_proto, inplace=True)
# df_s8["State"].replace(dict_state, inplace=True)

# Cenario 6
df_s6["Dir"].replace(dict_dir, inplace=True)
# Cenario 8
# df_s8["Dir"].replace(dict_dir, inplace=True)

# Drop Inf e NaN ---> df.replace([np.inf, -np.inf], np.nan).dropna(axis=
                                        1)
df_s6 = df_s6.replace([np.inf, -np.inf], 0)
# df_s8 = df_s8.replace([np.inf, -np.inf], 0)
df_s6 = df_s6.fillna(0)
# Salvar em CSV
df_s6.to_csv("./data/dataset_port_scanning/nmap_sV_features.csv", index
                                        = False)
# df_s8.to_csv("./data/ataques/cenario_8__ataque_features_extra.csv",
                                        index = False)

# print(df_s8['Label'].unique())
# print(df_s6['Label'].value_counts())
```

A.2 Gerar Modelos

gerar_modelos.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import pickle

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn import metrics

# Setar que as saidas so possuam duas casas decimais
float_formatter = "{:.2f}".format
np.set_printoptions(formatter={'float_kind':float_formatter})
np.set_printoptions(precision=2)

n_state = 74
# Load Data
df_c6 = pd.read_csv("./data/dataset_full/ataque/cenario_6__ataque.csv" ,
                                    low_memory=False)
df_c8 = pd.read_csv("./data/dataset_full/ataque/cenario_8__ataque.csv",
                                    low_memory=False)

df_c6 = df_c6.fillna(0)
df_c8 = df_c8.fillna(0)

# Separar uma proporcco de labels 0 (background), labels 1 (background-
                                    botnet) e labels 2 (botnet)
df6_l0 = df_c6.loc[df_c6['Label'] == 0] # 513527 instancias
df6_l1 = df_c6.loc[df_c6['Label'] == 1] # 7433 instancias
df6_l2 = df_c6.loc[df_c6['Label'] == 2] # 230 instancias
df8_l2 = df_c8.loc[df_c8['Label'] == 2] # 397 instancias

# Pegar as 1k primeiras instancias do label 0, 1k do label 1 e todos do
                                    label 2 em ambos os cenarios
df6_l0 = df6_l0[:25000]
# Concatenar os dataframes
frames = [df6_l0, df6_l1, df6_l2, df8_l2]
df_final = pd.concat(frames)

# separar X- features e y-labels
x_c6 = df_final.drop(['Label'],axis=1).values
y_c6 = df_final['Label'].values

# Separando data de treino e de validacco
train_x_c6, test_x_c6, train_y_c6, test_y_c6 = train_test_split(x_c6,
                                    y_c6, test_size = 0.25, random_state
                                     = n_state)
```

```
scaling = MinMaxScaler(feature_range=(-1,1)).fit(train_x_c6)
train_x_c6 = scaling.transform(train_x_c6)
test_x_c6 = scaling.transform(test_x_c6)


def save_pickle(nome,clf):
    pickle.dump(clf, open("./modelos_treinados/trafego_f_extra/"+ nome +
                                        "_f.pkl", 'wb'))
    pass

def plot_conf_matrix(nome_modelo,clf_plot):

    titles_options = [("Matrix de Confusco, sem Normalizacco - "+
                                        nome_modelo, None), ("Matrix de
                                        Confusco Normalizada - "+
                                        nome_modelo, 'true')]
    for title, normalize in titles_options:
        disp = plot_confusion_matrix(clf_plot, test_x_c6, test_y_c6,cmap
                                            =plt.cm.Blues, normalize=
                                            normalize)
        disp.ax_.set_title(title)

        print(title)
        print(disp.confusion_matrix)

    plt.show()

def plt_roc_biclass(dt,nb,rf,rlog,y_test):
    plt.figure(figsize=(12,7))


    preds_dt = dt[:,1]
    preds_nb = nb[:,1]
    preds_rf = rf[:,1]
    preds_rlog = rlog[:,1]

    fpr_dt, tpr_dt, threshold_dt = metrics.roc_curve(y_test, preds_dt)
    roc_auc_dt = metrics.auc(fpr_dt, tpr_dt)
    plt.plot(fpr_dt, tpr_dt,label="AUC - Decision Tree: = %0.2f" %
                                        roc_auc_dt, color='blue',
                                        linewidth=2)

    fpr_nb, tpr_nb, threshold_nb = metrics.roc_curve(y_test, preds_nb)
    roc_auc_nb = metrics.auc(fpr_nb, tpr_nb)
    plt.plot(fpr_nb, tpr_nb,label="AUC - Naive Bayes: = %0.2f" %
                                        roc_auc_nb, color='orange',
                                        linewidth=2)

    fpr_rf, tpr_rf, threshold_rf = metrics.roc_curve(y_test, preds_rf)
    roc_auc_rf = metrics.auc(fpr_rf, tpr_rf)
    plt.plot(fpr_rf, tpr_rf,label="AUC - Random Forest: = %0.2f" %
                                        roc_auc_rf, color='green',
                                        linewidth=2)

    fpr_rlog, tpr_rlog, threshold_rlog = metrics.roc_curve(y_test,
                                        preds_rlog)
    roc_auc_rlog = metrics.auc(fpr_rlog, tpr_rlog)
```

```
    plt.plot(fpr_rlog, tpr_rlog,label="AUC - Logistic Regression: = %0.
                                        2f" % roc_auc_rlog, color='
                                        purple', linewidth=2)


    plt.plot([0, 1], [0, 1], 'r--', lw=1)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.title('Receiver Operating Characteristic - Dataset 2')
    plt.xlabel('Taxa de Falso Positivo')
    plt.ylabel('Taxa de Verdadeiro Positivo')
    plt.grid(True)
    plt.legend(loc="lower right")
    plt.show()

# ##########################################
# Algoritmos ML
# ##########################################

def dt():
    clf_dt = DecisionTreeClassifier().fit(train_x_c6, train_y_c6)
    y_model_dt = clf_dt.predict(test_x_c6)
    save_pickle("dt",clf_dt)
    print("Decision Tree accuracy: ", accuracy_score(test_y_c6,
                                        y_model_dt))
    print(cross_val_score(clf_dt, train_x_c6, train_y_c6))

    plot_conf_matrix("Decision Tree",clf_dt)
    probs = clf_dt.predict_proba(test_x_c6)
    return probs

def nb():
    clf_nb = GaussianNB().fit(train_x_c6, train_y_c6)
    y_model_nb = clf_nb.predict(train_x_c6)

    save_pickle("nb",clf_nb)

    print("Naive Bayes accuracy : ",accuracy_score(train_y_c6,
                                        y_model_nb))
    print(cross_val_score(clf_nb, train_x_c6, train_y_c6))

    plot_conf_matrix("Naive Bayes",clf_nb)

    probs = clf_nb.predict_proba(test_x_c6)
    return probs

def random_forest():
    clf_rf = RandomForestClassifier(n_estimators=1000, random_state=1,
                                        criterion='entropy', bootstrap=
                                        True, oob_score=True, verbose=0,
                                         n_jobs=-1).fit(train_x_c6,
                                        train_y_c6)
    y_pred_R =clf_rf.predict(test_x_c6)

    save_pickle("rf",clf_rf)

    print("Random Forest Accuracy:", accuracy_score(test_y_c6, y_pred_R)
                                        )
    print(cross_val_score(clf_rf,train_x_c6, train_y_c6))
```

```
    plot_conf_matrix("Random Forest",clf_rf)

    probs = clf_rf.predict_proba(test_x_c6)
    return probs

def rlog():
    clf_rlog = LogisticRegression(random_state=0, max_iter=2500).fit(
                                        train_x_c6, train_y_c6)
    y_pred_rlog = clf_rlog.predict(test_x_c6)

    save_pickle("rlog",clf_rlog)

    print("R Log Accuracy:", accuracy_score(test_y_c6, y_pred_rlog))
    print(cross_val_score(clf_rlog,train_x_c6, train_y_c6))

    plot_conf_matrix("Logistic Regression",clf_rlog)

    probs = clf_rlog.predict_proba(test_x_c6)
    return probs


# Main
prob_dt = dt()
prob_nb = nb()
prob_rf = random_forest()
prob_rlog = rlog()
plt_roc_biclass(prob_dt,prob_nb,prob_rf,prob_rlog,test_y_c6)
```

A.3 Detectar Botnet

detect_botnet.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pickle import load

from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn import metrics

# Setar que as saidas so possuam duas casas decimais
float_formatter = "{:.2f}".format
np.set_printoptions(formatter={'float_kind':float_formatter})
np.set_printoptions(precision=2)

def pred(inx):
```

```
    dt_y = clf_dt.predict(inx)
    nb_y = clf_nb.predict(inx)
    rf_y = clf_rf.predict(inx)
    rlog_y = clf_rlog.predict(inx)

    print('\nDT: ', dt_y)
    print(np.unique(dt_y))

    print('\nNB: ', nb_y)
    print(np.unique(nb_y))

    print('\nRF: ', rf_y)
    print(np.unique(rf_y))

    print('\nRLOG: ', rlog_y)
    print(np.unique(rlog_y))

nc = pd.read_csv("./data/dataset_port_scanning/nc_p22.csv",low_memory=
                                    False).values
nmap_sV = pd.read_csv("./data/dataset_port_scanning/nmap_sV.csv",
                                    low_memory=False).values
nmap_T5 = pd.read_csv("./data/dataset_port_scanning/nmap_T5.csv",
                                    low_memory=False).values


clf_dt =  load(open('./modelos_treinados/ataque/dt.pkl', 'rb')) # 3
                                    labels
clf_nb =  load(open('./modelos_treinados/ataque/nb.pkl', 'rb')) # 3
                                    labels
clf_rf =  load(open('./modelos_treinados/ataque/rf.pkl', 'rb')) # 3
                                    labels
clf_rlog =  load(open('./modelos_treinados/ataque/rlog.pkl', 'rb')) # 3
                                    labels

print("\nnc",pred(nc))
print("\nnmap sV",pred(nmap_sV))
print("\nnmap T5",pred(nmap_T5))
```