UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Instituto de Biociências, Letras e Ciências Exatas
Programa de Pós-Graduação em Ciência da Computação

Luan Nunes Utimura

Aplicação em Tempo Real de Técnicas de Aprendizado de Máquina no Snort IDS

São José do Rio Preto
2020

Luan Nunes Utimura

Aplicação em Tempo Real de Técnicas de Aprendizado de Máquina no Snort IDS

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Instituto de Biociências, Letras e Ciências Exatas, Campus São José do Rio Preto.

Financiadora: CAPES

Orientador: Prof. Dr. Kelton Augusto Pontara da Costa

São José do Rio Preto
2020

Utimura, Luan Nunes.
Aplicação em Tempo Real de Técnicas de Aprendizado de Máquina no Snort IDS / Luan Nunes Utimura. – São José do Rio Preto, 2020
97 f. : il., tabs.

Orientador: Prof. Dr. Kelton Augusto Pontara da Costa
Dissertação (mestrado) - Universidade Estadual Paulista “Júlio de Mesquita Filho", Instituto de Biociências, Letras e Ciências Exatas

1. Sistemas de Detecção de Intrusão. 2. Aprendizado de Máquina. 3. Detecção de Anomalias. 4. Snort. I. da Costa, Kelton Augusto Pontara II. Universidade Estadual Paulista “Júlio de Mesquita Filho", Instituto de Biociências, Letras e Ciências Exatas. III. Título.

CDU – 518.72:76

Ficha catalográfica elaborada pela biblioteca do IBILCE
UNESP - Câmpus de São José do Rio Preto

Luan Nunes Utimura

Aplicação em Tempo Real de Técnicas de Aprendizado de Máquina no Snort IDS

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Instituto de Biociências, Letras e Ciências Exatas, Campus São José do Rio Preto.

Financiadora: CAPES

Comissão Examinadora

Prof. Dr. Kelton Augusto Pontara da Costa
UNESP - Câmpus de Bauru
Orientador

Prof. Dr. Danillo Roberto Pereira
UNOESTE - Universidade do Oeste Paulista

Prof. Dr. João Paulo Papa
UNESP - Câmpus de Bauru

Bauru
20 de março de 2020

Agradecimentos

Primeiramente, agradeço aos meus pais, Cesar e Flávia, pelo imensurável amor e carinho que eles sempre demonstraram por mim. Sou eternamente grato a eles pelos ensinamentos, valores e virtudes que me passaram, que essencialmente moldaram quem sou hoje. Espero um dia poder retribuir por tudo aquilo que eles fizeram por mim.

Agradeço aos meus irmãos, Luma e Lucas, pelas risadas, pelos bons momentos e, sobretudo, pelo amor e companheirismo que nós três temos. Sou muito grato por ter vocês em minha vida.

Agradeço ao meu orientador, Kelton, pela oportunidade de poder me aprofundar na área, desenvolvendo este e outros projetos, além de sempre estar à disposição quando precisei.

Agradeço a todos os meus amigos de São Paulo e Mogi das Cruzes que, ao longo desses dois anos, sempre estiveram comigo, independente da distância.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

"Every process has a genesis and ends with a revelation."
(Shingo Annen)

Resumo

À medida que a Internet cresce com o passar dos anos, é possível observar um aumento na quantidade de dados que trafegam nas redes de computadores do mundo todo. Em um contexto onde o volume de dados encontra-se em constante renovação, sob a perspectiva da área de Segurança de Redes de Computadores torna-se um grande desafio assegurar, em termos de eficácia e eficiência, os sistemas computacionais da atualidade. Dentre os principais mecanismos de segurança empregados nestes ambientes, destacam-se os Sistemas de Detecção de Intrusão em Rede. Muito embora a abordagem de detecção por assinatura seja suficiente no combate de ataques conhecidos, com a eventual descoberta de novas vulnerabilidades, faz-se necessário a utilização de abordagens de detecção por anomalia para amenizar o dano de ataques desconhecidos. No campo acadêmico, diversos trabalhos têm explorado o desenvolvimento de abordagens híbridas com o intuito de melhorar a acurácia dessas ferramentas, com o auxílio de técnicas de Aprendizado de Máquina. Nesta mesma linha de pesquisa, o presente trabalho propõe a aplicação destas técnicas para a detecção de intrusão em um ambiente tempo real mediante uma ferramenta popular e amplamente utilizada, o Snort. Os resultados obtidos mostram que em determinados cenários de ataque, a abordagem de detecção baseada em anomalia pode se sobressair em relação à abordagem de detecção baseada em assinatura, com destaque às técnicas *AdaBoost*, Florestas Aleatórias, Árvore de Decisão e Máquina de Vetores de Suporte Linear.

Palavras-chave: Sistemas de Detecção de Intrusão. Aprendizado de Máquina. Detecção de Anomalias. Snort.

Abstract

As the Internet grows over the years, it is possible to observe an increase in the amount of data that travels on computer networks around the world. In a context where data volume is constantly being renewed, from the perspective of the Network Security area it becomes a great challenge to ensure, in terms of effectiveness and efficiency, today's computer systems. Among the main security mechanisms employed in these environments, stand out the Network Intrusion Detection Systems. Although the signature-based detection approach is sufficient to combat known attacks, with the eventual discovery of new vulnerabilities, it is necessary to use anomaly-based detection approaches to mitigate the damage of unknown attacks. In the academic field, several works have explored the development of hybrid approaches in order to improve the accuracy of these tools, with the aid of Machine Learning techniques. In this same line of research, the present work proposes the application of these techniques for intrusion detection in a real time environment using a popular and widely used tool, the Snort. The obtained results shows that in certain attack scenarios, the anomaly-based detection approach may outperform the signature-based detection approach, with emphasis on the techniques AdaBoost, Random Forests, Decision Tree and Linear Support Vector Machine.

Keywords: Intrusion Detection Systems. Machine Learning. Anomaly Detection. Snort.

Lista de ilustrações

Figura 1 – Um Sistema de Detecção de Intrusão Simples 19
Figura 2 – Perfis dos Comportamentos de Invasores e Usuários Autorizados 20

Figura 3 – Exemplo de *underfitting*, um bom treinamento e *overfitting* 29
Figura 4 – Exemplo de *Naive Bayes* 31
Figura 5 – Exemplo de Árvore de Decisão 34
Figura 6 – Representação do *Bootstrap* 36
Figura 7 – Exemplo de Florestas Aleatórias 37
Figura 8 – Mapeamento da função Φ 38
Figura 9 – Exemplo de Hiperplano 40
Figura 10 – Exemplo de Matriz de Confusão 42

Figura 11 – Fases de Preparação e Execução do Projeto 55
Figura 12 – Infraestrutura do Ambiente de Testes do CICIDS2017 58
Figura 13 – Criação e Atualização de Fluxos de Tráfego de Rede 63
Figura 14 – *Timeout* de Fluxos de Tráfego de Rede 64

Figura 15 – Resultados Gerais de Acurácia, Precisão, *Recall* e *F-Score* 77
Figura 16 – Resultados Gerais de Falso-Positivos e Falso-Negativos 78
Figura 17 – Resultados Gerais de Tempo de Treinamento e Teste 79
Figura 18 – Infraestrutura do Ambiente de Testes da Segunda Etapa dos Experimentos 80

Lista de tabelas

Tabela 1 – Exemplo de funções *kernel* 39

Tabela 2 – Tipos de *plugins* no Snort 3 60
Tabela 3 – Tipos de *Inspectors* 61
Tabela 4 – Características dos fluxos de tráfego de rede (1 a 26) 65
Tabela 5 – Características dos fluxos de tráfego de rede (27 a 52) 66
Tabela 6 – Características dos fluxos de tráfego de rede (53 a 77) 67

Tabela 7 – Resultados com Naive Bayes Bernoulli 69
Tabela 8 – Matriz de Confusão da Naive Bayes Bernoulli 69
Tabela 9 – Resultados com Naive Bayes Gaussiano 70
Tabela 10 – Matriz de Confusão da Naive Bayes Gaussiano 70
Tabela 11 – Hiperparâmetros da Árvore de Decisão 71
Tabela 12 – Resultados com Árvore de Decisão 71
Tabela 13 – Matriz de Confusão da Árvore de Decisão 72
Tabela 14 – Hiperparâmetros das Florestas Aleatórias 72
Tabela 15 – Resultados obtidos com Florestas Aleatórias 73
Tabela 16 – Matriz de Confusão das Florestas Aleatórias 73
Tabela 17 – Hiperparâmetro da Máquina de Vetores de Suporte 74
Tabela 18 – Resultados obtidos com Máquina de Vetores de Suporte 74
Tabela 19 – Matriz de Confusão da Máquina de Vetores de Suporte 74
Tabela 20 – Hiperparâmetros do *AdaBoost* 75
Tabela 21 – Matriz de Confusão das *AdaBoost* 75
Tabela 22 – Resultados obtidos com *AdaBoost* 76
Tabela 23 – Comparativo de Todas as Técnicas 77
Tabela 24 – Aplicações, Serviços e Protocolos do Ambiente de Testes 81
Tabela 25 – Resultados da Detecção do DoS Slowloris 82
Tabela 26 – Resultados da Detecção do DoS SlowHTTPTest 83
Tabela 27 – Resultados da Detecção do DoS Hulk 84
Tabela 28 – Resultados da Detecção do Port Scan 85
Tabela 29 – Resultados da Detecção da Força Bruta SSH 86

Lista de quadros

Quadro 1 – Resumo dos Principais IDS de Código Aberto 26
Quadro 2 – Resumo dos Trabalhos Correlatos 54
Quadro 3 – Exemplos de *FlowIDs* 62

Lista de abreviaturas e siglas

CSA	*Clonal Selection Algorithm*
GA	*Genetic Algorithm*
GP	*Genetic Programming*
HIDS	*Host-based Intrusion Detection System*
ICMP	*Internet Control Message Protocol*
IDS	*Intrusion Detection System*
MIDS	*Mixed Intrusion Detection System*
ML	*Machine Learning*
MLP	*Multilayer Perceptron*
NBA	*Network Behaviour Analysis*
NIDS	*Network Intrusion Detection System*
NSA	*Negative Selection Algorithm*
OPF	*Optimum-Path Forest*
RNA	Redes Neurais Artificiais
SMOTE	*Synthetic Minority Oversampling TEchnique*
SVM	*Support Vector Machine*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*
WIDS	*Wireless-based Intrusion Detection System*

Lista de símbolos

α	Letra grega minúscula alfa
γ	Letra grega minúscula gamma
μ	Letra grega minúscula mi
σ	Letra grega minúscula sigma
θ	Letra grega minúscula teta
ξ	Letra grega Csi
Φ	Letra grega Fi

Sumário

1 Introdução 15
1.1 Motivação e Justificativa 16
1.2 Objetivos 17
1.2.1 Objetivo Geral 17
1.2.2 Objetivos Específicos 17
1.3 Estrutura da Dissertação 17

2 Sistemas de Detecção de Intrusão 19
2.1 Abordagens de Detecção em IDS 20
2.2 Técnicas de Detecção por Anomalia 21
2.3 Tipos de IDS 22
2.4 IDS de Código Aberto 23
2.4.1 Snort 23
2.4.2 Suricata 24
2.4.3 Bro IDS 25

3 Aprendizado de Máquina 27
3.1 Métodos de Aprendizado 27
3.1.1 Aprendizado Supervisionado 28
3.1.2 Aprendizado Não-Supervisionado 29
3.1.3 Aprendizado Semi-Supervisionado 29
3.1.4 Aprendizagem por Reforço 30
3.2 Algoritmos 30
3.2.1 Naive Bayes 30
3.2.2 Árvore de Decisão 32
3.2.3 Florestas Aleatórias 35
3.2.4 Máquina de Vetores de Suporte 37
3.2.5 AdaBoost 39
3.3 Métricas para Análise de Eficácia 40

4 Trabalhos Correlatos 43

5 Metodologia 55
5.1 Base de Dados CICIDS2017 56
5.2 Balanceamento da Base de Dados 58
5.3 Biblioteca *scikit-learn* 59

5.4 Plugin *ml_classifiers* . . . 60
5.4.1 Gerenciamento de Fluxos de Tráfego de Rede . . . 61
5.4.2 Classificação de Fluxos de Tráfego de Rede . . . 62
5.4.3 Configuração do Plugin . . . 64

6 Experimentos e Resultados . . . 68
6.1 Primeira Etapa dos Experimentos . . . 69
6.1.1 Naive Bayes . . . 69
6.1.2 Árvore de Decisão . . . 71
6.1.3 Florestas Aleatórias . . . 72
6.1.4 Máquina de Vetores de Suporte . . . 73
6.1.5 AdaBoost . . . 75
6.1.6 Comparação das Técnicas de Classificação . . . 76
6.2 Segunda Etapa dos Experimentos . . . 80
6.2.1 DoS Slowloris . . . 82
6.2.2 DoS SlowHTTPTest . . . 83
6.2.3 DoS Hulk . . . 83
6.2.4 Port Scan . . . 84
6.2.5 Força Bruta SSH . . . 85

7 Considerações Finais . . . 88
7.1 Trabalhos Futuros . . . 89

Referências . . . 90

1 Introdução

Com o advento da Internet, a humanidade foi rapidamente conquistada por inúmeras facilidades que intrinsecamente transformaram o cotidiano das sociedades contemporâneas. Dentre os principais atrativos dessa infraestrutura de escala mundial, destaca-se a tangibilidade das informações disponíveis no meio digital, independentemente da localização do usuário que deseja acessá-las.

Por mais que o alcance seja um dos principais atributos que proporcionaram o surgimento de aplicações fantásticas, ele é, também, um dos motivos preponderantes pelo qual a segurança de sistemas computacionais é, atualmente, imprescindível. A partir do momento em que uma aplicação faz uso da Internet para o envio e recebimento de dados, a mesma fica sujeita à intervenção de terceiros mal-intencionados que podem comprometer a integridade dessas informações. Sendo assim, a garantia da segurança em sistemas computacionais é, naturalmente, um processo contínuo em constante renovação.

Dentre as ferramentas utilizadas para a proteção de redes de computadores, vale mencionar os Sistemas de Detecção de Intrusão (*Intrusion Detection Systems* - IDS). Caracterizados pelas funções de monitoramento, alerta e prevenção de ataques e violações de políticas internas, a eficiência dos IDSs pode ser analisada a partir de três critérios (PORRAS; VALDES, 1998): *(i) acurácia*, *(ii) desempenho* e *(iii) completude*. Dependendo da metodologia de detecção utilizada pelo sistema, alguns critérios podem se sobressair em relação aos demais. Um IDS que realiza detecções por assinatura, isto é, que busca por padrões conhecidos – comumente chamados de *assinaturas* – nos dados analisados, possui uma boa taxa de acurácia por ser efetivo contra ataques conhecidos, todavia, sua completude é questionável uma vez que é preciso atualizar, constantemente, a base de conhecimento de ataques (GARCIA-TEODORO et al., 2009; ALJAWARNEH; ALDWAIRI; YASSEIN, 2018). Por outro lado, um IDS que realiza detecções por anomalia, ou seja, que estima o comportamento normal do sistema monitorado e busca por desvios que, potencialmente, representam anormalidades, possui dificuldades para alcançar altas taxas de acurácia, devido a maior susceptibilidade de gerar falso-positivos. Contudo, por não se limitar às assinaturas de ataques conhecidos, essa metodologia é capaz de identificar ameaças desconhecidas e, portanto, possui uma maior completude em relação à detecção por assinatura (GARCIA-TEODORO et al., 2009; ALJAWARNEH; ALDWAIRI; YASSEIN, 2018).

É possível classificar os IDSs levando em consideração o tipo de ambiente monitorado. Nos primeiros IDSs desenvolvidos, a detecção de intrusos era exclusivamente baseada em *host*, ou seja, a busca por eventos suspeitos ocorria localmente na máquina

em questão sendo monitorada pelo Sistema de Detecção de Intrusão Baseado em Host (*Host-based Intrusion Detection System* - HIDS). Porém, com a evolução da área e com a popularização da Internet, os IDSs focaram-se na detecção de ataques direcionados às redes de computadores, consolidando uma classe conhecida por Sistemas de Detecção de Intrusão em Rede (*Network Intrusion Detection Systems* - NIDS) (DEBAR; DACIER; WESPI, 1999).

1.1 Motivação e Justificativa

Embora no âmbito comercial seja predominante a utilização de NIDSs baseados em metodologias de detecção por assinatura, muitos estudos recentes da área acadêmica têm buscado a melhoria de metodologias de detecção por anomalia, através da aplicação de técnicas inteligentes de Aprendizado de Máquina. Nessas técnicas, por meio de uma etapa de treinamento, o IDS é capaz de aprender comportamentos normais e anômalos e, a partir deste conhecimento, realizar a predição/classificação de qualquer pacote/fluxo de tráfego de rede com base nas suas características. Em Shah e Issac (2018), um *plugin* adaptativo foi desenvolvido para melhorar a acurácia do Snort IDS (ROESCH et al., 1999). Por meio de uma abordagem híbrida, algoritmos de aprendizado de máquina foram utilizados para reduzir a taxa de falso-positivos provenientes de alarmes gerados pela solução base da ferramenta. O estudo mostrou-se eficaz ao mitigar de forma significativa a taxa de falso-positivos através de abordagens envolvendo Máquina de Vetores de Suporte (*Support Vector Machine* - SVM) com Lógica Fuzzy e SVM com Algoritmo do Vaga-lume (*Firefly Algorithm* - FA).

Em Saied, Overill e Radzik (2016), com um enfoque maior para a detecção de ataques de Negação de Serviço Distribuída (*Distributed Denial-of-Service* - DDoS) conhecidos e desconhecidos, foi desenvolvido um *plugin* portador de uma rede neural artificial para o Snort-AI (BEDÓN; SAIED, 2009). Em um ambiente controlado, o projeto proposto obteve uma acurácia de até 98%, superando a solução base do próprio Snort, isto é, sem modificações, e de outros trabalhos correlatos.

Apesar desses serem apenas alguns dos trabalhos desenvolvidos na área nos últimos anos, eles mostram que a detecção de intrusão baseada em anomalia ainda é uma das maiores tendências no que diz respeito ao estudo de IDSs. Sendo assim, neste trabalho, a proposta inicial é justamente realizar o levantamento bibliográfico de outros estudos dessa mesma natureza além, é claro, de contextualizar os principais conceitos ligados não só aos IDSs mas também às técnicas inteligentes empregadas nessas ferramentas. Na sequência, identificados os possíveis pontos de contribuição para a comunidade científica, propõe-se a capacitação, em termos de detecção de intrusão baseada em anomalia, do IDS de código aberto Snort.

1.2 Objetivos

1.2.1 Objetivo Geral

Substituir o esquema de detecção baseado em assinatura da última versão do Snort (Snort 3) por um esquema de detecção baseado em anomalia para classificar, em tempo real, fluxos de tráfego de rede mediante a utilização de técnicas de aprendizado de máquina.

1.2.2 Objetivos Específicos

Para facilitar a análise do desenvolvimento do trabalho e, consequentemente, a validação da metodologia proposta, foram definidos objetivos específicos, descritos a seguir:

a) Estudar a arquitetura do Snort 3, visando a compreensão dos seus principais meios de extensão;

b) Estudar as técnicas de aprendizado de máquina a serem utilizadas neste trabalho: Árvore de Decisão, Florestas Aleatórias, *AdaBoost*, *Naive Bayes* e Máquina de Vetores de Suporte;

c) Estudar sobre as particularidades da base de dados escolhida, CICIDS2017, principalmente no que diz respeito à forma como suas características foram obtidas (através da ferramenta *CICFlowMeter*);

d) Treinar as técnicas de aprendizado de máquina com a base de dados CICIDS2017 e avaliar seus respectivos resultados;

e) Desenvolver um *plugin* que habilite no Snort 3 o gerenciamento e a classificação em tempo real de fluxos de tráfego de rede usando as técnicas de aprendizado de máquina treinadas previamente;

f) Testar e validar a metodologia proposta através de uma bateria de experimentos com o Snort 3 modificado (com o *plugin*) e o Snort 3 não modificado (sem o *plugin*);

g) Verificar e comparar os resultados obtidos.

1.3 Estrutura da Dissertação

Esta dissertação de mestrado está estruturada da seguinte forma: o Capítulo 2 apresenta os principais conceitos e definições sobre os IDSs e algumas das soluções de código aberto mais populares disponíveis atualmente. O Capítulo 3 apresenta os principais conceitos e definições sobre as técnicas da área de Aprendizado de Máquina. O Capítulo 4

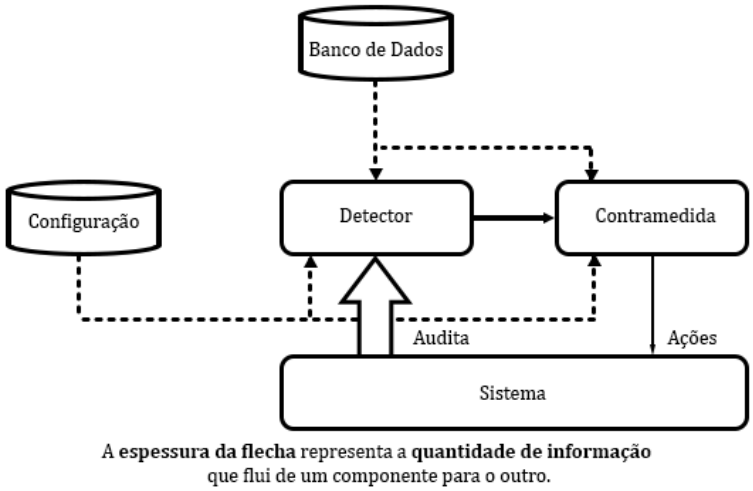
apresenta um levantamento bibliográfico e cronológico sobre os principais trabalhos desenvolvidos que uniram o Snort e a área de Aprendizado de Máquina. O Capítulo 5 apresenta a metodologia proposta para o desenvolvimento de um *plugin* portador de técnicas de aprendizado de máquina, capaz de classificar fluxos de tráfego de rede em tempo real. O Capítulo 6 apresenta os experimentos e os respectivos resultados obtidos nesta dissertação. Finalmente, o Capítulo 7 apresenta as considerações finais desta dissertação e, também, os próximos passos a serem tomados.

2 Sistemas de Detecção de Intrusão

Um IDS é responsável por monitorar dinamicamente as ações tomadas em um dado ambiente e, também, decidir se estas ações apresentam sintomas de um possível ataque ou se constituem um uso legítimo do ambiente (DEBAR; DACIER; WESPI, 1999). Eventualmente, caso sejam detectadas ações maliciosas no sistema, o objetivo é notificar o administrador da rede por meio de alertas (mensagens no terminal, *e-mail*, etc).

O IDS, de modo generalizado, também pode ser visto como um detector que utiliza três tipos de informação em seu processamento: 1) informação de longo-termo relacionada à técnica utilizada para detectar intrusões, 2) informação de configuração do estado atual do sistema e, finalmente, 3) informação de auditoria descrevendo os eventos que ocorreram no sistema (Figura 1). A partir desses três elementos, o IDS é capaz de filtrar informações desnecessárias e produzir uma visão sintética contendo somente as ações relacionadas à segurança tomadas pelos usuários. Desta forma, uma decisão final pode ser tomada para avaliar as chances destas ações constituírem (ou não) um sintoma de intrusão no sistema (DEBAR; DACIER; WESPI, 1999).

Figura 1: Um Sistema de Detecção de Intrusão Simples

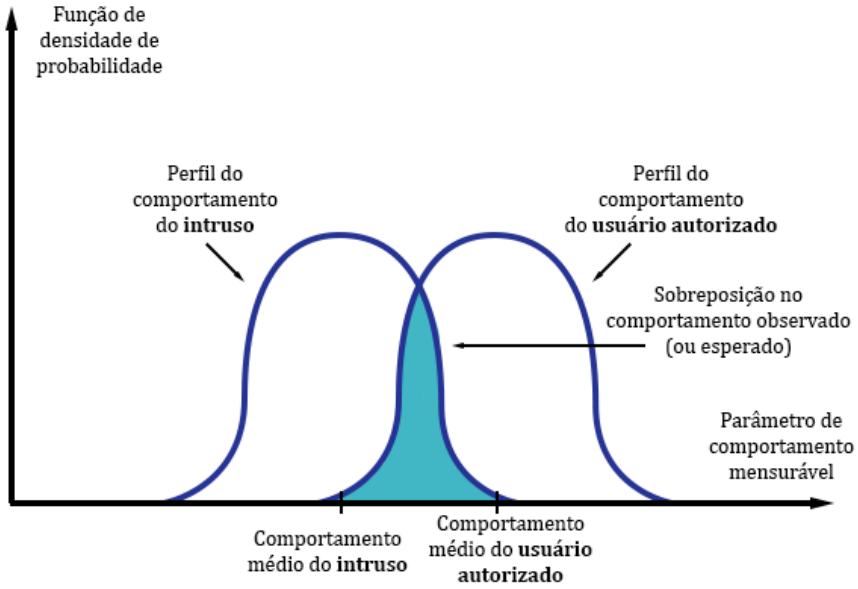


Fonte: Adaptada de Debar, Dacier e Wespi (1999).

A detecção de intrusão parte do pressuposto de que é possível diferenciar o comportamento de um invasor do comportamento de um usuário legítimo, de maneiras quantificáveis (STALLINGS, 2000). Entretanto, não existe uma delimitação exata que indica o término de um comportamento e o início de outro. Por outro lado, observa-se a existência de uma região comportamental comum entre os perfis de usuários (Figura 2), podendo dificultar a interpretação por parte dos IDS, que dependendo da abordagem utilizada, pode

aumentar a quantidade de falso-positivos (usuários normais classificados como invasores) ou falso-negativos (invasores classificados como usuários normais).

Figura 2: Perfis dos Comportamentos de Invasores e Usuários Autorizados



Fonte: Adaptada de Stallings (2000).

2.1 Abordagens de Detecção em IDS

Atualmente, a abordagem de detecção mais utilizada em IDS é a baseada em assinaturas. Os IDSs dessa natureza mantêm bases de dados extensas com as assinaturas dos ataques. Cada assinatura, na prática, é representada por um conjunto de regras que descreve tanto pacotes individuais como grupos de pacotes utilizados em ataques conhecidos. Desta forma, à medida que o IDS analisa os pacotes que passam por ele, se forem encontradas equivalências entre esses pacotes e as assinaturas da base de dados, são gerados alertas para notificar o administrador da rede. Além da incapacidade de detectar ataques desconhecidos, um IDS baseado em assinaturas pode enfrentar dificuldades relacionadas ao desempenho da aplicação, dependendo do número de comparações de assinaturas realizadas por cada pacote processado (KUROSE; ROSS, 2009).

Apesar de ser menos utilizada, a abordagem de detecção por anomalia tem chamado atenção nos últimos anos no campo científico. Partindo da criação de um perfil comportamental considerado "normal" do sistema computacional, a metodologia concentra-se em identificar variações que ultrapassam certos limites definidos no IDS, que potencialmente indicariam a ocorrência de uma anormalidade (GARCIA-TEODORO et al., 2009). Por mais que essa alternativa descarte a necessidade de manter uma base de dados de ataques conhecidos, a dificuldade dessa abordagem reside na busca por um modelo bem

estruturado que represente, da melhor forma possível, as particularidades do sistema monitorado, conciliando alta taxa de acurácia e baixa taxa de falso-positivos.

2.2 Técnicas de Detecção por Anomalia

Garcia-Teodoro et al. (2009) e Hodo et al. (2017) definem, ainda, três principais categorias de técnicas de detecção por anomalia: baseada em estatística, baseada em conhecimento e baseada em aprendizado de máquina (*machine learning*). Em técnicas baseadas em estatística, a atividade do tráfego de rede é capturado e um perfil representando o comportamento estocástico é criado. Tal perfil baseia-se em métricas como a taxa de tráfego, taxa de conexões, número de pacotes para cada protocolo, número de diferentes endereços IP, dentre outros. O processo de detecção por anomalia conta com dois perfis: um correspondendo ao perfil atual, monitorado ao longo do tempo, e outro correspondendo ao perfil estatístico treinado previamente. Na medida em que ocorrem eventos na rede, o perfil atual é atualizado e um escore de anomalia é estimado por meio da comparação entre os dois comportamentos. Representando o grau de irregularidade de determinado evento, caso o escore de anomalia ultrapasse um limite pré-estabelecido, é registrada a ocorrência de uma anormalidade.

Em técnicas baseadas em conhecimento, a abordagem mais utilizada é a de sistemas especialistas (*expert systems*). Com o intuito de classificar dados de acordo com um conjunto de regras, os sistemas especialistas realizam três passos. Primeiro, diferentes atributos e classes são identificados a partir dos dados de treinamento. Em seguida, um conjunto de regras de classificação, parâmetros ou procedimentos são deduzidos. Por fim, os dados são classificados de acordo com as regras utilizadas.

Existem, também, os métodos de anomalia baseados em especificações, onde o modelo desejado é construído manualmente a partir de um especialista humano, em termos de um conjunto de regras (especificações) que vislumbram determinar o comportamento legítimo do sistema. Em alternativa ao especialista humano, as especificações podem ser desenvolvidas por meio de ferramentas formais, como a metodologia da máquina de estados finitos (*Finite State Machine* - FSM), que por lidar com sequências de estados e as transições entre eles, torna-se adequada para modelar protocolos de rede (ESTEVEZ-TAPIADOR; GARCIA-TEODORO; DIAZ-VERDEJO, 2003).

Finalmente, em técnicas baseadas em aprendizado de máquina, o estabelecimento de modelos explícitos ou implícitos permitem que padrões aprendidos sejam categorizados. Uma característica singular dessas técnicas é a necessidade de dados rotulados para treinar tais modelos comportamentais. Dentre os principais esquemas baseados em aprendizado de máquina aplicados aos IDSs, podemos citar a utilização de Redes Neurais Artificiais (RNAs). Por se tratar de uma abordagem resiliente à mudanças no ambiente, RNAs já

foram aplicadas para a criação de perfis de usuários (FOX, 1990), predição de comandos com base em sequências de comandos anteriores (DEBAR; DACIER; LAMPART, 1998), identificação de comportamento intrusivo de padrões de tráfego (CANSIAN et al., 1997), dentre outras.

Com a utilização de Algoritmos Genéticos (*Genetic Algorithms* - GA), Li (2004) propôs a implementação de um esquema que considera informações temporais e espaciais de conexões de rede para codificá-las em termos de regras de um IDS. Desta forma, entende-se que o objetivo final da aplicação do GA é gerar regras que, de certo modo, sejam aplicáveis somente às conexões anômalas. Assim, essas regras são testadas em conexões históricas e utilizadas para filtrar novas conexões em busca de tráfego de rede suspeito.

2.3 Tipos de IDS

Levando em consideração o tipo do sistema monitorado pelo IDS, é possível classificá-lo como (ALHEETI, 2011; SONG et al., 2004; SUNDARAM, 1996; CROTHERS, 2003; KAZIENKO; DOROSZ, 2004):

- **NIDS**: é uma plataforma independente que analisa, examina e monitora o *backbone*[1] de uma rede, em busca de ataques. O posicionamento de um NIDS é estratégico, estando normalmente conectado a *switches*[2] configurados com espelhamento de portas[3] (*Port Mirroring*). O NIDS protege, portanto, um segmento inteiro de rede.
- **HIDS**: restringe-se a um computador em particular e oferece proteção por meio do monitoramento do sistema operacional e do sistema de arquivos, em busca de sinais de intrusão. Sua análise abrange chamadas ao sistema, *logs* de aplicações, modificações no sistema de arquivos, dentre outras atividades.
- **Híbrido de NIDS e HIDS**: foca-se na análise conjunta de dados provenientes dos *hosts* e da própria rede, de modo a obter uma visão compreensiva do sistema. A necessidade de utilização de sistemas híbridos torna-se evidente em redes criptografadas, onde é preciso identificar correlações entre os dados que trafegam e que são descriptografados nos *hosts*.

Em Liao et al. (2013) pontuam-se também os Sistemas de Detecção de Intrusão baseados em Wireless (*Wireless-based Intrusion Detection Systems* - WIDS), Análise de Comportamento de Rede (*Network Behaviour Analysis* - NBA) e Sistemas de Detecção

[1] *Backbone* pode ser entendido como a estrutura responsável por interligar diversos elementos de rede, permitindo o tráfego de dados de uma rede para outra.
[2] *Switch* é um dispositivo, comutador, que realiza a redistribuição de pacotes dentro de uma rede.
[3] *Port Mirroring* é um método de monitoramento do tráfego de rede onde cópias de todos os pacotes enviados são redirecionadas para uma porta específica.

de Intrusão Mistos (*Mixed Intrusion Detection System* - MIDS). Os WIDSs, assim como os NIDSs, capturam o tráfego de rede mas restringem-se à análise do ambiente sem-fio. O NBA, diferentemente do NIDS ou WIDS, realiza uma inspeção no tráfego de rede com o intuito de detectar ameaças com base *exclusiva* no comportamento inesperado deste evento. Por fim, a adoção de múltiplas tecnologias resulta em um MIDS, que pode contribuir com detecções mais completas e acuradas.

2.4 IDS de Código Aberto

Tendo em vista que a maioria das soluções de IDS adotadas em ambientes corporativos e domésticos utilizam, primariamente, abordagens de detecção por assinatura (DEBAR; DACIER; WESPI, 1999), é imprescindível a manutenção de bases de dados de ataques com o intuito de mantê-las sempre atualizadas. Por conta da constante renovação de ataques existentes e, também, do surgimento de novos ataques, tal processo de manutenção é desafiador ao ter de se adaptar, em tempo hábil, à tais mudanças no cenário de detecção de intrusão.

Em IDSs de código aberto (*open-source*) essa dificuldade é contornada de modo eficiente pela própria comunidade da ferramenta que, coletivamente, ajuda na construção e atualização de bases de dados diversificadas e completas. Nesta seção, são apresentadas algumas das principais ferramentas de código aberto disponíveis no mercado: Snort, Suricata e Bro IDS.

2.4.1 Snort

Criado em 1998 por Martin Roesch, Snort é o Sistema de Detecção e Prevenção de Intrusão (*Intrusion Detection and Prevention System* - IDPS) mais popular da atualidade. Sua abordagem é baseada em assinaturas, utilizando regras e pré-processadores para analisar o tráfego de rede. As regras oferecem um mecanismo simples e flexível para a criação de assinaturas e examinação de pacotes. Por outro lado, pré-processadores permitem extensas manipulações de dados que não podem ser feitas por meio de regras. Pré-processadores podem realizar tarefas como: a desfragmentação de IP, detecção de *portscan*, normalização de tráfego web, dentre outras. (NORTHCUTT; NOVAK, 2002).

As principais qualidades da ferramenta são (TACTICALFLEX, 2019):

- *Escalabilidade*: Passível de instalação em qualquer ambiente de rede.
- *Flexibilidade e Usabilidade*: Multi-plataforma (Linux, Windows, Mac OS X).
- *Tempo Real*: Capaz de entregar informações sobre eventos de tráfego de rede em tempo real.

- *Flexibilidade na Implantação*: Pode ser customizado para operar com diversas bases de dados, sistemas de *log* e ferramentas de terceiros.
- *Velocidade na Detecção e Responsividade para Ameaças*: Junto à firewall e outras camadas de infraestrutura de segurança, o Snort apresenta alta eficácia na detecção e prevenção de invasores, *worms*, vulnerabilidades de rede, etc.
- *Mecanismo de Detecção Modular*: Os sensores do Snort são modulares e podem monitorar diversas máquinas a partir de uma localização física e lógica. O Snort pode ser configurado antes, depois ou próximo de um firewall, atuando em diferentes segmentos da rede, de acordo com a necessidade da organização.

2.4.2 Suricata

Desenvolvido em 2009 pela *Open Information Security Foundation* (OISF), o Suricata é uma das soluções mais semelhantes ao Snort, por também utilizar regras e pré-processadores na detecção por assinatura, com o diferencial de ter uma arquitetura *multi-thread*. Tal característica resulta em maior velocidade e eficiência na análise do tráfego de rede, além de contribuir para a distribuição da carga de trabalho conforme as necessidades de processamento da ferramenta.

Em um estudo comparativo de Albin e Rowe (2012), três experimentos foram realizados para comparar o desempenho do Suricata e do Snort. No primeiro experimento, a capacidade *multi-threading* do Suricata foi responsável pela baixa taxa de pacotes descartados pela ferramenta (7%) em relação ao Snort (53%). No segundo experimento, com ambas ferramentas instaladas em um computador de alto-desempenho, melhorias significativas foram observadas no Suricata. Por fim, no terceiro e último experimento, altas taxas de detecção foram observadas em ambas as ferramentas, com um leve destaque para o Snort, por ter cometido menos erros de classificação.

Assim, pode-se destacar como sendo as principais qualidades da ferramenta (TACTICALFLEX, 2019):

- *Código Aberto*: Assim como no Snort, a comunidade da ferramenta desempenha um importante papel na documentação de novos ataques descobertos, com uma efetividade muito maior do que uma única organização seria capaz de fazer.
- *Multi-thread*: Uma arquitetura *multi-thread* permite que a ferramenta tire vantagem dos múltiplos núcleos e multiprocessadores dos sistemas da atualidade.
- *Suporte à Reputação de IP*: Incorporando reputação e assinaturas em seu mecanismo, o Suricata é capaz de marcar o tráfego proveniente de fontes ruins conhecidas.

- *Detecção de Protocolo Automatizada*: Pré-processadores identificam automaticamente o protocolo utilizado em um fluxo de tráfego de rede e aplicam as regras apropriadas para cada um deles, independente da porta numérica. O processo automatizado também previne erros e enganos comuns de usuários.

2.4.3 Bro IDS

Originalmente escrito por Vern Paxson em 1995, Bro IDS é a solução que mais difere do Snort e Suricata. A ferramenta é composta por dois principais componentes. O primeiro é o mecanismo de eventos (*event engine*), responsável por reduzir o fluxo de pacotes a uma série de eventos de alto nível. Esses eventos refletem a atividade de rede em termos neutros de política, ou seja, eles descrevem "o que" foi visto, mas não o "porquê" ou se é significativo (BRO, 2018a).

A análise semântica desses eventos é realizada pelo segundo componente da ferramenta, o interpretador de scripts (*script interpreter*), que utiliza uma linguagem própria. Os scripts são responsáveis por determinar ações de acordo com o tipo de atividade observada pelo IDS. A partir do tráfego de entrada, qualquer propriedade ou estatística pode ser obtida por meio do *scripting* (BRO, 2018a).

As principais qualidades da ferramenta são (BRO, 2018b):

- *Adaptável*: A linguagem de *scripting* própria do Bro IDS permite políticas de monitoramento de locais específicos.
- *Eficiência*: O Bro IDS foca-se em redes de alto desempenho e é utilizado operacionalmente em uma variedade de sites grandes.
- *Flexível*: O Bro IDS não se restringe a qualquer abordagem de detecção em particular e não depende de assinaturas tradicionais, tais como o Snort e Suricata utilizam.
- *Interfaces Abertas*: A ferramenta realiza interfaces com outras aplicações para troca de informações em tempo real.
- *Análise Aprofundada*: O Bro IDS possui analisadores para diversos protocolos, permitindo análises semânticas de alto nível na camada de aplicação.

O Quadro 1 mostra um resumo dos IDSs de código aberto apresentados anteriormente (Seções 2.4.1, 2.4.2 e 2.4.3). Devido à familiaridade dos autores com o Snort 3 (UTIMURA; COSTA, 2018) e ao fato dele ser o NIDS de código aberto mais empregado na atualidade, optou-se por sua utilização neste trabalho.

Quadro 1: Resumo dos Principais IDS de Código Aberto

Parâmetros	**Snort**	**Suricata**	**BroIDS**
Data de Criação	1998	2009	1995
Plataforma	Linux, Windows, Mac OS X	Linux, Windows, Mac OS X	Sistemas compatíveis com Unix
Abordagem de Detecção	Assinatura (Regras)	Assinatura (Regras)	Assinatura e Anomalia (Eventos)
Suporte a Redes de Alta Velocidade	Médio	Alto	Alto
Multi-thread	Snort 2 (Não) Snort 3 (Sim)	Sim	Não

Fonte: Elaborado pelo autor.

3 Aprendizado de Máquina

O Aprendizado de Máquina, do inglês *Machine Learning* (ML), pode ser definido em sua essência como a extração de conhecimento a partir de bases de dados. O ML é um tópico composto pela intersecção de diversas outras áreas, como estatística, inteligência artificial e ciência da computação, abrangendo campos como a *análise preditiva* e o *aprendizado estatístico* (BAKSHI; BAKSHI, 2018). Nos últimos anos, o ML tem possibilitado o desenvolvimento de aplicações como: carros autônomos, processamento de linguagem natural, mecanismos eficientes de busca, sistemas inteligentes de recomendação, dentre outros.

A aplicação do ML tem por objetivo construir novos algoritmos e aprimorar os existentes de forma que esses sejam capazes de aprender a partir de bases de dados, construindo modelos generalizáveis capazes de realizar predições precisas e/ou encontrar padrões, a partir de dados desconhecidos (BAKSHI; BAKSHI, 2018).

O processo de aprendizado consiste em melhorar um sistema de conhecimento através da ampliação ou rearranjo da base de conhecimento/motor de inferência (RUSSELL; NORVIG, 2016). O aprendizado de máquina compreende métodos computacionais para adquirir novos conhecimentos, novas habilidades e novas maneiras de organizar o conhecimento existente (TYUGU, 2011).

Os problemas de aprendizado podem variar bastante em termos de complexidade, desde a aprendizagem paramétrica (que visa o aprendizado de valores para determinados parâmetros) até formas complicadas de aprendizado simbólico (que visa o aprendizado de conceitos, gramáticas, funções e até mesmo comportamentos) (TYUGU, 2011).

3.1 Métodos de Aprendizado

Independentemente da técnica de ML escolhida para resolver um determinado problema, o aprendizado dela está intrinsecamente ligado a sua capacidade em reconhecer e classificar padrões (*pattern recognition* e *pattern classification* respectivamente). A criação de classificadores envolve, então, postular alguma forma geral de modelo, ou forma do classificador, e usar padrões de treinamento para aprender ou estimar os parâmetros desconhecidos do modelo (DUDA; HART; STORK, 2012). Na prática, o aprendizado pode ocorrer de quatro maneiras distintas, detalhadas a seguir.

3.1.1 Aprendizado Supervisionado

No aprendizado supervisionado (*supervised learning*), um *supervisor* fornece um rótulo categórico (ou custo) para cada padrão do conjunto de treinamento, de modo que o objetivo do problema passa a ser a redução do somatório de custos para estes padrões (DUDA; HART; STORK, 2012).

Na prática, os algoritmos de aprendizado supervisionado já são bem compreendidos na área e fáceis de avaliar o desempenho. De modo geral, se uma aplicação é modelada como um problema de aprendizado supervisionado e são fornecidas as saídas necessárias no conjunto de treinamento, é provável que o ML supervisionado resolva o problema (BAKSHI; BAKSHI, 2018).

Os algoritmos de aprendizado supervisionado podem ser interpretados como *classificadores* ou *regressores*, dependendo da natureza do problema (BISHOP, 2006; BAKSHI; BAKSHI, 2018). Na classificação, o objetivo é prever um rótulo de classe, que é uma escolha de uma lista predefinida de possibilidades. A classificação pode, ainda, ser binária ou multiclasse, isto é, lidar com apenas duas classes pré-definidas ou até mais de duas classes. Um exemplo de problema de classificação binária seria a classificação de conexões em uma rede, como *normais* ou *ataques*. Já um exemplo de problema de classificação multiclasse seria a classificação de frutas como *maçãs*, *pêras* ou *laranjas*. Note que em ambos os casos a classificação é exclusiva, ou seja, uma amostra não pode pertencer a duas classes ao mesmo tempo.

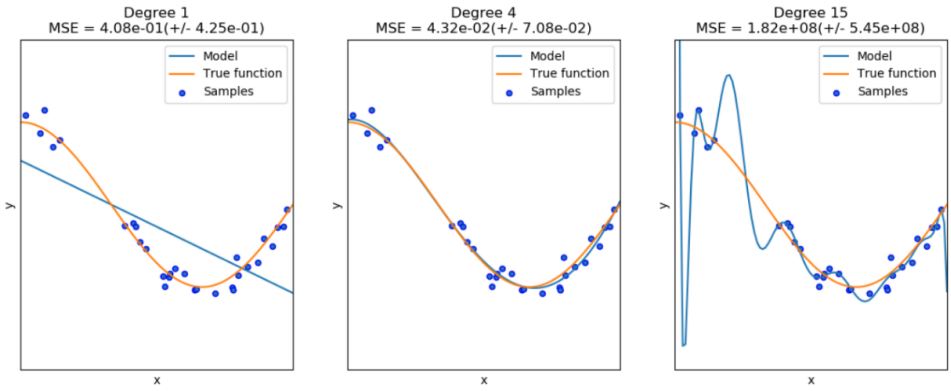
Na regressão, o objetivo é prever uma ou mais variáveis contínuas. Um exemplo de problema de regressão seria a predição do salário anual de um indivíduo a partir do seu nível de educação, idade e local de residência. Note que o salário, neste caso, pode ser qualquer valor dentro de um determinado intervalo e, por existir esta continuidade, trata-se de um problema de regressão e não de classificação.

Na etapa de aprendizado, entende-se por ideal a criação do modelo mais simples possível capaz de realizar predições com uma satisfatória taxa de acurácia. Quando são criados modelos muito complexos para uma determinada base de dados, diz-se que houve um super-treinamento (*overfitting*). O super-treinamento ocorre quando os modelos são criados demasiadamente próximos da natureza da própria base de dados de treinamento. Neste cenário, apesar do modelo funcionar bem para os dados do conjunto de treinamento, ele é incapaz de estrapolar boas predições para dados desconhecidos (BAKSHI; BAKSHI, 2018).

Por outro lado, quando são criados modelos muito simples, dificilmente eles serão capazes de capturar todos os aspectos relevantes das respectivas bases de dados e, neste caso, diz-se que houve um sub-treinamento (*underfitting*) (BAKSHI; BAKSHI, 2018). A visualização de um *underfitting*, um bom treinamento e um *overfitting* é demonstrada na

Figura 3.

Figura 3: Exemplo de *underfitting*, um bom treinamento e *overfitting*



Fonte: Scikit-learn (2014).

3.1.2 Aprendizado Não-Supervisionado

No aprendizado não-supervisionado (*unsupervised learning*), como o próprio nome já diz, não existe um *supervisor* para instruir o algoritmo de aprendizado (BAKSHI; BAKSHI, 2018). Isso significa que, na prática, em problemas de aprendizado não-supervisionado o objetivo pode ser a descoberta de exemplos similares nos dados (*clustering*), determinar a distribuição dos dados dentro do espaço de entrada (*density estimation*) ou projetar os dados de um espaço de n-dimensões para outro espaço de duas ou três dimensões, permitindo a *visualização* do mesmo (BISHOP, 2006).

Dentre todas estas aplicações do aprendizado não-supervisionado, a visualização é, potencialmente, a mais utilizada em problemas de natureza não-supervisionada. Destacam-se as técnicas: *Principal Component Analysis* (PCA), *Non-negative Matrix Factorization* (NMF), *Singular Value Decomposition* (SVD), *Linear Discriminant Analysis* (LDA) e t-SNE (BAKSHI; BAKSHI, 2018).

3.1.3 Aprendizado Semi-Supervisionado

Apesar dos classificadores tradicionais utilizarem dados rotulados nos seus respectivos processos de aprendizagem, a aquisição destes dados nem sempre é viável, podendo ser difícil, custosa e/ou, ainda, demandar um bom tempo e esforço de profissionais especializados na área do problema em questão. Por outro lado, dados não rotulados são relativamente fáceis de se obter, porém não possuem as mesmas possibilidades de uso como os dados rotulados.

Desta forma, conciliando os pontos positivos de ambos os tipos de dados, o aprendizado semi-supervisionado (*semi-supervised learning*) explora a utilização de grandes quantidades de dados não rotulados com dados rotulados, com o intuito de construir

melhores classificadores. Na prática, os dados rotulados são utilizados para adquirir informações do problema e guiar o aprendizado a partir de dados não rotulados (BRUCE, 2001).

Destacam-se as técnicas: Algoritmo de Maximização de Expectativas (*Expectation-Maximization* – EM) com Modelos de Misturas Generativas, *Self-training*, *Co-training*, *Transductive Support Vector Machines* (TSVM) e métodos baseados em grafos, como o *Particle Competition and Cooperation* (PCC) (ZHU, 2006; BREVE et al., 2011).

3.1.4 Aprendizagem por Reforço

Na aprendizagem por reforço (*reinforcement learning*), existe uma interação semelhante ao do aprendizado supervisionado no que diz respeito ao *modo* como um *agente* influencia no aprendizado do *ambiente*. Todavia, nesta abordagem, o único *feedback* que é dado em relação à predição do classificador é se a tentativa foi *correta* ou *errada*, não revelando, portanto, a verdadeira categoria do dado analisado (DUDA; HART; STORK, 2012). Deste modo, diferentemente do aprendizado supervisionado, as saídas ótimas do problema são descobertas pelo classificador através de um processo de tentativa e erro (BISHOP, 2006).

Existem duas principais estratégias para resolver problemas de aprendizagem por reforço. A primeira envolve a realização de buscas no espaço de comportamentos, a fim de encontrar um que desempenhe bem no ambiente. Normalmente, esta estratégia é implementada através de algoritmos genéticos e programação genética. Na segunda, técnicas estatísticas e métodos de programação dinâmica são utilizados para estimar a utilidade de tomar determinadas ações em estados do mundo. Neste contexto, Processos de Decisão de Markov (*Markov Decision Process* – MDP) são comumente utilizados para formular os ambientes do problema (KAELBLING; LITTMAN; MOORE, 1996).

3.2 Algoritmos

Nesta seção, são apresentados os algoritmos de aprendizado de máquina, supervisionados, utilizados no desenvolvimento deste trabalho.

3.2.1 Naive Bayes

Os classificadores *Naive Bayes* são um conjunto de algoritmos de aprendizados supervisionados baseados no Teorema de Bayes. Esses algoritmos partem do pressuposto de que existe uma independência condicional entre todos os atributos (ou características) dos exemplos, dado o contexto da classe. Por esse motivo, eles são considerados os mais simples dos modelos probabilísticos bayesianos (MCCALLUM; NIGAM et al., 1998). Ainda

sim, *Naive Bayes* se destacam como uns dos algoritmos de aprendizado indutivo mais eficientes e efetivos na área de aprendizado de máquina e mineração de dados (ZHANG, 2004).

Seguindo a notação de Zhang (2004), um exemplo E pode ser representado por uma tupla de valores de atributos $(x_1, x_2, \ldots, x_n)$, onde x_i é o valor do atributo X_i. Seja C a variável de classificação e c o valor de C. Supondo que existam apenas duas classes, uma positiva $(+)$ e outra negativa $(-)$, a probabilidade de um exemplo $E = (x_1, x_2, \ldots, x_n)$ pertencer à classe c é:

$$p(c \mid E) = \frac{p(E \mid c)p(c)}{p(E)}. \tag{3.1}$$

E pode ser classificado como $C = +$, isto é, pertencente à classe positiva, se, e somente se

$$f_b(E) = \frac{p(C = + \mid E)}{p(C = - \mid E)} \geq 1, \tag{3.2}$$

onde f_b é o classificador Bayesiano. Visto que todos os atributos são assumidos independentes entre si dado o valor da variável de classe, ou seja,

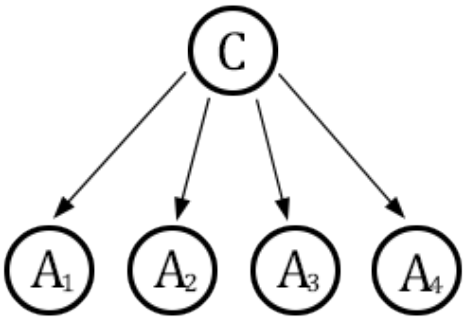
$$p(E \mid c) = p(x_1, x_2, \ldots, x_n \mid c) = \prod_{i=1}^{n} p(x_i \mid c), \tag{3.3}$$

o classificador resultante é dado por

$$f_{nb}(E) = \frac{p(C = +)}{p(C = -)} \prod_{i=1}^{n} \frac{p(x_i \mid C = +)}{p(x_i \mid C = -)}. \tag{3.4}$$

Na literatura, a função f_{nb} é comumente chamada de classificador *Naive Bayes* ou simplesmente *Naive Bayes*. Na Figura 4, é mostrado um exemplo de *Naive Bayes*. Note que, devido à suposição mencionada anteriormente sobre a independência condicional dos atributos, um *nó atributo* não possui qualquer outro *pai* exceto o *nó classe*.

Figura 4: Exemplo de *Naive Bayes*



Fonte: Elaborada pelo autor.

Os classificadores *Naive Bayes* podem diferir uns dos outros quanto às suposições feitas a respeito da distribuição de $p(x_i \mid c)$. Em um classificador *Naive Bayes Gaussiano*,

tem-se que

$$p(x_i \mid c) = \frac{1}{\sqrt{2\pi\sigma_c^2}} \exp\left(-\frac{(x_i - \mu_c)^2}{2\sigma_c^2}\right), \tag{3.5}$$

ou seja, que $p(x_i \mid c)$ é dado pela distribuição gaussiana. Além do *Naive Bayes Gaussiano*, existem também os classificadores *Naive Bayes Multinomial* e *Naive Bayes Bernoulli* (MCCALLUM; NIGAM et al., 1998).

No classificador *Naive Bayes Multinomial*, o modelo multinomial busca capturar a frequência com que determinados elementos de interesse aparecem nos exemplos *e.g.*, a frequência de palavras em documentos. Seguindo a notação de Scikit-learn (2011b), neste modelo, a distribuição é parametrizada por vetores $\theta_c = (\theta_{c1}, \ldots, \theta_{cn})$ para cada classe c, onde n é o número de atributos e θ_{ci} é a probabilidade $p(x_i \mid c)$ do atributo i aparecer no exemplo pertencente à classe c.

Os parâmetros θ_c são estimados por uma versão suavizada da máxima verossimilhança (*maximum likelihood*):

$$\hat{\theta_{ci}} = \frac{N_{ci} + \alpha}{N_c + \alpha n} \tag{3.6}$$

onde $N_{ci} = \sum_{x \in T} x_i$ é o número de vezes que o atributo i aparece em um exemplo da classe c no conjunto de treinamento T e $N_c = \sum_{i=1}^{n} N_{ci}$ a contagem total de todos os atributos para a classe c. A suavização prioriza os cálculos onde α[1] ≥ 0 para os atributos ausentes nos exemplos de aprendizado e evita probabilidades nulas em futuros cálculos (SCIKIT-LEARN, 2011b).

No classificador *Naive Bayes Bernoulli*, cada atributo é assumido como sendo uma variável de valor binário. A regra de decisão utilizada neste modelo é dada por:

$$p(x_i \mid c) = p(i \mid c)x_i + (1 - p(i \mid c))(1 - x_i) \tag{3.7}$$

Note que, diferentemente do *Naive Bayes Multinomial*, a regra de decisão do *Naive Bayes Bernoulli* penaliza explicitamente a não ocorrência de um atributo i que é um indicador para a classe c, ao invés de simplesmente ignorá-la (SCIKIT-LEARN, 2011b).

3.2.2 Árvore de Decisão

O classificador Árvore de Decisão é uma das possíveis abordagens para a tomada de decisão multiestágio. A ideia básica por trás de qualquer abordagem multiestágio é quebrar uma decisão complexa em uma série de decisões mais simples esperando que a solução final, obtida por meio dessa estratégia, se assemelhe com a solução esperada para o problema em questão (SAFAVIAN; LANDGREBE, 1991).

[1] Parâmetro de suavização aditivo. Quando configurado $\alpha = 1$, chama-se suavização de Laplace. Se $\alpha < 1$, chama-se suavização de Lidstone.

Na literatura, tal estratégia é chamada de *dividir e conquistar*. Esse processo de divisão de decisões, mencionado anteriormente, repete-se recursivamente inclusive nas decisões mais simples recém-criadas, até que não seja mais possível fazê-lo. Uma vez que o processo atinge esse limite, as soluções dos subproblemas de decisão passam a ser combinadas na forma de uma árvore (GAMA, 2004).

Os principais componentes de um modelo de árvore de decisão são os nós e as ramificações (ou galhos). Já para a construção do próprio modelo em si, as principais etapas são *splitting* (divisão), *stopping* (parada) e *prunning* (poda) (SONG; YING, 2015).

Em uma árvore de decisão, pode-se destacar três tipos de nós:

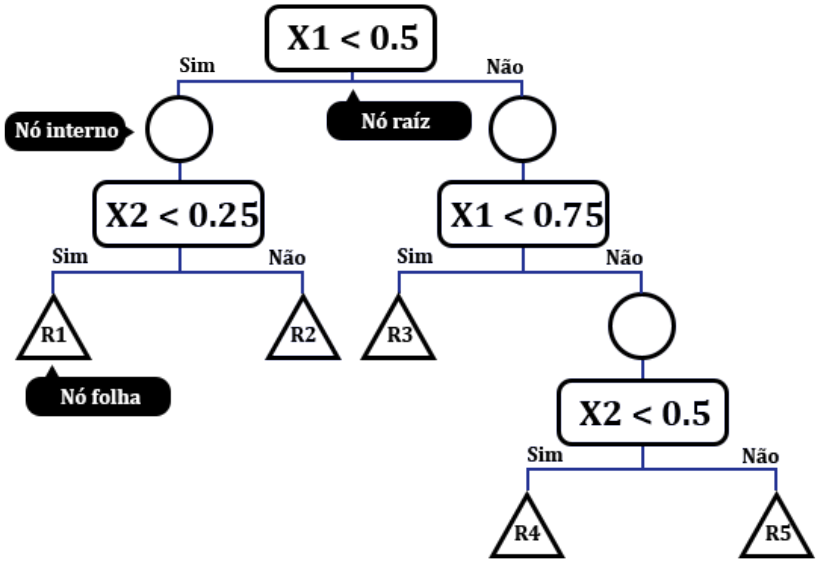
- **Nó raiz:** Também chamado de *nó de decisão*, representa a escolha que levará à subdivisão de todos os registros (ou exemplos, amostras) em dois ou mais subconjuntos mutualmente exclusivos;
- **Nó interno:** Também chamado de *nó de probabilidade*, representa uma das possíveis escolhas disponíveis neste ponto da estrutura em árvore. Nestes nós, a parte superior é conectada ao *nó pai* e a inferior é conectada aos *nós filhos* ou *nós folhas*;
- **Nó folha:** Também chamado de *nó terminal*, representa o resultado final de uma combinação de decisões ou eventos.

As ramificações representam os possíveis resultados que emanam dos nós raízes ou nós internos. Na prática, um modelo de árvore de decisão é construído usando uma hierarquia de ramificações. Cada caminho que parte de um nó raiz, passa pelos nós internos e chega a um nó folha representa uma *regra de classificação*. Esses caminhos também podem ser representados como regras *if-then e.g*, "*se* a condição 1 e a condição 2 ... e a condição k ocorrer, *então* o resultado j ocorre" (SONG; YING, 2015). Na Figura 5 é mostrada a estrutura de uma simples árvore de decisão.

Na figura acima, $X1$ e $X2$ são variáveis de entrada, contínuas, que variam de 0 a 1. Ambas variáveis poderiam ser interpretadas, por exemplo, como atributos ou características de um determinado exemplo ou amostra. Já $R1$, $R2$, $R3$, $R4$ e $R5$ são possíveis valores da variável alvo. Quando a variável alvo é discreta, os nós folhas representam rótulos de classes e, consequentemente, a estrutura como um todo é chamada de *árvore de classificação*. Em contrapartida, quando a variável alvo é contínua (neste caso, como $X1$ e $X2$), a estrutura é chamada de *árvore de regressão* (GAMA, 2004).

Na etapa de *splitting*, as principais variáveis de entrada do problema são identificadas e utilizadas para dividir os registros – nos nós raízes e nos subsequentes nós internos – em duas ou mais categorias, de acordo com o estado dessas variáveis (*e.g.*, $X1 < 0.5$ na Figura 5). Para escolher de maneira ótima qual variável de entrada levar em consideração

Figura 5: Exemplo de Árvore de Decisão



Fonte: Elaborada pelo autor.

no processo de divisão, o modelo utiliza características relacionadas ao grau de *pureza* dos nós filhos recém-gerados para diferenciar uma variável da outra. Esse processo segue até que um critério de homogeneidade ou parada seja satisfeito e não necessariamente todas as variáveis de entradas são utilizadas na construção da árvore de decisão (SONG; YING, 2015). Ademais, uma mesma variável pode ser utilizada em mais de um nível da estrutura em árvore *e.g.*, $X1 < 0.5$ e $X1 < 0.75$ na Figura 5.

Na etapa de *stopping*, são utilizadas estratégias para controlar a complexidade e a robustez dos modelos de árvores de decisão. Como mencionado na Seção 3.1.1, quanto mais complexo um modelo for, menos confiável ele é para a predição de futuros registros, pois não há generalização daquilo que foi aprendido. No contexto dos modelos de árvores de decisão, a situação de extrema complexidade ocorre quando o modelo se espalha o bastante para tornar os registros em cada nó folha 100% puros, isto é, a ponto de todos os registros de observações existentes alcançarem a classificação alvo. Segundo Song e Ying (2015), parâmetros comumente utilizados em regras de parada incluem: (i) o número mínimo de registros em um nó folha; (ii) o número mínimo de registros em um nó antes do *splitting*; e (iii) a profundidade de qualquer nó folha a partir do nó raiz.

Quando as regras de parada não funcionam como esperado, uma estratégia alternativa comumente empregada consiste em deixar que a árvore cresça bastante para, então, reduzi-la posteriormente a um tamanho ótimo através da poda (etapa de *pruning*) dos nós que proveem pouca informação adicional ao modelo construído. Para a seleção da melhor subárvore possível dentre muitos outros candidatos, pode-se levar em consideração a proporção de registros com erros de predição; usar uma base de dados para a validação; ou executar a validação cruzada sobre o modelo treinado (SONG; YING, 2015).

Dentre os principais algoritmos utilizados para a construção de uma árvore de decisão, pode-se citar o CART (*Classification and Regression Tree*), CHAID (*Chi-square Automatic Interaction Detector Decision Tree*), ID3 e C4.5 e C5.0 (SCIKIT-LEARN, 2011a; HONG et al., 2018).

3.2.3 Florestas Aleatórias

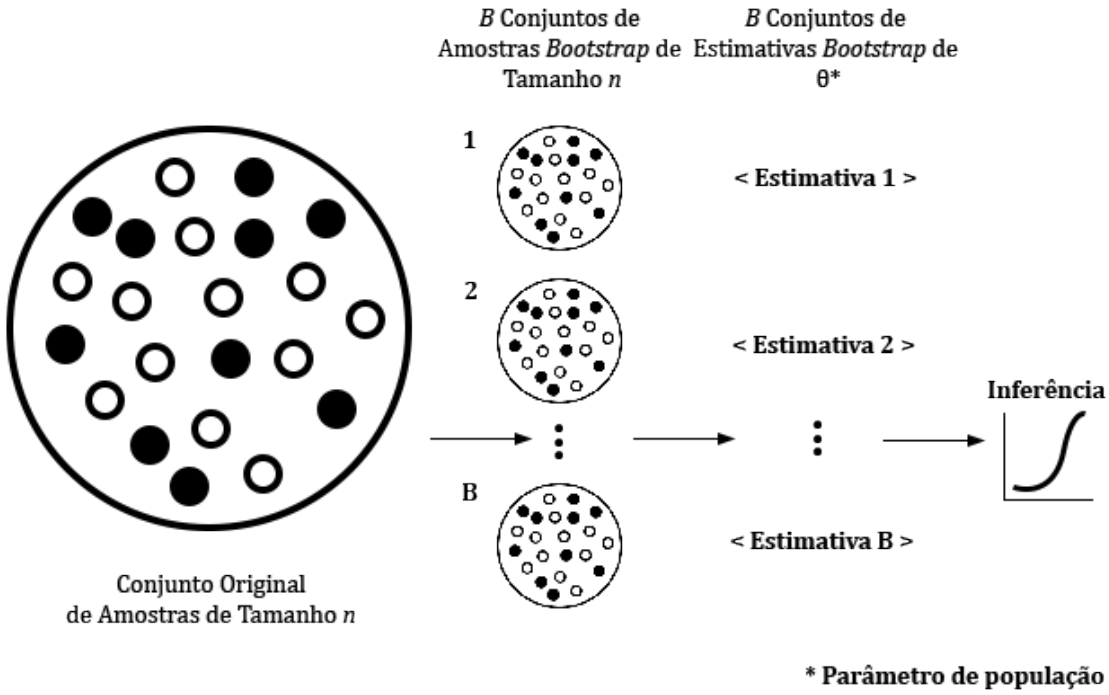
Para compreender melhor o algoritmo de Florestas Aleatórias, é preciso ter em mente a definição de *bagging* (ou *bootstrap aggregation*). *Bagging* é uma técnica comumente empregada na área de aprendizado de máquina para reduzir a variância de uma função de predição estimada. Ao longo dos anos, a técnica provou-se especialmente útil quando aliada a métodos que apresentam alta variância e baixo viés (*low-bias*), como as Árvores de Decisão (FRIEDMAN; HASTIE; TIBSHIRANI, 2001).

Em um problema de regressão, o *bagging* consiste em treinar diversas vezes uma mesma árvore de regressão com versões de amostras *bootstrap* dos dados de treinamento e calcular a média do resultado. Já em um problema de classificação, a técnica consiste em utilizar um *comitê* de árvores onde cada uma lança um voto para a classe predita. Independentemente do problema em questão, a ideia é a mesma: calcular a média de modelos com pouco viés e muito ruído para reduzir a variância (FRIEDMAN; HASTIE; TIBSHIRANI, 2001).

As amostras *bootstrap*, mencionadas anteriormente, são produtos da técnica *bootstrap* (Figura 6). O *bootstrap*, por sua vez, trata-se de uma técnica de simulação baseada em dados para inferência estatística (EFRON; TIBSHIRANI, 1994). É essencialmente um método de reamostragem através da amostragem independente, com substituição, de dados de amostras existentes com o mesmo tamanho de amostra n, realizando inferências entre esses dados reamostrados (YEN, 2019).

Dadas essas definições, entende-se que Florestas Aleatórias é uma modificação substancial do *bagging* que constrói uma grande coleção de árvores não correlacionadas e, então, calcula a média delas (BREIMAN, 2001; FRIEDMAN; HASTIE; TIBSHIRANI, 2001). Considerada uma das principais ferramentas para a predição e a análise de dados, a técnica é notável por apresentar uma alta acurácia, não ser suscetível ao super-treinamento e, também, ter a capacidade de lidar com amostras pequenas, espaços de características de alta dimensão e estruturas de dados complexas (BREIMAN, 2001; SCORNET et al., 2015). As etapas do algoritmo de Florestas Aleatórias, definidas por Friedman, Hastie e Tibshirani (2001), são descritas no Algoritmo 1. A Figura 7 ilustra o funcionamento do Algoritmo 1 em um cenário de classificação de uma amostra desconhecida.

Figura 6: Representação do *Bootstrap*



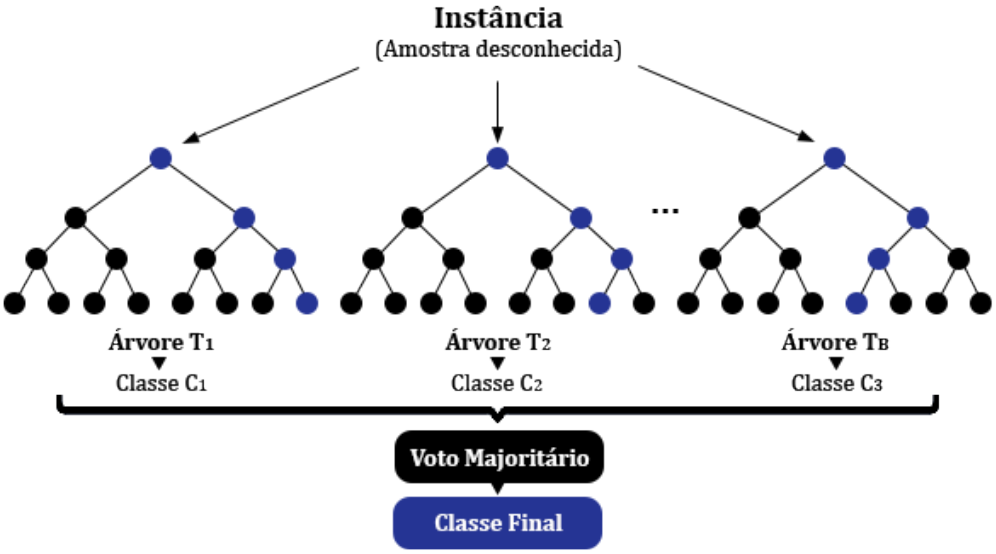
Fonte: Elaborada pelo autor.

ALGORITMO 1: Florestas Aleatórias para Regressão ou Classificação

início
  para $b = 1 \rightarrow B$ **faça**
    (a) Retire uma amostra *bootstrap* $\mathbf{Z}^*$ de tamanho N dos dados de treinamento.
    (b) Construa uma árvore da floresta aleatória T_b para os dados reamostrados através da repetição recursiva dos próximos passos para cada nó terminal da árvore, até que tamanho de nó mínimo n_{min} seja alcançado:
      i Selecione aleatoriamente m variáveis dentre p variáveis.
      ii Escolha a melhor variável/ponto de divisão dentre as m variáveis.
      iii Divida o nó em dois nós filhos.
  fim
  Gere a saída com o conjunto de árvores T_{b1}^{B}.
  Para fazer uma predição em um novo ponto x:
  Regressão: $\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.
  Classificação: Seja $\hat{C}_b(x)$ ser a predição de classe da b-ésima árvore da floresta aleatória. Então $\hat{C}_{rf}^{B}(x) = \textit{voto majoritário}\ \hat{C}_b(x)_1^B$.
fim

Neste processo, a amostra desconhecida é fornecida como *entrada* em todas as n árvores que compõem a floresta, produzindo n *saídas* que, neste caso, são rótulos de classe. Então, através do *voto majoritário*, é definida a classe final da amostra desconhecida.

Figura 7: Exemplo de Florestas Aleatórias



Fonte: Elaborada pelo autor.

3.2.4 Máquina de Vetores de Suporte

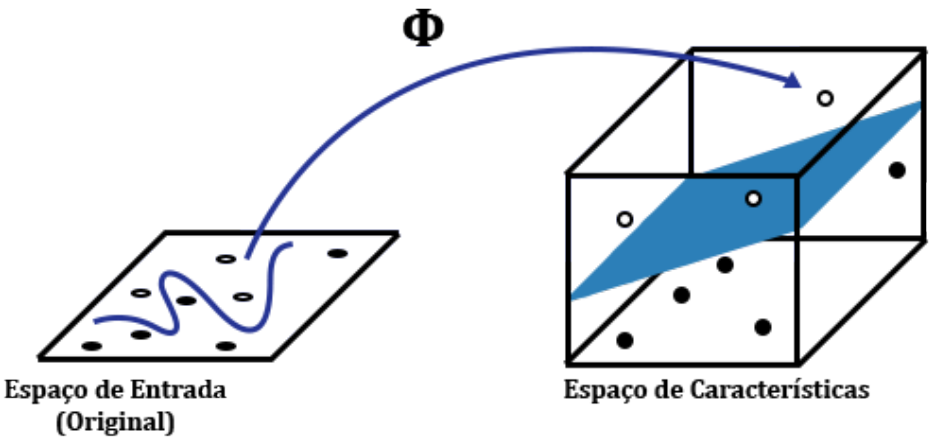
Originalmente propostas por Cortes e Vapnik (1995), as Máquinas de Vetores de Suporte (em inglês, *Support Vector Machines* - SVMs) tornaram-se extremamente populares após sua introdução na década de 90, especialmente na comunidade de Aprendizado de Máquina (BRERETON; LLOYD, 2010; JAMES et al., 2013). Trata-se de uma abordagem para classificação baseada no princípio da *minimização do risco estrutural*, da teoria da aprendizagem computacional, com grande poder de generalização em diversos domínios e robustez a dados de alta dimensionalidade (JOACHIMS, 1998; LORENA; CARVALHO, 2008).

A princípio, o algoritmo de SVMs foi desenvolvido para solucionar problemas envolvendo apenas duas classes. No entanto, com o passar dos anos, extensões permitiram que o algoritmo solucionasse, também, problemas multiclasse e problemas de regressão (neste caso, usa-se *Support Vector Regression*) (BRERETON; LLOYD, 2010).

Dado um conjunto de dados com n exemplos $(\vec{x_i}, y_i)$, onde cada $\vec{x_i}$ é um dado de entrada e $y_i \in \{-1, +1\}$ corresponde ao rótulo de $\vec{x_i}$, o SVM procura por um *hiperplano* $\vec{w} \cdot \vec{\Phi} + b = 0$, em um espaço de alta dimensionalidade, capaz de separar os dados das classes $+1$ e -1 com uma margem máxima.

Na definição acima, dada por Lorena e Carvalho (2008), $\vec{w}$ é um vetor de peso ortogonal ao hiperplano, b é um termo de *offset* e $\vec{\Phi}$ é uma função responsável por mapear os dados para um espaço de alta dimensionalidade (ou espaço de características) onde eles podem ser separados por um hiperplano com um baixo erro de treinamento. A Figura 8 mostra o mapeamento da função Φ.

Figura 8: Mapeamento da função Φ



Fonte: Elaborada pelo autor.

A maximização da margem mencionada anteriormente equivale à minimização da norma de $\vec{w}$. Consequentemente, os SVMs são treinados para resolver o seguinte problema de otimização:

$$\begin{aligned} \text{Minimizar} \quad & \|\vec{w}\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{Sujeito a:} \quad & y_i(\vec{w}\cdot\vec{\Phi}(\vec{x_i}+b) \geq 1-\xi_i \text{ e } \xi_i \geq 0 \\ & \text{para } i = 1,\ldots,n. \end{aligned} \tag{3.8}$$

onde C é um parâmetro de regularização que impõe um balanceamento entre o erro de treinamento e a generalização e ξ_i são variáveis de folga. Se, por um lado, as restrições são impostas para garantir que nenhum dado fique dentro das margens do hiperplano, as variáveis de folga, por outro lado, são usadas justamente para relaxar essa imposição, evitando o super-treinamento do modelo. A fronteira de decisão obtida com a resolução do problema de otimização (Equação 3.8) é dada por:

$$f(\vec{x}) = \sum_{\vec{x_i}\in SV} y_i\alpha_i\vec{\Phi}(\vec{x_i})\vec{\Phi}(\vec{x}) + b \tag{3.9}$$

onde as constantes α_i são chamadas de multiplicadores de Lagrange, determinadas no processo de otimização, e SV é o conjunto de vetores de suporte, isto é, de exemplos de treinamento cujos respectivos multiplicadores de Lagrange são maiores que zero.

Para todos os outros exemplos de treinamento, os multiplicadores de Lagrange associados são nulos e, portanto, elas não contribuem para a determinação da hipótese

final. Para obter a classificação de um exemplo desconhecido x, uma função sinal é aplicada a $f(\vec{x})$ de modo que, se $f(\vec{x}) > 0$ a classe predita é $+1$, se $f(\vec{x}) < 0$ a classe predita é -1 e, se $f(\vec{x}) = 0$ a classe é dita desconhecida ou atribuída aleatoriamente (LORENA; CARVALHO, 2008).

Note que, na Equação 3.9, a única informação necessária sobre a função de mapeamento é como calcular o produto escalar entre dois exemplos mapeados. Esse produto pode ser alcançado através de funções *kernel*, que permitem acesso à espaços de alta dimensão sem precisar conhecer a função de mapeamento explicitamente, o que normalmente é complexo (LORENA; CARVALHO, 2008). Na Tabela 1 são mostrados alguns exemplos de função *kernel*, onde d, r e γ são parâmetros arbitrários (SCIKIT-LEARN, 2011c).

Tabela 1: Exemplo de funções *kernel*

Nome	**Função *kernel***
Linear	$\langle x, x' \rangle$
Polinomial	$(\gamma \langle x, x' \rangle + r)^d$
Radial Basis Function (RBF)	$\exp(-\gamma \|x - x'\|^2)$
Sigmóide	$\tanh(\gamma \langle x, x' \rangle + r)$

Fonte: Elaborada pelo autor.

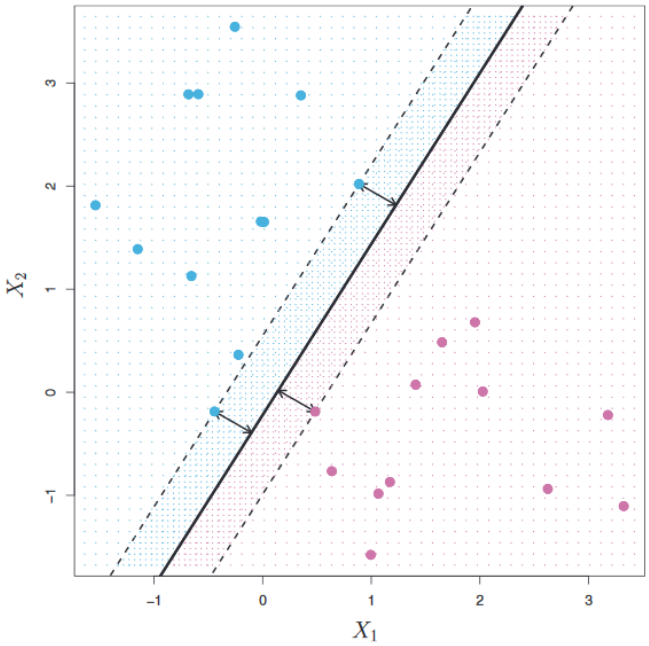
Na Figura 9 é mostrado um hiperplano que separa duas classes, identificadas pela cor azul e roxa. O hiperplano é representado por uma linha sólida e, as margens, pela distância (em ambos os lados) entre a linha sólida e a tracejada. Os três pontos próximos às margens, indicados pelas setas bidirecionais, são os chamados vetores de suporte.

Devido a diversas complexidades intrínsecas aos problemas multiclasse, normalmente evita-se a utilização de uma única formulação de SVM para resolvê-los. Alternativamente, são usadas estratégias como: *one-against-all* (1AA), *all-against-all* (AAA) e *error-correcting output coding* (ECOC) (DUAN; KEERTHI, 2005; LORENA; CARVALHO, 2008; ZHU, 2006). Em todas essas estratégias, a ideia básica por trás delas é a combinação de vários classificadores SVMs binários (para duas classes) para resolver um dado problema multiclasse (DUAN; KEERTHI, 2005).

3.2.5 AdaBoost

Segundo Schapire (2013), *boosting* é uma abordagem para o aprendizado de máquina baseada no princípio de que é possível criar uma regra de predição altamente precisa através da combinação de muitas regras relativamente fracas e imprecisas. Dito isso, o algoritmo de *AdaBoost* (abreviação de *Adaptative Boosting*) de Freund e Schapire (1997) foi o primeiro algoritmo prático de *boosting* e, até hoje, é amplamente estudado e aplicado nas mais diversas áreas do conhecimento.

Figura 9: Exemplo de Hiperplano



Fonte: James et al. (2013).

Em uma primeira etapa, o algoritmo de *AdaBoost* tem por objetivo observar o desempenho de um determinado conjunto de classificadores sobre um conjunto de treinamento. Nessa etapa, todos os "acertos" e "erros" dos classificadores são registrados e, em ambos os casos, ocorre a penalização dos classificadores, sendo mais rigorosa quando é registrado um "erro" de classificação.

O *AdaBoost* procede, então, sistematicamente extraindo um classificador do conjunto a cada iteração do algoritmo. Todos os elementos do conjunto de dados são ponderados de acordo com sua atual relevância em cada iteração, sendo que, no início, todos os elementos são ponderados com o mesmo valor.

À medida que o processo de extração de classificadores continua, quanto mais difíceis forem os exemplos, maior serão os seus pesos. Desta forma, o algoritmo foca na seleção de novos classificadores para o comitê que, de certo modo, ainda colaboram positivamente para a classificação de exemplos classificados erroneamente (ROJAS, 2009).

3.3 Métricas para Análise de Eficácia

Dada a diversidade de algoritmos de aprendizado de máquina que podem ser aplicados em um mesmo contexto para resolver problemas, faz-se necessária a utilização de métricas para a avaliação e a comparação de seus respectivos desempenhos. Em trabalhos que unem a utilização desses algoritmos com a tarefa de detectar intrusões – vide aqueles apresentados na próxima seção – é comum a utilização de métricas relacionadas às taxas

de falso-positivos e falso-negativos obtidas em cada técnica ou, então, no próprio sistema de classificação como um todo *e.g.*, IDSs. A título de exemplo, em Pietraszek e Tanner (2005) é possível encontrar uma análise geral sobre as particularidades de se lidar com técnicas de mineração de dados e de aprendizado de máquina, no âmbito dos IDSs, na busca pela redução de taxas de falso-positivos nessas ferramentas.

Para melhor entender o significado dessas taxas, é preciso ter em mente que, na detecção de intrusão, as atividades benignas, *i.e*, normais, são consideradas *condições negativas* enquanto que as atividades malignas, *i.e.*, ataques, são consideradas *condições positivas*. Desta forma, quando um sistema de detecção de intrusão ou um classificador identifica uma instância *verdadeiramente* benigna como sendo "normal" (classificação correta), essa contribui para a taxa de *verdadeiro-negativos* (VN). De maneira análoga, quando uma instância *verdadeiramente* maligna é identificada como sendo "ataque", essa contribui para a taxa de *verdadeiro-positivos* (VP).

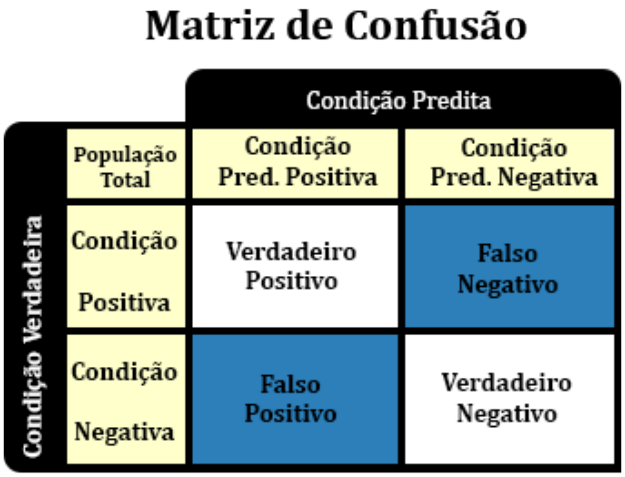
Seguindo essa lógica, quando uma instância *verdadeiramente* benigna é identificada *falsamente* como sendo "ataque", essa contribui para a taxa de *falso-positivos* (FP). Por fim, quando uma instância *verdadeiramente* maligna é identificada *falsamente* como sendo "normal", essa contribui para a taxa de *falso-negativos* (FN).

Por conveniência, essas taxas são normalmente apresentadas na forma de uma Matriz de Confusão, como mostrada na Figura 10. A partir dessas mesmas taxas, é possível derivar algumas métricas populares para a avaliação de modelos e sistemas no geral:

- **Acurácia:** $\frac{VP+VN}{VP+VN+FP+FN}$. É a razão entre as observações preditas corretamente e o total de observações;
- **Precisão:** $\frac{VP}{VP+FP}$. É a razão entre as observações *positivas* preditas corretamente e o total de observações *positivas* preditas. Em outras palavras, levando em consideração o âmbito dos IDSs, é a razão entre o número de ataques corretamente preditos e o número total de ataques preditos;
- ***Recall*:** $\frac{VP}{VP+FN}$. É a razão entre as observações *positivas* preditas corretamente e o total de observações na classe atual. Novamente, no âmbito dos IDSs, é a razão entre o número de ataques corretamente preditos e o número total de ataques existentes.
- ***F-Score*:** $2 \cdot (\frac{\text{Precisão} \cdot \textit{Recall}}{\text{Precisão} + \textit{Recall}})$. É a média ponderada da Precisão e do *Recall*. Consequentemente, é uma métrica que leva em consideração ambas as taxas de falso-positivos e falso-negativos (relacionadas aos erros de classificação).

Ressalta-se que a métrica de acurácia nem sempre é confiável, principalmente quando o conjunto de dados é desbalanceado (como é o caso do problema de detecção de intrusão). Por exemplo, dado um conjunto de dados composto por 95 instâncias benignas

Figura 10: Exemplo de Matriz de Confusão



Fonte: Elaborada pelo autor.

e 5 instâncias malignas, um classificador qualquer que classificasse todas as instâncias malignas como benignas teria uma acurácia de 95% que, superficialmente, aparenta ser um bom resultado. No entanto, como as instâncias malignas representam possíveis ataques à rede e, esse classificador não foi capaz de detectá-las no conjunto de dados, a qualidade do modelo é muito inferior àquela que ele aparenta ser sob a ótica da acurácia. Nesse mesmo exemplo, o *recall*, que é voltado para a análise das classificações feitas sobre ataques em relação à verdadeira quantidade de ataques existentes, seria de 0%.

4 Trabalhos Correlatos

Por estar no mercado há mais de uma década, o Snort já foi alvo de inúmeras pesquisas que não apenas avaliavam o seu desempenho em base de dados populares, para efeito de *benchmarking*, mas que também buscavam propor melhorias na ferramenta, seja no processamento de regras, na redução de falso-positivos ou na detecção de ataques conhecidos e desconhecidos. Neste capítulo, são apresentados em ordem cronológica alguns destes trabalhos que propuseram melhorias, com um maior destaque aos que utilizaram técnicas inteligentes de Aprendizado de Máquina.

Uma das primeiras grandes contribuições à área que merece destaque - apesar de não ter utilizado técnicas de *Machine Learning* - foi um projeto desenvolvido na Silicon Defense, uma companhia fundada pela *Defense Advanced Research Projects Agency* (DARPA), conhecido por SPADE (*Statistical Packet Anomaly Detection Engine*). Em Staniford, Hoagland e McAlerney (2002), Biles (2003), foi criado um pré-processador capaz de detectar o tráfego de rede que desvia do comportamento "normal" da rede monitorada pelo Snort.

Para detectar o tráfego "anormal" (ou os pacotes maliciosos), o SPADE mantinha internamente tabelas de probabilidades sobre a ocorrência de diferentes tipos de pacotes ao longo do tempo na rede, sendo que as ocorrências mais recentes tinham um peso maior do que as mais antigas, que gradativamente eram descartadas pelo pré-processador.

A partir destas probabilidades, era possível calcular um "escore de anomalia" para cada ocorrência registrada no pré-processador, podendo ser um *valor bruto* ($a(X)$), dado por $-log2(P(X))$ (onde $P(X)$ é a probabilidade de um dado pacote X), ou um *valor relativo* ($A(X)$), dado pela razão entre $a(X)$ e o maior *valor bruto* possível.

Desta forma, o *valor relativo* podia assumir valores de 0 à 1, sendo 0 "completamente normal" e 1 "completamente anormal". Na configuração do pré-processador, era possível especificar para cada pacote X um determinado limite de $A(X)$ que, uma vez ultrapassado, geraria alertas para notificar o comportamento "anormal" na rede.

Apesar de ter sido uma imensa contribuição à área, por aplicar no Snort o conceito de detecção *zero-day*, o SPADE não era capaz de dizer se uma anormalidade era um "ataque" ou um "erro de configuração", justamente por não analisar a intenção do pacote, apenas dizer se ele está dentro ou fora do comportamento "normal" da rede.

Em Silva et al. (2004), foi criado um método de detecção de intrusão em rede para identificar e classificar informações ilegítimas no *payload* de pacotes TCP/IP, baseado no conjunto de assinaturas do Snort que representa possíveis ataques à rede.

A metodologia contou com a utilização da rede neural artificial *Hamming Net* aliada à MAXNET, devido ao seu rápido aprendizado em relação a outras técnicas – Perceptron Multicamadas (*Multilayer Perceptrons* - MLP) com Retropropagação, Mapas de Konohen – e por não exigir treinamentos exaustivos.

A partir do arquivo de regras do Snort, foram extraídas as assinaturas dos ataques e criados os chamados arquivos "exemplares", em um formato compreendido pela *Hamming Net*. Através dos dados ilegítimos contidos nestes arquivos exemplares e, também, dos dados legítimos provenientes de uma base de dados, foi criado um arquivo de *payloads* de pacotes TCP/IP.

À medida que o arquivo de *payloads* era processado pela Hamming Net – previamente configurada a partir dos arquivos exemplares – as saídas da rede neural eram repassadas para a MAXNET, que tinha como saída o maior valor dentre aqueles concedidos na entrada. Desta forma, a MAXNET eliminava, gradualmente, todos os valores inferiores ao valor máximo.

Este processo repetia-se até que restasse apenas um único valor não-nulo que, quando comparado a um limite, definia se a atividade era ou não maliciosa, concluindo o processo de classificação. Com a realização dos experimentos, constatou-se que, em média, 70% dos dados de entrada (contendo informações ilegítimas) foram satisfatoriamente classificados, dando destaque à velocidade com que estas classificações ocorreram. No entanto, os autores enfatizaram o desafio que é encontrar o limite ideal para obter uma melhor taxa de detecção e, ainda, propuseram como trabalho futuro aplicar esta metodologia em dados capturados em tempo real, uma vez que, neste trabalho, ela foi aplicada em um ambiente *offline* processando dados gravados.

Visando a redução da quantidade de alertas falsos em IDSs, Weon et al. (2005) apresentaram uma abordagem baseada em comportamento com um algoritmo de aprendizado supervisionado baseado em memória. O sistema proposto foi originalmente desenvolvido para combinar dois IDSs heterogêneos: um sistema baseado no Snort, desenvolvido pela *Eletronics and Telecommunication Research Institute* (ETRI), e outro sistema baseado em anomalia, desenvolvido por co-autores deste mesmo artigo.

Os autores usaram um variante do algoritmo de Aprendizado Baseado em Instâncias (*Instance-based Learning* – IBL) como investigador comportamental do Snort. Sua tarefa era analisar os alertas gerados pelo Snort (com relevância para o administrador do sistema) e, também, gerar alertas utilizando um processo de análise próprio do IBL. Ou seja, mesmo que o Snort não gerasse um alerta sobre um determinado evento, ainda havia a possibilidade do IBL gerar um alerta para o mesmo evento.

Na prática, o IBL armazenava um subconjunto de instâncias reais dos exemplos de treinamento identificadas como "normais" ou "anormais". Assim, o conhecimento que ele

tinha era o conjunto de instâncias consideradas “importantes” para o problema. Sempre que um evento ocorria na rede, o IBL iniciava uma busca na base de conhecimento pela instância mais similar àquela referente ao evento em questão, classificando-a de acordo com o seu aprendizado.

Os resultados obtidos pelos autores mostram que o sistema híbrido foi capaz de reduzir a taxa de alertas falsos do Snort em até 60%, mantendo a maioria dos alertas verdadeiros. O sistema híbrido também conseguiu detectar um número de ataques novos que o Snort, por si só, não conseguiu. No entanto, os autores comentam que a taxa de redução ainda está longe do considerado “satisfatório”. Um dos motivos que pode ter levado a tais resultados é a utilização de um conjunto pobre de assinaturas no Snort, pois muitos alertas são acionados a partir do mesmo conjunto de ataques.

No trabalho de Silva (2007), foi desenvolvida uma ferramenta com o intuito de facilitar a implementação de diversos tipos de RNAs dentro do Snort. Uma vez que as RNAs estavam treinadas, a ideia principal do trabalho era substituir a abordagem convencional de detecção do IDS (baseada em regras) por outra baseada exclusivamente nas técnicas inteligentes para classificação de pacotes individuais.

No processo de criação e treinamento destas técnicas, o autor optou por utilizar a ferramenta de código aberto JavaNNS (*Java Neural Network Simulator*), um simulador com interface gráfica que oferece meios práticos para o manuseio de RNAs. Adaptando o código deste simulador para a linguagem C, foi possível aproveitar as estruturas criadas e treinadas por ele e importá-las no Snort. Desta forma, com algumas modificações no núcleo do Snort, à medida que o IDS processava os pacotes individualmente, eram feitas chamadas às funções escritas pelo autor que recorriam às classificações das RNAs importadas.

Apesar do trabalho ter sido concluído com sucesso, em termos de implementação, o autor reconheceu a dificuldade em utilizar RNAs em IDS, tendo em vista que a taxa de detecção do IDS passa a estar intrinsecamente ligada à configuração dos parâmetros destas técnicas, que nem sempre são fáceis de calibrar para obter o desempenho ótimo. Além disso, também foi ressaltado que a utilização de características de grupos de pacotes, isto é, de conexões, produziriam melhores resultados em comparação à utilização de características individuais de pacotes, tal como foi feito neste trabalho.

Em Makanju, Zincir-Heywood e Milios (2008), é apresentada uma comparação, em termos de capacidade de detecção, entre o Snort-Wireless, baseado em assinatura, e uma solução baseada em Programação Genética (*Genetic Programming* – GP). Os critérios utilizados foram: a adaptabilidade em frente a ataques modificados, a detecção independente de infraestrutura e a adaptabilidade a ataques similares desconhecidos.

Os experimentos foram realizados em três redes físicas distintas. Enquanto a Rede I e a Rede II são similares quanto à infraestrutura (diferem apenas nos pontos de acesso

sem fio utilizados), a Rede III foi exclusivamente configurada como *Extended Service Set* (ESS). O objetivo destas configurações diversificadas era reproduzir a natureza dos ambientes reais de rede, essencialmente heterogêneos.

Em todas as redes foram gerados ataques com o *void11*[1] a partir de uma máquina atacante. Na máquina de monitoramento, todos os dados dos ataques foram gravados através de um mecanismo de registro do *Kismet Wireless*. Posteriormente, todos os registros foram então processados pelo Snort-Wireless e a solução baseada em GP, sendo utilizados inclusive nas etapas de treinamento e teste desta última.

Os resultados obtidos pelos autores mostraram que os IDSs baseados em GP são mais adaptáveis que os IDSs convencionais na detecção de ataques modificados. Além do mais, os IDSs baseados em GP também mostraram-se capazes de realizar detecções independentes de infraestrutura e detectar ataques similares desconhecidos. No entanto, é preciso prestar atenção à representação das características das bases de dados utilizadas no treinamento e teste do algoritmo de aprendizado.

De modo geral, os resultados obtidos também mostram que um IDS baseado em aprendizado de máquina, tal como o GP, não apenas tem alta confiabilidade como também capacidade de sobrevivência diferentemente dos IDSs convencionais.

No trabalho de Ding, Xiao e Liu (2009), foi desenvolvido uma metodologia híbrida de detecção de intrusão que combina a abordagem baseada em assinatura com a abordagem baseada em anomalia. O HIDS proposto possuia três módulos: o módulo de detecção por assinatura, o módulo de detecção por anomalia e o módulo de geração de assinaturas. A base do módulo de detecção por assinatura é o motor e o conjunto de regras do Snort. O módulo de detecção por anomalia é construído utilizando-se Regras de Episódios Frequentes (*Frequent Episode Rules* – FER). Por fim, o módulo de geração de assinaturas é baseado em um variante do algoritmo *apriori*.

O funcionamento ideal do sistema é obtido no cenário em que os ataques não reconhecidos pelo módulo de detecção por assinatura são identificados pelo módulo de detecção por anomalia que, por sua vez, repassa as informações de tais eventos para o módulo de geração de assinatura. Desta forma, esses mesmos ataques podem ser identificados futuramente pelo próprio módulo de detecção por assinatura.

No módulo de detecção por anomalia, o algoritmo FER foi treinado e testado a partir de conexões provenientes da base de dados KDD Cup '99. Em média, a acurácia do algoritmo FER foi de 94,07%. Segundo os autores, tal resultado indica que o algoritmo FER faz uma boa descrição das características dos eventos de conexão usando *slide scanning windows* para a mineração de episódios frequentes normais. Apesar do HIDS proposto ter obtido bons resultados em um ambiente *offline*, não foi possível manter uma

[1] *void11* é uma ferramenta utilizada para forçar a desautenticação de clientes dos seus respectivos pontos de acesso sem fio.

acurácia alta em um ambiente tempo real.

O estudo de Li e Liu (2010) apresenta uma análise sobre a aplicação de Máquinas de Vetores de Suporte (*Support Vector Machines* – SVMs) no Snort. Segundo os autores, um IDS baseado em SVM consiste de três elementos: 1) um pré-processador de dados de auditoria, 2) um classificador SVM e 3) um sistema de decisão. No cenário levado em consideração, o conjunto de dados base utilizados são provenientes do próprio tráfego normal de rede, enquanto os ataques foram adicionados (em menor quantidade) nas categorias: *Probing*, *Denial of Service* (DoS), *User-to-Root* (U2R) e *Local-to-Remote* (L2R).

No esquema proposto, à medida que os dados são processados pelo pré-processador, as características dos pacotes são repassadas para um classificador SVM da "primeira camada", com o intuito de determinar se aquelas características indicam ou não uma tentativa de intrusão. Caso seja constatado que aqueles dados em análise são uma tentativa de intrusão, as mesmas características são então repassadas para um classificador SVM de "segunda camada", caso contrário, são descartadas e o próximo dado de auditoria é processado.

Nesta segunda etapa da classificação, o objetivo é identificar o tipo do ataque observado, para que medidas preventivas possam ser tomadas de acordo (por exemplo, em um *firewall*). Apesar do trabalho oferecer uma arquitetura interessante e, aparentemente, eficiente para o uso de SVM em IDS (a divisão da classificação em duas etapas diminui o processamento da ferramenta), por não ter sido aplicado é difícil estimar até que ponto esta solução é viável, no próprio contexto de detecção de ataques, pelos mesmos problemas de calibração das técnicas citados nos trabalhos mencionados anteriormente.

Em Fang e Liu (2011), é apresentado um estudo sobre a integração de RNAs no Snort para que a ferramenta possa se adaptar às redes e detectar anomalias. Neste projeto, uma *Elman Network* foi incorporada em um pré-processador para a detecção inteligente de *port scan*, treinada com o *Sttutgard Neural Network Simulator* (SNNS).

Segundo os autores, a escolha da rede de *feedback* parcial *Elman Network* deve-se justamente ao fato desse *feedback* poder detectar e identificar "modelos que mudam com o tempo" (*time-changing models*). Como o ataque de *port scanning* pode causar diferentes tempos de conexão entre endereços IP, a natureza desta técnica recai sobre os modelos tangíveis pela *Elman Network* e, portanto, seria conveniente utilizá-la para identificá-lo.

O SNNS treina a *Elman Network* a partir de duas base de dados: uma com o tráfego normal e outra com o tráfego anormal da rede (referente aos *port scans*). Uma vez treinada, o código da rede (em linguagem C) é integrado no pré-processador desenvolvido para o Snort. Consequentemente, o IDS passa a ser mais inteligente na detecção de novos ataques ou de variantes daqueles ataques já conhecidos.

Devi e Nagpal (2012) apresentam um estudo sobre como o GA poderia melhorar

o desempenho de um NIDS existente, no caso, o Snort. Com a implementação da técnica, o Snort passou a ser capaz de gerar e atualizar os seu próprio conjunto de regras em tempo real, fazendo-se o uso do GA para reduzir a taxa de falso-positivos ao longo do tempo. Nota-se que este trabalho apresentou uma implementação única de GA por levar em consideração ambas informações *temporais* e *espaciais* de uma conexão de rede durante a codificação do problema. Posteriormente, um trabalho semelhante foi desenvolvido em Kumar e Punia (2013), também aplicado sobre o Snort.

O estudo de Jongsawat e Decharoenchitpong (2015) apresenta uma proposta para a utilização de uma abordagem bayesiana para detectar relações entre variáveis de uma base de dados de tráfego de rede. A partir destas relações, o objetivo é gerar regras baseadas em comportamento para o Snort.

Antes de criar as regras para a ferramenta, foi realizada uma análise baseada em comportamento sobre o tráfego de rede da base de dados. Para isso, dois algoritmos de aprendizado de Redes Bayesianas foram aplicados sobre a base para a criação dos modelos de Redes Bayesianas: *Essential Graph Search* e *Bayesian Search*.

Em seguida, foi realizada uma Inferência Bayesiana para examinar possíveis relações entre as variáveis. Com base na intensidade de uma relação e/ou impacto entre as variáveis, os autores criaram, então, as regras para o Snort, levando em consideração o ambiente de rede deles e as necessidades destacadas para a construção de um IDS eficaz. Os autores deixaram, como trabalho futuro, provar que tal metodologia é eficaz, uma vez que não o fizeram neste estudo.

Em Naik (2015), é apresentado um IDS baseado em inferência fuzzy, o FI-Snort. Tal IDS é composto por dois componentes principais: o Snort e um sistema de inferência fuzzy. Nesta arquitetura, o Snort é responsável por coletar o tráfego de rede e realizar as atividades de monitoramento e análise, sendo que, os valores de alguns parâmetros de rede são repassados para o sistema de inferência fuzzy, como o tempo médio de pacote e os números de pacotes enviados e recebidos. O sistema de inferência fuzzy ajuda o Snort a decidir o nível de gravidade de um ataque *port scan* usando os valores destes parâmetros, algo que o Snort não realizaria por conta própria.

A coleta de dados para os experimentos envolveu seis computadores ao todo. Cinco computadores realizavam os ataques de *port scanning* em um único *host*, com as ferramentas *Nmap* e *Advanced Port Scanner*. Foram realizadas cinco baterias de ataques, variando o número de atacantes em cada uma delas (muito baixo, baixo, médio, alto e muito alto). Os resultados obtidos confirmaram o sucesso do FI-Snort e indicaram o nível de gravidade das ameaças juntamente dos alertas gerados pelo Snort. Além disso, o FI-Snort também foi capaz de reduzir o número de falso-positivos e falso-negativos na ferramenta.

Como visto em Naik (2015), a aplicação da inferência fuzzy no Snort demonstrou

ser uma boa alternativa para a criação de um sistema de detecção de intrusão inteligente, visto que a versão modificada do IDS fez com que ataques de *port scanning* passassem a ser classificados em diversos níveis de gravidade e as taxas de falso-positivos e falso-negativos fossem reduzidas significativamente.

No entanto, o FI-Snort pode ter sua acurácia comprometida por basear-se em um conjunto estático de regras fuzzy pois, ao longo do tempo, tal conjunto pode vir a se tornar obsoleto em relação aos ataques mais recentes de *port scanning*.

Para contornar este problema, os autores propuseram um novo sistema – também baseado no Snort – que utilizasse a interpolação dinâmica de regras fuzzy, dando origem ao D-FRI-Snort (NAIK; DIAO; SHEN, 2016). A operação deste novo sistema é semelhante ao do trabalho anterior, no quesito de também realizar a classificação de alertas em determinados níveis de gravidade, de acordo com o conjunto base de regras fuzzy. No entanto, em casos onde nenhuma regra fuzzy tem um *matching* com o alerta em análise, o novo sistema realiza uma etapa de interpolação entre as regras já existentes com o intuito de gerar os resultados necessários.

Naturalmente, todas as regras e resultados gerados na etapa de interpolação são armazenados e, de tempos em tempos, o sistema dinamicamente os promove como novas regras fuzzy do conjunto base. Tal característica permite que o Snort (modificado) se adapte às condições atuais da rede monitorada, algo que foi não previsto no trabalho anterior. Este sistema também reduziu as taxas de falso-positivos e falso-negativos na ferramenta.

No trabalho de Seelammal e Devi (2016), é apresentado um IDS baseado em detecção por anomalia que utiliza a análise de *big data* do framework *Hadoop*. Em um primeiro instante, com o Snort, foi capturado na forma de base de dados o comportamento da rede, que compreende: registros, comportamento de usuários e informações críticas de antivírus, de dispositivos inteligentes e do próprio sistema. Após a coleta destas informações, a base foi então analisada pelo framework *Hadoop* utilizando a classificação da Árvore de Decisão C4.5.

O sistema proposto foi analisado levando em consideração o tempo de execução e o ganho de velocidade (com e sem o *Hadoop*). Os resultados obtidos mostraram que, sem o *Hadoop*, o tempo de execução tende a crescer se o número de instâncias de treinamento aumenta, o que é esperado. No entanto, com o *Hadoop*, o sistema passou a aprender o comportamento de padrões vistos anteriormente e, consequentemente, mesmo com o aumento das instâncias de treinamento o tempo de execução tendeu a cair gradualmente. O ganho de velocidade observado nos experimentos foi linear. Tais resultados demonstram ganhos no desempenho do IDS e na capacidade de prover respostas rápidas a diversos tipos de ataques à rede.

O estudo de Mahmoud, Ali e Elshafie (2016) apresenta um NIDS com detecção por assinatura e anomalia baseado no Snort e no Algoritmo de Seleção Negativa (*Negative Selection Algorithm* – NSA). Para avaliar a eficiência do NIDS híbrido, o mesmo foi testado com a base de dados DARPA 1999.

Os resultados experimentais mostraram que o sistema proposto teve um desempenho superior ao do Snort não modificado, isto é, sem o auxílio do NSA. A construção do NIDS híbrido teve duas etapas principais: a geração de detectores e a combinação. Na primeira etapa, os dados de treinamento da base de dados DARPA 1999 foram utilizados para gerar os detectores de intrusão seguindo o próprio algoritmo de geração de detectores do NSA.

Na segunda etapa, através de um pré-processador, o conjunto de detectores gerados e o restante do algoritmo do NSA foram incorporados ao Snort, consolidando a solução híbrida de NIDS. O processo de experimentação do sistema proposto contou com o processamento da base DARPA 1999. Dos 201 ataques contidos na base de dados, 33 deles foram detectados pelo Snort não modificado (sem o algoritmo NSA). Já no sistema proposto, 102 deles foram detectados. O aumento no número de ataques detectados pela solução híbrida deve-se ao fato do NSA realizar a detecção de anomalias no cabeçalho dos pacotes e detectar ataques que o Snort não consegue detectar usando somente os arquivos de definição de regras.

Em Kumar et al. (2017), é apresentado um algoritmo de aprendizado de máquina usando *Anova F-Test* para combater *malwares* polimórficos. As características selecionadas pelo *Anova F-Test* foram implantadas no Snort para a construção de um sistema de defesa efetivo.

Para a geração de *malwares* polimórficos, os autores optaram pela utilização do framework *msfvenom* disponível na distribuição *Kali Linux*. A escolha da distribuição deve-se às inúmeras opções de *payloads* e *encoders* para a exploração de falhas. Ao todo, os experimentos contaram com 600 arquivos executáveis, sendo metade deles benigna.

Para cada arquivo executável utilizado nos experimentos, foram extraídas suas respectivas características em representação *2grams*. Posteriormente, com o algoritmo *Modified KMP* (MKMP), foi construída uma matriz de frequência que armazenava o número de vezes que determinada característica *2grams* aparecia em um *malware* polimórfico ou arquivo benigno.

Através da técnica de extração de características *Anova F-Test*, foram escolhidos os k melhores *2grams* da matriz de frequência. Os k melhores *2grams* foram utilizados tanto para a criação de assinaturas no Snort como, também, o treinamento de um modelo de Regressão Logística. Consequentemente, na fase de testes, à medida que os arquivos executáveis eram baixados de um servidor Apache, o Snort gerava alertas em alguns destes

pacotes. Os executáveis referentes a estes pacotes eram, então, processados pelo modelo de Regressão Logística, que predizia os seus respectivos rótulos.

Os resultados obtidos mostraram que o *Anova F-Test* contribuiu positivamente não somente para a precisão do sistema proposto, mas também para o *recall*, o *f-score* e a acurácia do mesmo. Utilizando o *2grams*, *3grams* e *4grams*, o modelo testado obteve acurácias de 97,7%, 91% e 85%, respectivamente.

Em Shah e Issac (2018), é apresentado um estudo comparativo sobre o desempenho do Snort e do Suricata. Na comparação realizada, foi observado que o Suricata é mais eficiente que o Snort no processamento de tráfegos de redes de alta velocidade, por ter uma taxa de queda de pacotes menor. Por outro lado, o Snort também teve seu destaque devido à taxa de acurácia de detecção superior a do Suricata. Tal característica motivou a escolha do Snort para a implementação de algoritmos de aprendizado de máquina para fins de redução de falso-positivos e falso-negativos no IDS.

Ao todo, cinco algoritmos foram considerados: SVM, Árvores de Decisão, Lógica Fuzzy, *BayesNet* e *NaiveBayes*. Aplicando-os em três bases de dados distintas (NSA, DARPA e NSL-KDD), constatou-se que o SVM foi o algoritmo de maior sucesso sob os critérios de taxa de detecção, acurácia de detecção e taxa de falso-positivos.

Em um experimento empírico, foi desenvolvido um *plugin* adaptativo com SVM para o Snort, visando a redução de falso-positivos e falso-negativos, conforme mencionado anteriormente. Em uma versão híbrida, foi possível melhorar o desempenho do *plugin* através da combinação do SVM com Lógica Fuzzy, o segundo algoritmo com o melhor desempenho observado. A solução híbrida de melhor resultado foi alcançada através da otimização do SVM com o Algoritmo do Vaga-lume (*Firefly Algorithm*).

Em Utimura e Costa (2018), foi realizado um estudo sobre a aplicação do MLP e um classificador baseado em *Optimum-Path Forest* (OPF) no Snort 3 (a versão mais recente da ferramenta). O objetivo do trabalho foi fornecer uma análise comparativa entre o desempenho destas técnicas em uma base de dados conhecida, a ISCX IDS 2012 e, também, verificar a aplicabilidade delas ao substituir, integralmente, o esquema de detecção convencional do IDS, realizando classificações de conexões em tempo real.

Para o treinamento de ambas as técnicas foi utilizado 70% da base de dados, sendo que foram propostas diversas configurações de treinamento/teste para analisar as acurácias de ambas as técnicas de acordo com o cenário trabalhado. O processo de experimentação sobre a base de dados revelou que, para todos os casos, o OPF apresentou melhores resultados (em termos de acurácia e consistência) comparado ao MLP, sendo que, quanto à aplicabilidade, ambas as técnicas funcionaram de acordo com o esperado realizando classificações em tempo real no Snort 3.

No entanto, como trabalho futuro, os autores propuseram o retreinamento da MLP

(para explorar novos parâmetros) e, também, a realização de uma comparação com a solução base do IDS, para explorar possíveis cenários onde a detecção por técnicas inteligentes foi essencial para conter ataques desconhecidos que não teriam sido detectados somente pelo Snort 3.

No estudo de Elshafie, Mahmoud e Ali (2019), é apresentada uma proposta de melhoria do Snort através do Algoritmo de Seleção Clonal (*Clonal Selection Algorithm* – CSA). O sistema proposto foi avaliado através da base de dados DARPA 1999 e comparado – em termos de *recall*, precisão e *f-score* – com a solução base do Snort e o Snort melhorado com NSA.

Seguindo a lógica por trás dos algoritmos de Sistema Imune Artificial (*Artificial Immune System* – AIS), o NSA foi utilizado para criar a primeira geração de detectores. Já para as gerações subsequentes, foi usado o CSA. Os resultados obtidos pelos autores mostram que, com a adição da primeira geração de detectores (NSA) ao Snort, é nítido o aumento do número de ataques detectados (como um reflexo do aumento do *recall*) em relação à solução base do IDS. No entanto, foi observado um aumento no número de alarmes falsos (como um reflexo da redução da precisão).

Com a adição da segunda, terceira e quarta geração de detectores (CSA) ao Snort, o aumento no *recall* torna-se ainda mais expressivo. Desta vez, foi possível reduzir o número de alarmes falsos a ponto de superar, neste quesito, a solução base do Snort (logo na segunda geração o sistema proposto mostrou-se, como um todo, mais eficaz que a solução base do Snort).

Para facilitar a visualização das contribuições feitas na área, o Quadro 2 apresenta um resumo dos trabalhos correlatos citados neste capítulo, dando destaque ao protocolo de comunicação utilizado (PT), à base de dados utilizada (BD), à(s) técnica(s) de inteligência computacional utilizada(s) (TC), ao fato dele ser ou não em tempo real (TR) e, finalmente, à versão do Snort utilizada (VR).

O indicador TR, em especial, foi suposto em alguns casos pelo fato dos autores não especificarem, de maneira clara, até que ponto o projeto em questão era em tempo real. Nos casos onde foi explicitamente dito que um *plugin* (ou pré-processador) foi desenvolvido para o Snort, assumiu-se que o mesmo era automaticamente capaz de execução em tempo real, mesmo que nos experimentos ele só tenha sido utilizado para processar uma base de dados. No entanto, nos casos onde não foi explicitamente especificado o desenvolvimento de um *plugin* ou, ainda, onde foi empregada uma metodologia à parte do Snort, por exemplo, para a criação de regras antes da execução propriamente dita do Snort, assumiu-se que o projeto não era capaz de executar ou exercer tal metodologia em tempo real.

Dado os trabalhos correlatos apresentados neste capítulo, é possível destacar alguns pontos passíveis de contribuição por esta dissertação de mestrado. Pelo Quadro 2, pode-se

perceber que a grande maioria dos trabalhos – inclusive os mais recentes – se baseia em bases de dados antigas que evidentemente não refletem mais os atuais cenários de redes de computadores e de ataques no geral. Além disso, nota-se que a versão predominantemente utilizada do Snort nos trabalhos correlatos é a 2, apesar do Snort 3 ter sido anunciado, em estágio alfa, em dezembro de 2014.

Tendo em vista esses pontos e, também, o fato de que muitos desses trabalhos não disponibilizaram publicamente o projeto para a comunidade científica e o público em geral, crê-se que, por meio desta dissertação, é possível contribuir para a área através do desenvolvimento de um *plugin* para o Snort 3, com técnicas de inteligência computacional treinadas sobre uma base de dados atualizada, para a classificação em tempo real de fluxos de tráfego de rede.

Quadro 2: Resumo dos Trabalhos Correlatos

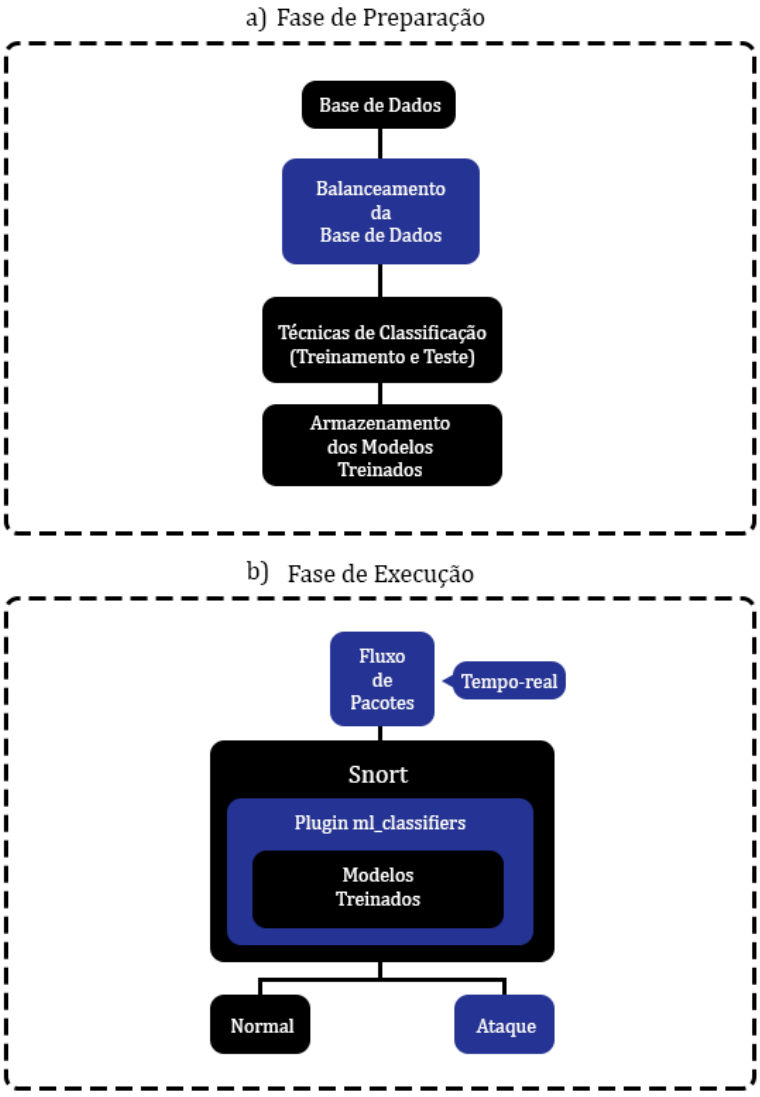
Referência	Propósito	PT	BD	TC	TR	VR
(STANIFORD; HOAGLAND; MCALERNEY, 2002; BILES, 2003)	Um pré-processador capaz de detectar o tráfego de rede que desvia do comportamento "normal" da rede monitorada pelo Snort	-	-	-	Sim	1
(SILVA et al., 2004)	Um método de detecção de intrusão em rede para identificar e classificar informações ilegítimas no *payload* de pacotes TCP/IP, baseado no conjunto de assinaturas do Snort que representa possíveis ataques à rede	TCP/ IP	-	*Hamming Net* e MAXNET	Não	2
(WEON et al., 2005)	Uma abordagem baseada em comportamento com um algoritmo de aprendizado supervisionado baseado em memória para a redução de alertas falsos	-	DARPA 1998	*Instance-based Learning*	Não	2
(SILVA, 2007)	Uma ferramenta para facilitar a implementação de diversos tipos de RNAs dentro do Snort	TCP, UDP, ICMP	-	-	Sim	2
(MAKANJU; ZINCIR-HEYWOOD; MILIOS, 2008)	Uma comparação, em termos de capacidade de detecção, entre o Snort-Wireless, baseado em assinatura, e uma solução baseada em GP	-	-	GP	Sim	2
(DING; XIAO; LIU, 2009)	Uma metodologia híbrida de detecção de intrusão que combina a abordagem baseada em assinatura com a abordagem baseada em anomalia	-	KDD Cup'99	Regras de Episódios Frequentes	Sim	2
(LI; LIU, 2010)	Uma análise sobre a aplicação de SVMs no Snort	-	-	SVMs	Sim	2
(FANG; LIU, 2011)	Um estudo sobre a integração de RNAs no Snort para que ele possa se adaptar às redes e detectar anomalias	-	-	*Elman Network*	Sim	2
(DEVI; NAGPAL, 2012; KUMAR; PUNIA, 2013)	Um estudo sobre como o GA poderia melhorar o desempenho do Snort	-	-	GA	Sim	2
(JONGSAWAT; DECHAROEN-CHITPONG, 2015)	Uma proposta para a utilização de uma abordagem bayesiana para detectar relações entre variáveis de uma base de dados de tráfego de rede	-	-	*Essential Graph Search*, *Bayesian Search*, Inferência Bayesiana	Não	2
(NAIK, 2015)	Um IDS baseado em inferência fuzzy, o FI-Snort	-	-	Inferência Fuzzy	Sim	2
(NAIK; DIAO; SHEN, 2016)	Sucessor do FI-Snort, com interpolação dinâmica de regras fuzzy	-	-	Inferência Fuzzy, Interpolação Dinâmica de Regras Fuzzy	Sim	2
(SEELAMMAL; DEVI, 2016)	Um IDS baseado em detecção por anomalia que utiliza a análise de *Big Data* do framework *Hadoop*	-	KDD Cup'99	Árvores de Decisão C4.5	Não	2
(MAHMOUD; ALI; ELSHAFIE, 2016)	Um NIDS com detecção por assinatura e anomalia baseado no Snort e no NSA	-	DARPA 1999	NSA	Sim	2
(KUMAR et al., 2017)	Um algoritmo de aprendizado de máquina usando *Anova F-Test* para combater *malwares* polimórficos	-	-	Regressão Logística e *Anova F-Test*	Não	2
(SHAH; ISSAC, 2018)	Um estudo comparativo sobre o desempenho do Snort e do Suricata. Contou com a implementação de algoritmos de aprendizado de máquina para fins de redução de falso-positivos e falso-negativos no Snort	TCP, UDP, ICMP	NSA Snort IDS Alert Logs, DARPA 2000, NSL-KDD	SVM, Árvores de Decisão, Lógica Fuzzy, *BayesNet*, *NaiveBayes*	Sim	2
(UTIMURA; COSTA, 2018)	Um estudo sobre a aplicação de um MLP e um classificador baseado em OPF no Snort	TCP, UDP, ICMP	ISCX2012	MLP, OPF	Sim	3
(ELSHAFIE; MAHMOUD; ALI, 2019)	Uma proposta de melhoria do Snort através do CSA	-	DARPA 1999	NSA, CSA	Sim	2

Fonte: Elaborado pelo autor.

5 Metodologia

Este capítulo descreve a metodologia utilizada para o desenvolvimento desta dissertação de mestrado, abordando os recursos, ferramentas e técnicas – incluindo as introduzidas na Seção 3.2 – necessárias para tal. Visto que o objetivo geral deste trabalho é a substituição do esquema de detecção por assinatura do Snort 3, para a classificação em tempo real de fluxos de tráfego de rede usando técnicas de inteligência computacional, e devido ao fato do Snort ser incapaz de interpretar bases de dados rotuladas por conta própria, foi preciso dividir o desenvolvimento do projeto em duas fases: de *preparação* e de *execução* (Figura 11).

Figura 11: Fases de Preparação e Execução do Projeto



Fonte: Elaborada pelo autor.

Na fase de preparação (Figura 11 (a)), que ocorre externamente ao Snort, o objetivo principal é realizar o pré-processamento da base de dados escolhida e, também, o treinamento das técnicas de classificação sobre essa base de dados. O pré-processamento da base de dados garante que a mesma não possua valores nulos, os dados estejam devidamente normalizados e as classes balanceadas (Seção 5.2). Já o treinamento das técnicas de classificação visa a criação dos seus respectivos modelos de aprendizado de máquina em um formato conhecido que permita, no *plugin ml_classifiers* (e consequentemente no Snort 3), a classificação dos fluxos de tráfego de rede em tempo real.

Já na fase de execução (Figura 11 (b)), o objetivo principal é o desenvolvimento do *plugin* propriamente dito que, inclusive, fará uso dos modelos das técnicas de classificação treinadas na fase de preparação. Se o Snort fosse capaz de interpretar bases de dados rotuladas, o treinamento das técnicas de classificação não precisaria ser externo à ferramenta, podendo ser realizado dentro do próprio *plugin*. Nas próximas seções, os elementos que compõem ambas as fases são descritos em maiores detalhes.

5.1 Base de Dados CICIDS2017

Como visto nos trabalhos correlatos, existe uma grande variedade de bases de dados disponíveis para a avaliação de IDSs. Em Sharafaldin, Lashkari e Ghorbani (2018), os autores apresentam uma análise sobre as principais bases de dados disponíveis publicamente: DARPA (1998-99), KDD'99 (1998-99), DEFCON (2000-2002), CAIDA (2002-16), LBNL (2004-05), CDX (2009), Kyoto (2009), Twente (2009), UMASS (2011), ISCX2012 (2012) e ADFA (2013). Dentre as múltiplas deficiências encontradas na grande maioria dessas bases de dados, com ênfase nas mais antigas, os autores destacaram como principais pontos críticos a dissimilaridade entre o tráfego simulado e o tráfego real, aquele observado nas atuais redes de computadores, e a carência no volume e na diversidade dos ataques presentes nessas bases de dados.

Dadas as particularidades de um projeto que envolve a utilização de técnicas supervisionadas de aprendizado de máquina, para que este tenha relevância no meio científico, além dos pontos mencionados anteriormente é de extrema importância que a base de dados escolhida seja rotulada e distinga adequadamente o tráfego benigno do tráfego malicioso, pois assim, as técnicas podem aprender a reconhecer tais padrões da forma mais clara e inequívoca possível.

Das bases de dados mencionadas no trabalho de Sharafaldin, Lashkari e Ghorbani (2018), uma das mais indicadas é a ISCX2012, por suprir grande parte das deficiências apontadas anteriormente. No entanto, recentemente, os autores da ISCX2012 publicaram uma nova base de dados: a CICIDS2017 (SHARAFALDIN; LASHKARI; GHORBANI, 2018). Trata-se de uma base de dados mais atualizada que, diferentemente da ISCX2012,

possui suporte ao protocolo HTTPS (*Hyper Text Transfer Protocol Secure*) e fornece – convenientemente em formato CSV – mais de 80 características[1] a respeito dos fluxos de tráfego de rede para o aprendizado de técnicas de inteligência computacional. A disponibilidade de tais características foi o principal fator que levou à escolha dessa base de dados para a execução deste projeto.

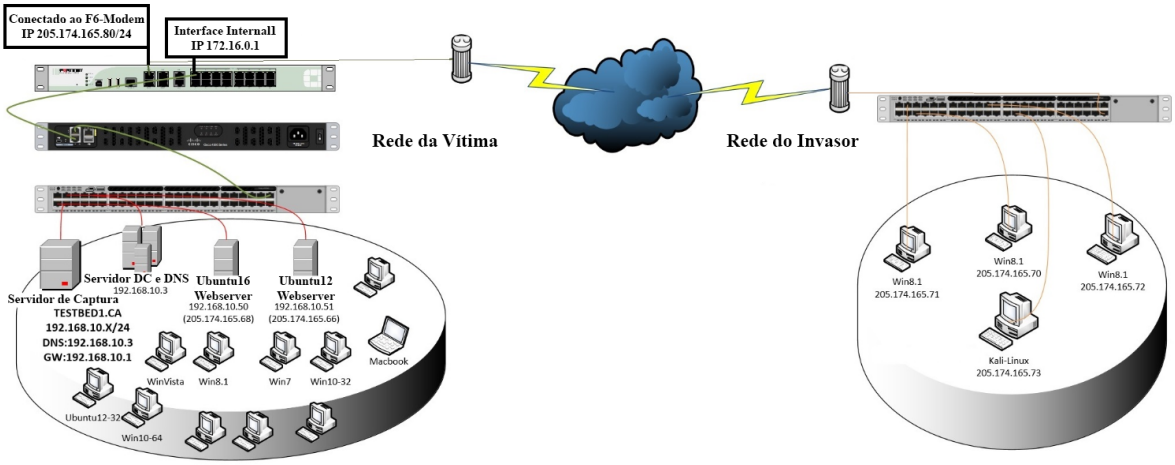
O CICIDS2017 conta com o tráfego completo capturado ao longo de cinco dias, totalizando 2.830.743 fluxos de tráfego de rede. Durante o processo de captura do tráfego, foi gerado o comportamento benigno correspondente à atividade de 25 usuários na rede, compreendendo protocolos como HTTP, HTTPS, FTP, SSH e outros protocolos ligados ao serviço de e-mail. Para o comportamento malicioso, foram utilizados seis perfis de ataques, descritos a seguir (SHARAFALDIN; LASHKARI; GHORBANI, 2018):

- **Força Bruta:** Usado não somente para a descoberta de senhas mas também de páginas e conteúdos escondidos em aplicações *web*;
- **Ataque *Heartbleed*:** Baseia-se em um *bug* na biblioteca de criptografia *OpenSSL*, uma implementação popular do protocolo TLS (*Transport Layer Security*). É normalmente explorada através do envio de uma requisição *heartbeat* malformada com o intuito de extrair informações sensíveis da resposta da vítima;
- **Botnet:** Um conjunto de dispositivos conectados através da Internet controlados por um atacante para desempenhar inúmeras tarefas, como roubar dados, enviar *spam*, dentre outras;
- **Negação de Serviço e Negação de Serviço Distribuída:** Consiste na inundação de pacotes voltados para uma única vítima, com o intuito de torná-la temporariamente indisponível, seja ela um *host* ou um recurso da rede. A versão distribuída normalmente conta com o uso de uma *botnet*;
- **Ataque *Web*:** Contempla ataques como *SQL Injection*, *Cross-Site Scripting (XSS)* e Força Bruta em HTTP. No SQL Injection, o atacante utiliza *strings* de comandos SQL para forçar respostas com informações do banco de dados. No XSS, o atacante utiliza técnicas para a injeção de *scripts*. Na Força Bruta em HTTP, o atacante usa a técnica com listas de senhas para tentar descobrir a senha do administrador;
- **Ataque de Infiltração:** A infiltração da rede, por dentro, normalmente ocorre através da exploração de um *software* vulnerável, *e.g.* Adobe Acrobat Reader. Após uma exploração bem-sucedida, é executado um *backdoor* no computador da vítima que, por sua vez, abre espaço para a condução de uma série de ataques à rede dela, como *portscans*, varredura de IPs, dentre outros.

[1] Obtidas através da ferramenta CICFlowMeter. Disponível em <http://www.netflowmeter.ca/>.

A infraestrutura do ambiente de testes utilizado para a geração da base de dados CICIDS2017 é mostrada na Figura 12. A distribuição dos fluxos de tráfego de rede, nas classes benigna e maliciosa, é de 80,30% e 19,70%, respectivamente. Portanto, é possível observar que há uma discrepância significativa entre o número de fluxos benignos e maliciosos, o que pode prejudicar no aprendizado das técnicas supervisionadas. Portanto, faz-se necessário o balanceamento da base de dados, discutido na seção a seguir.

Figura 12: Infraestrutura do Ambiente de Testes do CICIDS2017



Fonte: Adaptada de Sharafaldin, Lashkari e Ghorbani (2018).

5.2 Balanceamento da Base de Dados

Nesta condição onde há discrepância na distribuição das classes envolvidas, o principal problema que pode ser destacado é que os classificadores tendem a atingir uma boa acurácia na classe majoritária mas, em contrapartida, acurácias muito precárias nas classes minoritárias. Consequentemente, o desempenho do sistema como um todo é comprometido, porque há o enviesamento dos classificadores para a classe majoritária (GANGANWAR, 2012). Para combater o impacto do desbalanceamento entre classes no aprendizado de técnicas de inteligência computacional, dois métodos são comumente empregados (CHAWLA, 2009; YAP et al., 2014):

- ***Under-sampling*:** Instâncias da classe majoritária são retiradas aleatoriamente até que a distribuição se iguale a da classe minoritária;
- ***Over-sampling*:** Instâncias da classe minoritária são replicadas até que a distribuição se iguale a da classe majoritária.

No entanto, vale ressaltar que ambos os métodos aleatórios de *under-sampling* e *over-sampling* possuem imperfeições dignas de menção. Em métodos de remoção, como

o *under-sampling*, corre-se o risco de potencialmente eliminar importantes exemplos da classe majoritária, enquanto que, nos métodos de *over-sampling*, o risco maior é de ocorrer o super-treinamento do modelo.

Para não correr o risco de perder informações importantes a respeito dos fluxos benignos, foi aplicado o método de *over-sampling* sobre a base de dados CICIDS2017. Ademais, para evitar o super-treinamento dos modelos a serem aplicados posteriormente, foi utilizada uma técnica específica conhecida por *Synthetic Minority Oversampling TEchnique* (SMOTE) (CHAWLA et al., 2002), que realiza o *over-sampling* da classe minoritária mediante a criação de exemplos sintéticos, em oposição ao *over-sampling* com substituição. Nessa técnica, o *over-sampling* da classe minoritária se dá tomando cada amostra dessa classe e introduzindo exemplos sintéticos ao longo dos segmentos de reta que ligam elas a qualquer um dos (ou todos os) k-vizinhos mais próximos da classe minoritária (CHAWLA, 2009).

Neste trabalho, para a implementação do SMOTE e das demais técnicas de inteligência computacional foi utilizada a biblioteca *scikit-learn*[2] do *Python*[3]. A biblioteca é apresentada em maiores detalhes na próxima seção.

5.3 Biblioteca *scikit-learn*

Criado em 2007, o *scikit-learn* é desenvolvido por um time internacional composto por mais de uma dúzia de desenvolvedores, em sua maioria, pesquisadores de diversas áreas como: ciência da computação, neurociência, astrofísica, dentre outras (BUITINCK et al., 2013).

Trata-se de uma biblioteca escrita em *Python* que integra uma gama de algoritmos de aprendizado de máquina no estado-da-arte para a resolução de problemas supervisionados, de média escala, e problemas não-supervisionados (PEDREGOSA et al., 2011). A biblioteca foi projetada para se encaixar, harmonicamente, ao lado de um conjunto de pacotes numéricos e científicos centrados em torno das bibliotecas *NumPy* e *SciPy*, que mediante inúmeras funções, proveem estruturas de dados e métodos voltados para a manipulação e computação rápida de matrizes e de outras operações numéricas comuns (BUITINCK et al., 2013).

Tratando-se da utilidade do *scikit-learn* para este projeto, pode-se destacar a fácil utilização dos módulos referentes aos classificadores e, principalmente, a interface padronizada para o treinamento das técnicas (função *fit*), realização de predições (função *predict*) e obtenção de métricas (funções *accuracy_score*, *confusion_matrix* e *precision_recall_fscore_support*), que permite a criação de um único laço para a avaliação de todas as téc-

[2] Disponível em: <https://scikit-learn.org/stable/>
[3] Disponível em: <https://www.python.org/>

nicas sob as mesmas circunstâncias e métricas. Na próxima seção, é apresentado o *plugin* (módulo de extensão) desenvolvido para habilitar, no Snort 3, a detecção de intrusão com uma abordagem baseada em anomalia em tempo real usando as técnicas de inteligência computacional introduzidas na Seção 3.2.

5.4 Plugin *ml_classifiers*

Tendo em vista que o Snort (Snort 2 e Snort 3), até o presente momento, não possui um módulo nativo para realizar a detecção de intrusão com uma abordagem baseada em anomalia, foi preciso desenvolver um *plugin* para estender suas capacidades, chamado *ml_classifiers*[4]. Conforme consta na documentação do Snort 3 (SNORT, 2019), existem diversos tipos de *plugins* que atuam em diferentes partes do *pipeline* de processamento da ferramenta, como é mostrado na Tabela 2.

Tabela 2: Tipos de *plugins* no Snort 3

Plugin	**Função**
Codec	Para decodificação e codificação de pacotes
Inspector	Similares aos pré-processadores do Snort 2, são utilizados para a examinação de pacotes, normalização do tráfego de rede, dentre outras funções
IPSOption	Para detecção em regras do Snort
IPSAction	Para ações customizáveis
Logger	Para manipulação de eventos
Mpse	Para casamento rápido de padrões (*fast pattern matching*)
So	Para regras dinâmicas

Fonte: Adaptada de Snort (2019).

Uma vez que seriam carregados, para dentro do *plugin*, os modelos das técnicas de classificação previamente treinadas na fase de preparação, o *plugin* desenvolvido teria de lidar diretamente com os pacotes sendo processados pelo Snort 3, justamente para a criação do conceito de fluxos – até então inexistente, dentro da ferramenta, para a classificação mediante técnicas de inteligência computacional – e a extração de suas respectivas características. Dadas essas circunstâncias, optou-se pelo desenvolvimento de um *plugin* do tipo *Inspector*. Como é mostrado na Tabela 3, existem diferentes tipos de *Inspectors* que realizam funções específicas.

Com base na experiência de estudos anteriores feitos sobre o Snort 3 (UTIMURA; COSTA, 2018), foi desenvolvido um *Inspector* do tipo *IT_PROBE*. Tal tipo possui uma boa abrangência sobre os pacotes de rede e permite, com uma certa facilidade, a extração de todas as características necessárias para a criação e a manipulação dos fluxos de tráfego

[4] Disponível em: <https://github.com/lnutimura/ml_classifiers>

Tabela 3: Tipos de *Inspectors*

Inspectors	**Função**
IT_BINDER	Determina quais *inspectors* se aplicam a determinados fluxos
IT_WIZARD	Determina qual *inspector* de serviço usar se nenhum for vinculado explicitamente
IT_PACKET	Para processar todos os pacotes antes do processamento de sessão e serviço (*e.g.* normalização)
IT_NETWORK	Para processar pacotes sem serviço
IT_STREAM	Para rastreamento de fluxos, defragmentação de IP e remontagem TCP
IT_SERVICE	Para HTTP, FTP, TELNET, dentre outros
IT_PROBE	Para processar os pacotes após o processamento de todos os *inspectors* mencionados acima (*e.g.* port_scan)

Fonte: Adaptada de Snort (2019).

de rede. Nas próximas seções, é discutido em maiores detalhes a forma como esses fluxos são criados, gerenciados e, por fim, classificados dentro do *ml_classifiers*.

5.4.1 Gerenciamento de Fluxos de Tráfego de Rede

Para poder gerenciar os fluxos de tráfego de rede no *ml_classifiers* foi preciso entender, de antemão, como os fluxos eram criados e gerenciados dentro da ferramenta *CICFlowMeter*[5] (DRAPER-GIL et al., 2016; LASHKARI et al., 2017), a responsável pela geração dos arquivos de características dos fluxos da base de dados CICIDS2017. O motivo para isso é bastante simples. Visto que esses arquivos foram utilizados para o treinamento das técnicas de inteligência computacional na fase de preparação, faz-se necessário o uso da mesma metodologia de criação e gerenciamento de fluxos na fase de execução, pois caso contrário, existiriam incongruências na forma como os vetores de características dos fluxos seriam montados durante o treinamento, fora do Snort, e durante a execução, dentro do Snort, potencialmente resultando em classificações malfeitas.

Na metodologia utilizada para o desenvolvimento deste *plugin*, todos os fluxos de tráfego de rede são identificados por uma *string* única chamada *FlowID*, construída a partir da concatenação de algumas características do fluxo – protocolo, endereços da origem e do destino, portas da origem e do destino – o qual ela está associada. No Quadro 3, é mostrada a forma genérica de um *FlowID* e três exemplos para os protocolos TCP (*Transmission Control Protocol*), UDP (*User Datagram Protocol*) e ICMP (*Internet*

[5] *CICFlowMeter* é uma ferramenta voltada para a geração e a análise de tráfegos de rede bidirecionais para a detecção baseada em anomalia. Ela tem, como uma de suas principais funções, a extração de características de fluxos de tráfego de rede em formato CSV a partir de arquivos de captura de pacotes (PCAPs).

Control Message Protocol), respectivamente.

Quadro 3: Exemplos de *FlowIDs*

Forma Genérica
`Protocolo-Endereço_IP_Origem:Porta_Origem-Endereço_IP_Destino:Porta_Destino`
Protocolo TCP
`TCP-192.168.0.100:5050-192.168.0.105:80`
Protocolo UDP
`UDP-192.168.0.100:42000-192.168.0.105:6060`
Protocolo ICMP
`ICMP-192.168.0.100:0-192.168.0.105:0-256`

Fonte: Elaborado pelo autor.

Pelo fato do protocolo ICMP não fazer a utilização de portas, as mesmas são definidas como 0 no *FlowID*. No entanto, adicionalmente, é concatenada uma informação extra referente ao campo de identificação do cabeçalho ICMP. No exemplo anterior, essa informação corresponde ao valor 256.

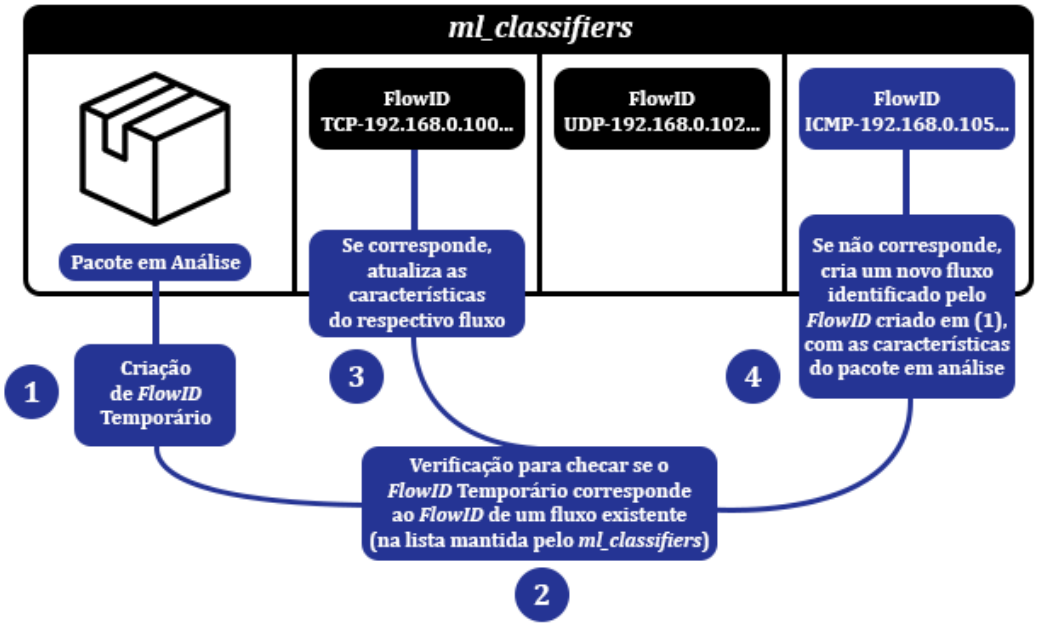
Sempre que um novo pacote é analisado pelo Snort 3 e, consequentemente, pelo *plugin*, é preciso verificar se ele pertence a um fluxo existente ou, então, se ele é o primeiro pacote de um novo fluxo que há de ser criado. O processo de verificação se dá através da criação de um novo *FlowID* (temporário), a partir das características do pacote em análise, e da utilização desse *FlowID* como uma chave de busca em uma lista de fluxos mantida pelo *plugin*. Caso não haja uma correspondência entre o *FlowID* recém-criado e qualquer outro *FlowID* presente na lista mantida pelo *plugin*, um novo fluxo é criado, inicializado com as características do pacote em análise, e anexado à lista de fluxos com o *FlowID* criado anteriormente (antes temporário, agora permanente). Por outro lado, se houver correspondência entre os *FlowIDs*, o fluxo identificado pelo respectivo *FlowID* é, então, atualizado com as características do pacote em análise. Esse processo é melhor representado na Figura 13.

5.4.2 Classificação de Fluxos de Tráfego de Rede

Tendo em vista que, até este ponto, os fluxos de tráfego de rede estão sendo criados e atualizados pelo *plugin*, para que ele possa realizar as classificações em tempo real é necessário o estabelecimento de uma política que selecione determinados fluxos para o processo de classificação pelas técnicas de inteligência computacional.

Para um protocolo orientado à conexão, como o TCP, uma primeira abordagem para essa política seria selecionar, para o processo de classificação, os fluxos que já atingiram o estado final deste protocolo, isto é, que já encerraram a conexão (mediante a troca de pacotes com as *flags* FIN e ACK). No entanto, em ambientes reais de redes de

Figura 13: Criação e Atualização de Fluxos de Tráfego de Rede



Fonte: Elaborada pelo autor.

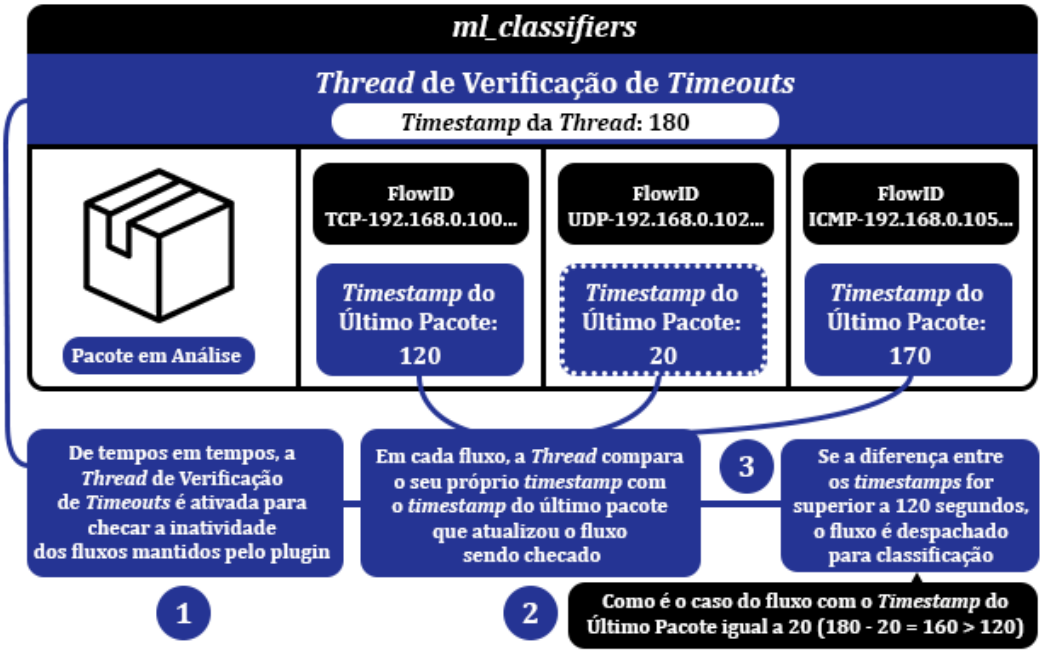
computadores, muitas conexões TCP mal chegam nesse estado, seja pela ocorrência de eventos inesperados que encerram a conexão prematuramente ou pelo simples prolongamento da conexão. Portanto, faz-se necessária a utilização de um mecanismo de *timeout* que possa despachar preemptivamente conexões TCP inativas para as técnicas de classificação. Para protocolos não orientados à conexão, como UDP e ICMP, a única alternativa é a utilização deste mesmo mecanismo de *timeout*, uma vez que não existe o conceito de estado de conexão nesses protocolos. Seguindo a metodologia utilizada na ferramenta *CICFlowMeter*, para todos os protocolos foi estabelecido um limite para o *timeout* de 120 segundos. Em Lashkari et al. (2017), esse limite produziu um dos melhores resultados em um problema de classificação envolvendo duas classes, com características obtidas pela mesma ferramenta (*CICFlowMeter*).

A verificação de *timeouts* é feita por meio de uma *thread*[6] que, de tempos em tempos, confere a atividade (ou a inatividade) dos fluxos de tráfego de rede presentes na lista mantida pelo *plugin*. Quando a diferença entre o *timestamp* (marca temporal) da *thread* – que representa o tempo atual de execução – e o *timestamp* do fluxo (ou melhor, do último pacote que atualizou ele) é maior que o limite pré-estabelecido para o *timeout*, o mesmo é retirado da lista mantida pelo *plugin* e despachado para as técnicas de classificação. Esse processo é melhor representado na Figura 14.

Nesta fase de classificação, todos os fluxos despachados têm suas características

[6] *Thread*, na computação, é um mecanismo utilizado para executar subtarefas de um programa concorrencialmente.

Figura 14: *Timeout* de Fluxos de Tráfego de Rede



Fonte: Elaborada pelo autor.

codificadas na forma de um vetor que, na prática, possui 77 características, com informações relacionadas ao fluxo – bidirecional – e, também, às direções *forward* (da origem para o destino) e *backward* (do destino para a origem). As características são descritas, na íntegra, nas Tabelas 4, 5 e 6.

Dependendo da configuração do *plugin*, esse vetor de características pode ser repassado para diferentes técnicas de classificação, como é discutido na próxima seção.

5.4.3 Configuração do Plugin

Através dos arquivos de configuração do Snort 3, é possível criar e disponibilizar o acesso a diversos parâmetros de inicialização nos *plugins*. Para o *ml_classifiers*, foi criada uma única chave que permite a escolha da técnica de classificação a ser utilizada. Desta forma, no arquivo de configuração (*snort.lua*), é possível escolher da seguinte forma:

```
ml_classifiers = {key = <dt|rf|ab|bnb|gnb|svc>}
```

onde `dt` é Árvore de Decisão, `rf` é Floresta Aleatória, `ab` é AdaBoost, `bnb` é Naive Bayes Bernoulli, `gnb` é Naive Bayes Gaussiano e `svc` é Máquina de Vetores de Suporte. Uma vez definido o classificador a ser utilizado, todas as classificações feitas por ele serão exibidas em tempo real no terminal executando o Snort 3 com o *ml_classifiers* carregado.

Tabela 4: Características dos fluxos de tráfego de rede (1 a 26)

#	**Característica**	**Descrição**
1	*DestinationPort*	Porta do destino
2	*FlowDuration*	Duração do fluxo (segundos)
3	*TotalFwdPackets*	Número total de pacotes enviados na direção *forward*
4	*TotalBwdPackets*	Número total de pacotes enviados na direção *backward*
5	*TotalLengthOfFwdPackets*	Comprimento total dos pacotes enviados na direção *forward*
6	*TotalLengthOfBwdPackets*	Comprimento total dos pacotes enviados na direção *backward*
7	*FwdPacketLengthMax*	Comprimento máximo dos pacotes enviados na direção *forward*
8	*FwdPacketLengthMin*	Comprimento mínimo dos pacotes enviados na direção *forward*
9	*FwdPacketLengthMean*	Comprimento médio dos pacotes enviados na direção *forward*
10	*FwdPacketLengthStd*	Desvio padrão dos comprimentos dos pacotes enviados na direção *forward*
11	*BwdPacketLengthMax*	Comprimento máximo dos pacotes enviados na direção *backward*
12	*BwdPacketLengthMin*	Comprimento mínimo dos pacotes enviados na direção *backward*
13	*BwdPacketLengthMean*	Comprimento médio dos pacotes enviados na direção *backward*
14	*BwdPacketLengthStd*	Desvio padrão dos comprimentos dos pacotes enviados na direção *backward*
15	*FlowBytesPerSec*	Quantidade de *bytes* enviados por segundo no fluxo
16	*FlowPacketsPerSec*	Quantidade de pacotes enviados por segundo no fluxo
17	*FlowIATMean*	Média de tempo de chegada entre pacotes no fluxo
18	*FlowIATStd*	Desvio padrão de tempo de chegada entre pacotes no fluxo
19	*FlowIATMax*	Máximo de tempo de chegada entre pacotes no fluxo
20	*FlowIATMin*	Mínimo de tempo de chegada entre pacotes no fluxo
21	*FwdIATTotal*	Total de tempo de chegada entre pacotes na direção *forward*
22	*FwdIATMean*	Média de tempo de chegada entre pacotes na direção *forward*
23	*FwdIATStd*	Desvio padrão de tempo de chegada entre pacotes na direção *forward*
24	*FwdIATMax*	Máximo de tempo de chegada entre pacotes na direção *forward*
25	*FwdIATMin*	Mínimo de tempo de chegada entre pacotes na direção *forward*
26	*BwdIATTotal*	Total de tempo de chegada entre pacotes na direção *backward*

Fonte: Elaborada pelo autor.

Tabela 5: Características dos fluxos de tráfego de rede (27 a 52)

#	**Característica**	**Descrição**
27	*BwdIATMean*	Média de tempo de chegada entre pacotes na direção *backward*
28	*BwdIATStd*	Desvio padrão de tempo de chegada entre pacotes na direção *backward*
29	*BwdIATMax*	Máximo de tempo de chegada entre pacotes na direção *backward*
30	*BwdIATMin*	Mínimo de tempo de chegada entre pacotes na direção *backward*
31	*FwdPSHFlags*	Total de *flags* PSH na direção *forward*
32	*BwdPSHFlags*	Total de *flags* PSH na direção *backward*
33	*FwdPSHFlags*	Total de *flags* URG na direção *forward*
34	*BwdPSHFlags*	Total de *flags* URG na direção *backward*
35	*FwdHeaderLength*	Comprimento total de cabeçalhos de pacotes na direção *forward*
36	*BwdHeaderLength*	Comprimento total de cabeçalhos de pacotes na direção *backward*
37	*FwdPacketsPerSec*	Quantidade de pacotes enviados por segundo na direção *forward*
38	*BwdPacketsPerSec*	Quantidade de pacotes enviados por segundo na direção *backward*
39	*PacketLengthMin*	Comprimento mínimo dos pacotes enviados no fluxo
40	*PacketLengthMax*	Comprimento máximo dos pacotes enviados no fluxo
41	*PacketLengthMean*	Comprimento médio dos pacotes enviados no fluxo
42	*PacketLengthStd*	Desvio padrão dos comprimentos dos pacotes enviados no fluxo
43	*PacketLengthVariance*	Variância dos comprimento dos pacotes enviados no fluxo
44	*FINFlagCount*	Quantidade de *flags* FIN no fluxo
45	*SYNFlagCount*	Quantidade de *flags* SYN no fluxo
46	*RSTFlagCount*	Quantidade de *flags* RST no fluxo
47	*PSHFlagCount*	Quantidade de *flags* PSH no fluxo
48	*ACKFlagCount*	Quantidade de *flags* ACK no fluxo
49	*URGFlagCount*	Quantidade de *flags* URG no fluxo
50	*CWEFlagCount*	Quantidade de *flags* CWE no fluxo
51	*ECEFlagCount*	Quantidade de *flags* ECE no fluxo
52	*DownUpRatio*	Razão entre o *download* e o *upload*

Fonte: Elaborada pelo autor.

Tabela 6: Características dos fluxos de tráfego de rede (53 a 77)

#	**Característica**	**Descrição**
53	*AveragePacketSize*	Tamanho médio dos pacotes
54	*AvgFwdSegmentSize*	Tamanho médio observado na direção *forward*
55	*AvgBwdSegmentSize*	Tamanho médio observado na direção *backward*
56	*FwdAvgBytesPerBulk*	Média da taxa de volume de *bytes* na direção *forward*
57	*FwdAvgPacketsPerBulk*	Média da taxa de volume de *pacotes* na direção *forward*
58	*FwdAvgBulkRate*	Média da taxa de volume na direção *forward*
59	*BwdAvgBytesPerBulk*	Média da taxa de volume de *bytes* na direção *backward*
60	*BwdAvgPacketsPerBulk*	Média da taxa de volume de *pacotes* na direção *backward*
61	*BwdAvgBulkRate*	Média da taxa de volume na direção *backward*
62	*SubflowFwdPackets*	Número médio de pacotes em um subfluxo na direção *forward*
63	*SubflowFwdBytes*	Número médio de *bytes* em um subfluxo na direção *forward*
64	*SubflowBwdPackets*	Número médio de pacotes em um subfluxo na direção *backward*
65	*SubflowBwdBytes*	Número médio de *bytes* em um subfluxo na direção *backward*
66	*InitWinBytesForward*	Total de *bytes* enviados na janela inicial na direção *forward*
67	*InitWinBytesBackward*	Total de *bytes* enviados na janela inicial na direção *backward*
68	*ActDataPktForward*	Número de pacotes com pelo menos 1 *byte* de *payload* TCP na direção *forward*
69	*MinSegSizeForward*	Tamanho mínimo de segmento observado na direção *forward*
70	*ActiveMean*	Tempo médio que um fluxo estava ativo antes de ficar inativo
71	*ActiveStd*	Desvio padrão do tempo que um fluxo estava ativo antes de ficar inativo
72	*ActiveMax*	Tempo máximo que um fluxo estava ativo antes de ficar inativo
73	*ActiveMin*	Tempo mínimo que um fluxo estava ativo antes de ficar inativo
74	*ActiveMean*	Tempo médio que um fluxo estava inativo antes de ficar ativo
75	*ActiveStd*	Desvio padrão do tempo que um fluxo estava inativo antes de ficar ativo
76	*ActiveMax*	Tempo máximo que um fluxo estava inativo antes de ficar ativo
77	*ActiveMin*	Tempo mínimo que um fluxo estava inativo antes de ficar ativo

Fonte: Elaborada pelo autor.

6 Experimentos e Resultados

Este capítulo descreve os experimentos realizados para a avaliação deste trabalho, bem como apresenta e discute os resultados obtidos. Os experimentos são divididos em duas etapas, onde, na primeira, é feita a avaliação das técnicas de classificação na fase de *preparação*, isto é, fora do Snort, e na segunda, é feita a avaliação das técnicas de classificação na fase de *execução*, dentro do Snort. Para a execução dos experimentos foi utilizado um computador com o sistema operacional Debian 9 (64 bits), processador Intel i7-4790 de 3.60GHz, placa de vídeo GeForce GTX 970 e 16GB de memória RAM DDR3 de 1333MHz.

Na primeira etapa, foi realizado o treinamento e o teste de todas as técnicas de classificação sobre a base de dados CICIDS2017, com uma proporção de 2/3 e 1/3 para os conjuntos de treinamento e teste, respectivamente. O conjunto de treinamento, originalmente composto por 1.894.676 fluxos de tráfego de rede, passou a ter 2.739.808 amostras após a aplicação do algoritmo de balanceamento SMOTE, com uma proporção aproximada de 50% para ambas as classes presentes na CICIDS2017. O conjunto de teste, composto por 933.200 fluxos de tráfego de rede, possui uma proporção aproximada de 80,32% amostras da classe benigna (normal) e 19,68% amostras da classe maliciosa (ataque).

A fim de garantir maior consistência nos resultados de cada técnica de classificação, todos os classificadores passaram por 15 rodadas de treinamento, sendo extraídas, ao término delas, as respectivas médias e desvios padrão dos resultados obtidos. Ademais, para todas as técnicas de classificação, os dados do conjunto de treinamento e de teste foram normalizados para o intervalo de $[0, 1]$, através do estimador *MinMaxScaler* da biblioteca *scikit-learn*.

Na segunda etapa, foi realizada uma comparação entre o Snort 3 modificado – com o *ml_classifiers* – e o Snort 3 não modificado em um ambiente de testes, para a identificação de possíveis cenários de ataque onde a solução modificada se sobressai em relação à solução base do Snort 3, e vice-versa. No *plugin ml_classifiers*, foram consideradas as técnicas de aprendizado de máquina que apresentaram os melhores resultados na primeira etapa dos experimentos, sendo elas: *AdaBoost*, Florestas Aleatórias, Árvore de Decisão e Máquina de Vetores de Suporte Linear.

Nas próximas seções, são apresentados os resultados obtidos na primeira e segunda etapa dos experimentos.

6.1 Primeira Etapa dos Experimentos

Nesta seção, são apresentados os resultados individuais do treinamento e teste de cada técnica de classificação sobre a base de dados CICIDS2017. Posteriormente, os resultados individuais são comparados entre si e são feitas algumas considerações a respeito dos desempenhos dos classificadores.

6.1.1 Naive Bayes

Para o *Naive Bayes* foram utilizadas as configurações padrão na biblioteca *scikit-learn* para o *BernoulliNB* e *GaussianNB*, pois não haviam muitos parâmetros para calibrar. A partir dessa configuração, foram obtidos os resultados mostrados nas Tabelas 7, 8, 9 e 10.

Tabela 7: Resultados com Naive Bayes Bernoulli

Treinamento	**Acurácia (%)**	**Precisão (%)**	**Recall (%)**	**F-Score (%)**	**Tempo de Trein. (s)**	**Tempo de Teste (s)**
1	66.5986	34.7809	80.1174	48.5047	2.0966	0.5234
2	66.5899	34.8490	80.1536	48.5776	2.1381	0.5740
3	66.5463	34.7798	80.0826	48.4973	2.4365	0.5292
4	66.6173	34.9024	80.0331	48.6072	2.2666	0.5278
5	66.6300	34.8026	80.0979	48.5222	2.4469	0.5214
6	66.5978	34.8969	80.1458	48.6226	2.2786	0.5249
7	66.6121	34.9095	80.1766	48.6406	1.9353	0.5234
8	66.6064	34.8534	80.1848	48.5875	2.2982	0.5705
9	66.6960	34.9338	80.1132	48.6524	2.4319	0.5228
10	66.5752	34.8029	79.9909	48.5029	2.2881	0.5196
11	66.5333	34.7443	80.0725	48.4609	2.4275	0.5212
12	66.6499	34.8480	80.1259	48.5715	2.2823	0.5225
13	66.6122	34.8236	80.0699	48.5375	2.0796	0.5262
14	66.6133	34.9282	80.0747	48.6399	2.1042	0.5247
15	66.5691	34.8195	80.1076	48.5405	2.0814	0.5245
Média	**66.6032**	**34.8450**	**80.1031**	**48.5644**	**2.2395**	**0.5304**
Desvio Padrão	**0.0399**	**0.0587**	**0.0522**	**0.0603**	**0.1595**	**0.0172**

Fonte: Elaborada pelo autor.

Tabela 8: Matriz de Confusão da Naive Bayes Bernoulli

	Positivo	**Negativo**	**Total**
Positivo	147140 (15.77%)	36538 (3.92%)	**183678**
Negativo	275439 (29.52%)	474083 (50.80%)	**749522**
Total	**422579**	**510621**	**933200**

Fonte: Elaborada pelo autor.

Tabela 9: Resultados com Naive Bayes Gaussiano

Treinamento	**Acurácia (%)**	**Precisão (%)**	**Recall (%)**	**F-Score (%)**	**Tempo de Trein. (s)**	**Tempo de Teste (s)**
1	47.3297	26.9616	98.4511	42.3306	2.4968	0.7586
2	45.7038	26.3778	98.1418	41.5800	2.5856	0.7920
3	43.0616	25.4557	98.2663	40.4365	3.1408	0.7508
4	44.2243	25.8880	98.1133	40.9666	2.9563	0.8062
5	44.0314	25.7794	98.4784	40.8620	3.1447	0.7536
6	50.7637	28.3625	98.0895	44.0019	3.0073	0.8050
7	43.5765	25.6903	98.3524	40.7392	2.4551	0.7455
8	44.2909	25.9128	98.4846	41.0300	3.0701	0.9878
9	48.6551	27.4808	98.0565	42.9302	3.0999	0.7558
10	44.4684	25.9308	98.1463	41.0231	3.0287	0.8023
11	42.4752	25.3288	98.9453	40.3329	3.0983	0.7615
12	47.1613	26.8786	98.1332	42.1989	2.9440	0.8077
13	44.3652	25.8946	98.2513	40.9869	2.3951	0.7494
14	45.7155	26.4546	98.2849	41.6882	2.4254	0.7560
15	45.4732	26.2884	98.1342	41.4682	2.4351	0.8076
Média	**45.4197**	**26.3123**	**98.2886**	**41.5050**	**2.8189**	**0.7893**
Desvio Padrão	**2.2385**	**0.8180**	**0.2318**	**1.0000**	**0.3066**	**0.0604**

Fonte: Elaborada pelo autor.

Tabela 10: Matriz de Confusão da Naive Bayes Gaussiano

	Positivo	**Negativo**	**Total**
Positivo	180251 (19.32%)	3427 (0.37%)	**183678**
Negativo	505417 (54.16%)	244105 (26.16%)	**749522**
Total	**685668**	**247532**	**933200**

Fonte: Elaborada pelo autor.

A partir das tabelas é possível perceber que ambas as técnicas não apresentaram um desempenho satisfatório na classificação de fluxos de tráfego de rede. Tomando como exemplo os resultados obtidos com o *Naive Bayes Bernoulli*, é possível observar que apesar dessa técnica ter classificado corretamente uma quantidade considerável de ataques (80,10% de *recall* médio), ela, por outro lado, apresentou uma taxa expressiva de falso-positivos (29,52%), isto é, classificou erroneamente como pertencente à classe de ataque uma quantidade significativa de fluxos normais, resultando em uma precisão média de 34,84%.

Nos resultados obtidos com o *Naive Bayes Gaussiano*, é possível observar que a técnica apresentou um comportamento similar ao do *Naive Bayes Bernoulli*. Apesar dela ter classificado corretamente praticamente todos os fluxos de ataque do conjunto de teste (98,28% *recall* médio), ela, por outro lado, classificou erroneamente muitos fluxos normais como pertencentes à classe de ataque, resultando em uma taxa de falso-positivos

de 54,16% e uma precisão média de 26,31%.

6.1.2 Árvore de Decisão

Para a Árvore de Decisão (*DecisionTreeClassifier*), foi aplicada a técnica *GridSearchCV* da biblioteca *scikit-learn* para determinar a profundidade máxima da árvore (*max_depth*), o número mínimo de amostras necessárias para dividir um nó interno (*min_samples_split*), o número mínimo de amostras necessárias para estar em um nó folha (*min_samples_leaf*) e o número máximo de características para levar em consideração na busca pela melhor divisão (*max_features*), conforme mostrado na Tabela 11. A partir dessa configuração, foram obtidos os resultados apresentados nas Tabelas 12 e 13.

Tabela 11: Hiperparâmetros da Árvore de Decisão

Hiperparâmetro	**Valores Explorados**	**Valor Ótimo**
max_depth	{Nenhum, 3}	Nenhum
min_samples_split	$\lceil \{0.1, 1.0, 10.0\} \cdot n\_samples \rceil$	0.1
min_samples_leaf	$\lceil \{0.1, 0.2, 0.3, 0.4, 0.5\} \cdot n\_samples \rceil$	0.1
max_features	$\lfloor \{0.1, 0.5, 1.0\} \cdot n\_samples \rfloor$	1.0
Escore Médio da Validação Cruzada do Melhor Estimador		94.9753%
Duração do *GridSearchCV*		57 min 25 s

Tabela 12: Resultados com Árvore de Decisão

Treinamento	**Acurácia (%)**	**Precisão (%)**	**Recall (%)**	**F-Score (%)**	**Tempo de Trein. (s)**	**Tempo de Teste (s)**
1	92.6161	73.0617	98.8343	84.0159	44.5650	0.1240
2	92.6049	73.0790	98.8559	84.0352	38.6748	0.1236
3	92.6073	73.0727	98.8324	84.0225	35.0792	0.1233
4	92.6011	73.1026	98.8668	84.0547	29.9511	0.1213
5	92.5874	72.9822	98.8370	83.9643	29.4476	0.1223
6	92.6194	73.1614	98.8296	84.0801	30.1560	0.1235
7	92.6398	73.2121	98.8403	84.1175	29.8973	0.1191
8	92.6343	73.1381	98.8902	84.0866	30.6659	0.1221
9	92.6271	73.1485	98.8498	84.0789	30.5102	0.1222
10	92.6165	73.0982	98.8619	84.0500	30.5807	0.1211
11	92.6126	73.0603	98.8559	84.0228	29.6311	0.1201
12	92.6451	73.1546	98.8573	84.0857	29.6091	0.1205
13	92.5968	73.0354	98.8453	84.0025	29.6960	0.1207
14	92.6301	73.2025	98.8618	84.1189	29.4131	0.1204
15	92.6364	73.1695	98.8273	84.0846	29.9301	0.1235
Média	**92.6183**	**73.1119**	**98.8497**	**84.0547**	**31.8538**	**0.1219**
Desvio Padrão	**0.0170**	**0.0647**	**0.0171**	**0.0439**	**4.3405**	**0.0015**

Fonte: Elaborada pelo autor.

Tabela 13: Matriz de Confusão da Árvore de Decisão

	Positivo	**Negativo**	**Total**
Positivo	181524 (19.45%)	2154 (0.23%)	**183678**
Negativo	66563 (7.13%)	682959 (73.18%)	**749522**
Total	**248087**	**685113**	**933200**

Fonte: Elaborada pelo autor.

De modo geral, nota-se que a técnica apresentou um certo balanceamento no que diz respeito à classificação dos fluxos de tráfego de rede de ambas as classes envolvidas. Observa-se que tanto os fluxos normais quanto os fluxos de ataques foram, em sua grande maioria, classificados corretamente, apresentando não só uma acurácia média alta (92,62%) como, também, valores razoáveis de precisão (73,11%) e *recall* (98,85%) médios, indicando que as taxas de falso-positivos (7,13%) e falso-negativos (0,23%) foram moderadas.

6.1.3 Florestas Aleatórias

Para as Florestas Aleatórias (*RandomForestClassifier*), foi aplicada a técnica *GridSearchCV* da biblioteca *scikit-learn* para determinar o número de estimadores (*n_estimators*), a profundidade máxima de cada árvore (*max_depth*), o número mínimo de amostras necessárias para dividir um nó interno (*min_samples_split*), o número mínimo de amostras necessárias para estar em um nó folha (*min_samples_leaf*) e o número máximo de características para levar em consideração na busca pela melhor divisão (*max_features*), conforme mostrado na Tabela 14. A partir dessa configuração, foram obtidos os resultados apresentados nas Tabelas 15 e 16.

Tabela 14: Hiperparâmetros das Florestas Aleatórias

Hiperparâmetro	**Valores Explorados**	**Valor Ótimo**
n_estimators	{10, 50, 100}	100
max_depth	{Nenhum, 3}	Nenhum
min_samples_split	$\lceil \{0.1, 0.2, \ldots, 0.9, 1.0\} \cdot n\_samples \rceil$	0.2
min_samples_leaf	$\lceil \{0.1, 0.2, 0.3, 0.4, 0.5\} \cdot n\_samples \rceil$	0.1
max_features	$\lfloor \{0.1, 0.5, 1.0\} \cdot n\_samples \rfloor$	0.5
Escore Médio da Validação Cruzada do Melhor Estimador		94.9814%
Duração do *GridSearchCV*		28 h 25 min 2 s

Tabela 15: Resultados obtidos com Florestas Aleatórias

Treinamento	**Acurácia (%)**	**Precisão (%)**	**Recall (%)**	**F-Score (%)**	**Tempo de Trein. (s)**	**Tempo de Teste (s)**
1	93.8698	77.7107	96.4400	86.0682	1458.4946	4.4243
2	92.8972	74.5290	97.1132	84.3353	1260.5495	4.4235
3	93.8668	77.7427	96.4215	86.0805	861.8964	4.3614
4	93.8649	77.7858	96.4379	86.1134	730.3742	4.3553
5	93.8300	77.6469	96.2992	85.9730	738.3054	4.3428
6	92.6431	74.0720	96.4594	83.7962	732.3004	4.3443
7	92.9833	74.8743	96.9509	84.4944	728.4759	4.3438
8	93.9599	78.0312	96.4654	86.2746	754.4641	4.3645
9	93.9151	77.9080	96.4552	86.1952	733.7280	4.3406
10	92.8681	74.5407	96.8291	84.2354	733.6966	4.3379
11	92.7460	74.2664	96.5321	83.9480	905.3532	4.3913
12	93.8820	77.7772	96.4235	86.1024	897.5509	4.3789
13	92.5010	73.3844	97.0709	83.5819	728.3351	4.3462
14	93.9103	77.9404	96.4552	86.2150	731.6788	4.3447
15	93.8877	77.8475	96.3686	86.1235	721.5761	4.3623
Média	**93.4417**	**76.4038**	**96.5815**	**85.3025**	**847.7853**	**4.3641**
Desvio Padrão	**0.5758**	**1.8253**	**0.2671**	**1.0683**	**220.5650**	**0.0285**

Fonte: Elaborada pelo autor.

Tabela 16: Matriz de Confusão das Florestas Aleatórias

	Positivo	**Negativo**	**Total**
Positivo	177008 (18.97%)	6670 (0.71%)	**183678**
Negativo	50370 (5.40%)	699152 (74.92%)	**749522**
Total	**227378**	**705822**	**933200**

Fonte: Elaborada pelo autor.

A partir das tabelas é possível perceber que a técnica foi satisfatoriamente eficaz na classificação dos fluxos de tráfego de rede, apresentando valores significativos de acurácia média (93,44%), precisão média (76,40%), *recall* médio (96,58%) e, consequentemente, de *f-score* médio (85,30%), decorrentes das taxas relativamente baixas de falso-positivos (5,40%) e falso-negativos (0,71%) obtidas a partir das classificações realizadas.

6.1.4 Máquina de Vetores de Suporte

Para a Máquina de Vetores de Suporte (*LinearSVC*), foi utilizada a seguinte configuração de parâmetros:

- **Função *kernel*:** Linear;
- **Máximo de Iterações:** 1000;

- **Custo:** 100;

Em especial, para o parâmetro de Custo (C), foi utilizada a função *GridSearchCV* da biblioteca *scikit-learn* para encontrar o valor de melhor desempenho na classificação dos fluxos de tráfego de rede, conforme mostrado na Tabela 17. A partir dessa configuração, foram obtidos os resultados apresentados nas Tabelas 18 e 19.

Tabela 17: Hiperparâmetro da Máquina de Vetores de Suporte

Hiperparâmetro	**Valores Explorados**	**Valor Ótimo**
C	{0.1, 1.0, 10.0, 100.0}	100.0
Escore Médio da Validação Cruzada do Melhor Estimador		92.9445%
Duração do *GridSearchCV*		31 min 1 s

Tabela 18: Resultados obtidos com Máquina de Vetores de Suporte

Treinamento	**Acurácia (%)**	**Precisão (%)**	**Recall (%)**	**F-Score (%)**	**Tempo de Trein. (s)**	**Tempo de Teste (s)**
1	91.1275	70.0394	95.7867	80.9142	411.7389	0.0814
2	91.2281	70.3733	95.7612	81.1274	548.8340	0.0772
3	91.2074	70.2988	95.7487	81.0734	699.7636	0.0819
4	91.1533	70.2427	95.6865	81.0138	369.1318	0.0770
5	91.1506	70.1168	95.7284	80.9450	569.7235	0.0767
6	91.1683	70.2556	95.7590	81.0484	296.0983	0.0771
7	91.1982	70.3117	95.8260	81.1097	357.6652	0.0764
8	91.1606	70.1682	95.8185	81.0115	316.2488	0.0769
9	91.1660	70.2209	95.7506	81.0223	350.9271	0.0770
10	91.1831	70.2608	95.7024	81.0315	515.4708	0.0766
11	91.1399	70.1062	95.7294	80.9383	493.2333	0.0844
12	91.1396	70.0937	95.7905	80.9518	465.1130	0.0816
13	91.1562	70.1567	95.7609	80.9832	444.4584	0.0774
14	91.1531	70.2456	95.7458	81.0370	388.6276	0.0774
15	91.1572	70.1822	95.7562	80.9985	345.0413	0.0822
Média	**91.1660**	**70.2049**	**95.7567**	**81.0137**	**438.1384**	**0.0788**
Desvio Padrão	**0.0274**	**0.0922**	**0.0381**	**0.0616**	**111.3202**	**0.0027**

Fonte: Elaborada pelo autor.

Tabela 19: Matriz de Confusão da Máquina de Vetores de Suporte

	Positivo	**Negativo**	**Total**
Positivo	175883 (18.85%)	7795 (0.84%)	**183678**
Negativo	74726 (8.01%)	674796 (72.31%)	**749522**
Total	**250609**	**682591**	**933200**

Fonte: Elaborada pelo autor.

De modo geral, a técnica teve um desempenho satisfatório na classificação dos fluxos de tráfego de rede, classificando corretamente a grande maioria dos fluxos em ambas as classes envolvidas, apresentando uma acurácia média alta (91,17%) e, também, valores significativos de precisão média (70,20%) e *recall* médio (95,76%). Nota-se que a técnica cometeu mais erros na classificação de fluxos normais do que na classificação de fluxos de ataques, apresentando uma taxa de falso-positivos de 8,01% e uma taxa de falso-negativos de 0,84%.

6.1.5 AdaBoost

Para o *AdaBoost* (*AdaBoostClassifier*), foi utilizada a seguinte configuração:

1. **Estimador base:** Árvore de Decisão;
2. **Algoritmo:** SAMME.R (HASTIE et al., 2009).

Foi utilizada a função *GridSearchCV* da biblioteca *scikit-learn* para determinar o número de estimadores (*n_estimators*) e a taxa de aprendizado (*learning_rate*), conforme mostrado na Tabela 20. A partir dessa configuração, foram obtidos os resultados apresentados nas Tabelas 21 e 22.

Tabela 20: Hiperparâmetros do *AdaBoost*

Hiperparâmetro	**Valores Explorados**	**Valor Ótimo**
n_estimators	{10, 50, 100}	100
learning_rate	{0.01, 0.05, 0.1, 1.0}	1.0
Escore Médio da Validação Cruzada do Melhor Estimador		99.0632%
Duração do *GridSearchCV*		3 h 14 min 42 s

Tabela 21: Matriz de Confusão das *AdaBoost*

	Positivo	**Negativo**	**Total**
Positivo	182206 (19.52%)	1472 (0.16%)	**183678**
Negativo	8236 (0.88%)	741286 (79.43%)	**749522**
Total	**190442**	**742758**	**933200**

Fonte: Elaborada pelo autor.

Tabela 22: Resultados obtidos com *AdaBoost*

Treinamento	**Acurácia (%)**	**Precisão (%)**	**Recall (%)**	**F-Score (%)**	**Tempo de Trein. (s)**	**Tempo de Teste (s)**
1	98.7439	94.6347	99.2283	96.8771	2170.4828	8.3686
2	98.7516	94.8228	99.0682	96.8990	2021.4550	8.3796
3	98.8849	95.2691	99.2596	97.2234	1183.9737	8.3268
4	98.6826	94.5534	99.0254	96.7377	1351.4776	8.2612
5	98.6144	93.9208	99.3757	96.5713	1039.4386	8.2614
6	98.7891	94.8293	99.2730	97.0003	1033.9353	8.2771
7	98.7955	94.8528	99.2794	97.0156	1030.2138	8.2520
8	98.7158	94.4075	99.3602	96.8205	1057.7547	8.2765
9	98.8953	95.3310	99.2519	97.2520	1044.9052	8.2720
10	98.8245	94.9178	99.3454	97.0812	1037.5660	8.2536
11	98.7923	94.9991	99.0691	96.9914	1372.2913	8.2634
12	98.7519	94.9195	98.9461	96.8910	1352.4101	8.2864
13	98.9249	95.4501	99.2643	97.3199	1029.6709	8.2680
14	98.8422	95.3366	98.9769	97.1227	1189.5021	8.2845
15	98.9597	95.6753	99.1986	97.4051	1037.7236	8.3600
Média	**98.7979**	**94.9280**	**99.1948**	**97.0139**	**1263.5200**	**8.2927**
Desvio Padrão	**0.0937**	**0.4488**	**0.1412**	**0.2269**	**362.5832**	**0.0436**

Fonte: Elaborada pelo autor.

A partir das tabelas é possível observar que a técnica é extremamente eficaz e balanceada no que diz respeito à classificação dos fluxos de tráfego de rede, apresentando uma acurácia (98,80%), precisão (94,93%), *recall* (99,19%) e *f-score* (97,01%) médios satisfatórios. De modo geral, houveram poucos erros de classificação, vide as taxas de falso-positivos (0,88%) e falso-negativos (0,16%).

6.1.6 Comparação das Técnicas de Classificação

Dados os resultados obtidos individualmente em cada técnica de classificação (Seções 6.1.1, 6.1.2, 6.1.3, 6.1.4 e 6.1.5), é possível compará-las em termos das métricas apresentadas anteriormente, como é mostrado na Tabela 23 e nas Figuras 15, 16 e 17. A Tabela 23 destaca em negrito as duas melhores técnicas de classificação, em termos de *f-score*, levando em consideração o Teste de Wilcoxon (WILCOXON, 1946) com significância de $0,05$.

De modo geral, levando em consideração a acurácia, precisão, *recall* e *f-score*, percebe-se que as técnicas mais consistentes na primeira etapa dos experimentos foram as Florestas Aleatórias e o *AdaBoost*, enquanto que, em contrapartida, as técnicas menos consistentes foram as duas variações do *Naive Bayes*: Bernoulli e Gaussiano (Tabela 23 e Figura 15).

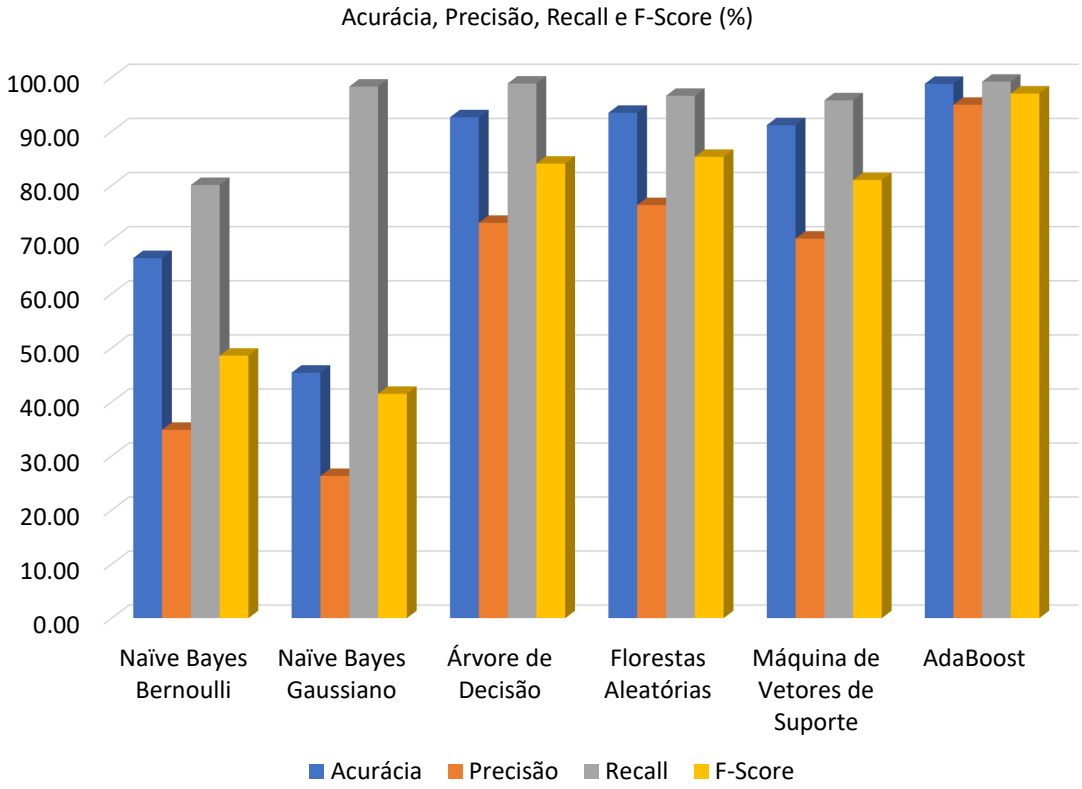
Tratando-se do *AdaBoost*, é nítido o fato que, dentre as demais técnicas, ela é a

Tabela 23: Comparativo de Todas as Técnicas

Técnica	Acurácia (%)	Precisão (%)	Recall (%)	F-Score (%)	Tempo de Trein. (s)	Tempo de Teste (s)
NB Bernoulli	66.60 ± 0.04	34.84 ± 0.06	80.10 ± 0.05	48.56 ± 0.06	2.24 ± 0.16	0.53 ± 0.02
NB Gaussiano	45.42 ± 2.24	26.31 ± 0.82	98.29 ± 0.23	41.51 ± 1.00	2.82 ± 0.31	0.79 ± 0.06
AD	92.62 ± 0.02	73.11 ± 0.06	98.85 ± 0.02	84.05 ± 0.04	31.85 ± 4.34	0.12 ± 0.00
FA	$\mathbf{93.44 \pm 0.58}$	$\mathbf{76.40 \pm 1.83}$	$\mathbf{96.58 \pm 0.27}$	$\mathbf{85.30 \pm 1.07}$	$\mathbf{847.79 \pm 220.56}$	$\mathbf{4.36 \pm 0.03}$
SVM	91.17 ± 0.03	70.20 ± 0.02	95.76 ± 0.04	81.01 ± 0.06	438.14 ± 111.32	0.08 ± 0.00
AB	$\mathbf{98.80 \pm 0.09}$	$\mathbf{94.93 \pm 0.45}$	$\mathbf{99.19 \pm 0.14}$	$\mathbf{97.01 \pm 0.23}$	$\mathbf{1263.52 \pm 362.58}$	$\mathbf{8.29 \pm 0.04}$

Fonte: Elaborada pelo autor.

Figura 15: Resultados Gerais de Acurácia, Precisão, *Recall* e *F-Score*

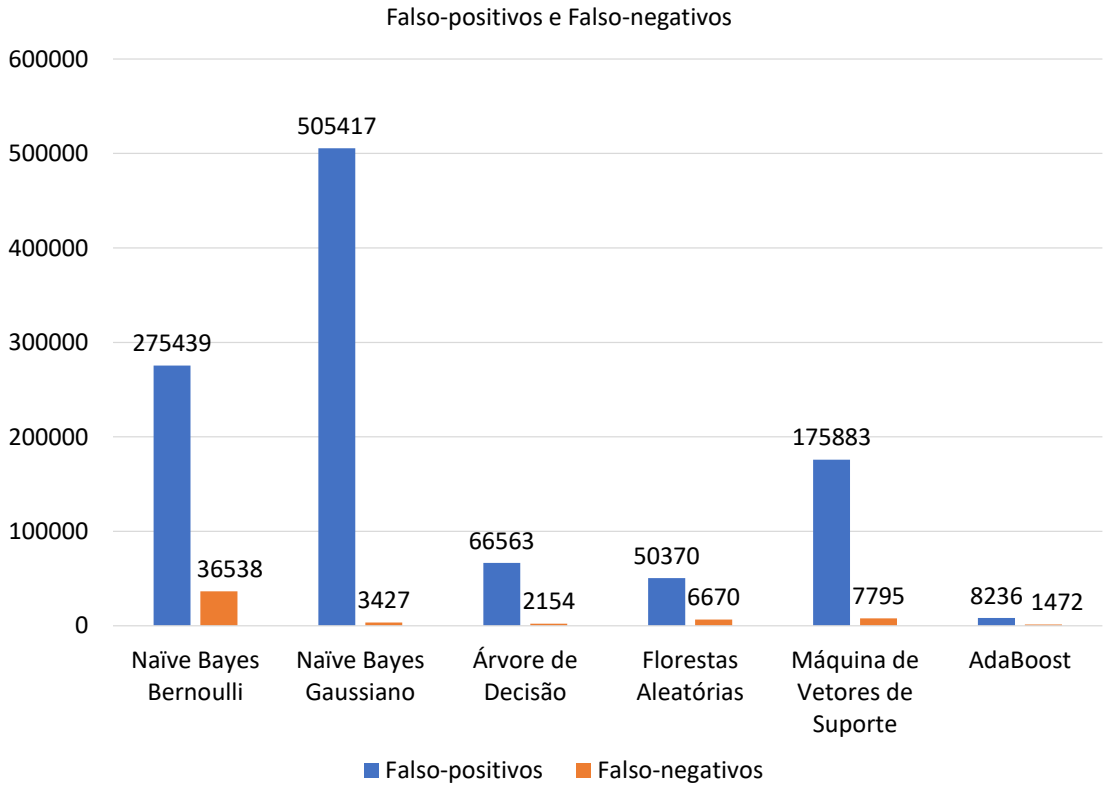


Fonte: Elaborada pelo autor.

mais indicada para a classificação de fluxos de tráfego de rede em um ambiente fechado, vide não apenas a acurácia e o *f-score* obtido mas, também, as ínfimas taxas de falso-positivos e falso-negativos observadas (Figura 16). Do mesmo modo destaca-se a técnica Florestas Aleatórias que, assim como as *AdaBoost*, cometeu poucos erros na classificação dos fluxos de tráfego de rede e, sobretudo, apresentou um alto *f-score* demonstrando eficácia na classificação correta dos fluxos de ataques.

Já o *Naive Bayes Bernoulli* e *Naive Bayes Gaussiano* não se destacaram na classificação dos fluxos de tráfego de rede, sendo as duas técnicas que mais erraram ao longo dos testes conduzidos na primeira etapa dos experimentos. O *Naive Bayes Gaussiano*, por exemplo, apesar de ter classificado corretamente a grande maioria dos fluxos de ataques

Figura 16: Resultados Gerais de Falso-Positivos e Falso-Negativos



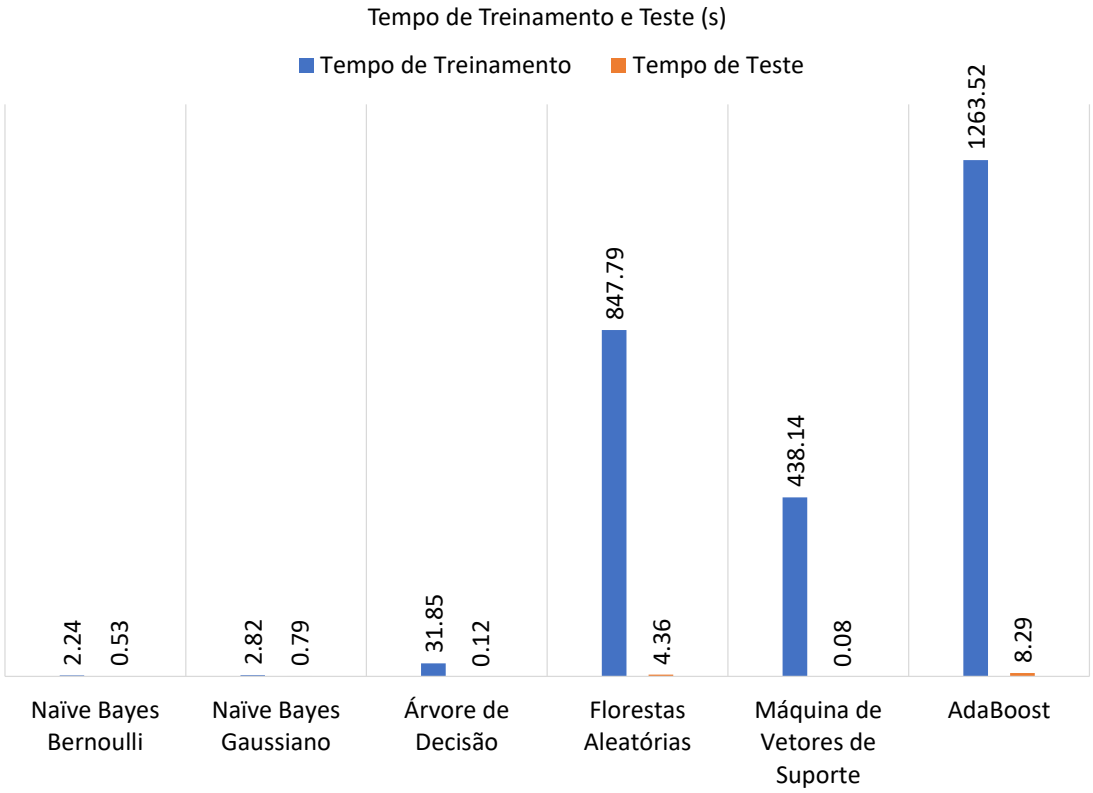
Fonte: Elaborada pelo autor.

presentes no conjunto de teste, foi a técnica que mais errou na classificação dos fluxos normais, apresentando um total de falso-positivos de 505.417, isto é, aproximadamente 67,43% dos fluxos normais foram classificados erroneamente (Figura 16). O *Naive Bayes Bernoulli*, apesar de ter cometido menos erros na classificação dos fluxos normais se comparado ao *Naive Bayes Gaussiano*, ainda assim foi a segunda técnica com o maior número de falso-positivos, 275.439, indicando que aproximadamente 36,74% dos fluxos normais foram classificados erroneamente (Figura 16).

Dado o número de características e o fato de que determinados subconjuntos de características são mais relevantes para a detecção de determinados ataques (SHARAFALDIN; LASHKARI; GHORBANI, 2018), a inconsistência do *Naive Bayes* pode estar relacionada à própria suposição da técnica de que todas as características utilizadas são independentes entre si, algo que, no mundo real, dificilmente é verdade. Por outro lado, tratando-se da consistência das Florestas Aleatórias e do *AdaBoost*, uma possível justificativa para o equilíbrio dessas técnicas pode estar relacionada ao fato de ambas utilizarem um comitê de estimadores para obter resultados mais precisos e, também, de serem utilizados critérios para moderar as impurezas e entropias dos respectivos modelos, algo que não existe no *Naive Bayes*.

Em um espectro mais ponderado das métricas utilizadas, encontram-se as técnicas

Figura 17: Resultados Gerais de Tempo de Treinamento e Teste



Fonte: Elaborada pelo autor.

Árvore de Decisão e Máquina de Vetores de Suporte. Como esperado, o desempenho da Árvore de Decisão foi inferior ao das técnicas que a utilizaram na forma de um comitê para obter melhores resultados (Florestas Aleatórias e *AdaBoost*). No entanto, percebe-se que o desempenho individual da Árvore de Decisão foi superior ao desempenho da Máquina de Vetores de Suporte, apresentando não apenas uma acurácia superior como, também, um *f-score* maior, atestando que a técnica é mais eficaz na classificação de fluxos de ataques que, no contexto de detecção de intrusão, representam a classe de maior interesse (ou peso).

Em termos dos tempos de treinamento e de teste de cada técnica de classificação nesta primeira etapa dos experimentos, o Naive Bayes Bernoulli, o Naive Bayes Gaussiano e a Árvore de Decisão foram as técnicas com o menor tempo de treinamento em relação às demais (Figura 17), enquanto o *AdaBoost* e as Florestas Aleatórias foram, disparadamente, as técnicas com o maior tempo de treinamento. Em relação aos tempos de teste, novamente, as técnicas com os maiores tempos de teste observados foram o *AdaBoost* e as Florestas Aleatórias, sendo que, de modo geral, não houve diferenças significativas entre as técnicas de classificação neste quesito, levando em consideração a quantidade de amostras testadas e, também, o fato desta etapa ser realizada em um ambiente *offline* de detecção de intrusão. Em um ambiente de detecção de intrusão em tempo real, a tendência é que

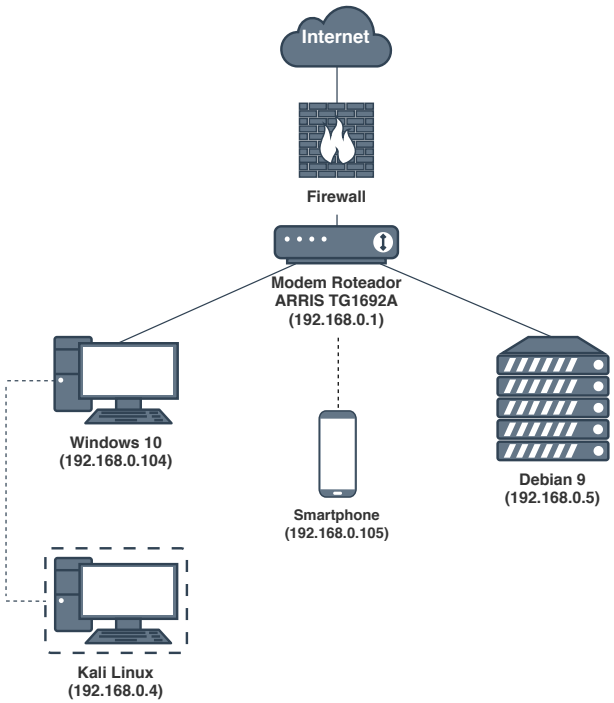
a questão do tempo de teste se torne ainda mais irrelevante, visto que a quantidade de fluxos de tráfego de rede classificados ao longo do tempo é substancialmente menor.

Na segunda etapa dos experimentos, o objetivo é comparar as técnicas que se destacaram na primeira etapa – como o *AdaBoost* e as Florestas Aleatórias – com as técnicas que apresentaram um desempenho mediano – como a Árvore de Decisão e a Máquina de Vetores de Suporte Linear – com o intuito de verificar seus respectivos desempenhos em um ambiente de detecção de intrusão em tempo real e, principalmente, pôr à prova a capacidade de generalização dos respectivos modelos de aprendizado de máquina.

6.2 Segunda Etapa dos Experimentos

Nesta seção, são apresentados os resultados obtidos com o Snort 3 em sua versão modificada – com as técnicas *AdaBoost*, Florestas Aleatórias, Árvore de Decisão e Máquina de Vetores de Suporte Linear – e não modificada, isto é, com a abordagem de detecção por assinatura, na tarefa de classificação em tempo real de fluxos de tráfego de rede em um ambiente de testes. Nesta etapa, foi utilizada uma infraestrutura de rede diferente daquela que originou a base de dados CICIDS2017, justamente para avaliar a capacidade de generalização dos modelos de aprendizado de máquina. A infraestrutura utilizada na segunda etapa dos experimentos é mostrada na Figura 18.

Figura 18: Infraestrutura do Ambiente de Testes da Segunda Etapa dos Experimentos



Fonte: Elaborada pelo autor.

Neste ambiente de testes, o tráfego benigno é, em sua maior parte, gerado a partir

do computador de mesa (Windows 10), do *smartphone* (iOS 13) e do servidor (Debian 9), em razão da utilização de diferentes aplicações, serviços e protocolos comumente observados em redes de computadores, conforme mostrado na Tabela 24.

Tabela 24: Aplicações, Serviços e Protocolos do Ambiente de Testes

Aplicação/Serviço	**Protocolo**
Google Chrome (Navegador *web*)	HTTP e HTTPS
Spotify (*Streaming* de áudio)	TCP e UDP
Skype e *Discord* (VoIP)	TCP e UDP
Cliente de E-mail (*Gmail* - iOS 13)	IMAP, POP3 e SMTP
Cliente e Servidor SSH	SSH
Cliente e Servidor DHCP	DHCP
Sincronização de Hórario com Servidor	NTP

Fonte: Elaborada pelo autor.

O tráfego referente aos ataques realizados no ambiente de testes, por sua vez, é gerado por uma máquina virtualizada com o sistema operacional Kali Linux[1], responsável pela a execução de ataques de DoS (Slowloris, SlowHTTPTest e Hulk), de *Port Scan* (Nmap) e de Força Bruta SSH (Patator) no servidor *web*. Comparada à infraestrutura do ambiente de testes que originou a base de dados CICIDS2017, a infraestrutura desta etapa dos experimentos (Figura 18) é significantemente menor, em termos do número de *hosts*, e mais simples, em termos do grau de complexidade da rede. Por esses motivos, não foram replicados os ataques da base de dados que utilizavam *botnets* – portanto, ataques distribuídos – e, também, aqueles cujas configurações e procedimentos não foram disponibilizados pelos autores da CICIDS2017. Ainda assim, DoS, *Port Scan* e Força Bruta SSH, são tópicos frequentemente explorados na área de Segurança de Redes de Computadores (KUERBAN et al., 2016; PATEL; SONKER, 2016; ZHANG et al., 2017; SADASIVAM; HOTA; ANAND, 2018; FAUST, 2018; ALMSEIDIN; AL-KASASSBEH; KOVACS, 2019).

O servidor – principal alvo da máquina virtual com Kali Linux – além de servir páginas *web* com o Apache, também é responsável por executar o Snort 3 e, consequentemente, por monitorar todos os fluxos de tráfego de rede durante o andamento da segunda etapa dos experimentos. A seguir, são apresentados em maiores detalhes os ataques conduzidos nesta segunda etapa dos experimentos e discutidos os resultados obtidos com o Snort 3, tanto em sua versão modificada – com as técnicas de aprendizado de máquina – quanto em sua versão não modificada, isto é, usando a abordagem de detecção baseada

[1] Kali Linux é uma distribuição Linux, derivada do Debian, projetada para análise forense digital e testes de penetração.

em assinatura[2].

6.2.1 DoS Slowloris

O Slowloris é um programa de ataque de DoS que permite que um invasor sobrecarregue um servidor de destino através da abertura e do mantimento de diversas conexões HTTP simultâneas entre o invasor e o alvo (CLOUDFLARE, 2017). Para a execução deste experimento, foi utilizada uma versão da ferramenta disponibilizada em um repositório público do GitHub[3]. A linha de comando utilizada para a execução do ataque direcionado ao servidor (192.168.0.5) e à porta 80 foi:

```
$ ./slowloris.py 192.168.0.5 -p 80 -v
```

Onde `-p` determina a porta alvo e `-v` o modo verboso. Os resultados acerca da detecção do ataque com o Snort 3 foram:

Tabela 25: Resultados da Detecção do DoS Slowloris

Abordagem	**Detectado**	**Tempo Médio de Classificação***
Snort 3 + AB	Sim	0.0135
Snort 3 + FA	Sim	0.0047
Snort 3 + AD	Sim	0.0001
Snort 3 + SVM	Sim	0.0001
Snort 3 Não Modificado	Não	-

* Refere-se ao tempo médio que as técnicas de aprendizado de máquina levaram para classificar todos os fluxos de tráfego de rede referentes ao ataque.

Fonte: Elaborada pelo autor.

Nos resultados apresentados na Tabela 25, considerou-se que o ataque foi *detectado* com sucesso se ao menos um fluxo de tráfego de rede referente ao ataque foi classificado corretamente – no caso das técnicas de aprendizado de máquina – ou se ao menos um alerta foi gerado pelo Snort 3 não modificado durante a execução do ataque (180 segundos).

Dados os resultados obtidos acerca das detecções realizadas, observa-se que enquanto o Snort 3 não modificado foi incapaz de detectar o DoS Slowloris, todas as técnicas de aprendizado de máquina obtiveram êxito na detecção do ataque, com destaque para a Árvore de Decisão e a Máquina de Vetores de Suporte Linear por serem as técnicas mais rápidas na classificação dos fluxos de tráfego de rede, o que é condizente com os resultados apresentados na primeira etapa dos experimentos.

[2] Na versão não modificada do Snort 3, foi utilizado o arquivo de regras da comunidade. Disponível em: <https://www.snort.org/downloads/community/snort3-community-rules.tar.gz>. Acesso em: 9 jan. 2020.

[3] Disponível em:<https://github.com/gkbrk/slowloris>. Acesso em: 10 jan. 2020.

6.2.2 DoS SlowHTTPTest

O SlowHTTPTest é uma ferramenta altamente configurável que simula ataques de DoS em nível de aplicação através do prolongamento de conexões HTTP de diferentes formas. Para a execução deste experimento, foi utilizada uma versão da ferramenta disponibilizada em um repositório público do GitHub[4]. A linha de comando utilizada para a execução do ataque direcionado ao servidor (192.168.0.5) e à porta 80 foi:

```
$ slowhttptest -c 50 -H -g -o slowhttp -i 10 -r 200 -t GET \
-u http://192.168.0.5:80 -x 24
```

Onde `-c` determina o número de conexões, `-H` o modo de teste *slow headers*, `-g` e `-o` as opções de relatório, `-i` o intervalo entre os dados de seguimento (em segundos), `-r` o número de conexões por segundo, `-t` o verbo para usar nas requisições, `-u` a URL do alvo e `-x` o comprimento máximo de cada par aleatório de nome/valor dos dados de seguimento por *tick*. Os resultados acerca da detecção do ataque com o Snort 3 foram:

Tabela 26: Resultados da Detecção do DoS SlowHTTPTest

Abordagem	**Detectado**	**Tempo Médio de Classificação***
Snort 3 + AB	Sim	0.0106
Snort 3 + FA	Sim	0.0050
Snort 3 + AD	Sim	0.0001
Snort 3 + SVM	Sim	0.0001
Snort 3 Não Modificado	Não	-

* Refere-se ao tempo médio que as técnicas de aprendizado de máquina levaram para classificar todos os fluxos de tráfego de rede referentes ao ataque.

Fonte: Elaborada pelo autor.

Dados os resultados obtidos acerca das detecções realizadas, conforme apresentados na Tabela 26, observa-se que eles foram bem próximos àqueles obtidos com a execução do DoS Slowloris, justamente por se tratar de técnicas conceitualmente semelhantes. Novamente, o Snort 3 não modificado foi incapaz de detectar o ataque, enquanto as demais técnicas de aprendizado de máquina obtiveram êxito em detectá-lo.

6.2.3 DoS Hulk

O Hulk é uma ferramenta desenvolvida para fins de pesquisa que conduz ataques de DoS. Ela foi projetada para gerar volumes de tráfego único e ofuscado em um servidor *web*, ignorando os mecanismos de *cache* e, assim, atingindo o conjunto de recursos diretos do servidor (SHTEIMAN, 2012). Para a execução deste experimento, foi utilizada uma

[4] Disponível em:<https://github.com/shekyan/slowhttptest>. Acesso em: 10 jan. 2020.

versão da ferramenta disponibilizada em um repositório público do GitHub[5]. A linha de comando utilizada para a execução do ataque direcionado ao servidor (192.168.0.5) e à porta 80 foi:

```
$ python hulk.py http://192.168.0.5:80
```

Os resultados acerca da detecção do ataque com o Snort 3 foram:

Tabela 27: Resultados da Detecção do DoS Hulk

Abordagem	**Detectado**	**Tempo Médio de Classificação***
Snort 3 + AB	Sim	0.0098
Snort 3 + FA	Sim	0.0051
Snort 3 + AD	Sim	0.0001
Snort 3 + SVM	Sim	0.0001
Snort 3 Não Modificado	Sim	-

* Refere-se ao tempo médio que as técnicas de aprendizado de máquina levaram para classificar todos os fluxos de tráfego de rede referentes ao ataque.

Fonte: Elaborada pelo autor.

Dados os resultados obtidos acerca das detecções realizadas, conforme apresentados na Tabela 27, observa-se, novamente, que eles foram bem próximos àqueles obtidos com a execução do DoS Slowloris e do DoS SlowHTTPTest. No entanto, desta vez, além das técnicas de aprendizado de máquina, o Snort 3 não modificado também foi capaz de detectar o DoS Hulk. De modo geral, nota-se uma tendência no que diz respeito à detecção de ataques de negação de serviço, visto que todas as técnicas de aprendizado de máquina foram capazes de detectá-los independentemente da abordagem empregada pelo ataque.

6.2.4 Port Scan

Port Scanning é uma das técnicas mais populares utilizadas por invasores para descobrir serviços passíveis de exploração em um *host* e/ou uma rede. Com o *port scanning*, um invasor pode descobrir diversas informações a respeito de um sistema alvo, como: quais serviços está executando, quais usuários possuem esses serviços, se autenticações anônimas são suportadas, se determinados serviços de rede exigem autenticação, dentre outras (CHRISTOPHER, 2001). Para a execução deste experimento, foi utilizado o utilitário gratuito e de código aberto Nmap[6], projetado para descoberta de rede e auditoria de segurança. A linha de comando utilizada para a execução do ataque direcionado ao servidor (192.168.0.5) foi:

[5] Disponível em:<https://github.com/grafov/hulk>. Acesso em: 10 jan. 2020.
[6] Disponível em:<https://nmap.org/>. Acesso em: 10 jan. 2020.

```
$ nmap -sS -sV -Pn -O 192.168.0.5
```

Onde `-sS` determina a técnica de escaneamento (TCP SYN), `-sV` indica que portas abertas serão examinadas para adquirir informações de serviços/versões, `-Pn` indica que todos os *hosts* serão tratados como online e `-O` habilita a detecção do sistema operacional. Os resultados acerca da detecção do ataque com o Snort 3 foram:

Tabela 28: Resultados da Detecção do Port Scan

Abordagem	**Detectado**	**Tempo Médio de Classificação***
Snort 3 + AB	Sim	0.0097
Snort 3 + FA	Sim	0.0049
Snort 3 + AD	Sim	0.0001
Snort 3 + SVM	Sim	0.0001
Snort 3 Não Modificado	Sim	-

* Refere-se ao tempo médio que as técnicas de aprendizado de máquina levaram para classificar todos os fluxos de tráfego de rede referentes ao ataque.

Fonte: Elaborada pelo autor.

Dados os resultados obtidos acerca das detecções realizadas, conforme apresentados na Tabela 28, observa-se que todas as técnicas de aprendizado de máquina e o próprio Snort 3 não modificado foram capazes de detectar o ataque de *Port Scan*. Por ser um dos ataques com o maior número de instâncias na base de dados CICIDS2017 e, também, por ser um dos ataques mais clássicos utilizados por invasores, é de se esperar que as técnicas de aprendizado de máquina identifiquem com certa facilidade o padrão de *port scanning* e que o Snort 3, por si só, tenha a assinatura desse ataque e de possíveis variações em seu arquivo de regras da comunidade.

6.2.5 Força Bruta SSH

A força bruta – ou busca exaustiva – é uma das técnicas de *hacking* mais antigas da história, além de ser considerada um dos ataques automatizados mais simples que existem, por exigir pouco conhecimento e intervenção mínima do invasor. O ataque consiste em realizar inúmeras tentativas de autenticação utilizando uma base de dados (ou um dicionário) de nomes de usuários e senhas até que haja um casamento entre eles (VANNEY, 2019). Para a execução deste experimento, foi utilizada a ferramenta Patator[7], um *brute-forcer* multiuso, com *design* modular[8] e utilização flexível para a execução de um ataque de Força Bruta SSH. A linha de comando utilizada para a execução do ataque direcionado ao servidor (192.168.0.5) e à porta 22 foi:

[7] Disponível em:<https://github.com/lanjelot/patator>. Acesso em: 10 jan. 2020.

[8] O Patator possui mais de 20 módulos para a execução de diferentes ataques de força bruta, por exemplo, para os protocolos FTP, SSH, SMTP, POP, dentre outros.

```
$ ./patator.py ssh_login host=192.168.0.5 user=lnutimura \
password=FILE0 0=/root/Desktop/wordlist-master/passlist.txt \
-x ignore:mesg='Authentication failed.'
```

Onde `ssh_login` especifica o módulo a ser utilizado, `host` a máquina alvo, `user` os nomes de usuários a serem testados (neste caso, apenas um, `lnutimura`), `password` as senhas a serem testadas (neste caso, várias, especificadas no arquivo `passlist.txt`) e `-x ignore` um parâmetro para ignorar as mensagens de falha de autenticação. Os resultados acerca da detecção do ataque com o Snort 3 foram:

Tabela 29: Resultados da Detecção da Força Bruta SSH

Abordagem	**Detectado**	**Tempo Médio de Classificação***
Snort 3 + AB	Não	-
Snort 3 + FA	Não	-
Snort 3 + AD	Sim	0.0001
Snort 3 + SVM	Sim	0.0001
Snort 3 Não Modificado	Não	-

* Refere-se ao tempo médio que as técnicas de aprendizado de máquina levaram para classificar todos os fluxos de tráfego de rede referentes ao ataque.

Fonte: Elaborada pelo autor.

Dados os resultados obtidos acerca das detecções realizadas, conforme mostrado na Tabela 29, observa-se que, desta vez, apenas duas técnicas de aprendizado de máquina – Árvore de Decisão e Máquina de Vetores de Suporte Linear – foram capazes de detectar o ataque de Força Bruta SSH. Nota-se, ainda, que as duas técnicas que obtiveram êxito na detecção deste ataque não foram aquelas que apresentaram os melhores resultados na primeira etapa dos experimentos com a base de dados CICIDS2017, mostrando que, diante de fluxos de tráfego de rede desconhecidos, a Árvore de Decisão e a Máquina de Vetores de Suporte Linear possuem uma maior capacidade de generalização quando comparadas ao *AdaBoost* e às Florestas Aleatórias.

Entende-se, portanto, que se o objetivo é detectar ataques de DoS, *Port Scan* e de Força Bruta SSH, é muito mais eficiente e eficaz optar pela utilização do Snort 3 aliado às técnicas Árvore de Decisão e Máquina de Vetores de Suporte Linear para tal fim, considerando-se que as técnicas não apenas detectaram todos os ataques, como também, apresentaram um tempo médio de classificação significantemente inferior ao do *AdaBoost* e das Florestas Aleatórias, o que pode ser um fator decisivo para a escolha de uma técnica de aprendizado de máquina para um ambiente de rede de alta velocidade.

Ressalta-se, ainda, que um ponto forte da abordagem de detecção de intrusão baseada em anomalia é que, diferentemente da abordagem baseada em assinatura (ou regras),

não foi preciso configurar o Snort 3 com o *ml_classifiers* para monitorar – especificamente – a infraestrutura da Figura 18. Isto é, não foram configurados parâmetros ligados à infraestrutura deste ambiente de testes no Snort 3 e no *plugin* para que as técnicas de aprendizado de máquina detectassem os ataques desta etapa dos experimentos, sendo necessário apenas o treinamentos delas sobre a base de dados CICIDS2017.

Por fim, apesar do Snort 3 não modificado ser incapaz de detectar ataques como o DoS Slowloris, o DoS SlowHTTPTest e a Força Bruta SSH – com o arquivo de regras da comunidade – a abordagem de detecção por assinatura (ou regra) ainda é de extrema importância para a manutenção e a proteção de redes de computadores. Na grande maioria dos casos onde IDSs baseados em assinaturas são configurados corretamente, levando em consideração as particularidades da rede que planeja-se monitorar, eles costumam ser menos suscetíveis à alertas falsos do que as abordagens baseadas em anomalias. No entanto, nada impede que IDSs híbridos sejam implantados em projetos de redes de computadores. Em uma configuração híbrida do Snort 3, por exemplo, a abordagem baseada em assinatura poderia ser utilizada para a contenção de ameaças estáticas, de fácil mitigação, enquanto que, por outro lado, a abordagem baseada em anomalia poderia se concentrar na contenção de ameaças mais complexas, dinâmicas, onde, neste cenário, um modelo de classificação poderia ser mais apropriado, como observado nos experimentos conduzidos ao longo desta etapa.

7 Considerações Finais

Neste estágio evolutivo em que a tecnologia se encontra atualmente, a capacidade de um dispositivo de se conectar à Internet é vista como um pré-requisito indispensável para que ele seja considerado relevante no contexto das novidades tecnológicas. Como consequência desse fenômeno – que diz muito a respeito da forma como a tecnologia tange o cotidiano das pessoas – percebe-se que as redes de computadores encontram-se em um estado de constante renovação, não apenas para sustentar a adesão de novos dispositivos mas, também, para garantir a segurança e a integridade de todos os componentes que compõem suas arquiteturas.

O crescimento das redes de computadores, no entanto, caminha em direção contrária à segurança delas, uma vez que, à medida que elas tornam-se mais heterogêneas, aumentam-se as suas superfícies de ataque, isto é, as suas amplitudes de penetrabilidade. Na área de detecção de intrusão, os ataques conhecidos são normalmente contidos – com certa eficácia – através de abordagens baseadas em assinatura, empregadas na grande maioria dos Sistemas de Detecção de Intrusão em Rede, como o Snort. Por outro lado, os ataques desconhecidos – cada vez mais relevantes visto a frequência com que eles surgem com o passar dos anos – são contidos através de abordagens baseadas em anomalia, normalmente fazendo-se o uso de técnicas de aprendizado de máquina.

Tendo em vista o desafio que é lidar com ataques desconhecidos e, consequentemente, adaptar ferramentas já existentes para a contenção destes ataques, este trabalho propôs a substituição do esquema de detecção baseado em assinatura da última versão do Snort (Snort 3) por um esquema de detecção baseado em anomalia, para habilitar a classificação em tempo real de fluxos de tráfego de rede mediante o uso de técnicas de aprendizado de máquina, que até o presente momento, é inexistente no IDS. Para tanto, foi desenvolvido um *plugin* para a tarefa em questão, chamado *ml_classifiers*.

Para a avaliação da metodologia proposta por este trabalho, os experimentos foram conduzidos em duas etapas. Na primeira etapa, foi realizado o treinamento das técnicas *Naive Bayes Bernoulli*, *Naive Bayes Gaussiano*, Árvore de Decisão, Florestas Aleatórias, Máquina de Vetores de Suporte Linear e *AdaBoost* sobre a base de dados CICIDS2017. Uma vez treinadas, a ideia é que, posteriormente, as melhores técnicas fossem incorporadas ao *ml_classifiers* a fim de utilizá-las para a classificação em tempo real de fluxos de tráfego de rede em um ambiente de testes. A partir dos resultados parciais obtidos (Tabela 23 e Figura 15) concluiu-se, em um primeiro momento, que as melhores técnicas para a classificação de fluxos de tráfego de rede eram o *AdaBoost* e as Florestas Aleatórias, com acurácias médias de 98,80% e 93,44% e *f-scores* médios de 97,01% e 85,30%,

respectivamente.

Na segunda etapa, foi realizado um comparativo entre o Snort 3 modificado – com as técnicas *AdaBoost*, Florestas Aleatórias, Árvore de Decisão e Máquina de Vetores de Suporte Linear – e o Snort 3 não modificado – com o arquivo de regras da comunidade – para a detecção dos ataques DoS Slowloris, DoS SlowHTTPTest, DoS Hulk, *Port Scan* e Força Bruta SSH. A partir dos resultados parciais obtidos, concluiu-se que, de modo geral, o Snort 3 com o *ml_classifiers* foi muito mais eficiente e eficaz que o Snort 3 não modificado na detecção destes ataques, com destaque à Árvore de Decisão e à Máquina de Vetores de Suporte Linear por serem as únicas técnicas capazes de detectar todos os ataques satisfatoriamente, demonstrando um maior poder de generalização e aplicabilidade em comparação ao *AdaBoost* e às Florestas Aleatórias.

Entende-se, portanto, que é oportuno o estudo de metodologias e de abordagens que empregam técnicas de aprendizado de máquina para a detecção de ataques conhecidos e desconhecidos em redes de computadores. Além do mais, conforme as considerações finais da Seção 6.2.5, percebe-se que o desenvolvimento de IDSs híbridos torna-se, cada vez mais, uma alternativa sólida para a detecção de ataques em redes de computadores, visto que a abordagem baseada em assinatura pode ser empregada para a detecção de ataques de fácil mitigação e, por outro lado, a abordagem baseada em anomalia pode ser empregada para a detecção de ataques mais complexos, dinâmicos, mediante a utilização de modelos de classificação, tal como foram propostos e empregados nesta dissertação.

7.1 Trabalhos Futuros

Como trabalhos futuros, planeja-se, em um primeiro momento, explorar metodologias para a otimização dos hiperparâmetros das técnicas de aprendizado de máquina. Além deste processo de otimização, planeja-se, também, incluir outros ataques para a avaliação em tempo real do *ml_classifiers*, bem como introduzir uma etapa adicional para a seleção de características de fluxos de tráfego de rede.

Por fim, outro ponto a ser considerado é a reestruturação do código do *ml_classifiers* de modo que seja possível combiná-lo, de forma mais harmônica, ao próprio esquema de detecção baseado em assinatura do Snort 3, tornando-o verdadeiramente híbrido.

Referências

ALBIN, E.; ROWE, N. C. A realistic experimental comparison of the suricata and snort intrusion-detection systems. In: IEEE. *Advanced Information Networking and Applications Workshops (WAINA), 2012 26th International Conference on*. [S.l.], 2012. p. 122–127.

ALHEETI, K. M. Intrusion detection system and artificial intelligent. In: *Intrusion Detection Systems*. [S.l.]: InTech, 2011.

ALJAWARNEH, S.; ALDWAIRI, M.; YASSEIN, M. B. Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model. *Journal of Computational Science*, v. 25, p. 152 – 160, 2018. ISSN 1877-7503. Disponível em: <http://www.sciencedirect.com/science/article/pii/S1877750316305099>.

ALMSEIDIN, M.; AL-KASASSBEH, M.; KOVACS, S. Detecting slow port scan using fuzzy rule interpolation. In: IEEE. *2019 2nd International Conference on new Trends in Computing Sciences (ICTCS)*. [S.l.], 2019. p. 1–6.

BAKSHI, K.; BAKSHI, K. Considerations for artificial intelligence and machine learning: Approaches and use cases. In: IEEE. *2018 IEEE Aerospace Conference*. [S.l.], 2018. p. 1–9.

BEDÓN, C.; SAIED, A. *Snort-AI (Version 2.4.3) "Open Source Project"*. 2009. Disponível em: <http://snort-ai.sourceforge.net/index.php>.

BILES, S. Detecting the unknown with snort and the statistical packet anomaly detection engine (spade). *Computer Security Online Ltd., Tech. Rep*, 2003.

BISHOP, C. M. *Pattern recognition and machine learning*. [S.l.]: springer, 2006.

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

BRERETON, R. G.; LLOYD, G. R. Support vector machines for classification and regression. *Analyst*, The Royal Society of Chemistry, v. 135, p. 230–267, 2010. Disponível em: <http://dx.doi.org/10.1039/B918972F>.

BREVE, F.; ZHAO, L.; QUILES, M.; PEDRYCZ, W.; LIU, J. Particle competition and cooperation in networks for semi-supervised learning. *IEEE Transactions on Knowledge and Data Engineering*, IEEE, v. 24, n. 9, p. 1686–1698, 2011.

BRO. *Bro IDS Documentation*. 2018. Disponível em: <https://www.bro.org/sphinx-/intro/index.html>. Acesso em: 15 mar. 2018.

BRO. *The Bro Network Security Monitor*. 2018. Disponível em: <https://www.bro.org/>. Acesso em: 17 mar. 2018.

BRUCE, R. F. A bayesian approach to semi-supervised learning. In: *NLPRS*. [S.l.: s.n.], 2001. p. 57–64.

BUITINCK, L.; LOUPPE, G.; BLONDEL, M.; PEDREGOSA, F.; MUELLER, A.; GRISEL, O.; NICULAE, V.; PRETTENHOFER, P.; GRAMFORT, A.; GROBLER, J. Api design for machine learning software: experiences from the scikit-learn project. *arXiv preprint arXiv:1309.0238*, 2013.

CANSIAN, A.; MOREIRA, E. d. S.; CARVALHO, A. C. P. d. L.; JUNIOR, J. B. Network intrusion detection using neural networks. 1997.

CHAWLA, N. V. Data mining for imbalanced datasets: An overview. In: *Data mining and knowledge discovery handbook*. [S.l.]: Springer, 2009. p. 875–886.

CHAWLA, N. V.; BOWYER, K. W.; HALL, L. O.; KEGELMEYER, W. P. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, v. 16, p. 321–357, 2002.

CHRISTOPHER, R. *Port Scanning Techniques and the Defense Against Them*. 2001. Disponível em: <https://www.sans.org/reading-room/whitepapers/auditing/port-scanning-techniques-defense-70>. Acesso em: 10 jan. 2020.

CLOUDFLARE. *Slowloris DDoS Attack*. 2017. Disponível em: <https://www-.cloudflare.com/learning/ddos/ddos-attack-tools/slowloris/>. Acesso em: 10 jan. 2020.

CORTES, C.; VAPNIK, V. Support-vector networks. *Machine learning*, Springer, v. 20, n. 3, p. 273–297, 1995.

CROTHERS, T. *Implementing intrusion detection systems: a hands-on guide for securing the network*. [S.l.]: Wiley, 2003.

DEBAR, H.; DACIER, M.; LAMPART, S. *An experimentation workbench for intrusion detection systems*. [S.l.]: IBM TJ Watson Research Center, 1998.

DEBAR, H.; DACIER, M.; WESPI, A. Towards a taxonomy of intrusion-detection systems. *Computer Networks*, Elsevier, v. 31, n. 8, p. 805–822, 1999.

DEVI, S.; NAGPAL, R. Intrusion detection system using genetic algorithm-a review. *International Journal of Computing & Business Research*, 2012.

DING, Y.-X.; XIAO, M.; LIU, A.-W. Research and implementation on snort-based hybrid intrusion detection system. In: IEEE. *2009 International Conference on Machine Learning and Cybernetics*. [S.l.], 2009. v. 3, p. 1414–1418.

DRAPER-GIL, G.; LASHKARI, A. H.; MAMUN, M. S. I.; GHORBANI, A. A. Characterization of encrypted and vpn traffic using time-related. In: *Proceedings of the 2nd international conference on information systems security and privacy (ICISSP)*. [S.l.: s.n.], 2016. p. 407–414.

DUAN, K.-B.; KEERTHI, S. S. Which is the best multiclass svm method? an empirical study. In: SPRINGER. *International workshop on multiple classifier systems*. [S.l.], 2005. p. 278–285.

DUDA, R. O.; HART, P. E.; STORK, D. G. *Pattern classification*. [S.l.]: John Wiley & Sons, 2012.

EFRON, B.; TIBSHIRANI, R. J. *An introduction to the bootstrap*. [S.l.]: CRC press, 1994.

ELSHAFIE, H. M.; MAHMOUD, T. M.; ALI, A. A. Improving the performance of the snort intrusion detection using clonal selection. In: IEEE. *2019 International Conference on Innovative Trends in Computer Engineering (ITCE)*. [S.l.], 2019. p. 104–110.

ESTEVEZ-TAPIADOR, J. M.; GARCIA-TEODORO, P.; DIAZ-VERDEJO, J. E. Stochastic protocol modeling for anomaly based network intrusion detection. In: IEEE. *Information Assurance, 2003. IWIAS 2003. Proceedings. First IEEE International Workshop on*. [S.l.], 2003. p. 3–12.

FANG, X.; LIU, L. Integrating artificial intelligence into snort ids. In: IEEE. *2011 3rd International Workshop on Intelligent Systems and Applications*. [S.l.], 2011. p. 1–4.

FAUST, J. Distributed analysis of ssh brute force and dictionary based attacks. In: *Culminating Projects in Information Assurance*. [s.n.], 2018. v. 56. Disponível em: <https://repository.stcloudstate.edu/msia_etds/56>.

FOX, K. A neural network approach wowards intrusion detection. *Tech. Rep.*, Harris Corporation, 1990.

FREUND, Y.; SCHAPIRE, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, Elsevier, v. 55, n. 1, p. 119–139, 1997.

FRIEDMAN, J.; HASTIE, T.; TIBSHIRANI, R. *The elements of statistical learning*. [S.l.]: Springer series in statistics New York, 2001.

GAMA, J. Functional trees. *Machine Learning*, v. 55, n. 3, p. 219–250, Jun 2004. ISSN 1573-0565. Disponível em: <https://doi.org/10.1023/B:MACH.0000027782.67192.13>.

GANGANWAR, V. An overview of classification algorithms for imbalanced datasets. *International Journal of Emerging Technology and Advanced Engineering*, Citeseer, v. 2, n. 4, p. 42–47, 2012.

GARCIA-TEODORO, P.; DIAZ-VERDEJO, J.; MACIÁ-FERNÁNDEZ, G.; VÁZQUEZ, E. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, Elsevier, v. 28, n. 1-2, p. 18–28, 2009.

HASTIE, T.; ROSSET, S.; ZHU, J.; ZOU, H. Multi-class adaboost. *Statistics and its Interface*, International Press of Boston, v. 2, n. 3, p. 349–360, 2009.

HODO, E.; BELLEKENS, X. J. A.; HAMILTON, A. W.; TACHTATZIS, C.; ATKINSON, R. C. Shallow and deep networks intrusion detection system: A taxonomy and survey. *CoRR*, abs/1701.02145, 2017. Disponível em: <http://arxiv.org/abs/1701.02145>.

HONG, H.; LIU, J.; BUI, D. T.; PRADHAN, B.; ACHARYA, T. D.; PHAM, B. T.; ZHU, A.-X.; CHEN, W.; AHMAD, B. B. Landslide susceptibility mapping using j48 decision tree with adaboost, bagging and rotation forest ensembles in the guangchang area (china). *Catena*, Elsevier, v. 163, p. 399–413, 2018.

JAMES, G.; WITTEN, D.; HASTIE, T.; TIBSHIRANI, R. *An introduction to statistical learning*. [S.l.]: Springer, 2013.

JOACHIMS, T. Text categorization with support vector machines: Learning with many relevant features. In: NÉDELLEC, C.; ROUVEIROL, C. (Ed.). *Machine Learning: ECML-98*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998. p. 137–142. ISBN 978-3-540-69781-7.

JONGSAWAT, N.; DECHAROENCHITPONG, J. Creating behavior-based rules for snort based on bayesian network learning algorithms. In: IEEE. *2015 International Conference on Science and Technology (TICST)*. [S.l.], 2015. p. 267–270.

KAELBLING, L. P.; LITTMAN, M. L.; MOORE, A. W. Reinforcement learning: A survey. *Journal of artificial intelligence research*, v. 4, p. 237–285, 1996.

KAZIENKO, P.; DOROSZ, P. Intrusion detection systems (ids) part 2-classification; methods; techniques. *WindowsSecurity. com*, 2004.

KUERBAN, M.; TIAN, Y.; YANG, Q.; JIA, Y.; HUEBERT, B.; POSS, D. Flowsec: Dos attack mitigation strategy on sdn controller. In: IEEE. *2016 IEEE International Conference on Networking, Architecture and Storage (NAS)*. [S.l.], 2016. p. 1–2.

KUMAR, B. J.; NAVEEN, H.; KUMAR, B. P.; SHARMA, S. S.; VILLEGAS, J. Logistic regression for polymorphic malware detection using anova f-test. In: IEEE. *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*. [S.l.], 2017. p. 1–5.

KUMAR, K.; PUNIA, R. Improving the performance of ids using genetic algorithm. *International Journal of Computer Science and Communication*, v. 4, n. 2, 2013.

KUROSE, J. F.; ROSS, K. W. *Computer networking: a top-down approach*. [S.l.]: Addison Wesley Boston, USA, 2009.

LASHKARI, A. H.; DRAPER-GIL, G.; MAMUN, M. S. I.; GHORBANI, A. A. Characterization of tor traffic using time based features. In: *ICISSP*. [S.l.: s.n.], 2017. p. 253–262.

LI, H.; LIU, D. Research on intelligent intrusion prevention system based on snort. In: IEEE. *Computer, Mechatronics, Control and Electronic Engineering (CMCE), 2010 International Conference on*. [S.l.], 2010. v. 1, p. 251–253.

LI, W. Using genetic algorithm for network intrusion detection. *Proceedings of the United States Department of Energy Cyber Security Group*, v. 1, p. 1–8, 2004.

LIAO, H.-J.; LIN, C.-H. R.; LIN, Y.-C.; TUNG, K.-Y. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, Elsevier, v. 36, n. 1, p. 16–24, 2013.

LORENA, A. C.; CARVALHO, A. C. D. Evolutionary tuning of svm parameter values in multiclass problems. *Neurocomputing*, Elsevier, v. 71, n. 16-18, p. 3326–3334, 2008.

MAHMOUD, T. M.; ALI, A. A.; ELSHAFIE, H. M. A hybrid snort-negative selection network intrusion detection technique. *International Journal of Computer Applications*, Foundation of Computer Science, v. 975, p. 8887, 2016.

MAKANJU, A.; ZINCIR-HEYWOOD, N.; MILIOS, E. Adaptabilty of a gp based ids on wireless networks. In: IEEE. *2008 Third International Conference on Availability, Reliability and Security*. [S.l.], 2008. p. 310–318.

MCCALLUM, A.; NIGAM, K. et al. A comparison of event models for naive bayes text classification. In: CITESEER. *AAAI-98 workshop on learning for text categorization*. [S.l.], 1998. v. 752, n. 1, p. 41–48.

NAIK, N. Fuzzy inference based intrusion detection system: Fi-snort. In: IEEE. *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*. [S.l.], 2015. p. 2062–2067.

NAIK, N.; DIAO, R.; SHEN, Q. Application of dynamic fuzzy rule interpolation for intrusion detection: D-fri-snort. In: IEEE. *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. [S.l.], 2016. p. 78–85.

NORTHCUTT, S.; NOVAK, J. *Network intrusion detection*. [S.l.]: Sams Publishing, 2002.

PATEL, S. K.; SONKER, A. Internet protocol identification number based ideal stealth port scan detection using snort. In: IEEE. *2016 8th International Conference on Computational Intelligence and Communication Networks (CICN)*. [S.l.], 2016. p. 422–427.

PEDREGOSA, F.; VAROQUAUX, G.; GRAMFORT, A.; MICHEL, V.; THIRION, B.; GRISEL, O.; BLONDEL, M.; PRETTENHOFER, P.; WEISS, R.; DUBOURG, V. et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, v. 12, n. Oct, p. 2825–2830, 2011.

PIETRASZEK, T.; TANNER, A. Data mining and machine learning—towards reducing false positives in intrusion detection. *Information security technical report*, Elsevier, v. 10, n. 3, p. 169–183, 2005.

PORRAS, P. A.; VALDES, A. Live traffic analysis of tcp/ip gateways. In: *NDSS*. [S.l.: s.n.], 1998.

ROESCH, M. et al. Snort: Lightweight intrusion detection for networks. In: *Lisa*. [S.l.: s.n.], 1999. v. 99, n. 1, p. 229–238.

ROJAS, R. Adaboost and the super bowl of classifiers: A tutorial introduction to adaptive boosting. 2009.

RUSSELL, S. J.; NORVIG, P. *Artificial intelligence: a modern approach*. [S.l.]: Malaysia; Pearson Education Limited,, 2016.

SADASIVAM, G. K.; HOTA, C.; ANAND, B. Honeynet data analysis and distributed ssh brute-force attacks. In: *Towards Extensible and Adaptable Methods in Computing*. [S.l.]: Springer, 2018. p. 107–118.

SAFAVIAN, S. R.; LANDGREBE, D. A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, IEEE, v. 21, n. 3, p. 660–674, 1991.

SAIED, A.; OVERILL, R. E.; RADZIK, T. Detection of known and unknown ddos attacks using artificial neural networks. *Neurocomputing*, Elsevier, v. 172, p. 385–393, 2016.

SCHAPIRE, R. E. Explaining adaboost. In: *Empirical inference*. [S.l.]: Springer, 2013. p. 37–52.

SCIKIT-LEARN. *Decision Trees*. 2011. Disponível em: <https://scikit-learn.org/stable-/modules/tree.html>. Acesso em: 14 jul. 2019.

SCIKIT-LEARN. *Naive Bayes*. 2011. Disponível em: <https://scikit-learn.org/stable-/modules/naive_bayes.html>. Acesso em: 11 jul. 2019.

SCIKIT-LEARN. *Support Vector Machines*. 2011. Disponível em: <https://scikit-learn-.org/stable/modules/svm.html>. Acesso em: 17 jul. 2019.

SCIKIT-LEARN. *Underfitting vs. Overfitting*. 2014. Disponível em: <http://scikit-learn-.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html>. Acesso em: 11 jul. 2019.

SCORNET, E.; BIAU, G.; VERT, J.-P. et al. Consistency of random forests. *The Annals of Statistics*, Institute of Mathematical Statistics, v. 43, n. 4, p. 1716–1741, 2015.

SEELAMMAL, C.; DEVI, K. V. Computational intelligence in intrusion detection system for snort log using hadoop. In: IEEE. *2016 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*. [S.l.], 2016. p. 642–647.

SHAH, S. A. R.; ISSAC, B. Performance comparison of intrusion detection systems and application of machine learning to snort system. *Future Generation Computer Systems*, Elsevier, v. 80, p. 157–170, 2018.

SHARAFALDIN, I.; LASHKARI, A. H.; GHORBANI, A. A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: . [S.l.: s.n.], 2018.

SHTEIMAN, B. *HULK - HTTP Unbearable Load King*. 2012. Disponível em: <https:-//packetstormsecurity.com/files/112856/HULK-Http-Unbearable-Load-King.html>. Acesso em: 10 jan. 2020.

SILVA, J. R. C. d. *Redes Neurais Artificiais para Sistemas de Detecção de Intrusos*. Monografia (Bacharel em Ciência da Computação) — Faculdades Integradas de Caratinga, Minas Gerais, 2007.

SILVA, L. de S.; SANTOS, A. F. dos; SILVA, J. D. S. da; MONTES, A. A neural network application for attack detection in computer networks. In: IEEE. *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*. [S.l.], 2004. v. 2, p. 1569–1574.

SNORT. *Snort 3 User Manual*. 2019. Acesso em: 21 jul. 2019.

SONG, T.; KO, C.; ALVES-FOSS, J.; ZHANG, C.; LEVITT, K. Formal reasoning about intrusion detection systems. In: SPRINGER. *International Workshop on Recent Advances in Intrusion Detection*. [S.l.], 2004. p. 278–295.

SONG, Y.-Y.; YING, L. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, Shanghai Mental Health Center, v. 27, n. 2, p. 130, 2015.

STALLINGS, W. *Network Security Essentials: Applications and Standards, 4/e*. [S.l.]: Pearson Education India, 2000.

STANIFORD, S.; HOAGLAND, J. A.; MCALERNEY, J. M. Practical automated detection of stealthy portscans. *Journal of Computer Security*, IOS Press, v. 10, n. 1-2, p. 105–136, 2002.

SUNDARAM, A. An introduction to intrusion detection. *Crossroads*, ACM, v. 2, n. 4, p. 3–7, 1996.

TACTICALFLEX. *Snort vs Suricata*. 2019. Disponível em: <https://tacticalflex-.zendesk.com/hc/en-us/articles/360010678893-Snort-vs-Suricata>. Acesso em: 15 jul. 2019.

TYUGU, E. Artificial intelligence in cyber defense. In: IEEE. *Cyber Conflict (ICCC), 2011 3rd International Conference on*. [S.l.], 2011. p. 1–11.

UTIMURA, L. N.; COSTA, K. A. Aplicação e análise comparativa do desempenho de classificadores de padrões para o sistema de detecção de intrusão snort. In: *Anais do XXXVI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Porto Alegre, RS, Brasil: SBC, 2018. ISSN 2177-9384. Disponível em: <https://ojs.sbc.org.br/index.php/sbrc/article/view/2426>.

VANNEY, I. *Brute force against SSH and FTP services: attacking and defending SSH and FTP*. 2019. Disponível em: <https://linuxhint.com/bruteforce_ssh_ftp/>. Acesso em: 10 jan. 2020.

WEON, I.-Y.; SONG, D. H.; LEE, C.-H.; HEO, Y.-J.; KIM, K.-Y. A memory-based learning approach to reduce false alarms in intrusion detection. In: IEEE. *The 7th International Conference on Advanced Communication Technology, 2005, ICACT 2005.* [S.l.], 2005. v. 1, p. 241–245.

WILCOXON, F. Individual comparisons of grouped data by ranking methods. *Journal of economic entomology*, Oxford University Press Oxford, UK, v. 39, n. 2, p. 269–270, 1946.

YAP, B. W.; RANI, K. A.; RAHMAN, H. A. A.; FONG, S.; KHAIRUDIN, Z.; ABDULLAH, N. N. An application of oversampling, undersampling, bagging and boosting in handling imbalanced datasets. In: SPRINGER. *Proceedings of the first international conference on advanced data and information engineering (DaEng-2013)*. [S.l.], 2014. p. 13–22.

YEN, L. *An Introduction to the Bootstrap Method*. 2019. Disponível em: <https:/-/towardsdatascience.com/an-introduction-to-the-bootstrap-method-58bcb51b4d60>. Acesso em: 15 jul. 2019.

ZHANG, H. The optimality of naive bayes. *AA*, v. 1, n. 2, p. 3, 2004.

ZHANG, H.; QI, Y.; ZHOU, H.; ZHANG, J.; SUN, J. Testing and defending methods against dos attack in state estimation. *Asian Journal of Control*, Wiley Online Library, v. 19, n. 4, p. 1295–1305, 2017.

ZHU, X. Semi-supervised learning literature survey. *Computer Science, University of Wisconsin-Madison*, v. 2, n. 3, p. 4, 2006.