

**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**

**FELIPE YUZO AKASHI SATO**

**ANÁLISE DO DESEMPENHO DE ALGORITMOS CLASSIFICADORES PARA  
DETECÇÃO DE ANOMALIAS EM SISTEMAS DE DETECÇÃO DE INTRUSÃO**

**CORNÉLIO PROCÓPIO**

**2023**

**FELIPE YUZO AKASHI SATO**

**ANÁLISE DO DESEMPENHO DE ALGORITMOS CLASSIFICADORES PARA  
DETECÇÃO DE ANOMALIAS EM SISTEMAS DE DETECÇÃO DE INTRUSÃO**

**Performance Analysis of Classification Algorithms for Anomaly Detection in  
Intrusion Detection Systems**

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Lucas Dias Hiera Sampaio

**CORNÉLIO PROCÓPIO**

**2023**



Esta licença permite compartilhamento, remixe, adaptação e criação a partir do trabalho, mesmo para fins comerciais, desde que sejam atribuídos créditos ao(s) autor(es). Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.

[4.0 Internacional](#)

**FELIPE YUZO AKASHI SATO**

**ANÁLISE DO DESEMPENHO DE ALGORITMOS CLASSIFICADORES PARA  
DETECÇÃO DE ANOMALIAS EM SISTEMAS DE DETECÇÃO DE INTRUSÃO**

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação do Curso de Bacharelado em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 13/Junho/2023

---

Antônio Carlos Fernandes da Silva  
Doutor  
Universidade Tecnológica Federal do Paraná

---

Henrique Yoshikazu Shishido  
Doutor  
Universidade Tecnológica Federal do Paraná

**CORNÉLIO PROCÓPIO  
2023**

Aos futuros estudantes, pois sempre seremos  
alunos da vida.

## **AGRADECIMENTOS**

Agradeço primeiramente a Deus, que me permitiu chegar até aqui e estudar nesta grande instituição de ensino e pesquisa. Agradeço à minha família, que me apoiou incondicionalmente durante toda a graduação. Agradeço também aos meus familiares que me suportaram de diversas maneiras enquanto estive distante na graduação.

Agradeço aos professores e colaboradores da Universidade Tecnológica Federal do Paraná campus de Cornélio Procópio (UTFPR-CP) pelo suporte e conhecimento fomentado durante todo o período de graduação. Agradeço em especial ao meu orientador Lucas Dias Hiera Sampaio pela orientação e apoio no desenvolvimento deste trabalho. Agradeço também ao professor Armando Paulo da Silva e seus colaboradores nos projetos de extensão que tive o prazer em participar.

Agradeço ao meu amigo Mateus que me apoiou a seguir neste caminho e me incentivou a continuar me desenvolvendo. Agradeço a Talita, que sempre esteve comigo durante os momentos mais difíceis e complicados e me motivou durante toda a graduação. Ao apoio e motivação dados à mim, o meu muito obrigado.

## RESUMO

Com a expansão do uso de serviços em nuvem, a segurança na rede se tornou ponto de foco para o setor de tecnologia. Os ataques aos sistemas empresariais e governamentais através de *malwares* se intensificaram de forma proporcional ao crescimento do número de usuários que dependem do acesso aos dados e serviços em nuvem. Desta forma, os Sistemas de Detecção de Intrusão baseados em Rede (SDIR) servem como primeira ferramenta de proteção, com algoritmos processando o tráfego em pontos vitais da rede com o intuito de detectar e impedir ações maliciosas. Como técnica complementar aos Sistemas de Detecção por Assinatura, os Sistemas de Detecção por Anomalias analisam o comportamento do tráfego na rede com o objetivo de detectar estes ataques. Uma das formas utilizadas para esta detecção é o uso de algoritmos classificadores para analisar e classificar entre o tráfego normal ou anômalo dos pacotes da rede. O algoritmo de Naïve-Bayes é um classificador utilizado devido a sua simplicidade de implementação. O algoritmo *Adaboost*, por sua vez, apresenta uma possibilidade de desempenho ao algoritmo. Por fim, a Máquina de Vetores de Suporte (SVM), apresenta um algoritmo de maior complexidade, que servirá como comparativo para os outros algoritmos. Este trabalho irá analisar e comparar o desempenho dos algoritmos classificadores sobre o *dataset* NB-15 da Universidade de New South Wales de 2015, considerando os principais aspectos para possíveis implementações em dispositivos reais.

**Palavras-chave:** detecção de anomalias; algoritmo de naïve-bayes; algoritmo adaboost; svm; unsw nb15.

## ABSTRACT

With the expansion of cloud services usage, network security has become a focal point for the technology sector. Attacks on enterprise and government systems through malware have intensified in proportion to the growing number of users relying on cloud data and services access. Thus, Network-based Intrusion Detection Systems (NIDS) serve as the first line of defense, with algorithms processing traffic at critical points in the network to detect and prevent malicious actions. As a complementary technique to Signature-based Detection Systems, Anomaly-based Detection Systems analyze traffic behavior in the network to detect such attacks. One of the methods used for this detection is the use of classifier algorithms to analyze and classify network packet traffic as normal or anomalous. The Naïve-Bayes algorithm is a classifier commonly used due to its simplicity of implementation. The Adaboost algorithm, on the other hand, offers a potential performance boost to the algorithm. Finally, the Support Vector Machine (SVM) presents a more complex algorithm that will serve as a benchmark for the other algorithms. This work will analyze and compare the performance of these classifier algorithms on the NB-15 dataset from the University of New South Wales from 2015, considering key aspects for potential real-world implementations.

**Keywords:** anomaly detection; naïve-bayes algorithm; adaboost algorithm; svm; unsw nb15.

## LISTA DE FIGURAS

<b>Figura 1 – Exemplo do Classificador de Margens Rígidas <i>Support Vector Machine</i> (SVM) . . . . .</b>	<b>25</b>
<b>Figura 2 – Exemplo do Classificador de Margens Maleáveis SVM . . . . .</b>	<b>27</b>
<b>Figura 3 – Exemplo do Classificador SVM Não-Linear . . . . .</b>	<b>30</b>
<b>Figura 4 – Esquema de Construção da Rede . . . . .</b>	<b>34</b>
<b>Figura 5 – Fluxograma de Execução . . . . .</b>	<b>37</b>
<b>Figura 6 – Exemplo de Matriz de Confusão Normalizada . . . . .</b>	<b>40</b>
<b>Figura 7 – Exemplo de Curva ROC . . . . .</b>	<b>40</b>
<b>Figura 8 – Matriz de Confusão do Classificador de <i>Gaussian Naïve-Bayes</i> . . . . .</b>	<b>42</b>
<b>Figura 9 – Matriz de Confusão Normalizada do Classificador de <i>Gaussian Naïve-Bayes</i> . . . . .</b>	<b>43</b>
<b>Figura 10 – Matriz de Confusão do Classificador por <i>AdaBoost</i> . . . . .</b>	<b>44</b>
<b>Figura 11 – Matriz de Confusão Normalizada do Classificador por <i>AdaBoost</i> . . . . .</b>	<b>44</b>
<b>Figura 12 – Curva ROC do <i>AdaBoost Classifier</i> variando os parâmetros em 10% . . . . .</b>	<b>45</b>
<b>Figura 13 – Matriz de Confusão do Classificador por <i>Support Vector Machine</i> . . . . .</b>	<b>47</b>
<b>Figura 14 – Matriz de Confusão Normalizada do Classificador por <i>Support Vector Machine</i> . . . . .</b>	<b>47</b>
<b>Figura 15 – Curva ROC do Classificador por <i>Support Vector Machine</i> . . . . .</b>	<b>48</b>

## LISTA DE TABELAS

<b>Tabela 1 – Porcentagem de dados normais e anormais nos <i>datasets</i></b>	<b>34</b>
<b>Tabela 2 – Tamanho do Conjunto de Treino e Teste</b>	<b>38</b>
<b>Tabela 3 – Atributos selecionados</b>	<b>38</b>
<b>Tabela 4 – Resultados das métricas para o algoritmo de Naïve-Bayes Gaussiano</b>	<b>43</b>
<b>Tabela 5 – Resultados das métricas para o algoritmo <i>AdaBoost</i></b>	<b>45</b>
<b>Tabela 6 – Resultados do <i>GridsearchCV</i></b>	<b>46</b>
<b>Tabela 7 – Parâmetros para o SVM com otimização em Precisão</b>	<b>46</b>
<b>Tabela 8 – Resultados das métricas para o algoritmo SVM</b>	<b>48</b>
<b>Tabela 9 – Resultados das métricas para cada algoritmo</b>	<b>49</b>
<b>Tabela 10 – Quantidade de Pacotes processados por Segundo</b>	<b>49</b>

## **LISTA DE QUADROS**

<b>Quadro 1 – Funções <i>Kernel</i></b> . . . . .	<b>29</b>
<b>Quadro 2 – Resumo de Trabalhos Relacionados</b> . . . . .	<b>32</b>
<b>Quadro 3 – Características do Fluxo</b> . . . . .	<b>35</b>
<b>Quadro 4 – Características Básicas de Transmissão</b> . . . . .	<b>35</b>
<b>Quadro 5 – Características do Conteúdo</b> . . . . .	<b>35</b>
<b>Quadro 6 – Características Temporais</b> . . . . .	<b>36</b>
<b>Quadro 7 – Características de Propósito Geral</b> . . . . .	<b>36</b>
<b>Quadro 8 – Características de Conexão</b> . . . . .	<b>37</b>

## **LISTA DE ABREVIATURAS E SIGLAS**

### **Siglas**

AUC	<i>Area Under the Curve</i>
CIC	<i>Canadian Institute of Cybersecurity</i>
HIDS	<i>Host-based Intrusion Detection System</i>
IBM	<i>International Business Machine Corporation</i>
IDS	<i>Intrusion Detection System</i>
IoT	<i>Internet of Things</i>
ML	<i>Machine Learning</i>
NBA	<i>Network Behaviour Analysis</i>
NIDS	<i>Network-based Intrusion Detection System</i>
SV	<i>Support Vector</i>
SVM	<i>Support Vector Machine</i>
UNSW	<i>University of New South Wales</i>
WIDS	<i>Wireless-based Intrusion Detection System</i>

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO . . . . .</b>	<b>12</b>
<b>1.1</b>	<b>Considerações iniciais . . . . .</b>	<b>12</b>
<b>1.2</b>	<b>Objetivos . . . . .</b>	<b>13</b>
1.2.1	Objetivo geral . . . . .	13
1.2.2	Objetivos específicos . . . . .	13
<b>1.3</b>	<b>Justificativa . . . . .</b>	<b>13</b>
<b>1.4</b>	<b>Estrutura do trabalho . . . . .</b>	<b>14</b>
<b>2</b>	<b>REFERENCIAL TEÓRICO . . . . .</b>	<b>15</b>
<b>2.1</b>	<b><i>Intrusion Detection System (IDS)</i> . . . . .</b>	<b>15</b>
2.1.1	Arquitetura de <i>Intrusion Detection System (IDS)</i> . . . . .	15
2.1.2	Alvo da IDS . . . . .	16
2.1.3	Modo de Detecção da IDS . . . . .	16
<b>2.2</b>	<b>Algoritmos Classificadores . . . . .</b>	<b>17</b>
2.2.1	Algoritmo de Naïve-Bayes . . . . .	18
2.2.2	Algoritmo <i>Adaptive Boosting</i> . . . . .	19
2.2.3	Algoritmo SVM . . . . .	22
<b>3</b>	<b>TRABALHOS RELACIONADOS . . . . .</b>	<b>31</b>
<b>4</b>	<b>MATERIAIS E MÉTODOS . . . . .</b>	<b>33</b>
<b>4.1</b>	<b>Materiais . . . . .</b>	<b>33</b>
4.1.1	<i>Dataset</i> . . . . .	33
4.1.2	Ferramentas Utilizadas . . . . .	36
<b>4.2</b>	<b>Métodos . . . . .</b>	<b>37</b>
4.2.1	Extração de Parâmetros . . . . .	38
4.2.2	Métricas de Avaliação . . . . .	39
<b>5</b>	<b>RESULTADOS . . . . .</b>	<b>42</b>
<b>5.1</b>	<b>Aplicação do Algoritmo de Naïve-Bayes Gaussiano . . . . .</b>	<b>42</b>
<b>5.2</b>	<b>Aplicação do Algoritmo Classificador <i>AdaBoost</i> . . . . .</b>	<b>44</b>
<b>5.3</b>	<b>Aplicação do Algoritmo SVM . . . . .</b>	<b>45</b>
<b>5.4</b>	<b>Análise entre algoritmos . . . . .</b>	<b>48</b>
<b>6</b>	<b>CONCLUSÃO . . . . .</b>	<b>50</b>

6.1	<b>Trabalhos Futuros</b>	50
	<b>REFERÊNCIAS</b>	51

## 1 INTRODUÇÃO

A evolução humana e o seu desenvolvimento estão relacionados com a busca de melhorias e de inovações. Essas duas variáveis também incentivaram o processo de desenvolvimento e pesquisa por séculos. Segundo Fitzsimmons 1994, por todo o mundo a tecnologia se tornaria indispensável para as pessoas obterem as informações necessárias para o seu desenvolvimento pessoal e profissional, o que de fato ocorre. Conforme os dados coletados por Kemp 2022, o número de dispositivos conectados à Internet continua crescendo anualmente, principalmente com a crescente integração de dispositivos de *Internet of Things* (IoT). Com isso, pode-se perceber que há uma expansão das redes digitais, bem como a inclusão de mais dispositivos à Internet, levando à valorização das informações dos usuários, com bancos de dados e serviços sob demanda sendo as áreas mais afetadas.

### 1.1 Considerações iniciais

No período de 2020 e 2021, com a mudança do modelo de trabalho para o ambiente remoto e maior utilização do ambiente de *cloud* para as atividades corporativas, a maturidade das empresas que migraram para este ambiente não acompanharam de forma proporcional os requisitos. Esta baixa maturidade facilita com que as empresas sejam vítimas de criminosos cibernéticos, que veem estas empresas como vítimas fáceis e lucrativas. De acordo com a *International Business Machine Corporation* (IBM), para remediar estes problemas, foi observado que a automatização dos processos de segurança influenciam diretamente a perda sofrida por ataques cibernéticos e o tempo gasto entre a detecção de intrusão e a contenção da vulnerabilidade [International Business Machine Corporation 2021].

Ainda, segundo o relatório anual de ameaças da IBM, o perfil dos ataques cibernéticos no mundo se agravou em comparação com os anos anteriores com um crescimento de 33% em relação ao mesmo período de 2020 e 2021. Dado o aumento do tráfego de dados sensíveis na rede e consequente aumento no seu valor associado, a equipe de segurança da IBM catalogou as principais formas de ataque utilizadas por criminosos durante o ano de 2021. Em suas observações, o modo de invasão por *ramsonware* continua sendo o principal modelo de ataque, com o acesso inicial sendo feito por meio de *phishing* e de exploração de vulnerabilidades em 21% dos casos [International Business Machine Corporation 2022].

O cenário brasileiro não difere muito das tendências mundiais no aumento das atividades de invasões cibernéticas. Segundo a Distrito 2021, o Brasil foi o quinto país mais atingido por ataques cibernéticos no ano de 2021, indicando que as empresas brasileiras ainda são alvo para ataques. Ainda, segundo a Distrito 2021, é fato que a maturidade das empresas brasileiras de cibersegurança está em fase de desenvolvimento. Isto pode ser observado pelo nível de investimentos na área de cibersegurança, visto que, embora exista uma crescente tendência de investimentos, ainda não é um montante considerável dado o tamanho do mercado brasileiro e

sua demanda por soluções na área. Outro ponto que trabalha contra o desenvolvimento da segurança cibernética brasileira é a cultura das empresas brasileiras em buscar soluções pontuais para problemas específicos, o que aumenta a complexidade das soluções de segurança dado que elas necessitam ser compatíveis com outras soluções já empregadas.

## 1.2 Objetivos

Nesta seção estão apresentadas os objetivos gerais e específicos deste trabalho, indicando as técnicas empregadas e o *dataset* escolhido.

### 1.2.1 Objetivo geral

O objetivo geral deste trabalho é comparar o desempenho entre o Algoritmo de Naïve-Bayes Gaussiano [Chan, Golub e LeVeque 1979], o algoritmo *Adaboost* [Zhu *et al.* 2006] e o Algoritmo de SVM [Faceli *et al.* 2011] sobre o *dataset* *University of New South Wales* (UNSW) NB15 de Moustafa 2017 para a definição da base entre o comportamento normal dos dados coletados e o comportamento anômalo de invasões que foram capturadas no *dataset*.

### 1.2.2 Objetivos específicos

Para se atingir o objetivo geral do trabalho, serão necessários a conclusão dos seguintes objetivos específicos.

- Executar o balanceamento do *dataset*;
- Executar a normalização do *dataset*;
- Amostrar o *dataset* para treino e teste com *cross validation*;
- Executar os algoritmos classificadores sobre o *dataset* amostrado;
- Analisar e comparar o desempenho dos algoritmos com as métricas estabelecidas;

## 1.3 Justificativa

Com este estudo foi possível analisar a aplicabilidade de algoritmos de *Machine Learning* (ML) sobre o *dataset* UNSW NB-15, que, por ser recente, apresenta novas possibilidades de servir como *benchmark* para desenvolvimentos futuros sobre o assunto. Ainda, este trabalho aplica conceitos de *Data Science* e *Data Engineering* com o tratamento dos dados do dataset para a aplicação.

Desta forma, o trabalho apresenta um *template* para o teste de técnicas de ML sobre *datasets*, com a sua metodologia permitindo o teste de outras técnicas ou outros *datasets*.

#### 1.4 Estrutura do trabalho

Este trabalho está organizado da seguinte forma: No Capítulo 2 serão abordados os conceitos sobre IDS, o algoritmo de Naïve-Bayes, o algoritmo *AdaBoost*, o algoritmo SVM e suas propriedades, as métricas que serão utilizadas e trabalhos na área de detecção de anomalias através de algoritmos classificadores. No Capítulo 3 será apresentado o ciclo de vida do projeto, as características do *dataset* a separação entre os elementos de treinamento e teste, o balanceamento do *dataset* utilizado e as ferramentas e linguagens de programação que serão utilizadas. No Capítulo 4 serão apresentados os resultados preliminares da aplicação dos algoritmos de Naïve-Bayes e o algoritmo *AdaBoost*. Por fim, no Capítulo 5 serão apresentadas as conclusões preliminares da análise da aplicação dos algoritmos neste trabalho.

## 2 REFERENCIAL TEÓRICO

Com a transição do modelo de trabalho tradicional e o contínuo aumento da presença das empresas no meio digital, muitos grupos criminosos estão se aproveitando da atual imaturidade de muitas empresas para lucrarem com ataques às redes e aos dados que nela circulam. Esta ameaça por meios digitais às empresas e governos do mundo não são novidade, com diferentes relatórios voltados à análise e melhoria destes sistemas de segurança [International Business Machine Corporation 2022, Distrito 2021].

Neste contexto, o desenvolvimento de IDS baseados em detecção de anomalias se mostram como uma alternativa complementar às tecnologias tradicionais de detecção baseadas em assinaturas. O trabalho de Fernandes *et al.* 2019 apresenta uma compilação dos ramos de pesquisa e desenvolvimento para aplicação desta tecnologia até o ano de 2019. Os algoritmos de classificação empregam técnicas de ML para ensinar os algoritmos a reconhecerem e classificarem padrões de tráfego na rede em comportamentos normais ou anômalos.

Neste capítulo são apresentadas a definição dos conceitos e das tecnologias utilizadas bem como os trabalhos relacionados ao tema.

### 2.1 *Intrusion Detection System (IDS)*

Segundo Axelsson 2000, IDS são algoritmos que permitem ao sistema reconhecer e responder a ataques cibernéticos. Estes sistemas podem ser comparados com os alarmes de segurança, que permitem a ativação de sistemas dedicados para identificar com rapidez a iminência de um ataque e responder de forma apropriada. Desta forma, as IDS podem ser classificados quanto à arquitetura, o alvo e o modo de operação.

#### 2.1.1 Arquitetura de IDS

A arquitetura das IDS se baseiam na distribuição lógica ou geográfica dos módulos de detecção, podendo ser distribuída ou centralizada.

Na arquitetura centralizada, todo o tráfego da rede passa pelo IDS, com este servindo como funil das informações entrando na rede. Este tipo de arquitetura apresenta a vantagem de concentrar todo o tráfego a ser analisado em um único ponto, porém, ele requer que o roteador no qual a IDS está implementada tenha poder de processamento suficiente para evitar gargalos e gerar latência nos sistemas.

Já na arquitetura distribuída, os módulos de IDS estão espalhados por toda a rede, e estes módulos ficam analisando o tráfego para detectar possíveis invasões. Para tanto, os módulos podem operar de forma paralela, onde todos os módulos operam de forma independente,

analizando o tráfego presente na sua rede; ou de forma cooperativa, onde existe a comunicação entre os módulos para o compartilhamento dos dados e melhor detecção de intrusões.

O uso de uma arquitetura ou de outra é baseado principalmente no tamanho da rede, visto que para redes grandes, o custo da instalação de um único módulo de processamento centralizado é inviável.

### 2.1.2 Alvo da IDS

A classificação dos IDS quando ao seu alvo se refere à localização do IDS na rede. Para *Host-based Intrusion Detection System* (HIDS), o IDS fica focado no monitoramento de dispositivos que contenham informações sensíveis, analisando o tráfego de dados que entram e saem do dispositivo. Por outro lado, o *Network-based Intrusion Detection System* (NIDS) consiste em um ativo de rede que analisa todo o tráfego que transita por ele, buscando detectar possíveis tentativas de intrusão [Liao *et al.* 2013].

Já o *Wireless-based Intrusion Detection System* (WIDS) consiste na aplicação de sistemas NIDS em redes sem fio, que possuem as complexidades de conexões *ad-hoc*. Por fim, o *Network Behaviour Analysis* (NBA) analisa o tráfego em busca de reconhecer ataques com padrões de fluxo inesperados [Liao *et al.* 2013].

### 2.1.3 Modo de Detecção da IDS

Quando ao modo de detecção, os trabalhos de Axelsson 2000, Patcha e Park 2007 e Liao *et al.* 2013, apresentam duas grandes classificações: a Detecção por Assinatura e a Detecção por Anomalia.

#### Sistemas por Detecção de Assinaturas

Os IDS baseados em assinaturas são os mais simples de serem utilizados. Essas assinaturas são as características únicas que identificam o *malware*. Desta forma, os IDS buscam em um banco de dados as regras e referências para localizar traços da assinatura dos *malwares* que estejam tentando invadir o sistema.

Esta forma de detecção possui a vantagem de reconhecer com precisão ataques já classificados com baixo consumo computacional, pois seu funcionamento se baseia em uma busca em banco de dados que contém assinaturas de *malwares* anteriormente detectados e classificados. Porém, como sua detecção é limitada a ataques já reconhecidos, este modelo de IDS requer constante atualizações de seu banco de dados e não consegue detectar *Zero-day Attacks*, pois não possui a assinatura dos *malwares* sendo empregados.

## Sistemas por Detecção de Anomalias

Os IDS baseados em anomalias são algoritmos que analisam o comportamento do tráfego da rede e são capazes de reconhecer padrões de tráfego normais e anômalos para reconhecer padrões de tráfego e, com base nestes padrões identificar anomalias que representam possíveis ataques.

Estes algoritmos podem se utilizar de diferentes técnicas e abordagens para classificar os dados. Na abordagem estatística, os algoritmos buscam reconhecer um padrão através da análise estatística dos dados em um prolongado período de tempo. Esta abordagem possui como vantagem a capacidade reconhecer e detectar ataques mesmo que estes estejam distribuídos em um longo período de tempo. Outra vantagem é a capacidade de reconhecer ataques sem conhecimento prévio, o que permite os sistemas a detectarem *Zero-Day Attacks*. Entretanto, uma desvantagem desta abordagem é que usuários maliciosos podem treinar o algoritmo a reconhecer ataques como tráfego normal, inutilizando o IDS.

Outra abordagem se utiliza de técnicas de ML para identificar o comportamento normal da rede. Esta técnica apresenta a capacidade de detectar novas formas de invasões, visto que não é limitada ao reconhecimento de invasões anteriores como o modelo baseado em assinaturas. Entretanto, a principal desvantagem desta técnica é a alta taxa de falsos positivos que são detectados.

Desta forma, a redução da taxa de falso positivos se torna indispensável para a melhoria das técnicas de detecção de anomalias, o que implica na relevância maior da precisão como métrica para analisar os algoritmos classificadores.

## **2.2 Algoritmos Classificadores**

Algoritmos classificadores são algoritmos que buscam, através da análise dos dados em cada atributo, classificar as informações recebidas. Estes algoritmos normalmente consistem em uma fase de treinamento, onde o algoritmo é submetido a análise de um subconjunto teste para o refino dos atributos a serem estudados, e uma fase de execução, onde são aplicados dados no algoritmo para testar sua eficácia [Duda, Hart e Stork 2000, Faceli *et al.* 2011]

Este algoritmo de reconhecimento de padrões pode ser supervisionado ou não-supervisionado. Quando os dados sendo analisados são rotulados e estruturados, o reconhecimento é dito supervisionado, pois o algoritmo irá reconhecer os padrões através dos rótulos dos dados [Liao *et al.* 2013, Patcha e Park 2007].

Por outro lado, o algoritmo é dito não-supervisionado quando os dados não são rotulados, e o algoritmo emprega técnicas para identificar os padrões nos dados puros. Esta segunda abordagem possui a vantagem de permitir que o algoritmo utilize os dados adquiridos diretamente, sem a necessidade de um pré-processamento para rotular os dados [Liao *et al.* 2013, Patcha e Park 2007].

Abaixo estão descritos os algoritmos empregados neste trabalho: o algoritmo de Naïve-Bayes, o algoritmo *Adaboost* e o algoritmo SVM.

### 2.2.1 Algoritmo de Naïve-Bayes

A base para o algoritmo de Naïve-Bayes consiste no Teorema Fundamental de Bayes apresentado na Equação 1 abaixo, no qual são atualizados os valores de probabilidade com base em novas evidências [Duda, Hart e Stork 2000, Faceli *et al.* 2011].

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (1)$$

Onde  $P(A)$  é a probabilidade da classe,  $P(B)$  é a probabilidade dos dados apresentares os valores e  $P(B|A)$  são os objetos com os valores encontrados que pertencem a classe  $A$  [Faceli *et al.* 2011].

Assim, considerando o contexto de algoritmos classificadores, podemos reescrever a Equação 1 considerando a probabilidade de um dado pertencer a uma classe. Ao maximizarmos essa probabilidade obteremos uma precisão maior para a classificação de determinado dado. Para a execução desta função é utilizado o Algoritmo Classificador de Naïve-Bayes. Desta forma, podemos reescrever a Equação 1 conforme a Equação 2 abaixo.

$$P(y_i|\mathbf{x}) = \frac{P(\mathbf{x}|y_i)P(y_i)}{P(\mathbf{x})} \quad (2)$$

Onde  $P(y_i|\mathbf{x})$  é a probabilidade do exemplo  $\mathbf{x}$  pertencer a classe  $y_i$ ,  $P(y_i)$  é a probabilidade da classe  $y_i$  ocorrer,  $P(\mathbf{x}|y_i)$  é a probabilidade da classe  $y_i$  possuir o elemento  $\mathbf{x}$  e  $P(\mathbf{x})$  é a probabilidade do elemento  $\mathbf{x}$  ocorrer.

Desta forma, o Algoritmo Classificador de Naïve-Bayes assume que os valores dos atributos são independentes entre si com base em uma classe, o que permite decompor as probabilidades de cada atributo em um produtório como demonstrado na Equação 3 abaixo [Faceli *et al.* 2011].

$$P(y_i|\mathbf{x}) \propto P(y_i) \prod_{j=1}^d P(x_j|y_i) \quad (3)$$

Chegamos neste produtório através da suposição de que os atributos possuem dependências entre si quase nulas, uma suposição ideal que é indicada no nome do algoritmo como *Naïve*, ou *Ingênuo* em português [Duda, Hart e Stork 2000, Faceli *et al.* 2011].

Este algoritmo possui grandes vantagens em sua aplicação, como o baixo custo computacional, a capacidade de lidar com grandes quantidades de dados e dados incompletos e a baixa sensibilidade a ruídos e atributos irrelevantes, sendo seus principais pontos fortes [Faceli *et al.* 2011].

### Algoritmo 1 – Algoritmo de Naïve-Bayes Gaussiano

```

1: Entrada: Dados de treinamento  $X$  e rótulos  $Y$ 
2: Saída: Modelo Naïve-Bayes Gaussiano  $NBG$ 

3: Inicializar  $NBG$ 
4: Separar os dados de treinamento por classe
5: para cada classe  $C_i$  faz
6:   para cada atributo  $A_j$  faz
7:     Calcular a média  $\mu_{ij}$  e o desvio padrão  $\sigma_{ij}$  dos valores de  $A_j$  para a classe  $C_i$ 
8:   finaliza para
9: finaliza para
10: Calcular a probabilidade a priori  $P(C_i)$  para cada classe  $C_i$ 
11: retorna  $NBG$ 
```

**Fonte:** Autor, 2023.

### Algoritmo de Naïve-Bayes Gaussiano

Como forma de aplicação, o Algoritmo de Naïve-Bayes Gaussiano supõe que as probabilidades das características obedece uma curva Gaussiana, atualizando as probabilidades da Equação 3 [Scikit-Learn 2022]. Desta forma, temos a Equação 4.

$$P(y_i|\mathbf{x}) = \frac{1}{\sqrt{2\pi\sigma_x^2}} \exp\left(-\frac{(y_i - \mu_x)^2}{2\sigma_x^2}\right) \quad (4)$$

Onde  $\sigma_x$  e  $\mu_x$  são a variância e a média da variável  $y_i$  estimados pelo método da maior probabilidade dada uma classe de  $\mathbf{x}$ . Desta forma, o Algoritmo 1 abaixo apresenta o funcionamento do modelo de classificação.

Estes valores são obtidos através da técnica desenvolvida por Chan, Golub e LeVeque 1979, que consiste no cálculo da média e da variância dos elementos de um conjunto para reduzir o custo computacional do cálculo para conjuntos muito grandes.

#### 2.2.2 Algoritmo Adaptive Boosting

O algoritmo *Adaptive Boosting*, ou *AdaBoost*, se baseia na geração de vários classificadores de baixa precisão, normalmente árvores compostas de um nó de duas folhas, chamados de *stumps*, que são aplicados sobre um conjunto de teste por votação ponderada. Para cada iteração dos testes, o algoritmo altera o peso dos conjuntos de teste com base no erro dos classificadores [Faceli *et al.* 2011].

Durante a execução do algoritmo, para os elementos corretamente classificados, o algoritmo reduz o seu peso, e para os elementos classificados incorretamente o seu peso é incrementado. Este processo é repetido a cada iteração do algoritmo com o objetivo de reduzir o erro do classificador a cada execução. Ao final, o classificador ideal é dado pela combina-

### Algoritmo 2 – Algoritmo AdaBoost

```

1: Entrada:
2: Um algoritmo de aprendizado  $\phi$ 
3: Um conjunto de treinamento  $D = \{(\mathbf{x}_i, y_i), i = 1, \dots, n\}$ 
4: Número de Iterações  $Nr$ 
5: Um conjunto de testes com  $nt$  exemplos  $T = \{(\mathbf{x}_j, ?), j = 1, \dots, nt\}$ 
6: Saída:
7: Previsões para o conjunto de Teste

8: /* Fase de Treinamento */
9: para exemplo  $i \in D$  faz
10:    $w(i) \leftarrow \frac{1}{n}$ 
11: finaliza para
12: para  $l = 1; l < Nr$  faz
13:   para exemplo  $i \in D$  faz
14:      $p_l \leftarrow \frac{w_l(i)}{\sum_i w_l(i)}$ 
15:   finaliza para
16:   /* Chamada do Algoritmo de Aprendizado */
17:    $f_l \leftarrow \phi(p_l)$ 
18:   /* Calculo do Erro */
19:    $e_l = \sum_i p_l(i)[f_l(\mathbf{x}_i) \neq y_i]$ 
20:    $\beta_l \leftarrow \frac{e_l}{1 - e_l}$ 
21:   para exemplo  $i \in D$  faz
22:      $w_{l+1}(i) := w_l(i) \beta_l^{1-[f_l(\mathbf{x}_i) \neq y_i]}$ 
23:   finaliza para
24: finaliza para
25: /* Fase de Teste */
26: para  $j = 1; j < nt$  faz
27:    $y_j = \arg_{\max_y \in Y} \sum_{l=1}^{Nr} (\log(\frac{1}{\beta_l}) [f_l(\mathbf{x}_j \in T) = y])$ 
28: finaliza para
29: retorna Vetor de previsões  $Y$ 

```

**Fonte: [Faceli et al. 2011].**

ção dos classificadores obtidos em cada iteração. O Algoritmo 2, extraído de Faceli *et al.* 2011, apresenta o funcionamento do classificador *AdaBoost*.

Neste algoritmo,  $w(i)$  é a função do peso de cada exemplo para o classificador. Note que inicialmente todos os exemplos possuem o mesmo peso, de acordo com o primeiro *loop* de execução. Logo em seguida, é executado outro *loop* determinado pelo número de iterações do algoritmo, onde  $p_l(i)$  é o peso total para cada exemplo no conjunto de teste,  $f_l$  é o resultado do classificador  $\phi(p_l)$ ,  $e_l$  é a somatória dos pesos dos exemplos incorretamente classificados e  $\beta_l$  é a importância do resultado classificado. No final de cada iteração, o algoritmo atualiza o peso de cada exemplo para a próxima iteração com base no peso atual e na importância do resultado classificado. Por fim, é armazenado todas as classificações em um vetor através de uma somatória de logaritmos da importância de cada parâmetro.

### Algoritmo 3 – Algoritmo SAMME

```

1: Entrada:
2: Um algoritmo de aprendizado  $\phi$ 
3: Um conjunto de treinamento  $D = \{(y_i, \mathbf{x}_i), i = 1, \dots, n\}$ 
4: Número de Iterações  $Nr$ 
5: Um conjunto de testes com  $nt$  exemplos  $T = \{(y_j, ?), j = 1, \dots, nt\}$ 
6: Saída:
7: Previsões para o conjunto de Teste

8: /* Fase de Treinamento */
9: para exemplo  $i \in D$  faz
10:    $w(i) \leftarrow \frac{1}{n}$ 
11: finaliza para
12: para  $l = 1; l < Nr$  faz
13:   para exemplo  $i \in D$  faz
14:      $p_l \leftarrow \frac{w_l(i)}{\sum_i w_l(i)}$ 
15:   finaliza para
16:   /* Chamada do Algoritmo de Aprendizado */
17:    $f_l \leftarrow \phi(p_l)$ 
18:   /* Calculo do Erro */
19:    $e_l = \sum_i p_l(i)[f_l(y_i) \neq \mathbf{x}_i]$ 
20:    $\alpha_l \leftarrow \log(\frac{1 - e_l}{e_l}) + \log(n - 1)$ 
21:   para exemplo  $i \in D$  faz
22:      $w_{l+1}(i) := w_l(i) \exp \alpha_l \phi(f_l(y_i) \neq \mathbf{x}_i)$ 
23:   finaliza para
24: finaliza para
25: /* Fase de Teste */
26: para  $j = 1; j < nt$  faz
27:    $x_j = \arg_{\max_{x \in X}} \sum_{l=1}^{Nr} (\log(\alpha_l [f_l(y_j \in T) = \mathbf{x}])$ 
28: finaliza para
29: retorna Vetor de previsões  $X$ 

```

**Fonte:** Adaptado de [Zhu et al. 2006].

Segundo Faceli et al. 2011, a principal vantagem deste algoritmo é o fato do classificador conseguir lidar com diversos níveis de dificuldade de classificação, com os elementos próximos ao hiperplano possuindo uma dificuldade maior de classificação. Entretanto, é necessário que o algoritmo seja sensível para que consiga gerar diferentes hipóteses de classificação a cada alteração no conjunto de treinamento. Isto se deve ao fato do objetivo do algoritmo ser a redução da variância dos erros dos classificadores, buscando reduzir ao máximo para se obter o melhor classificador possível.

### Algoritmo AdaBoost com SAMME

Para otimizar o algoritmo *AdaBoost* para situações com múltiplas classes, Zhu et al. 2006 implementou o algoritmo Stagewise Additive Modeling using a Multi-class Exponential

loss function (SAMME), que consiste na execução do algoritmo baseado na função de perda exponencial para reduzir o erro dos classificadores fracos utilizados na classificação. Isto ocorre pois o classificador *AdaBoost* opera considerando o erro necessário como superior à uma escolha binária aleatória, ou seja, com a probabilidade de erro inferior a  $1/2$ . Para casos binários, esta classificação é simples e a probabilidade do erro serve para a classificação. Porém, como demonstrado pelo autor, em classificações com mais de duas classes, se obter a probabilidade de erro ser superior a  $1/2$  pode ser custoso.

Desta forma, os autores desenvolveram o algoritmo que, utilizando a função de perda exponencial, passa a considerar a probabilidade de erro necessária como superior a  $1/K$ , com  $K$  sendo a quantidade de classes. Como notado no desenvolvimento do algoritmo, quando  $K = 2$ , o novo algoritmo passa a se comportar como o algoritmo *AdaBoost* tradicional, porém para casos em que  $K > 2$ , o desempenho se apresenta superior, com a probabilidade de erro escalando conforme a quantidade de classes sendo analisadas.

### 2.2.3 Algoritmo SVM

Como parâmetro de comparação entre este trabalho e o trabalho de Tateisi 2022, foi reaplicado o algoritmo classificador SVM sobre o *dataset* UNSW NB15. Esta aplicação serve para validar o desempenho analisado neste trabalho e como os valores das métricas diferem ou não entre os trabalhos. Este algoritmo também serve como intersecção entre os algoritmos aplicados no trabalho de Tateisi 2022 com os aplicados neste trabalho.

Segundo o trabalho de Cortes e Vapnik 1995, as SVM são uma família de algoritmos de *Machine Learning* que buscam encontrar um hiperplano que separe o maior conjunto de vetores em uma das duas classes possíveis através do preprocessamento dos dados para em uma dimensionalidade maior que a dos dados originais [Duda, Hart e Stork 2000]. Esse modelo de classificação pode ser tanto rígido quanto flexível, com diversos mecanismos de alteração do comportamento do classificador para a delimitação do hiperplano.

A forma do hiperplano varia conforme a dimensionalidade do problema. Caso o problema tenha dimensão 1, o hiperplano será um ponto; caso seja 2, será uma curva; caso seja 3, será uma superfície; e caso seja maior que 3, será um hiperplano. Independentemente de sua forma, o seu objetivo continua sendo o de maximizar a margem entre os elementos de classes distintas. Estes elementos são chamados de *Support Vector* (SV) e são eles que definem a forma e tamanho da margem. Esta característica permite que o algoritmo seja menos influenciado pelos elementos distantes, pois os elementos que definem a forma do hiperplano são apenas os elementos SV [Mammone, Turchi e Cristianini 2009, Faceli *et al.* 2011].

Para a aplicação desta técnica, caso os dados sejam linearmente separáveis podemos tanto utilizar o cálculo do classificador de margens rígidas (HMC do inglês *Hard Margin Classifier*), em que a classificação é absoluta, e o cálculo do classificador de margens maleáveis (SMC do inglês *Soft Margin Classifier*), que possui uma variável de folga que permite que da-

dos estejam entre as margens ou que dados sejam classificados incorretamente, para que o erro do todo seja reduzido. Caso contrário é necessário implementar o classificador não linear com o uso de *Kernel Tricks* para executar a sua classificação [Duda, Hart e Stork 2000, Faceli *et al.* 2011]. Em ambos os casos, o objetivo é separar os dados do conjunto em duas classes distintas tal que os seus valores pertençam ao conjunto  $Y = \{-1, +1\}$ .

### Classificador de Margens Rígidas

Segundo o trabalho de Faceli *et al.* 2011, considerando  $\mathbf{X}$  como o conjunto de treinamento com  $n$  dados, tal que  $\mathbf{x}_i \in \mathbf{X}$ , cada qual associado a um elemento  $y_i$  de saída, onde  $y_i \in Y$ , temos que o cálculo do hiperplano se dá através Equação 5 da abaixo.

$$h(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b \quad (5)$$

Onde  $\mathbf{w} \cdot \mathbf{x}$  é o produto escalar entre os vetores  $\mathbf{x}$  e  $\mathbf{w}$ , tal que  $\mathbf{w} \in \mathbf{X}$  é o vetor normal ao hiperplano descrito e  $\frac{b}{\|\mathbf{w}\|}$  é a distância do hiperplano até a origem.

Podemos ainda reescrever a Equação 5, visto que o resultado será dado por  $y$ . Assim, temos a Equação 6

$$g(\mathbf{x}) = \text{sgn}(h(\mathbf{x})) = \begin{cases} +1 & \text{se } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ -1 & \text{se } \mathbf{w} \cdot \mathbf{x} + b < 0 \end{cases} \quad (6)$$

Ou ainda,

$$\begin{cases} \mathbf{w} \cdot \mathbf{x} + b \geq +1 & \text{se } y_i = +1 \\ \mathbf{w} \cdot \mathbf{x} + b \leq -1 & \text{se } y_i = -1 \end{cases} = y_i(\mathbf{w} \cdot \mathbf{x} + b) - 1 \geq 0, \forall (\mathbf{x}_i, y_i) \in \mathbf{X} \quad (7)$$

Para se obter a maior margem possível, podemos minimizar  $\|\mathbf{w}\|$ , obtendo o problema de otimização abaixo.

$$\text{Minimizar} \quad \frac{1}{2} \|\mathbf{w}\|^2 \quad (8)$$

$$\text{Com restrições: } y_i(\mathbf{w} \cdot \mathbf{x} + b) - 1 \geq 0, \forall i = 1, \dots, n \quad (9)$$

Para resolver este problema, utilizamos os coeficientes de Lagrange  $\alpha$  para obter uma função lagrangiana que relaciona o problema de otimização com as restrições, como a Equação 10 descrita abaixo.

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i (y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1) \quad (10)$$

Derivando a função lagrangiana em relação a  $b$  e  $\mathbf{w}$  para encontrar o seu mínimo, temos que,

$$\sum_{i=1}^n \alpha_i y_i = 0 \quad (11)$$

$$\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \quad (12)$$

Substituindo na Equação 10, obtemos a Equação 13 abaixo.

$$\text{Maximizar } \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \quad (13)$$

$$\text{Com as restrições: } \begin{cases} \alpha_i \geq 0, \forall i = 1, \dots, n \\ \sum_{i=1}^n \alpha_i y_i = 0 \end{cases} \quad (14)$$

Para a solução deste problema de otimização, podemos utilizar a propriedade da formulação dual e primal, com as condições de Kühn-Tucker, resolvendo primeiramente para o problema dual, e utilizando o resultado para a solução do problema primal. Desta forma, a Equação 13 e a Equação 14 representam a forma dual, com o problema original sendo a forma primal.

Considerando  $\alpha^*$  como solução da forma dual, e  $b^*$  e  $\mathbf{w}^*$  solução da forma primal, temos que com o valor de  $\alpha^*$  podemos utilizar a Equação 12 para encontrar o valor de  $\mathbf{w}^*$ . Ainda, através das condições de Kühn-Tucker e de  $\alpha^*$ , podemos reescrever a forma dual como a Equação 15.

$$\alpha_i^* (y_i (\mathbf{w}^* \cdot \mathbf{x}_i + b^*) - 1) = 0, \forall i = 1, \dots, n \quad (15)$$

Analizando a Equação 15, temos que para  $\alpha_i^* = 0$ , os elementos estão classificados em suas respectivas classes e, portanto não contribuem para o cálculo de  $\mathbf{w}^*$ . Para  $\alpha_i^* > 0$  os elementos são responsáveis pela construção e definição do hiperplano e são eles os elementos chamados de vetores de suporte SV. Para  $b^*$ , utilizamos a Equação 16 abaixo. Nela é calculado a média de todos os elementos  $\mathbf{x}_j$  tal que  $\alpha_j^* > 0$ . Onde  $n_{SV}$  é o número de SV e  $SV$  representa o conjunto dos SV.

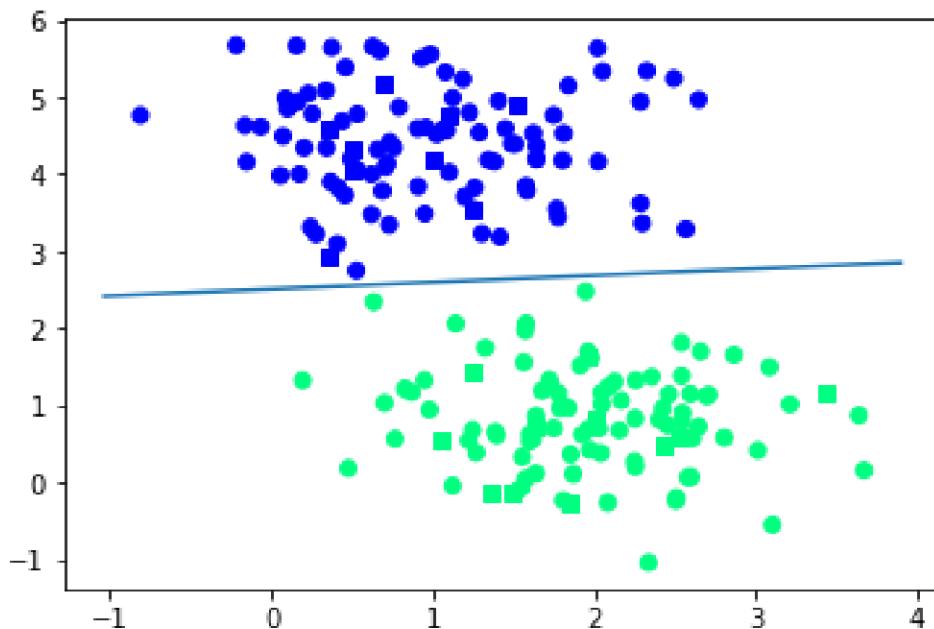
$$b^* = \frac{1}{n_{SV}} \sum_{x_j \in SV} \frac{1}{y_j} - \mathbf{w}^* \cdot \mathbf{x}_j = \frac{1}{n_{SV}} \sum_{x_j \in SV} \left( \frac{1}{y_j} - \sum_{x_i \in SV} \alpha_i^* y_i \mathbf{x}_i \cdot \mathbf{x}_j \right) \quad (16)$$

Com essas equações, temos que o problema inicial pode ser reescrito de acordo com a Equação 17.

$$g(\mathbf{x}) = \text{sgn}(h(\mathbf{x})) = \text{sgn}\left(\sum_{\mathbf{x}_i \in SV} y_i \alpha_i^* \mathbf{x}_i \cdot \mathbf{x} + b^*\right) \quad (17)$$

Assim, com a utilização destas equações é calculado a posição do classificador, com a Figura 1 abaixo apresentando graficamente o funcionamento do Classificador de Margens Rígidas sobre um conjunto de dados aleatórios e o Algoritmo 4 apresentando o pseudo código do classificador.

**Figura 1 – Exemplo do Classificador de Margens Rígidas SVM**



**Fonte:** Autor, 2022.

### Classificador de Margens Maleáveis

Embora o Classificador de Margens Rígidas seja interessante, o classificador pode ser sujeito aos efeitos de *bias* no conjunto. Para corrigir este problema, é utilizado uma nova variável no problema de otimização ( $\xi$ ) que é a tolerância do algoritmo em relação a *outliers* da classificação [Faceli *et al.* 2011].

Desta forma, as equações que definem o Classificador de Margens Maleáveis continuam as mesmas, porém com a presença da tolerância, como demonstrado na Equação 18 abaixo que permite que elementos possam ser classificados incorretamente quando analisado o conjunto como um todo.

$$y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq 1 - \xi_i, \xi_i \geq 0, \forall i = 1, \dots, n \quad (18)$$

#### Algoritmo 4 – SVM Hard Margin Classifier

```

1: Entrada: Dados de treinamento  $X$  e rótulos  $Y$ 
2: Saída: Vetor de pesos  $w$  e bias  $b$ 

3: Inicializar  $w$  e  $b$  com zeros
4: Definir taxa de aprendizado  $\eta$ 
5: Realizar o treinamento do modelo:
6: enquanto existir algum exemplo classificado incorretamente faca
7:   para cada exemplo  $(x_i, y_i)$  faca
8:     se  $y_i(w \cdot x_i + b) \leq 0$  então
9:       Atualizar  $w \leftarrow w + \eta y_i x_i$ 
10:      Atualizar  $b \leftarrow b + \eta y_i$ 
11:    finaliza se
12:  finaliza para
13: finaliza enquanto
14: retorna  $w$  e  $b$ 

```

**Fonte: Autor, 2023.**

Com o problema de otimização,

$$\text{Minimizar} \frac{1}{2} \|\mathbf{w}\|^2 + C \left( \sum_{i=1}^n \xi_i \right) \quad (19)$$

Onde  $C$  é o peso atribuído à tolerância.

Ao aplicarmos as constantes de Lagrange e construirmos a função lagrangiana, obtemos o problema de otimização abaixo.

$$\text{Maximizar} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \quad (20)$$

$$\text{Com as restrições: } \begin{cases} 0 \leq \alpha_i \leq C, \forall i = 1, \dots, n \\ \sum_{i=1}^n \alpha_i y_i = 0 \end{cases} \quad (21)$$

Este problema é idêntico ao classificador de margens rígidas, a não ser pela variável  $C$  que limita o valor possível de  $\alpha_i$ . Desta forma, considerando  $\alpha^*$  como solução da forma dual, e  $\mathbf{w}^*$ ,  $b^*$  e  $\xi^*$  são soluções da forma primal. O vetor  $\mathbf{w}^*$  continua sendo definido pela Equação 12. Para calcular  $\xi_i^*$ , utilizamos a Equação 22.

$$\xi_i^* = \max(0, 1 - y_i \sum_{j=1}^n y_j \alpha_j^* \mathbf{x}_j \cdot \mathbf{x}_i + b^*) \quad (22)$$

De semelhante modo, para  $b^*$ , utilizamos a Equação 23 e a Equação 24 para obter o seu valor.

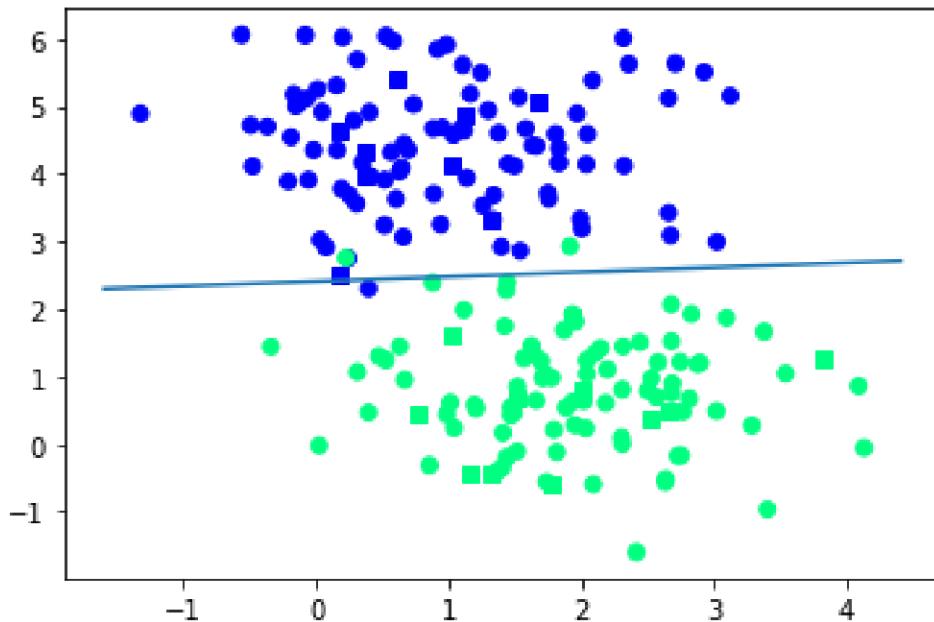
$$\alpha_i^* (y_i (\mathbf{w}^* \cdot \mathbf{x}_i + b^*) - 1 + \xi_i^*) = 0, \forall i = 1, \dots, n \quad (23)$$

$$(C - \alpha_i^*) \xi_i^* = 0 \quad (24)$$

Da forma semelhante ao método de margens rígidas,  $\alpha_i^* > 0$  define os vetores de suporte. Entretanto, no método de margens flexíveis, temos tipos diferentes de SV. Se  $\alpha_i^* < C$ , segundo a Equação 24,  $\xi_i^* = 0$ , o que implica pela Equação 23 que esses elementos estão sobre as margens. Caso  $\alpha_i^* > C$ , a variável  $\xi_i^*$  define três casos: se  $\xi_i^* > 1$ , os pontos são erros; se  $0 < \xi_i^* \leq 1$ , os pontos estão corretamente classificados, porém entre as margens; e se  $\xi_i^* = 0$  os pontos estão sobre as margens do hiperplano [Faceli et al. 2011].

Para o cálculo de  $b^*$  utilizamos a Equação 16 para todo  $\alpha_i^* < C$ . Desta forma, a solução do problema é a mesma dada pela Equação 17, porém as variáveis  $\alpha_i^*$  são definidas pela Equação 20 [Faceli et al. 2011]. A Figura 2 abaixo apresenta graficamente o funcionamento do classificador de margens maleáveis e o Algoritmo 4 apresentando o pseudo código do classificador.

**Figura 2 – Exemplo do Classificador de Margens Maleáveis SVM**



**Fonte:** Autor, 2022.

### Classificador SV Não-Linear

Os classificadores de margens rígidas e de margens flexíveis são classificadores lineares, ou seja, assumem que o problema dado seja linearmente separável. Porém, as situações por vezes não são linearmente separáveis, necessitando de técnicas para a aplicação do algoritmo nesses casos. Para resolver esta situação, todo o conjunto de treinamento é mapeado em um espaço de maior dimensão. Considerando  $\Phi : X \rightarrow \mathbb{S}$ , um mapeamento dos dados,

### Algoritmo 5 – SVM Soft Margin Classifier

```

1: Entrada: Dados de treinamento  $X$  e rótulos  $Y$ , parâmetro de regularização  $C$ 
2: Saída: Vetor de pesos  $w$  e bias  $b$ 

3: Inicializar  $w$  e  $b$  com zeros
4: Definir taxa de aprendizado  $\eta$ 
5: Definir número máximo de iterações  $max\_iter$ 
6: Definir tolerância para a margem  $tol$ 
7: Realizar o treinamento do modelo:
8: para  $iter$  de 1 até  $max\_iter$  faca
9:   para cada exemplo  $(x_i, y_i)$  faca
10:    se  $y_i(w \cdot x_i + b) < 1$  então
11:      Atualizar  $w \leftarrow (1 - \eta)w + \eta Cy_i x_i$ 
12:      Atualizar  $b \leftarrow b + \eta Cy_i$ 
13:    senão,
14:      Atualizar  $w \leftarrow (1 - \eta)w$ 
15:    finaliza se
16:  finaliza para
17:  se  $\|w\| < tol$  então
18:    Parar o treinamento
19:  finaliza se
20: finaliza para
21: retorna  $w$  e  $b$ 

```

**Fonte: Autor, 2023.**

temos que escolhendo  $\Phi$  corretamente permite que o conjunto em  $\mathcal{S}$  possa ser separado pelo SVM [Faceli et al. 2011].

Desta forma, podemos reescrever o problema de otimização dado pela Equação 13 segundo a Equação 25 com o classificador dado pela Equação 26 e o cálculo de  $b^*$  pela Equação 27.

$$\text{Maximizar} \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)) \quad (25)$$

$$g(\mathbf{x}) = \text{sgn}(h(\mathbf{x})) = \text{sgn}\left(\sum_{x_i \in SV} y_i \alpha_i^* \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}) + b^*\right) \quad (26)$$

$$b^* = \frac{1}{n_{SV}} \sum_{x_j \in SV} \left( \frac{1}{y_j} - \sum_{x_i \in SV} \alpha_i^* y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \right) \quad (27)$$

O *Kernel Trick* utiliza das funções *Kernel* para simplificar e reduzir o gasto de recursos empregados na execução do mapeamento  $\Phi$ , visto que o espaço  $\mathcal{S}$  pode ter dimensão infinita [Faceli et al. 2011]. Assim, a função *Kernel* recebe dois parâmetros tal que,

$$K(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) \quad (28)$$

### Algoritmo 6 – SVM Não Linear com Kernel

```

1: Entrada: Dados de treinamento  $X$  e rótulos  $Y$ , parâmetro de regularização  $C$ , função de kernel  $\phi$ 
2: Saída: Vetor de pesos  $w$  e bias  $b$ 

3: Inicializar  $w$  e  $b$  com zeros
4: Definir taxa de aprendizado  $\eta$ 
5: Definir número máximo de iterações  $max\_iter$ 
6: Definir tolerância para a margem  $tol$ 
7: Realizar o treinamento do modelo:
8: para  $iter$  de 1 até  $max\_iter$  faz
9:   para cada exemplo  $(x_i, y_i)$  faz
10:    se  $y_i(w \cdot \phi(x_i) + b) < 1$  então
11:      Atualizar  $w \leftarrow (1 - \eta)w + \eta Cy_i\phi(x_i)$ 
12:      Atualizar  $b \leftarrow b + \eta Cy_i$ 
13:    senão,
14:      Atualizar  $w \leftarrow (1 - \eta)w$ 
15:    finaliza se
16:  finaliza para
17:  se  $\|w\| < tol$  então
18:    Parar o treinamento
19:  finaliza se
20: finaliza para
21: retorna  $w$  e  $b$ 

```

**Fonte: Autor, 2023.**

A vantagem desta técnica é que a escolha e cálculo de  $\Phi(\mathbf{x})$  é implícita, além do fato que o cálculo do *Kernel* é relativamente simples e permite representar espaços abstratos [Faceli et al. 2011].

Para manter as características do problema de otimização, é necessário que as funções *Kernel* obedeçam as condições do teorema de Mercer [Faceli et al. 2011]. Deste teorema temos que a função *Kernel* é responsável por gerar matrizes semidefinidas  $K$  em que cada elemento  $K_{i,j}$  é definido por  $K_{i,j} = K(\mathbf{x}_i, \mathbf{x}_j)$  para todo  $i, j = 1, \dots, n$ .

Dentre as funções *Kernel* comuns estão os *Kernels* polinomiais, de função de base radial e os sigmodais apresentados no Quadro 1 abaixo.

**Quadro 1 – Funções Kernel**

<b>Tipo de Kernel</b>	<b>Função <math>K(\mathbf{x}_i, \mathbf{x}_j)</math></b>
Polinomial	$(\delta(\mathbf{x}_i, \mathbf{x}_j) + \kappa)^d$
Função de Base Radial	$\exp(-\sigma \ \mathbf{x}_i - \mathbf{x}_j\ ^2)$
Sigmoidal	$\tanh(\delta(\mathbf{x}_i \cdot \mathbf{x}_j) + \kappa)$

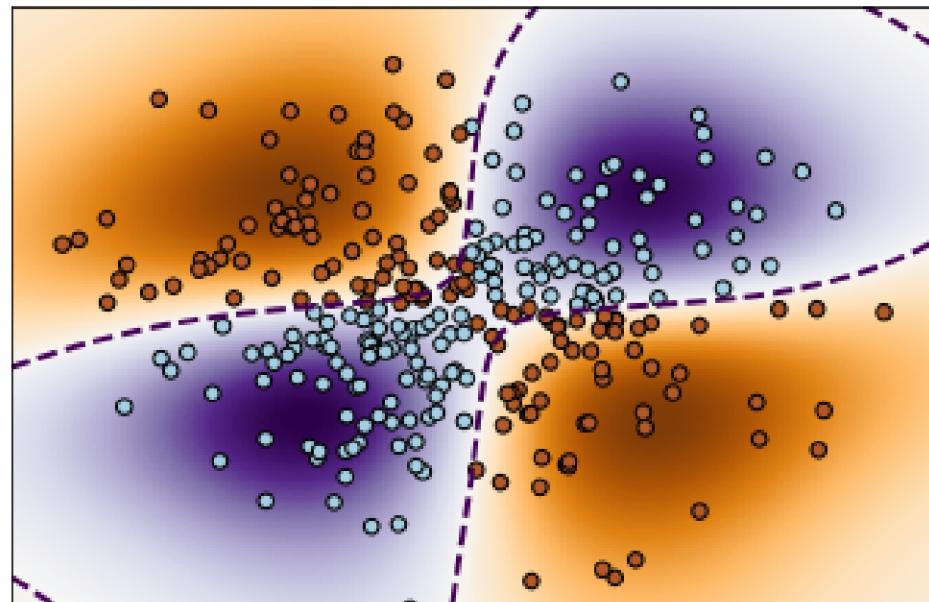
**Fonte: Faceli et al. 2011.**

Assim, o classificador é reescrito de acordo com a Equação 29 e a Equação 30 abaixo, resultando nos hiperplanos representados na Figura 3.

$$g(\mathbf{x}) = \operatorname{sgn}(h(\mathbf{x})) = \operatorname{sgn}\left(\sum_{\mathbf{x}_i \in SV} y_i \alpha_i^* K(\mathbf{x}_i, \mathbf{x}_j) + b^*\right) \quad (29)$$

$$b^* = \frac{1}{n_{SV}} \sum_{x_j \in SV} \left( \frac{1}{y_j} - \sum_{x_i \in SV} \alpha_i^* y_i K(\mathbf{x}_i, \mathbf{x}_j) \right) \quad (30)$$

**Figura 3 – Exemplo do Classificador SVM Não-Linear**



Fonte: Autor, 2022.

### 3 TRABALHOS RELACIONADOS

Singh e Banerjee 2020 aplicaram a comparação entre os algoritmos de Naïve-Bayes, SVM e Árvores de Decisão sobre o *dataset* NSL-KDD. Após o pré-processamento dos dados para a redução da dimensionalidade, os dados foram aplicados ao aos classificadores e os resultados foram classificados por sua acurácia e precisão que formam a base do cálculo do *f-measure*. Os resultados obtidos foram de que a técnica de árvore de Decisão apresentou a melhor acurácia com ambos os algoritmos de redução de dimensionalidade.

Já Liu *et al.* 2008 buscaram implementar o conceito de *time slicing* no algoritmo de Naïve-Bayes, levando o horário de aquisição do dado como parâmetro classificador para se aproveitar das características do tráfego com base na variação do horário. Desta forma, os autores conseguiram melhorar o desempenho da detecção do sistema em relação ao método não modificado, com os autores aprovando a sua implementação em ambientes complexos de redes.

Semelhantemente, Swarnkar e Hubballi 2016 buscaram aprimorar o algoritmo de Naïve-Bayes com a implementação do método *One Class Payload Anomaly Detection* (OCPAD), que busca otimizar o espaço utilizado focando a detecção das anomalias no conteúdo dos pacotes trafegando na rede. Porém, visto que a técnica demanda alto poder computacional, os autores recomendam o seu uso em um sistema de multicamadas.

Já Catania, Bromberg e Garino 2012 utilizaram algoritmos classificadores baseados em SVM em uma aplicação não-supervisionada para detectar ataques. Neste trabalho, os autores confirmaram as limitações do uso de SVM em *datasets* com alta porcentagem de ataques, com a técnica *SNORT-based SVM* (SbSVM). Neste método, os *datasets* foram pré-processados com o algoritmo SNORT, um algoritmo IDS baseado em assinatura, para remover ataques já conhecidos do *dataset*, permitindo o SbSVM focar na detecção de ataques desconhecidos.

Wang, Gu e Wang 2017 propuseram a técnica do *Logarithm Marginal Density Ratios Transformation* (LMDRT) com o SVM para melhorar o desempenho em *datasets* de alta dimensionalidade. Para isso, o LMDRT condensa os dados do *dataset*, o que permite reduzir o tempo gasto com o treinamento do algoritmo e, como consequência, o gasto computacional.

Kabir *et al.* 2018 propuseram um algoritmo para aprimorar a técnica *Least Square Support Vector Machine* (LS-SVM) com o pré-processamento dos dados através da amostragem de subdivisões arbitrárias do *dataset*. Com isso, através da variabilidade das amostras, é construído um esquema de *Optimal Allocation* para cada subdivisão, e estes dados são então inseridos no LS-SVM. O algoritmo proposto pelos autores foi validado através do *dataset* KDD99 e se apresentou como uma eficiente forma para detecção de anomalias para diversos tipos de ataques presentes no *dataset*.

Li e Li 2010 apresentam o uso do algoritmo de Naïve-Bayes como algoritmo classificador fraco sobre o *dataset* KDD99. Esta forma de aplicação dos algoritmos apresentam resultados

competitivos nos testes realizados, com a precisão e a sensibilidade do algoritmo se mostrando promissores e condizentes com o esperado.

A aplicação de Yuan, Kaklamanos e Hogrefe 2016 inova ao implementar três formas do algoritmo *AdaBoost* para analisar como as diferentes técnicas competem entre si. Foram implementadas *stumps* de decisão, que são compostas por um nó e duas folhas, como os algoritmos classificadores fracos, onde foram executadas 30 iterações de cada algoritmo sobre o *dataset* KDD99. Destes dados, foram analisados o desempenho do algoritmo com diferentes taxas de dados classificados e não-classificados para o aprendizado semi-supervisionado, com os resultados mostrando que mesmo com apenas 1% dos dados classificados, o algoritmo apresenta uma alta precisão e taxa de detecção.

Hu, Hu e Maybank 2008 apresentam a implementação do algoritmo *AdaBoost* sobre o *dataset* KDD99, utilizando como algoritmo classificador fraco *stumps* de decisão. O trabalho permite que ao ajustar o *overfitting* do algoritmo de classificação, pode se controlar o *tradeoff* entre a taxa de detecção e a taxa de falso-positivos detectados pelo algoritmo.

Por fim, Tateisi 2022 apresenta a aplicação dos algoritmos de Floresta Aleatória, SVM e Redes Neurais Artificiais sobre o *dataset* UNSW NB15. Utilizando como base esta aplicação, temos que as aplicações dos algoritmos apresentaram alta precisão e sensibilidade em todas as execuções. Isto aponta que o *dataset* utilizado apresenta a capacidade de alta performance para os algoritmos trabalhados, com outros algoritmos podendo estabelecer um novo *benchmark* para a análise do desempenho de algoritmos classificadores em Sistemas de Detecção de Intrusão.

Abaixo no Quadro 2 estão resumidos os trabalhos relacionados ao tema.

**Quadro 2 – Resumo de Trabalhos Relacionados**

<b>Autor</b>	<b>Algoritmo</b>	<b>Dataset</b>
Singh e Banerjee 2020	Naïve-Bayes, SVM e Árvores de Decisão	NSL-KDD
Liu <i>et al.</i> 2008	Naïve-Bayes	DARPA1998
Swarnkar e Hubballi 2016	Naïve-Bayes com OCPAD	(Não citado)
Catania, Bromberg e Garino 2012	<i>SNORT-based SVM</i>	DARPA1998
Wang, Gu e Wang 2017	LMDRT	NSL-KDD
Kabir <i>et al.</i> 2018	LS-SVM	KDD99
Li e Li 2010	<i>AdaBoost</i> com Naïve-Bayes como Classificador Fraco	KDD99
Yuan, Kaklamanos e Hogrefe 2016	<i>AdaBoost</i> Semi-Supervisionado	KDD99
Hu, Hu e Maybank 2008	<i>AdaBoost</i>	KDD99
Tateisi 2022	Floresta Aleatória, SVM e Redes Neurais Artificiais	UNSW-NB15
<b>Este trabalho</b>	<b>Naïve-Bayes, AdaBoost e SVM</b>	UNSW-NB15

**Fonte: Autor, 2022.**

## 4 MATERIAIS E MÉTODOS

Neste capítulo serão discutidos a escolha do *dataset* empregado, os parâmetros empregados para treinamento e teste dos algoritmos, as métricas de avaliação e a linguagem de programação utilizada para o desenvolvimento.

### 4.1 Materiais

Os principais materiais envolvidos no trabalho foram os dados para treinamento e teste, com estes sendo originados dos trabalhos de Moustafa e Slay 2015, Moustafa e Slay 2016 e Moustafa 2017, da *Canadian Institute of Cybersecurity* (CIC). Além dos dados, o trabalho também utilizou o Jupyter Notebooks com a linguagem Python para a construção dos algoritmos dos modelos, com a utilização da biblioteca Scikit Learn e os métodos nele implementados.

#### 4.1.1 Dataset

Segundo Renear, Sacchi e Wickett 2010, a definição para *dataset* é baseada em uma ideia comum em um alto nível de generalização, porém que depende no contexto e na comunidade na qual é utilizado. Para uma definição formal, *dataset* está relacionado com agrupamento, conteúdo, relacionamento e propósito [Renear, Sacchi e Wickett 2010].

Para a análise de desempenho de algoritmos de detecção de intrusão, é necessário a utilização de *datasets* para treinamento e validação. Neste contexto, o *dataset* KDDCUP99 e sua modificação KDD-NSL são utilizados como referência para a avaliação do desempenho de algoritmos para detecção de anomalias. Este *dataset*, no entanto, foi gerado em 1999, baseado no *dataset* DARPA 1998, o que limita a sua acurácia dado os avanços na área de segurança e rede em 20 anos [Tavallaei et al. 2009].

Para eliminar esta limitação de idade do *dataset* KDD99 e KDD-NSL, os trabalhos de Moustafa e Slay 2015 e Sharafaldin, Lashkari e Ghorbani 2018 apresentam o desenvolvimento de novos *datasets* para melhorar a acurácia da análise de desempenho de algoritmos, com o *dataset* UNSW NB15 de Moustafa e Slay 2015 e o CIC IDS2018 de Sharafaldin, Lashkari e Ghorbani 2018 buscando reduzir esta lacuna tecnológica.

Na Tabela 1 abaixo estão descritas as razões de dados normais e anormais em cada *dataset*.

Observando que os *datasets* mais recentes apresentam uma alta taxa de tráfego normal, para o trabalho, será utilizado o *dataset* UNSW NB15 para análise dos algoritmos devido a sua alta taxa de tráfego normal e as características da rede simulada [Moustafa e Slay 2015, Moustafa e Slay 2016, Moustafa 2017].

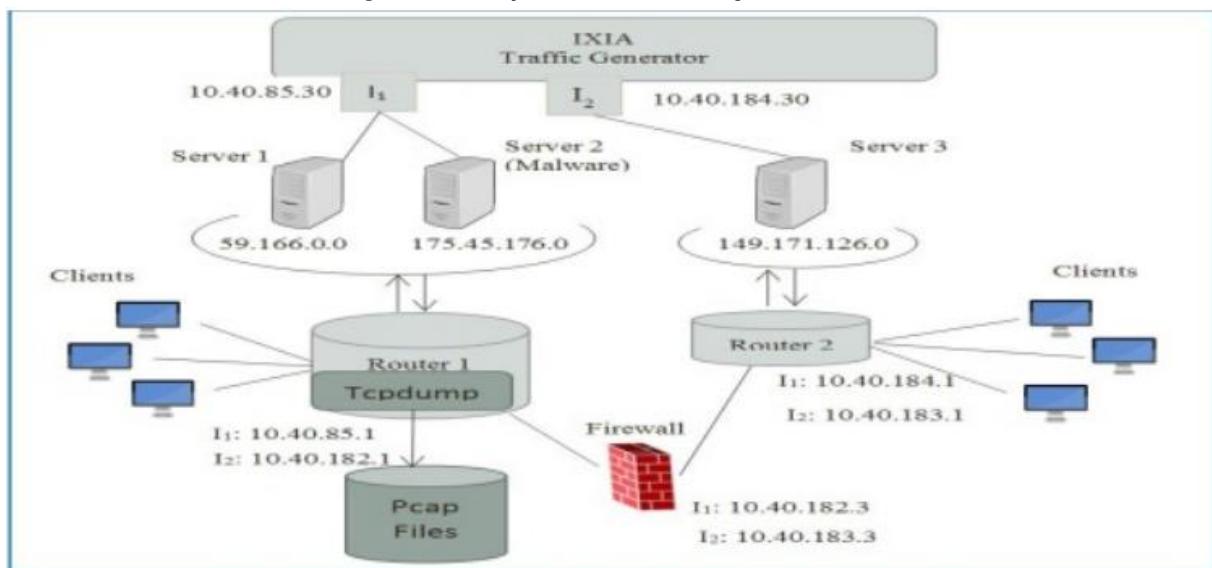
**Tabela 1 – Porcentagem de dados normais e anormais nos datasets.**

	KDD-99	CIC-IDS2018	UNSW-NB15
Normal	19,86	83,07	87,35
Anormal	80,14	16,93	12,65

**Fonte:** Adaptado de [Tavallaei et al. 2009, Sharafaldin, Lashkari e Ghorbani 2018, Moustafa e Slay 2015].

Em seu trabalho, Moustafa e Slay 2016 compararam o *dataset* UNSW NB15 desenvolvido com o principal *dataset* empregado até então para o *benchmark* dos algoritmos de classificação na área de segurança, o *dataset* KDD99 de 1999. Neste trabalho, através do emprego de explicação estatística, correlação de características e a capacidade de análise complexa através da execução de cinco algoritmos, foi observado a alta capacidade do *dataset* em representar cenários realísticos e que se aproximam do cenário real enfrentado,

Este *dataset* foi gerado através do uso da ferramenta *IXIA PerfectStorm* para simular tráfego normal e anômalo em uma rede simulada composta por três servidores, com o tráfego sendo capturado no Roteador 1 pela ferramenta TCPDump no formato de pacotes .pcap totalizando 100 Gigabytes (GBs) de dados como mostrado na Figura 4 [Moustafa e Slay 2015]. Estes dados foram processados gerando o *dataset*, com um total de 47 características separadas de acordo com o Quadro 3 ao Quadro 8 apresentados, onde o tipo da característica segue que N = Nominal, I = Inteiro, F = Decimal, T = Registro Temporal e B = Binário.

**Figura 4 – Esquema de Construção da Rede**

**Fonte:** Moustafa e Slay 2015.

O Quadro 3 apresenta as características do tráfego, como os endereços IP de origem e destino e as portas de origem e destino dos pacotes.

**Quadro 3 – Características do Fluxo**

#	Nome	Tipo	Descrição
1	<i>srcip</i>	Nominal	Endereço IP de Origem
2	<i>sport</i>	Inteiro	Porta de Origem
3	<i>dstip</i>	Nominal	Endereço IP de Destino
4	<i>dsport</i>	Inteiro	Porta de Destino
5	<i>proto</i>	Nominal	Protocolo de Transferência

**Fonte:** Adaptado de Moustafa e Slay 2015.

Já o Quadro 4 e o Quadro 5 apresentam as características do conteúdo da mensagem, e o Quadro 6 apresentando as características temporais do tráfego.

**Quadro 4 – Características Básicas de Transmissão**

#	Nome	Tipo	Descrição
6	<i>state</i>	Nominal	O estado e seu protocolo dependente (Ex.: ACC, CLO, ou (-))
7	<i>dur</i>	Decimal	Duração total de transmissão
8	<i>sbytes</i>	Inteiro	Bytes da requisição
9	<i>dbytes</i>	Inteiro	Bytes da resposta
10	<i>sttl</i>	Inteiro	Tempo de vida do pacote da requisição
11	<i>dttl</i>	Inteiro	Tempo de vida do pacote da resposta
12	<i>sloss</i>	Inteiro	Pacotes da requisição retransmitidos ou perdidos
13	<i>dloss</i>	Inteiro	Pacotes da resposta retransmitidos ou perdidos
14	<i>service</i>	Nominal	Serviço (http, ftp, ssh, dns, ...)
15	<i>sload</i>	Decimal	Taxa de bits por segundo da origem
16	<i>dload</i>	Decimal	Taxa de bits por segundo do destino
17	<i>spkt</i>	Inteiro	Quantidade de pacotes da requisição
18	<i>dpkt</i>	Inteiro	Quantidade de pacotes da resposta

**Fonte:** Adaptado de Moustafa e Slay 2015.

**Quadro 5 – Características do Conteúdo**

#	Nome	Tipo	Descrição
19	<i>swin</i>	Inteiro	Valor do anúncio da janela TCP de origem
20	<i>dwin</i>	Inteiro	Valor do anúncio da janela TCP de destino
21	<i>stcpb</i>	Inteiro	Número de sequência TCP da origem
22	<i>dtcpb</i>	Inteiro	Número de sequência TCP do destino
23	<i>smeansz</i>	Inteiro	Média do tamanho do pacote de dados transmitido pela origem
24	<i>dmeansz</i>	Inteiro	Média do tamanho do pacote de dados transmitido pelo destino
25	<i>trans_depth</i>	Inteiro	A profundidade da conexão da transação HTTP de requisição/resposta
26	<i>res_bdy_len</i>	Inteiro	O tamanho do conteúdo dos dados transferidos pelo servidor de serviços HTTP

**Fonte:** Adaptado de Moustafa e Slay 2015.

Através das características anteriores foi possível analisar novas características no Quadro 7 e Quadro 8. Estas novas características são utilizadas para associar as características do

**Quadro 6 – Características Temporais**

#	Nome	Tipo	Descrição
27	<i>sjit</i>	Inteiro	Jitter da origem (ms)
28	<i>djit</i>	Inteiro	Jitter do destino (ms)
29	<i>stime</i>	Inteiro	Tempo inicial de gravação
30	<i>ltime</i>	Inteiro	Tempo final de gravação
31	<i>sintpkt</i>	Inteiro	Tempo de chegada entre-pacotes da origem (ms)
32	<i>dintpkt</i>	Inteiro	Tempo de chegada entre-pacotes do destino (ms)
33	<i>tcprtt</i>	Inteiro	A somatória do <i>synack</i> e do <i>ackdat</i> do TCP
34	<i>synack</i>	Inteiro	Tempo entre os pacotes de SYN e SYN_ACK do TCP
35	<i>ackdat</i>	Inteiro	Tempo entre os pacotes de SYN_ACK e ACK do TCP

**Fonte:** Adaptado de Moustafa e Slay 2015.

fluxo de forma quantitativa para detectar formas de ataque baseadas na sobrecarga de requisições por endereços de IP ou portas específicas [Moustafa e Slay 2015, Moustafa e Slay 2016, Moustafa 2017].

**Quadro 7 – Características de Propósito Geral**

#	Nome	Tipo	Descrição
36	<i>is_sm_ips_ports</i>	Binário	Se os endereços IP da origem (1) e do destino (3) e as portas da origem (2) e destino (4) são iguais, então 1 (Verdadeiro), caso contrário, 0 (Falso)
37	<i>ct_state_ttl</i>	Inteiro	Número para cada estado (6) de acordo com um intervalo de valores para o tempo de sobrevivência dos pacotes de origem (10) e destino (11)
38	<i>ct_flw_http_mthd</i>	Inteiro	Número de fluxos que possuem métodos como GET e POST no serviço HTTP
39	<i>is_ftp_login</i>	Binário	Se a sessão FTP é acessada com um usuário e senha, então 1. Caso contrário, 0
40	<i>ct_ftp_cmd</i>	Inteiro	Tempo de chegada entre-pacotes da origem (ms)

**Fonte:** Adaptado de Moustafa e Slay 2015.

Desta forma, o *dataset* gerado consegue representar o tráfego de dados em uma rede considerando os avanços nas características dos ataques. Isto permite que os NIDS testados sobre este *dataset* apresentem um desempenho com maior acurácia quando utilizados para analisar o tráfego de uma rede real [Moustafa e Slay 2015, Moustafa e Slay 2016, Moustafa 2017].

#### 4.1.2 Ferramentas Utilizadas

Para a construção das análises e dos algoritmos foi utilizada a linguagem *Python* com o *Jupyter notebook* para a escrita dos algoritmos. A escolha da linguagem se deu pela disponibilidade de bibliotecas de Ciência de Dados e ML como as bibliotecas *Pandas*, *Numpy* e *Scikit-learn*.

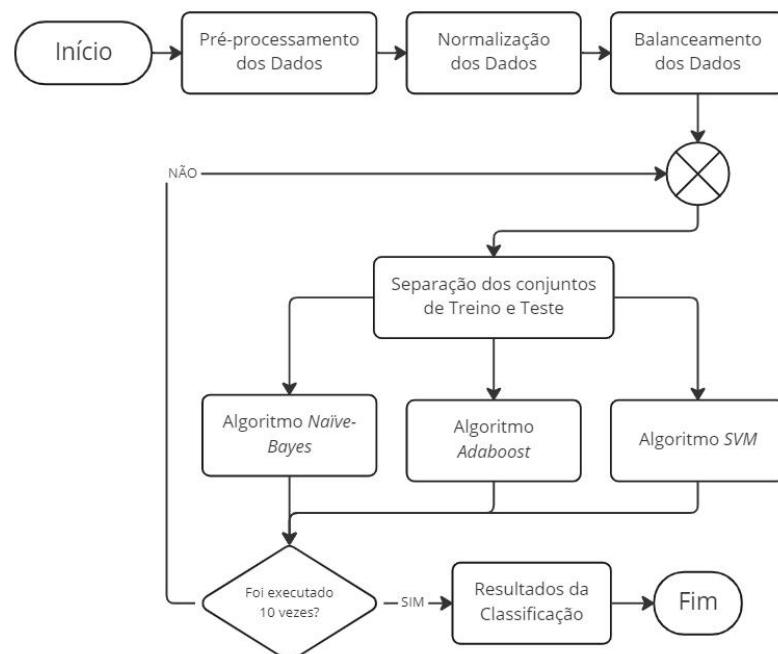
**Quadro 8 – Características de Conexão**

#	Nome	Tipo	Descrição
41	<i>ct_srv_src</i>	Inteiro	Número de conexões que contém o mesmo serviço (14) e endereço de origem (1) em cada 100 conexões de acordo com o tempo final
42	<i>ct_srv_dst</i>	Inteiro	Número de conexões que contém o mesmo serviço (14) e endereço de destino (3) a cada 100 conexões de acordo com o tempo final
43	<i>ct_dst_ltm</i>	Inteiro	Número de conexões do mesmo endereço de destino (3) a cada 100 conexões no último período
44	<i>ct_src_ltm</i>	Inteiro	Número de conexões do mesmo endereço de origem (1) a cada 100 conexões no último período
45	<i>ct_src_dport_ltm</i>	Inteiro	Número de conexões do mesmo endereço de origem (1) e porta de destino (4) a cada 100 conexões
46	<i>ct_dst_sport_ltm</i>	Inteiro	Número de conexões do mesmo endereço de destino (3) e porta de origem (2) a cada 100 conexões
47	<i>ct_dst_src_ltm</i>	Inteiro	Número de conexões com o mesmo endereço de origem (1) e destino (2) a cada 100 conexões no último período

**Fonte:** Adaptado de Moustafa e Slay 2015.

Para controle do desenvolvimento do algoritmo, foi utilizada a plataforma GitHub para controle de versões e implementação dos algoritmos nos repositórios online. Esta abordagem visou facilitar futuros trabalhos sobre os algoritmos implementados.

## 4.2 Métodos

**Figura 5 – Fluxograma de Execução**

**Fonte:** Autor, 2022.

A Figura 5 representa o processo de execução adotado. O preprocessamento consiste na normalização do *dataset* e o balanceamento dos dados através da Técnica de Superamostragem Sintética de Minorias (SMOTE) para balancear os dados que alimentarão os algoritmos classificadores [Chawla *et al.* 2002]. Após esta etapa os dados são separados em conjuntos aleatórios de treino e teste que são alimentados nos algoritmos classificadores. A Tabela 2 abaixo apresenta a quantidade de elementos utilizados para treino e teste.

**Tabela 2 – Tamanho do Conjunto de Treino e Teste.**

Conjunto	Tamanho	%
Treino	3549974	87.48
Teste	508003	12.52
Total	4057977	100.00

**Fonte:** Autor, 2022.

#### 4.2.1 Extração de Parâmetros

As características utilizadas para análise dos algoritmos foram extraídas utilizando o algoritmo *Random Forest* obtendo os atributos e seus pesos associados a importância para os modelos do algoritmo preditivo. Estas características estão apresentadas na Tabela 3 e foram também utilizados no trabalho de Tateisi 2022, que utilizou estes atributos no *dataset* UNSW NB15, o que nos permite desenvolver o trabalho com a análise em relação a diferentes algoritmos classificadores. Desta forma, temos uma base de comparação do desempenho de algoritmos classificadores diferentes sobre este *dataset* novo.

Para tanto, a metodologia de extração dos parâmetros segue o mesmo procedimento do trabalho da autora, com os dados sendo aplicados sobre algoritmos diferentes, com o algoritmo SVM servindo como base de ligação e comparação da metodologia utilizada pela autora.

**Tabela 3 – Atributos selecionados.**

Atributo	Importância
<i>srcip</i>	0.193031
<i>ct_state_ttl</i>	0.143258
<i>sttl</i>	0.132478
<i>dmeansz</i>	0.083276
<i>dbytes</i>	0.055197
<i>Sload</i>	0.054356
<i>dttl</i>	0.047501

**Fonte:** Autor, 2022.

#### 4.2.2 Métricas de Avaliação

As métricas de avaliação que serão utilizados serão a Precisão (Pr), o *Recall* (Rc) e o *f-measure* (F1) que foram utilizados para classificar os trabalhos de Sharafaldin, Lashkari e Ghorbani 2018 e Singh e Banerjee 2020 com resultados satisfatórios. Para cada métrica será analisada os valores de Verdadeiro Positivo (VP), Falso Positivo (FP), Verdadeiro Negativo (VN) e Falso Negativo (FN) de cada algoritmo implementado.

A Precisão será a principal métrica de análise e irá observar quanto dos dados classificados como anômalos são de fato tráfego malicioso na rede. Caso o seu valor seja muito baixo, isso poderá gerar lentidão na rede, visto que o algoritmo estará limitando o recebimento de dados normais na rede e gerando problemas quanto a utilização do sistema, removendo o propósito de sua aplicação [Sharafaldin, Lashkari e Ghorbani 2018, Singh e Banerjee 2020].

Para o cálculo da Precisão, foi utilizado a razão entre os ataques corretamente classificados e todos os ataques detectados pelo algoritmo, segundo a Equação 31.

$$Pr = \frac{VP}{VP + FP} \quad (31)$$

Onde VP são os dados anômalos classificados como maliciosos e FP são os dados normais classificados como maliciosos.

O *Recall*, também chamado de Sensibilidade ou Taxa de Verdadeiro Positivo (TVP), irá medir a razão entre os dados classificados como anômalos que foram classificados corretamente e todos os dados anômalos que os algoritmos classificaram como normais [Sharafaldin, Lashkari e Ghorbani 2018, Singh e Banerjee 2020]. Para o cálculo do *Recall*, foi utilizado a razão entre os ataques corretamente classificados e todos os ataques gerados, segundo a Equação 32.

$$Rc = \frac{VP}{VP + FN} \quad (32)$$

Onde FN são os dados anômalos que escaparam a detecção.

A Taxa de Falso Positivos (TFP), dada pela Equação 33 abaixo, é a probabilidade de alarmes falsos ou falsos positivos, que influenciam o algoritmo. Ela é aplicada no eixo x da Receiver Operator Characteristics (ROC) e serve como variável para a análise de desempenho. A forma de análise da Curva ROC será posteriormente nesta seção.

$$TFP = \frac{FP}{FP + VN} \quad (33)$$

Onde VN são os dados normais corretamente classificados.

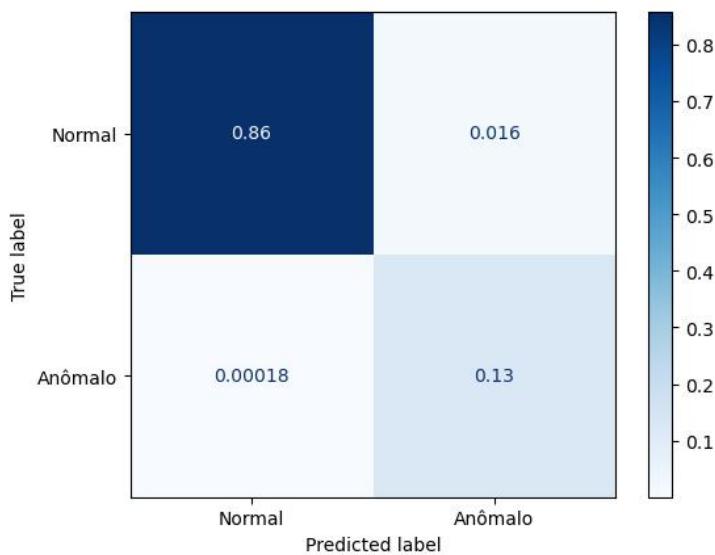
Através dos resultados anteriores, será feito o cálculo do *F1 Score*, que consiste em analisar a combinação harmônica entre a precisão e a sensibilidade do algoritmo, como descrito

abaixo na Equação 34.

$$F_1 = \frac{2}{\frac{1}{Pr} + \frac{1}{Rc}} \quad (34)$$

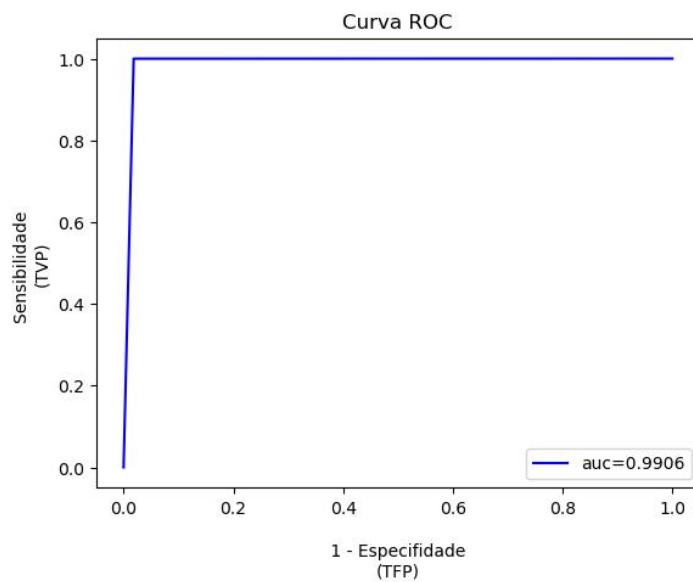
Com a análise do *F1 Score* é possível obter a confiabilidade dos algoritmos na sua detecção e classificação dos dados presentes na rede de forma consistente. Este valor serve como principal componente da análise do desempenho dos algoritmos, com suas componentes (*Pr* e *Rc*) indicando onde estão as limitações a serem trabalhadas [Sharafaldin, Lashkari e Ghorbani 2018, Singh e Banerjee 2020].

**Figura 6 – Exemplo de Matriz de Confusão Normalizada**



**Fonte:** Autor, 2022.

**Figura 7 – Exemplo de Curva ROC**



**Fonte:** Autor, 2022.

Para facilitar a visualização dos dados, será utilizado a Matriz de Confusão considerando os valores de VP, FP, VN e FN para a análise de cada algoritmo. A Matriz de Confusão se utiliza do sistema de coordenadas inerente às matrizes como forma de associar os dados a categorias definidas [Susmaga 2004]. No eixo x temos as classificações preditas, e no eixo y as classificações verdadeiras. Assim, o quadrante  $a_{00}$  representa os dados Verdadeiros Negativo, o quadrante  $a_{01}$  representa os dados Falsos Positivo, o quadrante  $a_{10}$  representa os dados Falsos Negativo, e o quadrante  $a_{11}$  representa os dados Verdadeiros Positivo.

A Figura 6 apresenta um exemplo de uma matriz de confusão normalizada da biblioteca do *Scikit Learn* do Python. Nela podemos observar como o valor de cada elemento da matriz está relacionado com o tom de azul do gráfico e a somatória totaliza 1.

Por fim, para permitir a análise da granularidade dos dados será utilizada a Curva Característica de Operação do Receptor, ou Curva ROC. Para gerar este gráfico, são plotados no eixo x a porcentagem de Falso Positivo dada pela Equação 33 e no eixo y a porcentagem de Verdadeiro Positivo, tal como mostrado na Figura 7 [Fawcett 2006].

Segundo Fawcett 2006, a *Area Under the Curve* (AUC) da curva ROC serve como discriminante para a qualidade do classificador, visto que esta métrica avalia a habilidade de discriminação do classificador. Esta curva é importante para compararmos o desempenho de diferentes classificadores com base na leniência em relação a taxa de falso positivos permitidos. Desta forma, é possível escolher, com base no nível de precisão necessária, qual o melhor algoritmo para ser adotado em aplicações reais baseado no seu desempenho e *trade-off* de eficiência e custo.

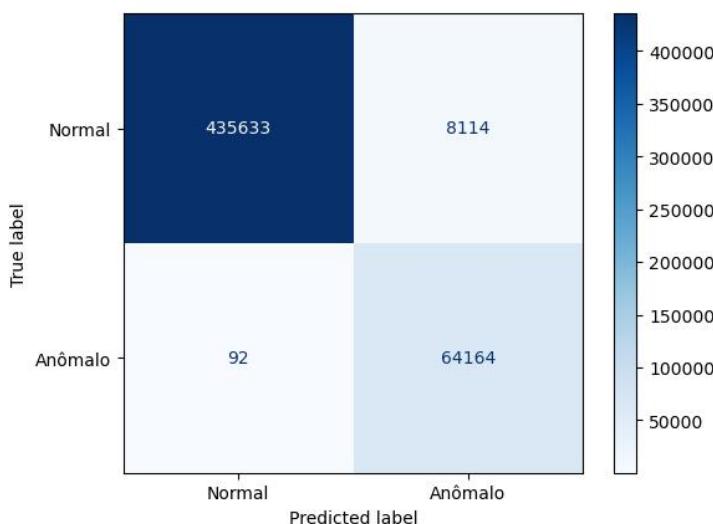
## 5 RESULTADOS

Neste capítulo serão discutidos sobre os resultados obtidos na aplicação dos algoritmos de Naïve-Bayes Gaussiano, *AdaBoost* e SVM e a análise dos resultados obtidos em para cada algoritmo.

### 5.1 Aplicação do Algoritmo de Naïve-Bayes Gaussiano

Seguindo o fluxograma da Figura 5, os dados foram divididos entre conjuntos de treinamento e teste utilizando a biblioteca *sklearn* do Python segundo a proporção indicada na Tabela 2. Após a separação dos conjuntos, foi realizada a execução do Algoritmo de Naïve-Bayes Gaussiano para gerar a matriz de confusão e a sua normalizada. Foram utilizados como parâmetros de execução as probabilidades *a priori* como padrão nulo, o que permite que estas probabilidades variem conforme os dados inseridos, e a suavização padrão de  $1e - 9$ . Os resultados podem ser observados na Figura 8 e Figura 9.

**Figura 8 – Matriz de Confusão do Classificador de *Gaussian Naïve-Bayes***

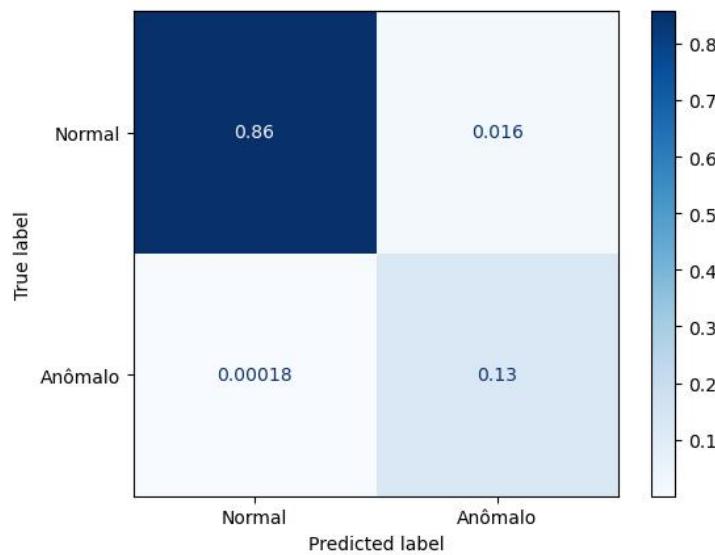


**Fonte: Autor, 2022.**

Como podemos observar na Figura 9, a proporção de dados classificados como Verdadeiros Negativos e Verdadeiros Positivos é semelhante à proporção de dados normais e anômalos do *dataset*, implicando que os conjuntos de treino e teste amostrados são representativos do universo do *dataset*.

Ainda, após as 10 iterações do algoritmo, temos que a taxa de Falso Positivos do algoritmo é inferior a 2%. Embora o resultado seja baixo, quando escalamos para o volume de pacotes transitando em uma rede, a quantidade de alarmes falsos são significativos.

Ao analisarmos as métricas obtidas durante as dez rodadas de treino e teste com conjuntos aleatorizados apresentados na Tabela 4, temos que a precisão e o *recall* do algoritmo

**Figura 9 – Matriz de Confusão Normalizada do Classificador de Gaussian Naïve-Bayes**

**Fonte: Autor, 2022.**

são altos. Ainda, o desvio padrão das métricas indicam uma baixa variação do desempenho do algoritmo com diferentes conjuntos de dados de treinamento e teste.

Quando consideramos uma conexão de 10 Gbps com pacotes de 32 Bytes, ocorrerão 625 mil de pacotes marcados como falsos positivos por segundo. Ainda, para cada um dos 625 mil pacotes descartados como falso positivo, será necessário o reenvio do mesmo, aumentando a latência da rede devido ao aumento do tráfego para ser analisado. Considerando a mesma rede, apenas 7032 pacotes maliciosos não seriam detectados pelo algoritmo.

**Tabela 4 – Resultados das métricas para o algoritmo de Naïve-Bayes Gaussiano.**

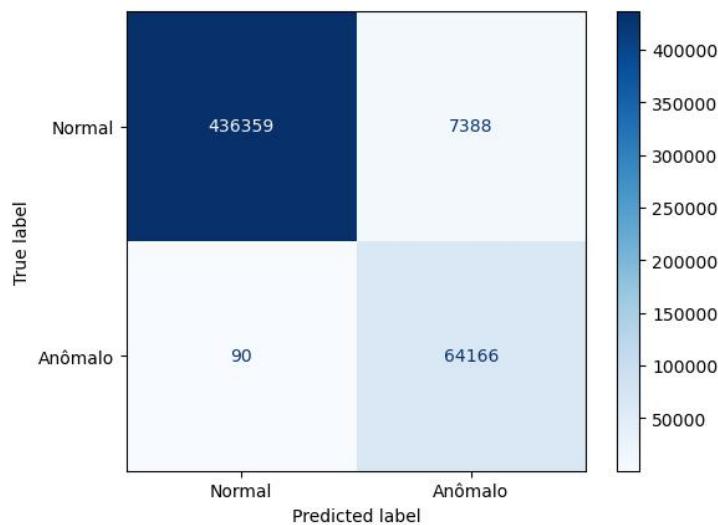
	Treino (s)	Teste (s)	Acurácia	Precisão	Recall	F1	AUC
1	0.65907	0.44037	0.98962	0.97854	0.99902	0.98867	0.99072
2	0.64974	0.32419	0.98980	0.97882	0.99912	0.98887	0.99081
3	0.64274	0.38479	0.98960	0.97829	0.99923	0.98865	0.99061
4	0.64340	0.38595	0.98995	0.97882	0.99944	0.98903	0.99086
5	0.64602	0.38557	0.98934	0.97778	0.99920	0.98837	0.99037
6	0.64755	0.38499	0.98972	0.97854	0.99923	0.98878	0.99070
7	0.64948	0.38892	0.98963	0.97831	0.99928	0.98868	0.99061
8	0.65412	0.38787	0.98992	0.97896	0.99923	0.98899	0.99089
9	0.64858	0.38300	0.98962	0.97828	0.99929	0.98867	0.99060
10	0.64775	0.41690	0.98960	0.97852	0.99797	0.98864	0.99070
Média	0.64885	0.39426	0.98968	0.97849	0.99920	0.99874	0.99069
Desvio Padrão	0.00460	0.01804	0.00017	0.00033	0.00013	0.00018	0.00015

**Fonte: Autor, 2022.**

## 5.2 Aplicação do Algoritmo Classificador AdaBoost

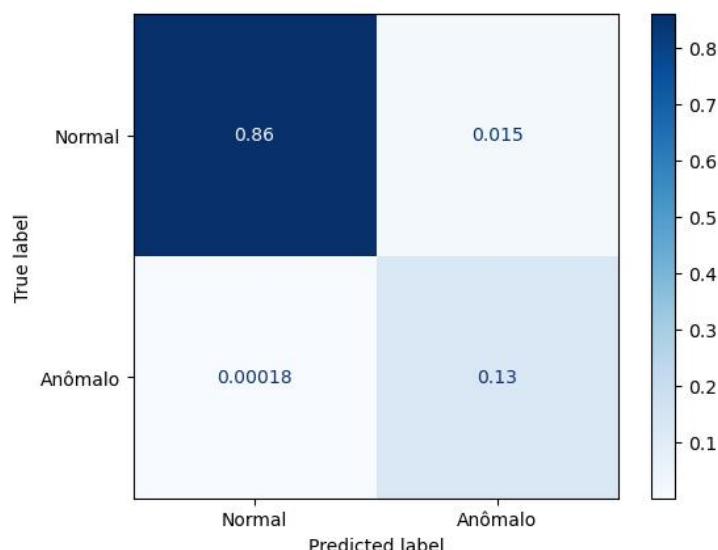
Para a aplicação do Algoritmo Classificador *AdaBoost* foi utilizado os parâmetros padrão na sua execução, com o algoritmo de estimativa sendo o algoritmo de Floresta Aleatória com uma profundidade de um nó, impulsionado através do algoritmo SAMME, com 50 classificadores fracos e a taxa de aprendizado sendo 1,0. A Figura 10 e Figura 11 apresentando a matriz de confusão resultante.

**Figura 10 – Matriz de Confusão do Classificador por *AdaBoost***



**Fonte: Autor, 2022.**

**Figura 11 – Matriz de Confusão Normalizada do Classificador por *AdaBoost***



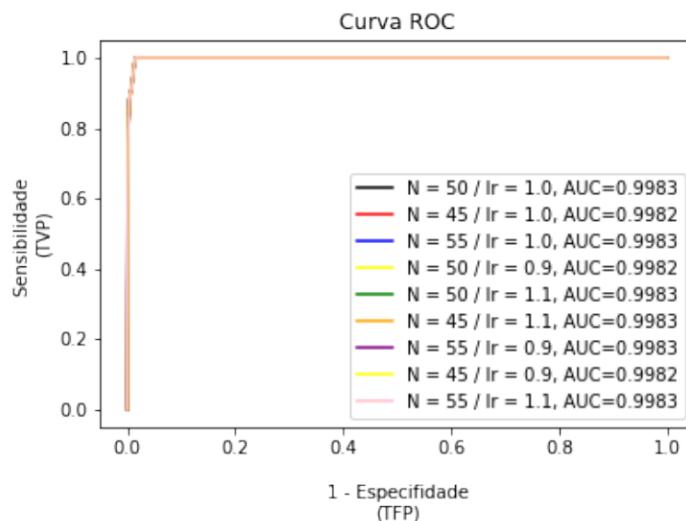
**Fonte: Autor, 2022.**

Ainda, foi analisado o desempenho do algoritmo variando-se em  $\pm 10\%$  os parâmetros, com os resultados apresentados na Figura 12.

**Tabela 5 – Resultados das métricas para o algoritmo AdaBoost.**

	Treino (s)	Teste (s)	Acurácia	Precisão	Recall	F1	AUC
1	92.4117	3.59225	0.99133	0.98200	0.99919	0.99052	0.99849
2	93.0978	3.55882	0.99165	0.98246	0.99941	0.99086	0.99875
3	92.6238	3.52898	0.99123	0.98157	0.99940	0.99041	0.99868
4	89.4219	3.47040	0.99140	0.98209	0.99924	0.99059	0.99878
5	89.7990	3.47365	0.99102	0.98118	0.99935	0.99018	0.99873
6	90.6961	3.44042	0.99133	0.98182	0.99934	0.99050	0.99872
7	89.6035	3.43079	0.99119	0.98179	0.99910	0.99037	0.99867
8	90.2569	3.46701	0.99152	0.98226	0.99934	0.99073	0.99884
9	89.4561	3.43568	0.99125	0.98167	0.99935	0.99043	0.99867
10	88.8153	3.49302	0.99134	0.98192	0.99928	0.99053	0.99851
Média	90.6162	3.48910	0.99132	0.98188	0.99930	0.99051	0.99868
Desvio Padrão	1.45786	0.05178	0.00017	0.00034	0.00009	0.00018	0.00011

Fonte: Autor, 2022.

**Figura 12 – Curva ROC do AdaBoost Classifier variando os parâmetros em 10%**

Fonte: Autor, 2022.

Como podemos observar, não existe diferença substancial na Curva ROC o que implica num desempenho similar para cada execução. Desta forma, a variação dos parâmetros iniciais em 10% não apresenta variação no desempenho visível.

Novamente, considerando uma conexão padrão de 10 Gbps, com pacotes de 32 Bytes, temos que aproximadamente 547 mil pacotes serão identificados falsamente como anomalias, mas apenas 1446 pacotes maliciosos não serão identificados.

### 5.3 Aplicação do Algoritmo SVM

Para a aplicação do algoritmo SVM, foi primeiramente utilizada a biblioteca *Grid Search CV* do *Scikit-learn* para identificar os melhores parâmetros para a execução do modelo. Como

parâmetros foram considerados o Parâmetro de Regularização (C), o Coeficiente do Kernel (Gamma) e a quantidade máxima de iterações que o modelo pode executar. Todos estes parâmetros foram analisados e classificados com base na Precisão de cada modelo, executando um *cross-validation* de 3 iterações, com o tempo necessário para cada teste sendo apresentados na Tabela 6 abaixo.

**Tabela 6 – Resultados do GridsearchCV.**

C	Gamma	Limite de Iterações	Tempo Consumido (min)
1	1	0.1	1000
2	1	0.1	1000
3	1	0.1	1000
4	1	0.1	5000
5	1	0.1	5000
6	1	0.1	5000
7	1	1.0	1000
8	1	1.0	1000
9	1	1.0	1000
10	1	1.0	5000
11	1	1.0	5000
12	1	1.0	5000
13	5	0.1	1000
14	5	0.1	1000
15	5	0.1	1000
16	5	0.1	5000
17	5	0.1	5000
18	5	0.1	5000
19	5	1.0	1000
20	5	1.0	1000
21	5	1.0	1000
22	5	1.0	5000
23	5	1.0	5000
24	5	1.0	5000

**Fonte:** Autor, 2023.

Os melhores parâmetros estão apresentados na Tabela 7 abaixo.

**Tabela 7 – Parâmetros para o SVM com otimização em Precisão.**

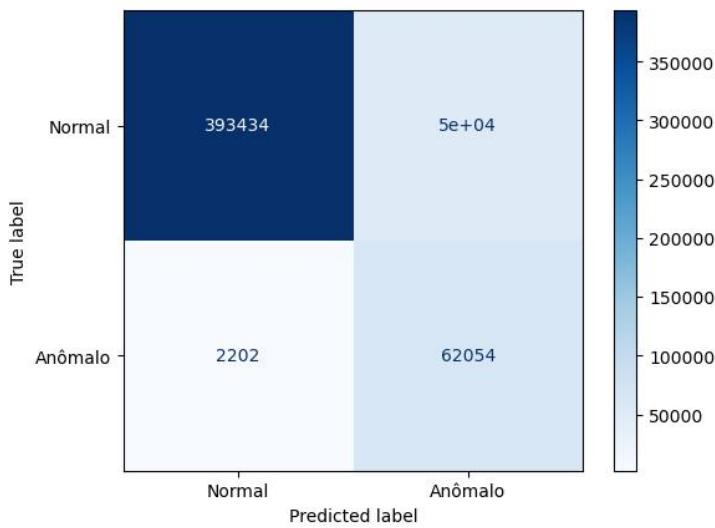
C	Gamma	Limite de Iterações	Precisão
1	5	1	1000

**Fonte:** Autor, 2023.

Assim, após a execução do algoritmo, o modelo calibrado foi testado no grupo de testes, gerando as matrizes de confusão da Figura 13 e Figura 14.

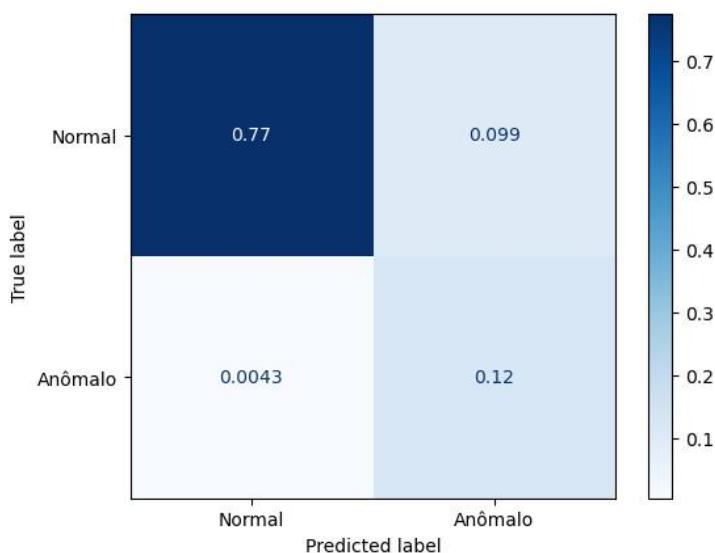
Como podemos observar, o algoritmo alcançou em média 99,079% de precisão e necessitou de aproximadamente 83.19 minutos por iteração de treino e 1,65 minutos para classificar todo os elementos de teste.

**Figura 13 – Matriz de Confusão do Classificador por *Support Vector Machine***



**Fonte:** Autor, 2023.

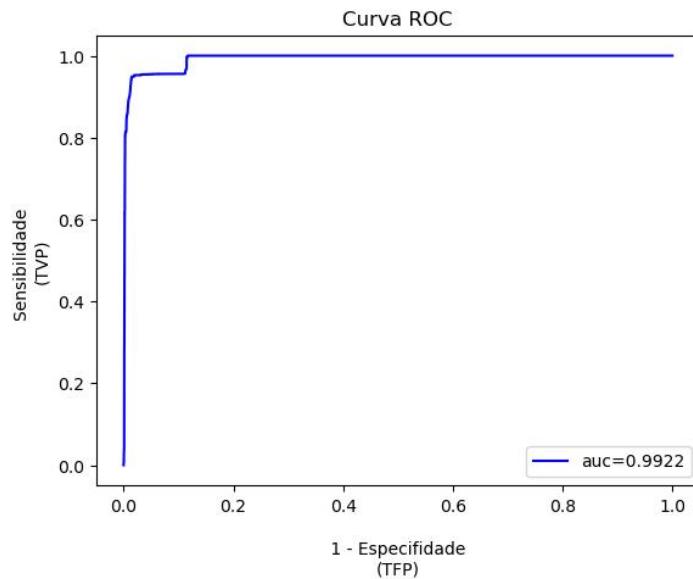
**Figura 14 – Matriz de Confusão Normalizada do Classificador por *Support Vector Machine***



**Fonte:** Autor, 2023.

Considerando o cenário hipotético de aplicação real em uma conexão padrão de 10 Gbps, com pacotes de 32 Bytes, temos que aproximadamente 3.87 milhões de pacotes serão identificados falsamente como anomalias, e 168 mil pacotes maliciosos não serão identificados.

Com base nos resultados, podemos observar que a implementação real do algoritmo SVM é inviável, visto que o tempo de treinamento e teste estão na escala de dias e as tendências do mercado estão focadas na redução da latência das conexões.

**Figura 15 – Curva ROC do Classificador por *Support Vector Machine***

**Fonte:** Autor, 2023.

**Tabela 8 – Resultados das métricas para o algoritmo SVM.**

	Treino (s)	Teste (s)	Acurácia	Precisão	Recall	F1	AUC
1	4949.69	100.012	0.91297	0.99084	0.81539	0.89459	0.98862
2	4967.14	100.105	0.91372	0.99108	0.81700	0.89566	0.99149
3	5005.36	99.1589	0.91304	0.99075	0.81576	0.89478	0.99156
4	4953.18	99.4567	0.91305	0.99086	0.81569	0.89478	0.99171
5	4957.55	99.3229	0.91276	0.99042	0.81540	0.89443	0.99316
6	4954.39	99.2064	0.91299	0.99066	0.81571	0.89472	0.99133
7	5069.88	99.8337	0.91295	0.99079	0.81557	0.89466	0.99154
8	4961.96	99.6393	0.91371	0.99111	0.81694	0.89564	0.99171
9	4989.94	99.7725	0.91290	0.99067	0.81552	0.89460	0.99153
10	5108.85	103.257	0.91300	0.99075	0.81566	0.89472	0.99443
Média	4991.49	99.2064	0.91310	0.99079	0.81586	0.89486	0.99171
Desvio Padrão	52.2871	1.13653	0.00032	0.00019	0.00056	0.00041	0.00139

**Fonte:** Autor, 2023.

#### 5.4 Análise entre algoritmos

Para a análise comparativa dos algoritmos serão utilizadas como métricas a Precisão, o Recall e o  $F_1$ , indicadas por Sharafaldin, Lashkari e Ghorbani 2018 e Singh e Banerjee 2020. Desta forma, analisando a Tabela 9, podemos observar que os resultados das métricas são semelhantes entre os classificadores de Naïve-Bayes Gaussiano e Adaboost, com o classificador por SVM apresentando uma performance inferior em todos os aspectos à exceção da precisão.

O desempenho similar entre os classificadores de Naïve-Bayes Gaussiano e Adaboost era esperado, visto que o classificador por Adaboost utilizou como classificador fraco o algoritmo de Naïve-Bayes. Entretanto, o custo em tempo da implementação é acima do esperado, visto

que, embora o algoritmo *Adaboost* apresente uma Precisão superior de 1,05% e  $F_1$  superior de 0,82%, a diferença em relação ao *Recall* é insignificante.

**Tabela 9 – Resultados das métricas para cada algoritmo.**

Métrica	Gaussian Naïve-Bayes	AdaBoost Classifier	Support Vector Machine
Treino (s)	0.648845 ± 0.004599	90.61619 ± 1.457855	4991.494 ± 52.28706
Teste (s)	0.394255 ± 0.018044	3.489101 ± 0.051776	99.97467 ± 1,136533
Acurácia	0.989682 ± 0.000169	0.991322 ± 0.000166	0.913104 ± 0.000316
Precisão	0.978488 ± 0.000330	0.988771 ± 0.000344	0.990788 ± 0.000192
Recall	0.999202 ± 0.000130	0.999299 ± 0.000093	0.815864 ± 0.000564
$F_1$	0.998736 ± 0.000183	0.990511 ± 0.000180	0.894858 ± 0.000407
AUC ROC	0.990685 ± 0.000145	0.998684 ± 0.000107	0.991706 ± 0.001391

**Fonte:** Autor, 2023.

O destaque se observa no desempenho do algoritmo de classificação por SVM, que apresentou uma precisão 1,26% superior em relação ao algoritmo de Naïve-Bayes Gaussiano e 0,20% superior em relação ao algoritmo *Adaboost*, porém apresentou um *Recall* 18,35% inferior ao algoritmo de Naïve-Bayes Gaussiano e 18,36% inferior ao algoritmo *Adaboost*. Estes resultados implicam na diferença no  $F_1$ , onde o algoritmo SVM apresentou um desempenho 10,40% inferior ao algoritmo de Naïve-Bayes Gaussiano e 9,66% inferior ao algoritmo *Adaboost*.

Ainda neste aspecto, embora as porcentagens apresentem resultados promissores, quando escalado no volume de dados trafegando em uma rede, temos que mesmo o melhor algoritmo em relação a desempenho temporal não está em condições de ser aplicado num cenário semelhante ao simulado neste *dataset*. Podemos observar pela Tabela 10 abaixo que, embora a quantidade de pacotes processados para teste entre os algoritmos classificadores por Gaussian Naïve-Bayes e *Adaboost* esteja em uma ordem de grandeza, o desempenho do algoritmo classificador por SVM é três a quatro ordens de grandeza inferior aos classificadores analisados.

**Tabela 10 – Quantidade de Pacotes processados por Segundo.**

Métrica	Gaussian Naïve-Bayes	AdaBoost Classifier	Support Vector Machine
Pacotes/s	1.288517 e+06	1.455971 e+05	1.017737 e+02

**Fonte:** Autor, 2023.

## 6 CONCLUSÃO

A crescente migração dos sistemas de informação para o ambiente de nuvem apresentam novos desafios para a segurança cibernética das empresas e instituições modernas. Estes desafios buscam agregar a segurança, confiabilidade e disponibilidade dos dados e o seu compartilhamento seguro e com a eficiência no consumo de recursos limitados. Para tanto, o uso de ferramentas de segurança eficientes para detecção e prevenção de invasões é essencial para os modernos sistemas de segurança como a primeira linha de defesa contra usuários maliciosos.

Neste aspecto, os algoritmos para Detecção de Anomalias em Sistemas de Detecção de Intrusão baseados em Rede são essenciais para identificar *malwares* que estão no ambiente de desenvolvimento e trabalho, com a detecção de *Zero Day Attacks* nas redes de comunicação modernas sendo outra vantagem desta técnica. Desta forma, este trabalho buscou analisar o desempenho dos algoritmos num *dataset* moderno e representativo do cenário experienciado pelos usuários atuais. Isto possibilitou detectar as forças e fraquezas nos algoritmos aplicados, tal como a precisão ou o *recall* ou o custo de recursos e tempo necessários para a aplicação de cada algoritmo, que são os principais problemas enfrentados na área.

Os resultados obtidos neste trabalho nos permitem observar o comportamento dos algoritmos quando aplicados sobre um *dataset* moderno podem nos direcionar quanto ao futuro do estado da arte em relação ao desenvolvimento de soluções de segurança cibernética baseadas na detecção de anomalias. O desempenho de cada algoritmo também permite a oportunidade de melhorias nos algoritmos aplicados com o objetivo de aperfeiçoar o desempenho desta técnica.

Portanto, quando observamos o desempenho dos algoritmos abordados, temos que a utilização de pré-processamento dos dados, como o trabalho de Catania, Bromberg e Garino 2012, e a otimização dos algoritmos para melhorar o seu desempenho são os principais desafios no caminho da implementação de dispositivos comerciais viáveis utilizando a tecnologia de Detecção de Intrusão por Anomalias.

### 6.1 Trabalhos Futuros

Como trabalho futuro temos a inclusão dos resultados de outros algoritmos que aplicam diferentes abordagens das utilizadas neste trabalho, como Classificação por Floresta Aleatória. Outra possibilidade de expansão é comparar o desempenho destes algoritmos em outros *datasets*, como o IDS 2017 do professor Sharafaldin *et al.* 2017.

## REFERÊNCIAS

- AXELSSON, S. Intrusion detection systems: A survey and taxonomy. Citeseer, p. 27, Março 2000.
- CATANIA, C. A.; BROMBERG, F.; GARINO, C. G. An autonomous labeling approach to support vector machines algorithms for network traffic anomaly detection. **Expert Systems with Applications**, v. 39, n. 2, p. 1822–1829, 2012. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417411011808>.
- CHAN, T. F.; GOLUB, G. H.; LEVEQUE, R. J. Updating formulae and a pairwise algorithm for computing sample variances. p. 349 – 360, 11 1979.
- CHAWLA, N. V. *et al.* Smote: Synthetic minority over-sampling technique. **Journal of Artificial Intelligence Research**, v. 16, p. 321–357, Fev 2002. Disponível em: <https://jair.org/index.php/jair/issue/view/1100>.
- CORTES, C.; VAPNIK, V. Support-vector networks. **Machine Learning**, v. 20, n. 3, p. 273–297, Sep 1995. ISSN 1573-0565. Disponível em: <https://doi.org/10.1007/BF00994018>.
- DISTRITO. **CyberTech Report**: O avanço dos ataques cibernéticos. 2021. 45 p.
- DISTRITO. **CyberTech Report**: Desafios da cibersegurança no brasil. 2021. 55 p.
- DUDA, R. O.; HART, P. E.; STORK, D. G. **Pattern Classification**. 2. ed. New York, NY: John Wiley & Sons, Inc, 2000. 654 p. ISBN 0471056693.
- FACELI, K. *et al.* **Inteligência artificial: uma abordagem de aprendizado de máquina**. 2. ed. LTC, 2011. 394 p. ISBN 9788521637509. Disponível em: [https://integrada\[minhabiblioteca\].com.br/#/books/9788521637509/](https://integrada[minhabiblioteca].com.br/#/books/9788521637509/).
- FAWCETT, T. An introduction to roc analysis. **Pattern Recognition Letters**, v. 27, n. 8, p. 861–874, 2006. ISSN 0167-8655. ROC Analysis in Pattern Recognition. Disponível em: <https://www.sciencedirect.com/science/article/pii/S016786550500303X>.
- FERNANDES, G. *et al.* A comprehensive survey on network anomaly detection. **Telecommunication Systems**, v. 70, p. 447–489, julho 2019.
- FITZSIMMONS, J. Information technology and the third industrial revolution. **The Eletronic Library**, v. 12, n. 5, p. 295–297, outubro 1994.
- HU, W.; HU, W.; MAYBANK, S. Adaboost-based algorithm for network intrusion detection. **IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)**, v. 38, n. 2, p. 577–583, 2008.
- INTERNATIONAL BUSINESS MACHINE CORPORATION. **Cost of a Data Breach**: A view from the cloud 2021. Armonk, 2021. 13 p.
- INTERNATIONAL BUSINESS MACHINE CORPORATION. **X-Force Threat Intelligence Index 2022**. Armonk, 2022. 59 p.
- KABIR, E. *et al.* A novel statistical technique for intrusion detection systems. **Future Generation Computer Systems**, v. 79, p. 303–318, 2018. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X17301371>.

- KEMP, S. **Digital 2022: Global Overview Report**. 2022. Disponível em: <https://datareportal.com>. Acesso em: 27 de maio de 2022.
- LI, W.; LI, Q. Using naive bayes with adaboost to enhance network anomaly intrusion detection. In: **2010 Third International Conference on Intelligent Networks and Intelligent Systems**. [S.I.: s.n.], 2010. p. 486–489.
- LIAO, H.-J. et al. Intrusion detection system: A comprehensive review. **Journal of Network and Computer Applications**, v. 36, n. 1, p. 16–24, 2013. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804512001944>.
- LIU, T. et al. Method for network anomaly detection based on bayesian statistical model with time slicing. **2008 7th World Congress on Intelligent Control and Automation**, p. 3359–3362, 2008.
- MAMMONE, A.; TURCHI, M.; CRISTIANINI, N. Support vector machines. **WIREs Computational Statistics**, v. 1, n. 3, p. 283–289, 2009. Disponível em: <https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.49>.
- MOUSTAFA, N.; SLAY, J. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In: **2015 Military Communications and Information Systems Conference (MilCIS)**. [S.I.: s.n.], 2015. p. 1–6.
- MOUSTAFA, N.; SLAY, J. The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set. **Information Security Journal: A Global Perspective**, Taylor & Francis, v. 25, n. 1-3, p. 18–31, 2016. Disponível em: <https://doi.org/10.1080/19393555.2015.1125974>.
- MOUSTAFA, N. M. A. **Designing an online and reliable statistical anomaly detection framework for dealing with large high-speed network traffic**. June 2017. Tese (Phd Thesis) — School of Engineering and Information Technology, Canberra, Australia, June 2017.
- PATCHA, A.; PARK, J.-M. An overview of anomaly detection techniques: Existing solutions and latest technological trends. **Computer Networks**, v. 51, n. 12, p. 3448–3470, 2007. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S138912860700062X>.
- RENEAR, A. H.; SACCHI, S.; WICKETT, K. M. Definitions of dataset in the scientific and technical literature. **Proceedings of the American Society for Information Science and Technology**, v. 47, n. 1, p. 1–4, 2010. Disponível em: <https://asistdl.onlinelibrary.wiley.com/doi/abs/10.1002/meet.14504701240>.
- SCIKIT-LEARN. **Gaussian Naïve-Bayes**. 2022. Disponível em: [https://scikit-learn.org/stable/modules/naive\\_bayes.html#gaussian-naive-bayes](https://scikit-learn.org/stable/modules/naive_bayes.html#gaussian-naive-bayes).
- SHARAFALDIN, I. et al. Towards a reliable intrusion detection benchmark dataset. **Software Networking**, v. 2017, p. 177–200, 01 2017.
- SHARAFALDIN, I.; LASHKARI, A. H.; GHORBANI, A. A. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: INSTICC. **Proceedings of the 4th International Conference on Information Systems Security and Privacy - ICISSP**. [S.I.]: SciTePress, 2018. p. 108–116. ISBN 978-989-758-282-0. ISSN 2184-4356.
- SINGH, S.; BANERJEE, S. Machine learning mechanisms for network anomaly detection system: A review. **2020 International Conference on Communication and Signal Processing (ICCP)**, p. 0976–0980, 2020.

- SUSMAGA, R. Confusion matrix visualization. In: KŁOPOTEK, M. A.; WIERZCHOŃ, S. T.; TROJANOWSKI, K. (Ed.). **Intelligent Information Processing and Web Mining**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004. p. 107–116. ISBN 978-3-540-39985-8.
- SWARNKAR, M.; HUBBALLI, N. Ocpad: One class naive bayes classifier for payload based anomaly detection. **Expert Systems with Applications**, v. 64, p. 330–339, 2016. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417416303839>.
- TATEISI, N. Y. **Caracterização de Tráfego e Anomalias em Redes**. 2022. 47 p. Monografia (Graduação) — Universidade Tecnológica Federal do Paraná, Avenida Alberto Carazzai, 1640 - Cornélio Procópio - Paraná, 2022.
- TAVALLAEE, M. et al. A detailed analysis of the kdd cup 99 data set. In: IEEE. **2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications**. [S.I.]: IEEE, 2009. p. 1–6. ISBN 978-1-4244-3763-4. ISSN 2329-6275.
- WANG, H.; GU, J.; WANG, S. An effective intrusion detection framework based on svm with feature augmentation. **Knowledge-Based Systems**, v. 136, p. 130–139, 2017. ISSN 0950-7051. Disponível em: <https://www.sciencedirect.com/science/article/pii/S095070511730415X>.
- YUAN, Y.; KAKLAMANOS, G.; HOGREFE, D. A novel semi-supervised adaboost technique for network anomaly detection. In: **Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems**. New York, NY, USA: Association for Computing Machinery, 2016. (MSWiM '16), p. 111–114. ISBN 9781450345026. Disponível em: <https://doi.org/10.1145/2988287.2989177>.
- ZHU, J. et al. Multi-class adaboost. **Statistics and its interface**, v. 2, p. 349 – 360, 02 2006.