UNIVERSIDADE FEDERAL DO CEARÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

FRANCISCO JEFERSON DA SILVEIRA PONTES

AVALIAÇÃO DO DESEMPENHO DE TÉCNICAS DE APRENDIZADO DE MÁQUINA NA DETECÇÃO DE MALWARE EM TRÁFEGO DE REDES IOT

SOBRAL

2024

FRANCISCO JEFERSON DA SILVEIRA PONTES

AVALIAÇÃO DO DESEMPENHO DE TÉCNICAS DE APRENDIZADO DE MÁQUINA NA DETECÇÃO DE MALWARE EM TRÁFEGO DE REDES IOT

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Da Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Da Computação.

Orientador: Prof. Dr. Wendley Souza da Silva

SOBRAL

2024

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

P858a Pontes, Francisco Jeferson da Silveira.
AVALIAÇÃO DO DESEMPENHO DE TÉCNICAS DE APRENDIZADO DE MÁQUINA NA DETECÇÃO DE MALWARE EM TRÁFEGO DE REDES IOT / Francisco Jeferson da Silveira Pontes. – 2024.
59 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Sobral, Curso de Engenharia da Computação, Sobral, 2024.
Orientação: Prof. Dr. Wendley Souza da Silva.

1. IoT. 2. Aprendizado de Máquina. 3. IA. 4. Malware. 5. Classificação. I. Título.

CDD 621.39

FRANCISCO JEFERSON DA SILVEIRA PONTES

AVALIAÇÃO DO DESEMPENHO DE TÉCNICAS DE APRENDIZADO DE MÁQUINA NA DETECÇÃO DE MALWARE EM TRÁFEGO DE REDES IOT

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Da Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Da Computação.

Aprovada em: 25/07/2024

BANCA EXAMINADORA

Prof. Dr. Wendley Souza da Silva (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Iális Cavalcante de Paula Júnior
Universidade Federal do Ceará (UFC)

Stefane Adna dos Santos
Universidade Federal do Ceará (UFC)

À minha família, por sempre acreditar e investir em mim. Mãe, foi seu cuidado e dedicação que me deram, em muitos momentos, a esperança para continuar. Pai, sua presença me deu a certeza de que nunca estou sozinho nesta jornada.

AGRADECIMENTOS

Agradeço aos meus pais por cuidarem de mim e nunca desistirem dos meus sonhos, mesmo diante de todas as dificuldades.

Às minhas irmãs, que sempre acreditaram em mim e apoiaram meus planos.

Aos familiares que me apoiaram e contribuíram para que meus objetivos se concretizassem.

Aos amigos Samuel Milanez e Ana Landim, que viveram comigo todos os momentos dessa jornada, por todos os conselhos e pela companhia diária.

À minha namorada, Bruna Kersia, por acreditar e me motivar mesmo diante dos maiores desafios.

Aos amigos Gabriel Souza, Livia Celita e Stefane Adna, que estiveram sempre junto a mim na graduação.

Ao Prof. Dr. Wendley Souza da Silva, por me orientar durante toda a graduação. Agradeço pela paciência, por todos os ensinamentos e, principalmente, por ser um exemplo de profissional.

Aos professores da Universidade Federal do Ceará (UFC), por todos os ensinamentos.

A todos da Universidade Federal do Ceará (UFC), por cada contribuição diária nesses anos de graduação.

“Bem, amigos, terminou!”

(Galvão Bueno)

RESUMO

Dado o aumento exponencial de dispositivos conectados à internet e a consequente ampliação da superfície de ataque, a segurança desses dispositivos se torna uma questão crítica. Nesse cenário, utiliza-se a base de dados IoT-23, que contém registros de tráfego de rede capturados em dispositivos *Internet of Things* (IoT), tanto benignos quanto infectados por *malware*, para criar modelos de classificação capazes de detectar fluxos infectados. O desenvolvimento envolve a limpeza e transformação dos dados, incluindo a normalização e seleção de atributos relevantes, para preparar o conjunto de dados para a aplicação dos algoritmos de classificação. Os algoritmos analisados incluem *Decision Tree*, *Random Forest*, *Extra Trees*, *k-Nearest Neighbors* e *Gaussian Naive Bayes*. Cada algoritmo foi implementado e testado utilizando a técnica de validação cruzada k-fold, que permite avaliar a capacidade de generalização dos modelos. Os resultados foram medidos em termos de acurácia, tempo de execução e quantidade de erros, proporcionando uma visão clara do desempenho de cada técnica. O estudo discute a eficácia das abordagens de aprendizado de máquina na detecção de *malware*, destacando que a técnica *Decision Tree* apresenta melhores resultados de acordo com as métricas estabelecidas. Além disso, a análise discute a relevância das transformações e segmentações dos dados, considerando o impacto no desempenho dos classificadores. Os resultados obtidos podem orientar o desenvolvimento de sistemas de detecção de intrusões mais eficientes e robustos, capazes de proteger os dispositivos IoT contra ataques. A pesquisa contribui para o campo da segurança em IoT e *Machine Learning* (ML), propondo direções futuras para a implementação de soluções de segurança baseadas em Inteligência Artificial (IA).

Palavras-chave: Intenet of Things; Malware; Aprendizado de Máquina; Classificador

ABSTRACT

Given the exponential increase in Internet-connected devices and the consequent expansion of the attack surface, the security of these devices becomes a critical issue. In this context, the IoT-23 dataset, which contains network traffic records captured from Internet of Things (IoT) devices, both benign and malware-infected, is utilized to create classification models capable of detecting infected flows. The development involves cleaning and transforming the data, including normalization and the selection of relevant attributes, to prepare the dataset for the application of classification algorithms. The analyzed algorithms include Decision Tree, Random Forest, Extra Trees, K-Nearest Neighbors (KNN), and Gaussian Naive Bayes. Each algorithm was implemented and tested using the k-fold cross-validation technique, which allows for the evaluation of the models' generalization capabilities. The results were measured in terms of accuracy, execution time, and the number of errors, providing a clear view of each technique's performance. The study discusses the effectiveness of machine learning approaches in malware detection, highlighting that the Decision Tree technique shows the best results according to the established metrics. Additionally, the analysis discusses the relevance of data transformations and segmentations, considering their impact on the classifiers' performance. The results obtained can guide the development of more efficient and robust intrusion detection systems capable of protecting IoT devices against attacks. The research contributes to the field of IoT security and machine learning, proposing future directions for the implementation of AI-based security solutions.

Keywords: Intenet of Things; Malware; Machine Learning; Classifier

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ANNs	Redes Neurais Artificiais
AS	Atributos Selecionados
CNNs	*Convolutional Neural Networks*
CPSs	*Cyber-Physical Systems*
DN	Dados Normalizados
DNNs	*Deep Neural Networks*
GANs	*Generative Adversarial Networks*
GPU	*Graphics Processing Unit*
IA	Inteligência Artificial
IAM	Gestão de Identidades e Acessos
IDS	Sistema de Detecção de Intrusões
IoT	*Internet of Things*
IP	*Internet Protocol*
KNN	*k-Nearest Neighbors*
LGPD	Lei Geral de Proteção de Dados Pessoais
ML	*Machine Learning*
NLP	Processamento de Linguagem Natural
OT	Tecnologia Operacional
PCA	*Principal Component Analysis*
RNNs	*Recurrent Neural Networks*
TI	Tecnologia da Informação

SUMÁRIO

1 INTRODUÇÃO

O termo IoT, abreviação para Internet das Coisas, refere-se a dispositivos elétricos ou eletrônicos de diferentes tamanhos e capacidades que podem se conectar à internet. Esses dispositivos podem ser empregados em uma variedade de configurações, abrangendo desde residências até ambientes industriais, que engloba áreas como manufatura, meio ambiente, saúde, energia elétrica e comunicação (SADHU *et al.*, 2022).

Até 2025, as projeções indicam que haverá aproximadamente 64 bilhões de dispositivos IoT conectados à internet. Essa ampla adoção desses dispositivos está, inegavelmente, transformando o mundo em um ambiente altamente interconectado. O paradigma da IoT, aliado às novas tecnologias de rede 5G, está abrindo caminho para novos cenários de aplicação e modelos de negócios nunca antes imaginados, como as Indústrias 4.0 e as Cidades Inteligentes, entre outros (REY *et al.*, 2022).

As soluções IoT podem ser utilizados de várias maneiras para auxiliar sistemas e empresas na simplificação, melhoria, automatização e controle de processos. Também podem ser usadas para fornecer dados importantes de desempenho de atividades ou até mesmo fatores ambientais que precisam ser monitorados de forma contínua e remotamente. As aplicações IoT podem, portanto, ajudar no desenvolvimento de novos sistemas e estratégias de negócios, bem como fornecer dados (SADHU *et al.*, 2022).

Um dos desafios mais relevantes na área da Internet das Coisas é a segurança. Esta questão é de suma importância, uma vez que implica garantir a integridade e a confidencialidade de todas as partes envolvidas no sistema, o que representa um desafio complexo para os desenvolvedores (VILLAMIL *et al.*, 2020).

Além disso, esses dispositivos são caracterizados por restrições econômicas e físicas, o que se traduz em recursos limitados tanto em termos de *hardware* quanto de *software*. Essa conjuntura, aliada à sua natureza distribuída e ao papel central que desempenham na evolução dos Sistemas Ciber-Físicos (CPSs), os torna alvos primários para ciberataques (FERENCZ *et al.*, 2021). Apenas no primeiro semestre de 2021, houve um registro de 1,5 bilhão de ataques direcionados a dispositivos IoT, dobrando a estimativa em relação ao primeiro semestre de 2020 (BOVENZI *et al.*, 2023).

Entretanto, ao mesmo tempo em que avanços em novas tecnologias estão ocorrendo, o número e a diversidade de ciberataques têm crescido nos últimos anos, tornando as abordagens de segurança atuais rapidamente obsoletas. Por essa razão, a garantia da segurança em futuros

ambientes de rede habilitados por tecnologias 5G apresenta desafios significativos que requerem técnicas modernas para serem superados.

Uma estratégia que tem ganhado destaque na detecção de dispositivos comprometidos por *malware* é o monitoramento das atividades desses dispositivos para gerar impressões digitais ou perfis comportamentais. Essas impressões digitais podem ser empregadas na detecção de anomalias causadas por ciberataques ou por modificações maliciosas de *software*. Nos dispositivos IoT, diversas fontes de comportamento heterogêneo podem ser monitoradas, incluindo comunicações de rede, utilização de recursos, eventos e ações de *software*, bem como interações de usuários (REY *et al.*, 2022).

Desse modo, para detecção de malware em redes IoT, propõe-se a implementação e análise dos resultados de técnicas de *machine learning* ultilizando o conjunto de dados IoT-23 (GARCIA *et al.*, 2021) para treino e teste, este estudo propõe analisar e comparar o desempenho de diferentes algoritmos de aprendizado de máquina na detecção de malware em tráfego de rede IoT. A implementação e avaliação desses algoritmos, utilizando o conjunto de dados IoT-23, permitirá identificar a técnica mais eficaz para detectar atividades maliciosas nesse ambiente.

Dessa forma, o presente estudo tem como objetivo analisar o desempenho de diferentes algoritmos de aprendizado de máquina - Decision Tree, Random Forest, Extra Trees, K-Nearest Neighbors e Gaussian Naive Bayes - na detecção de *malware* em tráfego de rede, classificando-o como *Malicious* ou *Benign*. Em resumo, busca-se implementar e avaliar qual técnica se mostra mais adequada ao contexto de detecção de malware em redes IoT.

1.1 Justificativa

A Internet das Coisas (IoT) representa uma revolução tecnológica que está transformando significativamente diversos setores, desde residências até ambientes industriais, abrangendo áreas como manufatura, saúde, energia e comunicação. A crescente interconexão de dispositivos IoT à internet cria um ambiente altamente interconectado, abrindo caminho para novos modelos de negócios e aplicações disruptivas.

No entanto, esse rápido avanço tecnológico também apresenta desafios significativos, sendo a segurança uma das principais preocupações. A natureza distribuída e as restrições de recursos dos dispositivos IoT os tornam alvos atrativos para ciberataques.

O *malware* é considerado a maior ameaça à cibersegurança, pois causa danos diretos aos sistemas ou captura informações sensíveis. Além disso, o *malware* representa o tipo

mais frequente de ataques a computadores, redes ou usuários, causando danos ou roubando informações sensíveis. Nos últimos anos, o número de *softwares* maliciosos aumentou em 22,9%, o que reflete um aumento alarmante nas ameaças aos usuários de computadores (ABOAOJA *et al.*, 2022).

Ao compreender e abordar esses desafios de segurança, este trabalho contribuirá para o desenvolvimento de soluções mais robustas e eficazes para proteger os dispositivos IoT e as redes em que estão inseridos. Além disso, os resultados obtidos podem orientar a implementação de medidas preventivas e de detecção de ameaças em ambientes de IoT, ajudando a garantir a integridade e a confiabilidade desses sistemas em um mundo cada vez mais conectado.

1.2 Trabalhos Relacionados

Nobakht *et al.* (2023) propõem o modelo DEMD-IoT (Deep Ensemble Malware Detection for IoT), que combina aprendizado profundo com aprendizado em conjunto para melhorar a precisão da detecção de *malware*. O DEMD-IoT utiliza três redes neurais convolucionais unidimensionais (1D-CNNs) para analisar padrões de tráfego de rede IoT. Um meta-aprendizado, implementado através do algoritmo *Random Forest*, é empregado para integrar os resultados das CNNs e fornecer a classificação final. Este modelo se destaca por sua capacidade de reduzir o tempo de pré-processamento e a sobrecarga computacional ao empregar 1D-CNNs em vez das mais comuns 2D-CNNs. Logo, ao combinar técnicas de otimização de hiperparâmetros e aprendizado em conjunto, obtem um resultado com precisão de detecção de *malware* de 99,9% ao ser testado no conjunto de dados IoT-23. Este desempenho supera significativamente o de modelos tradicionais de aprendizado de máquina e outros métodos de aprendizado profundo. Entretanto, a limitação do modelo é que, quando a quantidade de hiperparâmetros se elevar, o tempo de execução aumentará. Vale salientar que o tempo de execução não foi incluso nas métricas de avaliação.

Bovenzi *et al.* (2023) apresentam uma comparação detalhada de métodos de detecção de anomalias em redes IoT utilizando aprendizado profundo. O estudo avalia a performance e a robustez de diferentes arquiteturas de Deep Learning, incluindo AutoEncoders [1] clássicos. A análise é realizada em dois conjuntos de dados recentes e públicos (IoT-23 e Kitsune), adotando

[1] **AutoEncoder** é um tipo de arquitetura de rede neural projetada para comprimir (codificar) dados de entrada de forma eficiente em suas características essenciais e, em seguida, reconstruir (decodificar) a entrada original a partir dessa representação compactada. Disponível em: <https://www.ibm.com/topics/autoencoder>. Acesso em: 8 julho 2024.

uma abordagem adversarial para testar a robustez dos modelos contra ataques. Os resultados experimentais mostram que as arquiteturas propostas melhoram significativamente o desempenho em comparação com *baselines* conhecidos e propostas anteriores, validando a eficácia das soluções no contexto de segurança em IoT. Este estudo complementa a literatura existente ao fornecer uma análise comparativa abrangente de técnicas de detecção de anomalias em IoT, focando em métodos de aprendizado profundo e sua aplicabilidade em cenários adversariais. A abordagem do estudo se destaca por abordar desafios específicos, como a heterogeneidade dos dispositivos e o volume crescente de dados transmitidos, oferecendo soluções que combinam desempenho superior com robustez aprimorada.

Martínez *et al.* (2023) mapeiam as ameaças e mecanismos de segurança usando o framework MITRE ATT&CK, focando nos riscos associados a dispositivos médicos implantáveis e vestíveis (IWMDs) e à infraestrutura centralizada de saúde. A revisão cobre os requisitos de segurança e privacidade, analisando ameaças e mecanismos de defesa, e destaca a importância de regulamentações robustas e técnicas de inteligência artificial para melhorar a detecção e mitigação de ameaças. Os desafios futuros incluem a proteção contra ataques cibernéticos e a garantia da segurança do paciente, além de promover um ecossistema de saúde seguro que envolva pacientes, profissionais de saúde, seguradoras, familiares e autoridades públicas. Este estudo complementa a literatura existente ao oferecer uma análise atualizada das questões de segurança no setor de saúde, sendo um recurso valioso para pesquisadores e profissionais que buscam aprimorar a segurança e a privacidade em ambientes de saúde.

Douiba *et al.* (2023) apresentam um modelo aprimorado de detecção de anomalias para segurança em IoT utilizando algoritmos de árvore de decisão e gradient boosting. Este trabalho aborda a crescente vulnerabilidade dos dispositivos IoT devido ao aumento da sua adoção em diversas áreas, como sistemas de saúde, cidades inteligentes e indústrias. A pesquisa propõe um Sistema de Detecção de Intrusões (IDS), empregando o *Catboost*, um algoritmo de *gradient boosting* com código aberto. O modelo foi testado em diversos datasets, incluindo NSL-KDD, IoT-23, BoT-IoT e Edge-IIoT, utilizando *Graphics Processing Unit* (GPU) para melhorar a configuração experimental. Os resultados demonstraram resultados em termos de acurácia, recall e precisão, superiores a 99,9%, reduzindo significativamente o tempo de processamento. A metodologia utilizada neste estudo é particularmente relevante para a nossa pesquisa, pois oferece uma abordagem eficiente para a detecção de anomalias em ambientes IoT, que são caracterizados por um grande volume de dados e a necessidade de respostas rápidas. A integração do Catboost e

o uso de GPU para otimizar o tempo de treinamento e validação destacam-se como contribuições significativas, alinhando-se aos nossos objetivos de aprimorar a segurança em redes IoT através de técnicas avançadas de aprendizado de máquina. Portanto, este artigo fornece uma base robusta e um ponto de comparação valioso para a nossa pesquisa, especialmente no que tange à aplicação de técnicas de aprendizado de máquina para a detecção de ataques em sistemas IoT.

1.3 Objetivos

1.3.1 Objetivo geral

Este estudo tem como objetivo avaliar o desempenho de algoritmos de aprendizado de máquina na detecção de *malware* em ambientes IoT, a partir de uma base de dados pública.

1.3.2 Objetivos Específicos

- Explorar trabalhos que abordam o uso de técnicas de aprendizado de máquina para detecção de *malware* e trabalhos que utilizaram a base de dados IoT-23;
- Explorar a base de dados IoT-23, identificar as características dos atributos e adequar os dados para as implementações;
- Implementar os classificadores *Decision Tree, Random Forest, Extra Trees, K-Nearest Neighbors e Gaussian Naive Bayes* para a detecção de *malware* utilizando o conjunto de dados IoT-23;
- Analisar o desempenho dos algoritmos, considerando os critérios de precisão, acurácia e tempo de execução;
- Verificar qual das técnicas apresenta o melhor desempenho;

1.4 Organização do Documento

A Seção 2 destaca os conceitos teóricos fundamentais explorados neste trabalho. Em seguida, a Seção 3 apresenta as etapas metodológicas conduzidas na desenvolvimento deste estudo. Os resultados dos experimentos implementados são apresentados na Seção 4. Por último, a Seção 5 oferece as considerações finais deste projeto, além de elencar possíveis trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção, será apresentada a fundamentação teórica que aborda os principais conceitos explorados no desenvolvimento deste projeto.

2.1 *Internet of Things* - IoT

A IoT trata-se da interconexão de máquinas e dispositivos pela internet, gerando dados que alimentam análises e suportam inovações tecnológicas. Também pode ser descrita como um conjunto de objetos, fixos ou móveis, conectados via internet, aparelhados de sensores, atuadores e módulos de comunicação, ou um sistema de objetos físicos com comunicação autônoma. Outras definições mencionam que a IoT consiste em sistemas embarcados conectados à internet, atualizáveis e adaptáveis, capazes de coletar dados de áreas remotas, diagnosticar falhas e reiniciar sistemas de forma eficiente e econômica. Enquanto isso, outros autores a definem como um ecossistema global de tecnologias de informação e comunicação que visa conectar qualquer objeto à internet em qualquer momento e lugar. Por fim, a IoT é composta por numerosos nós sensores com capacidades limitadas de processamento, armazenamento e energia (RAY, 2018).

Um sistema IoT é dividido em vários blocos funcionais para segmentar diversas utilidades do sistema. São eles: sensoriamento, identificação, atuação, comunicação e gerenciamento. A seguir, as definições completas. (ALJABRI *et al.*, 2021).

2.1.1 Dispositivos

Um sistema de IoT é fundamentado em dispositivos que desempenham funções de sensoriamento, atuação, controle e monitoramento. Esses dispositivos podem trocar dados entre si e com aplicativos, coletar dados de outros dispositivos, processá-los localmente ou enviá-los para servidores centralizados ou aplicativos baseados em nuvem. Eles também podem realizar tarefas dependendo das restrições, como memória, capacidade de processamento, latências e velocidades de comunicação. Os dispositivos IoT podem ter várias interfaces comunicação com outros dispositivos, incluindo com fio, sem fio, E/S (Entrada/Saída) para sensores, conectividade com a Internet, memória e armazenamento, e áudio/vídeo. Além disso, abrangem uma variedade de tipos, como sensores vestíveis, relógios inteligentes, luzes LED, automóveis e máquinas industriais. Praticamente todos os dispositivos IoT geram dados que, quando processados por

sistemas de análise de dados, fornecem informações úteis para orientar ações adicionais, tanto localmente quanto remotamente (RAY, 2018).

2.1.2 Comunicação

Esse bloco é responsável pela comunicação entre dispositivos e servidores remotos. Os protocolos de comunicação IoT geralmente funcionam na camada de enlace de dados, na camada de rede, na camada de transporte e na camada de aplicação (RAY, 2018).

2.1.3 Serviços

Um sistema IoT serve vários tipos de funções, como serviços para modelagem de dispositivos, controle de dispositivos, publicação de dados, análise de dados e descoberta de dispositivos (RAY, 2018).

2.1.4 Gerenciamento

O bloco de gerenciamento fornece diferentes funções para governar um sistema IoT e buscar a governança subjacente do sistema IoT (RAY, 2018).

2.1.5 Segurança

O bloco funcional de segurança protege o sistema IoT fornecendo funções como autenticação, autorização, privacidade, integridade de mensagem, integridade de conteúdo e segurança de dados (RAY, 2018).

2.1.6 Aplicação

A camada de aplicação é a mais importante em termos de usuários, pois atua como uma interface que fornece os módulos necessários para controlar e monitorar vários aspectos do sistema IoT. As aplicações permitem que os usuários visualizem e analisem o status do sistema no estágio atual (RAY, 2018).

2.2 Industria 4.0

A Indústria 4.0, também conhecida como a Quarta Revolução Industrial, refere-se à transformação digital dos processos de manufatura e produção, integrando tecnologias avançadas como a IoT, IA, *Big Data*, e computação em nuvem. Essa integração permite a criação de "fábricas inteligentes"onde os sistemas físicos e digitais interagem e cooperam entre si de forma autônoma e em tempo real, otimizando a eficiência operacional e a flexibilidade da produção. Entretanto, desdobram-se inúmeros desafios de segurança na Indústria 4.0 (FERENCZ *et al.*, 2021).

2.2.1 Desafios de segurança na Indústria 4.0

2.2.1.1 Superfície de Ataque Ampliada

A integração de dispositivos IoT e *Cyber-Physical Systems* (CPSs) amplia significativamente a superfície de ataque, expondo mais pontos vulneráveis a possíveis ataques cibernéticos. Cada dispositivo conectado representa uma possível porta de entrada para atacantes, tornando a proteção de endpoints uma prioridade crítica (FERENCZ *et al.*, 2021)(LUTHRA; MANGLA, 2018).

2.2.1.2 Complexidade da Rede

As redes industriais na Indústria 4.0 são complexas, compostas por uma combinação de tecnologias de Tecnologia da Informação (TI) e Tecnologia Operacional (OT). A integração dessas tecnologias aumenta a complexidade da segurança, exigindo soluções que possam proteger ambos os ambientes simultaneamente (FERENCZ *et al.*, 2021).

2.2.1.3 Monitoramento em Tempo Real

A necessidade de monitoramento em tempo real para detecção e resposta a incidentes de segurança é crucial. As fábricas inteligentes dependem de dados em tempo real para operar eficientemente, e qualquer interrupção pode causar paradas significativas na produção. Implementar sistemas de monitoramento eficazes que possam detectar e mitigar ameaças imediatamente é um grande desafio (FERENCZ *et al.*, 2021).

2.2.1.4 Gestão de Identidades e Acessos

Com a proliferação de dispositivos e usuários, a Gestão de Identidades e Acessos (IAM) se torna mais complexa. Garantir que apenas usuários autorizados e dispositivos autenticados possam acessar os sistemas críticos é vital para evitar acessos não autorizados e proteger informações sensíveis (FERENCZ *et al.*, 2021).

2.2.1.5 Proteção de Dados

A Indústria 4.0 gera e utiliza grandes volumes de dados sensíveis, incluindo dados de produção, informações proprietárias e dados pessoais de funcionários. Proteger esses dados contra vazamentos e acessos não autorizados é um desafio constante, exigindo criptografia robusta e políticas de segurança de dados rigorosas (FERENCZ *et al.*, 2021).

2.2.1.6 Conformidade Regulamentar

As empresas precisam cumprir uma variedade de normas e regulamentos de segurança cibernética e privacidade de dados, como Lei Geral de Proteção de Dados Pessoais (LGPD). Manter a conformidade com essas regulamentações exige recursos significativos e um entendimento aprofundado das exigências legais (FERENCZ *et al.*, 2021).

2.2.1.7 Integração de Tecnologias Legadas

Muitas indústrias utilizam equipamentos e sistemas legados que não foram projetados com a segurança cibernética em mente. Integrar esses sistemas em um ambiente de Indústria 4.0 sem comprometer a segurança é um desafio significativo, muitas vezes exigindo a atualização ou substituição de equipamentos antigos (FERENCZ *et al.*, 2021).

2.2.1.8 Educação e Treinamento

A falta de conhecimento e habilidades em segurança cibernética entre os funcionários pode representar um risco significativo. Investir em educação e treinamento contínuos para todos os níveis da organização é essencial para criar uma cultura de segurança e garantir que todos estejam preparados para identificar e responder a ameaças (FERENCZ *et al.*, 2021).

2.3 *Malware*

O termo *"malware"* é uma abreviação de *"malicious software"* (programa malicioso) e se refere a qualquer programa ou arquivo prejudicial a um sistema de computador. O *malware* é projetado para causar danos, interromper operações, roubar dados confidenciais ou obter acesso não autorizado a sistemas. Ele assume várias formas, incluindo vírus, *worms, trojans, ransomware, spyware, adware e rootkits* (ABOAOJA *et al.*, 2022).

A proliferação de *malware* é impulsionada pelo aumento do uso da *Internet* e de serviços digitais, como bancos e compras *online*, que oferecem um ambiente propício para crimes cibernéticos. Criadores de *malware* utilizam técnicas de obfuscação e evasão para evitar a detecção, tornando o combate a essas ameaças um desafio contínuo para a comunidade de segurança cibernética.

Segundo Aboaoja *et al.* (2022), a análise de *malware* é crucial para entender suas características e funções principais. Existem três abordagens principais:

- Análise Estática: Avalia o código fonte do *malware* sem executá-lo. Embora seja rápida, pode ser evitada por códigos bem obfuscados.
- Análise Dinâmica: Observa o comportamento do *malware* durante sua execução. É mais difícil a evasão, mas consome mais recursos.
- Análise Híbrida: Combina as abordagens estática e dinâmica para aproveitar os benefícios de ambas.

Para a detecção de *malware*, as abordagens incluem:

- Detecção Baseada em Assinaturas: Compara o *malware* com uma base de dados de assinaturas conhecidas. Eficaz para *malware* conhecido, mas ineficaz contra novas variantes.
- Detecção Baseada em Comportamento: Monitora o comportamento dos programas para identificar atividades maliciosas. Mais robusta contra técnicas de obfuscação, mas pode ser contornada por *malware* sofisticado.
- Detecção Heurística: Utiliza regras específicas para detectar comportamentos maliciosos. É uma abordagem flexível, mas é limitada a comportamentos predefinidos.

A utilização de técnicas de aprendizado de máquina tem se mostrado promissora na detecção de *malware*, permitindo o processamento de grandes volumes de dados e a identificação de padrões que podem não ser evidentes para analistas humanos (ABOAOJA *et al.*, 2022).

2.4 *Machine Learning* - ML

Machine Learning-ML, ou Aprendizado de Máquina, é uma área da Inteligência Artificial - IA focada no desenvolvimento de técnicas computacionais que permitem a sistemas adquirir conhecimento de forma automática. Sua natureza, envolve aprender a partir de dados e melhorar o desempenho em tarefas específicas através da experiência. Segundo (PATTERN. . . , 2007), ML visa aprender automaticamente relações e padrões significativos a partir de exemplos e observações.

O estudo de técnicas de ML é multidisciplinar e abrange inúmeros domínios de pesquisa e aplicações. Atualmente, é aplicado em diversas áreas, como mineração de texto, detecção de spam, recomendação de vídeo, classificação de imagens, detecção de objetos, recomendação de produtos e diagnóstico médico (ALZUBAIDI *et al.*, 2021).

Os algoritmos de ML aprendem iterativamente a partir de dados, descobrindo *insights* e padrões complexos. Os principais tipos de aprendizado são supervisionado, não supervisionado e por reforço (JANIESCH *et al.*, 2021).

- Aprendizado Supervisionado: Requer um conjunto de dados de treinamento com entradas e saídas rotuladas. Exemplos incluem regressão (previsão de valores numéricos) e classificação (previsão de categorias).
- Aprendizado Não Supervisionado: Detecta padrões sem rótulos preexistentes, incluindo técnicas como agrupamento (*clustering*) e redução de dimensionalidade.
- Aprendizado por Reforço: O sistema aprende por tentativa e erro para maximizar uma recompensa, útil em ambientes como jogos e sistemas multiagentes.

2.4.1 Redes Neurais Artificiais - ANNs

Redes Neurais Artificiais (*Artificial Neural Networks* - ANNs) são modelos computacionais inspirados na estrutura e funcionamento das redes neurais biológicas. Elas são compostas por unidades de processamento interconectadas chamadas neurônios artificiais, que trabalham em conjunto para resolver problemas de reconhecimento de padrões, aprendizado e tomada de decisão. As ANNs são capazes de aprender com exemplos, ajustando suas conexões (pesos sinápticos) com base em dados de treinamento, sem a necessidade de regras explícitas para realizar as tarefas. Essas redes são utilizadas em diversas aplicações, como reconhecimento de fala, visão computacional e processamento de linguagem natural (YEGNANARAYANA, 1994).

2.4.2 Deep Learning (Aprendizado Profundo)

Uma subárea do ML que usa redes neurais profundas *Deep Neural Networks* (DNNs) com múltiplas camadas ocultas. Devido à sua capacidade de modelar relações não lineares e capturar padrões complexos, as DNNs são amplamente utilizadas em tarefas como reconhecimento de imagem, processamento de linguagem natural e detecção de fraudes (MONTAVON *et al.*, 2018; ALZUBAIDI *et al.*, 2021).

2.4.3 Desempenho e Aplicabilidade

Algoritmos de ML são amplamente aplicáveis em áreas como detecção de fraudes, análise de crédito, reconhecimento de fala e imagem, Processamento de Linguagem Natural (NLP), entre outros. Eles extraem regularidades de grandes bases de dados, produzindo decisões confiáveis e repetíveis (ALZUBAIDI *et al.*, 2021).

2.4.4 Desafios na Implementação

A implementação de sistemas de ML enfrenta desafios como a escolha adequada dos parâmetros de implementação, gestão de viés e inconsistência nos dados, usufruir das propriedades preconfiguradas dos modelos, e reutilização de modelos preconfigurados (como um serviço) (ALZUBAIDI *et al.*, 2021).

Ademais, modelos mais simples geralmente não tendem a ser flexíveis o suficiente para capturar regularidades e padrões não lineares que são relevantes para a tarefa de aprendizado. Por outro lado, modelos excessivamente complexos apresentam um risco maior de sobreajuste. Além disso, é mais difícil de interpretar, e eles tendem a ser computacionalmente mais caros. Os custos computacionais são expressos pelos requisitos de memória e pelo tempo de inferência necessário para executar um modelo em novos dados. Esses critérios são particularmente importantes ao avaliar redes neurais profundas, onde vários milhões de parâmetros do modelo podem ser processados e armazenados, impondo demandas especiais aos recursos de *hardware* (JANIESCH *et al.*, 2021).

2.5 Decision Tree

Árvores de Decisão são modelos preditivos utilizados para classificação e regressão. Esses modelos utilizam uma estrutura em forma de árvore, onde cada nó interno representa uma "decisão" baseada em uma característica dos dados. Cada ramo representa o resultado da escolha, e cada folha representa uma classe ou um valor de saída. A construção de uma árvore de decisão envolve a divisão recursiva dos dados de treinamento em subconjuntos, com o objetivo de maximizar a separação das classes ou minimizar o erro na predição (QUINLAN, 1986).

2.5.1 Estrutura de uma árvore de decisão

- Nó Raiz: O nó inicial, onde começa a primeira divisão dos dados.
- Nós Internos: Nós que representam atributos nos quais os dados são divididos.
- Ramos: Linhas que conectam nós e representam os resultados dos testes de decisão.
- Folhas: Nós terminais que indicam a classe ou valor de previsão final.

2.5.1.1 Construção da Árvore

- Seleção de Atributos: Em cada nó, o algoritmo seleciona o atributo que melhor divide os dados. Isso é geralmente feito usando métricas como ganho de informação, índice Gini ou redução de variância.
- Divisão dos Dados: Os dados são divididos com base no valor do atributo selecionado.
- Recursão: O processo é repetido recursivamente para cada subconjunto de dados gerado pela divisão, criando novos nós internos e ramos até que um critério de parada seja atendido (por exemplo, profundidade máxima da árvore ou número mínimo de amostras por folha).
- Podamento: Após a construção inicial, a árvore pode ser podada para reduzir a complexidade e evitar o *overfitting*[1], removendo ramos que têm pouca importância para a predição (QUINLAN, 1986).

[1] ***Overfitting***: O ajuste excessivo é um comportamento indesejável de aprendizado de máquina que ocorre quando o modelo de aprendizado de máquina fornece previsões precisas para dados de treinamento, mas não para novos dados. Disponível em: <https://aws.amazon.com/pt/what-is/overfitting/>. Acesso em: 14 jul. 2024

2.5.1.2 Vantagens e Desvantagens

- Vantagens: Simplicidade e interpretabilidade. Capacidade de lidar com dados categóricos e numéricos. Não requer normalização de dados.
- Desvantagens: Suscetibilidade ao *overfitting*. Instabilidade com pequenas variações nos dados. Pode gerar árvores muito complexas e de difícil interpretação.

(FILHO, 2023)

2.6 *Random Forest*

O algoritmo *Random Forest* é um dos métodos de aprendizado de máquina mais proeminentes e eficazes, especialmente em tarefas de previsão. Desenvolvido por Leo Breiman e Adele Cutler, *Random Fores*t baseia-se em métodos de *ensemble learning*, especificamente na técnica de *"bagging" (bootstrap aggregating)*, que visa aumentar a precisão preditiva e reduzir o *overfitting* de modelos individuais de árvore de decisão.

O algorítmo é composto por uma coleção de árvores de decisão, onde cada árvore é construída utilizando uma amostra *bootstrap* do conjunto de dados original. Esse processo de amostragem com reposição assegura que algumas observações sejam repetidas nas amostras enquanto outras são deixadas de fora (*out-of-bag*, OOB). Além disso, em cada nó de decisão dentro de uma árvore, um subconjunto aleatório de variáveis é selecionado para determinar a melhor divisão. Este procedimento é repetido para todas as árvores da floresta, garantindo a diversidade entre elas (SCHONLAU; ZOU, 2020).

As árvores de decisão particionam recursivamente o espaço de entrada em subespaços com base em critérios de divisão até que uma condição de parada predeterminada seja atingida. As folhas das árvores contêm as previsões finais. Para problemas de classificação, um critério de divisão frequentemente utilizado é a entropia, que mede a desordem ou incerteza dos dados. Para problemas de regressão, o critério geralmente é o erro quadrático médio. O processo de *bootstrap aggregating (bagging)* é essencial no *Random Forest*, onde cada árvore de decisão é treinada em uma amostra bootstrap do conjunto de dados original. Este método ajuda a reduzir o sobreajuste, pois a combinação de múltiplas árvores menos correlacionadas resulta em uma melhor generalização dos dados. As previsões finais são feitas agregando os resultados das árvores individuais, seja por votação majoritária (para classificação) ou pela média das previsões (para regressão)

Random Forest apresenta várias vantagens significativas, incluindo redução de *overfitting*, pois combinação de várias árvores de decisão reduz o risco de sobreajuste, um problema comum em árvores de decisão individuais. Alcança também, maior precisão de previsão, pois o algoritmo tende a oferecer previsões mais precisas do que métodos como a regressão linear, especialmente em casos de dados não lineares. Além disso, a taxa de erro do *Random Forest* pode ser estimada de forma precisa utilizando o erro *out-of-bag* (OOB), onde cada observação que não é incluída na amostra bootstrap para uma árvore é usada para validar essa árvore.

2.7 *Extra Trees*

O *Extra Trees Classifier*, também conhecido como *Extremely Randomized Trees*, é um método de aprendizado de máquina baseado em árvores de decisão, que utiliza a técnica de *ensemble*[2] para melhorar a precisão preditiva e controlar o *overfitting*. Este classificador é uma variação do *Random Forest* 2.6, diferenciando-se principalmente pela maneira como as árvores são construídas e pelas escolhas de divisões (*splits*) mais aleatórias. É um algoritmo que agrega os resultados de múltiplas árvores de decisão para realizar a classificação. Ele introduz uma camada adicional de aleatoriedade no processo de construção das árvores, que pode levar a uma redução da variância e, consequentemente, a um melhor desempenho em dados não vistos (generalização) (GEURTS *et al.*, 2006).

Ao contrário do *Random Forest*, que utiliza amostragem com reposição (*bootstrap sampling*) para selecionar subconjuntos de dados, o Extra Trees geralmente utiliza toda a amostra de treinamento disponível para construir cada árvore. Contudo, também é possível utilizar amostragem com reposição se configurado pelo usuário.

Para cada nó em uma árvore, o *Extra Trees Classifier* seleciona uma amostra aleatória dos atributos e escolhe aleatoriamente os pontos de divisão (*thresholds*) para cada atributo. Esta escolha não é baseada em um critério de otimização local, como a redução da impureza de Gini ou do índice de entropia, mas sim em *thresholds* aleatórios, o que introduz uma quantidade significativa de aleatoriedade no modelo. As árvores são crescidas até a máxima profundidade ou até que outros critérios de parada (como número mínimo de amostras por folha) sejam atendidos. Não há poda (*pruning*) das árvores, permitindo que cada árvore se ajuste completamente aos dados de treinamento (GEURTS *et al.*, 2006).

[2] ***Ensemble learning*** é uma técnica de aprendizado de máquina que agrega dois ou mais modelos (por exemplo, modelos de regressão, redes neurais) para obter melhores previsões. Disponível em: <https://www.ibm.com/topics/ensemble-learning>. Acesso em: 14 jul. 2024.

A predição final do *Extra Trees Classifier* é obtida pela agregação (geralmente por meio de votação majoritária) das predições de todas as árvores individuais. Este método de agregação ajuda a suavizar as predições e a reduzir o *overfitting* (GEURTS *et al.*, 2006).

2.8 *K-Nearest Neighbors* - KNN

O KNN é um algoritmo de aprendizado supervisionado amplamente utilizado em tarefas de classificação e regressão. Sua popularidade se deve à simplicidade de implementação e à eficácia em uma variedade de aplicações, especialmente em problemas onde as relações entre dados são não lineares e complexas. Ele opera com base no princípio da proximidade: ele assume que os exemplos de dados que estão próximos uns dos outros no espaço de características compartilham características semelhantes. Esse algoritmo armazena todos os exemplos de treinamento e, para fazer uma predição, identifica os k exemplos mais próximos no espaço de características (RASCHKA, 2018).

O processo do KNN pode ser dividido em duas fases principais: treinamento e predição.

- Treinamento: O KNN é um algoritmo baseado em instâncias que não requer uma fase de treinamento explícita. Em vez disso, ele simplesmente armazena os exemplos de treinamento.
- Predição: Para prever a saída para um novo ponto de dados, o algoritmo calcula a distância entre esse ponto e todos os pontos de treinamento. Os k pontos mais próximos (vizinhos) são então selecionados. Para classificação: A classe do ponto de dados é determinada pela classe mais frequente entre os k vizinhos. Para regressão: A saída é a média dos valores dos k vizinhos.

A precisão do KNN depende da métrica de distância escolhida. A distância Euclidiana é a mais comumente usada, definida como a equação 2.1:

$$\text{dist}(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2} \tag{2.1}$$

Outras métricas, como a distância de Manhattan, Minkowski e a distância de Mahalanobis, podem ser usadas dependendo da natureza dos dados.

O KNN é intuitivo e fácil de entender, e pode ser aplicado tanto em problemas de classificação quanto de regressão, além disso, é eficaz em dados onde não colinearidade entre

os atributos. Entretanto, há algumas desvantagens, como em espaços de alta dimensionalidade, cuja performance do KNN pode deteriorar significativamente pois as distâncias tornam-se menos significativas. Além disso, a necessidade de calcular a distância para todos os pontos de treinamento pode ser computacionalmente intensiva, especialmente para grandes conjuntos de dados. O KNN requer armazenamento de todos os exemplos de treinamento, o que pode ser inviável para grandes volumes de dados (RASCHKA, 2018). Esses problemas endossam a *Curse of Dimensionality*, explanada a seguir:

> A maldição da dimensionalidade(*Curse of Dimensionality*) refere-se ao impacto negativo que um número alto de dimensões tem no desempenho do KNN. À medida que o número de dimensões aumenta, o volume do espaço de características cresce exponencialmente, diluindo a densidade dos dados. Isso faz com que os pontos de dados fiquem, em média, equidistantes uns dos outros, tornando difícil diferenciar entre os vizinhos mais próximos e os mais distantes. Essa característica reduz a eficácia, pois a noção de proximidade perde seu significado (RASCHKA, 2018).

2.9 *Gaussian Naive Bayes*

O *Gaussian Naive Bayes* é um classificador probabilístico baseado no Teorema de *Bayes* com a suposição de independência ingênua entre as características. É uma variante do algoritmo *Naive Bayes* que assume que os dados contínuos se distribuem de acordo com uma distribuição Gaussiana (normal). Essa técnica é amplamente utilizada devido à sua simplicidade, eficiência e desempenho razoavelmente bom em muitas aplicações práticas, como classificação de texto, detecção de spam, e reconhecimento de padrões (WEINBERGER, 2018).

2.9.1 Teorema de Bayes

O Teorema de *Bayes* fornece uma maneira de calcular a probabilidade posterior de uma hipótese y dada uma evidência x:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \tag{2.2}$$

onde:

- $P(y|x)$ é a probabilidade posterior de y dado x.
- $P(x|y)$ é a probabilidade de x dado y (verossimilhança).
- $P(y)$ é a probabilidade a priori de y.
- $P(x)$ é a probabilidade marginal de x.

2.9.2 Suposição de Independência Ingênua

O *Naive Bayes* faz a Suposição "ingênua" de que as características são condicionalmente independentes, dado a classe y:

$$P(x|y) = \prod_{\alpha=1}^{d} P(x_\alpha|y) \tag{2.3}$$

onde x_α representa o valor da α-ésima característica. Esta suposição simplifica significativamente o cálculo da verossimilhança.

2.9.3 Modelo Gaussiano

Para dados contínuos, o *Gaussian Naive Bayes* assume que cada característica segue uma distribuição Gaussiana (normal):

$$P(x_\alpha|y=c) = \frac{1}{\sqrt{2\pi\sigma_{\alpha,c}^2}} \exp\left(-\frac{(x_\alpha - \mu_{\alpha,c})^2}{2\sigma_{\alpha,c}^2}\right) \tag{2.4}$$

onde:

- $\mu_{\alpha,c}$ é a média da α-ésima característica para a classe c.
- $\sigma_{\alpha,c}^2$ é a variância da α-ésima característica para a classe c.

2.9.4 Estimação dos Parâmetros

Os parâmetros $\mu_{\alpha,c}$ e $\sigma_{\alpha,c}^2$ são estimados a partir dos dados de treinamento:

$$\mu_{\alpha,c} = \frac{1}{n_c} \sum_{i=1}^{n} I(y_i = c) x_{i\alpha} \tag{2.5}$$

$$\sigma_{\alpha,c}^2 = \frac{1}{n_c} \sum_{i=1}^{n} I(y_i = c)(x_{i\alpha} - \mu_{\alpha,c})^2 \tag{2.6}$$

onde n_c é o número de amostras na classe c.

2.9.5 Classificação

Para classificar uma nova amostra x, calculamos a probabilidade posterior para cada classe y e escolhemos a classe que maximiza essa probabilidade:

$$h(x) = \arg\max_y P(y|x) = \arg\max_y P(x|y)P(y) \tag{2.7}$$

Com a suposição de independência ingênua e a distribuição Gaussiana, isso se traduz em:

$$h(x) = \arg\max_y \log(P(y)) + \sum_{\alpha=1}^{d} \log(P(x_\alpha|y)) \tag{2.8}$$

O *Gaussian Naive Bayes* pode ser visto como um classificador linear sob certas condições. Quando as variâncias das características são iguais entre as classes, o classificador resulta em uma fronteira de decisão linear. Isso é semelhante ao modelo de regressão logística:

$$P(y|x) = \frac{1}{1+\exp(-y(w^\top x+b))} \tag{2.9}$$

onde w e b são parâmetros que podem ser determinados a partir das probabilidades condicionais e a priori.

O *Gaussian Naive Bayes* é uma poderosa técnica de classificação que, apesar de suas suposições simplificadoras, frequentemente desempenha bem em muitos cenários práticos. Entre suas vantagens estão a simplicidade e a eficiência computacional, a necessidade de uma quantidade relativamente pequena de dados de treinamento para estimar os parâmetros necessários, e a escalabilidade para grandes conjuntos de dados. Contudo, apresenta desvantagens como a suposição de independência ingênua, que raramente se mantém na prática e pode levar a previsões subótimas, e a sensibilidade à precisão das estimativas de probabilidade, especialmente em dados com distribuição não Gaussiana (WEINBERGER, 2018).

2.10 Métricas de Avaliação

No desenvolvimento de algoritmos de aprendizado de máquina, é fundamental utilizar métricas de avaliação adequadas para medir o desempenho dos modelos. A qualidade de um modelo é determinada pelos valores dessas métricas, logo, é fundamental escolher ferramentas de avaliação adequadas ao modelo e que atinja os objetivos de análise.

Problemas de classificação envolvem a categorização de dados em diferentes classes, podendo ser binárias ou multiclasses. A classificação binária decide a qual de duas classes

possíveis uma nova observação pertence, enquanto a classificação multiclasses lida com mais de duas classes (NASER; ALAVI, 2021).

Em um modelo de classificação binária, existem duas classes: positiva (+) e negativa (-), que indicam a ocorrência ou não de um determinado evento. A avaliação de um modelo de classificação é feita comparando as classes preditas pelo modelo com as classes reais, utilizando a matriz de confusão (SAMMUT; WEBB, 2010).

A matriz de confusão resume o desempenho de um modelo de classificação em relação aos dados de teste. É uma matriz bidimensional indexada pela classe verdadeira e pela classe predita (MONARD; BARANAUSKAS, 2003).

Para um problema de classificação binária, a matriz de confusão inclui:

- **Verdadeiro Positivo (VP)**: número de exemplos da classe positiva classificados corretamente.
- **Verdadeiro Negativo (VN)**: número de exemplos da classe negativa classificados corretamente.
- **Falso Positivo (FP)**: número de exemplos da classe negativa classificados incorretamente como positivos.
- **Falso Negativo (FN)**: número de exemplos da classe positiva classificados incorretamente como negativos.

Então, o número total de elementos do conjunto é definido como (LORENA *et al.*, 2021):

$$\mathrm{n} = \mathrm{VP} + \mathrm{FN} + \mathrm{FP} + \mathrm{VN} \tag{2.10}$$

Tabela 1 – Matriz de Confusão

	Classe predita 0	**Classe predita 1**
Classe verdadeira 0	VN	FP
Classe verdadeira 1	FN	VP

Fonte: Adaptado de (LORENA *et al.*, 2021).

Diversas medidas de desempenho podem ser derivadas da matriz de confusão, incluindo:

- **Acurácia (acc)**: Proporção de previsões corretas em relação ao número total de amostras.

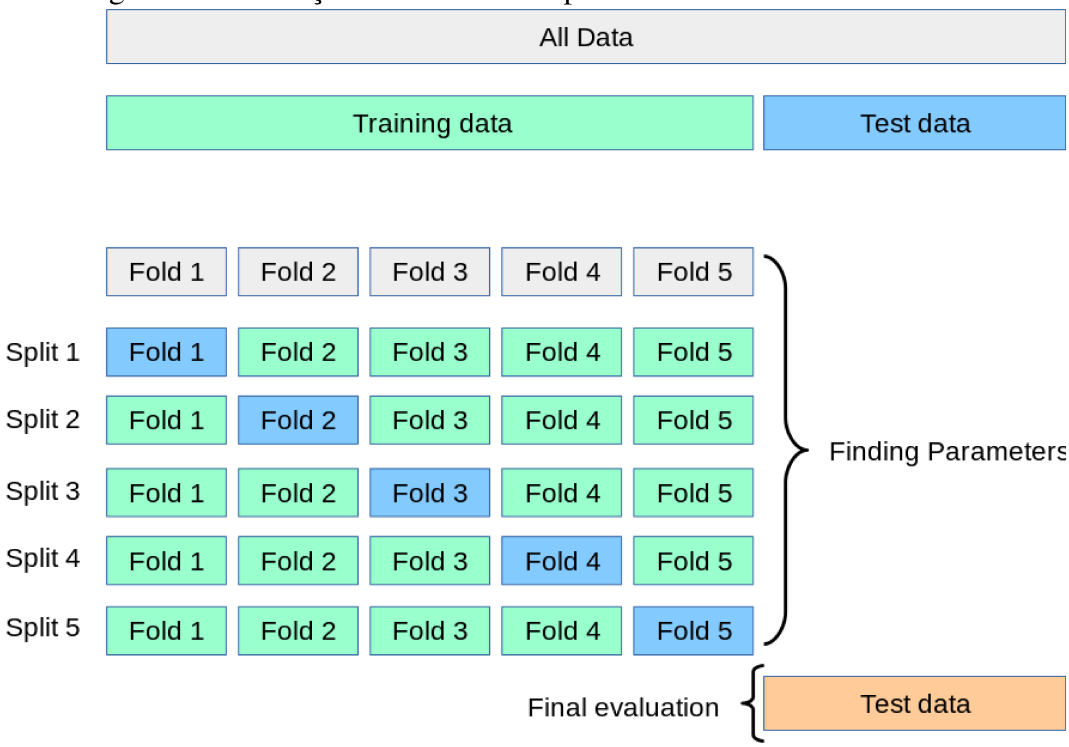
$$\mathrm{acc} = \frac{\mathrm{VP} + \mathrm{VN}}{n} \tag{2.11}$$

2.10.1 Validação Cruzada

A validação cruzada é um método de reamostragem de dados usado para avaliar a capacidade de generalização dos modelos e evitar *overfitting*. O *overfitting* ocorre quando o modelo se ajusta perfeitamente ao conjunto de dados disponível, mas falha em generalizar para novos dados (BERRAR, 2018).

O método básico de validação cruzada é o k-fold. Nesse método, o conjunto de dados é dividido aleatoriamente em k subconjuntos (*folds*). O modelo é treinado com k-1 folds e validado com o *fold* restante. Esse processo se repete até que cada um dos k subconjuntos tenha sido usado como conjunto de validação. A média do desempenho nos k subconjuntos determina o desempenho final da validação cruzada. A Figura 1 ilustra o processo da validação cruzada k-fold para k = 5.

Figura 1 – Validação Cruzada k-fold para k = 5



Fonte: (SCIKIT-LEARN, 2007)

3 METODOLOGIA

Nesta seção, encontra-se a metodologia empregada na elaboração deste trabalho. Ademais, serão abordados detalhes sobre a base de dados utilizada, os *softwares* e bibliotecas empregados, assim como o fluxo dos experimentos realizados no desenvolvimento deste projeto.

3.1 Fluxo de Desenvolvimento

Inicialmente, a Seção 3.2 apresenta a base de dados utilizada na construção do trabalho. Posteriormente, na Seção 3.3, são expostas as linguagens, bibliotecas e plataformas utilizadas na implementação deste projeto. Logo após, a Seção 3.4 descreve as transformações aplicadas aos dados. Por fim, a Seção 3.5 contém os detalhes da etapa de classificação. Nesse estágio, para realizar o treinamento dos modelos, foram empregadas os conjuntos de dados resultantes das etapas anteriores.

3.2 Base de Dados

Kaur *et al.* (2023) exploram diversas bases de dados relacionadas à segurança no contexto IoT, destacando suas principais características e aplicações, além de comparar o desempenho de diferentes técnicas de ML. Entre essas bases, a IoT-23 se sobressai pelo grande volume de dados e pelo foco nos atributos de conexão de rede. No entanto, faltam informações sobre a implementação de técnicas de ML nessa base. Por isso, a IoT-23 é selecionada para a avaliação das técnicas de ML.

A base de dados IoT-23(GARCIA *et al.*, 2021) ultilizada no desenvolvimentos dos experimentos, é um conjunto de dados de tráfego de rede de dispositivos IoT. Ela contém 20 capturas de tráfego infectado por *malware* e 3 capturas de tráfego inofensivo, em dispositivos IoT. Foi publicado pela primeira vez em janeiro de 2020, com capturas entre 2018 e 2019 no *Stratosphere Research Laboratory*[1]. Seu objetivo é fornecer um grande conjunto de dados de ataques reais e rotulados de *Malicious* e *Benign* para que os pesquisadores desenvolvam algoritmos de aprendizado de máquina (GARCIA *et al.*, 2021).

O conjunto de dados IoT-23 consiste em vinte e três capturas de diferentes tráfegos

[1] ***Stratosphere Research Laboratory***. *A cybersecurity group of the Artificial Intelligence Centre, Faculty of Electrical Engineering at the Czech Technical University in Prague. The group works at the intersection of cybersecurity, machine learning, and helping others*. Disponível em: <https://www.stratosphereips.org>. Acesso em: 04 jun. 2024

de rede IoT. Esses cenários estão divididos em vinte capturas de rede (arquivos pcap) de dispositivos IoT infectados e três capturas de rede do tráfego real de dispositivos IoT. Em cada cenário malicioso, foi executado um malware específico em uma Raspberry Pi[2], com vários protocolos e ações distintas (GARCIA *et al.*, 2021).

O tráfego de rede capturado foi obtido ao capturar o tráfego de rede de três dispositivos IoT diferentes: uma lâmpada inteligente Philips HUE[3], um assistente pessoal inteligente Amazon Echo[4] e uma fechadura inteligente Somfy[5]. Vale ressaltar que esses três dispositivos são hardwares reais. Ambos os casos *malicious* e *benign* foram executados em um ambiente de rede controlado com conexão à Internet irrestrita, assim como qualquer outro dispositivo IoT real (GARCIA *et al.*, 2021).

A base de dados contém 25.003.363 registros e 23 atributos, sendo um deles a *label* que identifica a classe *Malicious* ou *Benign*. A Tabela 2 contém a descrição de todos os atributos da base de dados.

3.3 Linguagens, Bibliotecas e Ferramentas

Para o desenvolvimento dos classificadores, foi utilizada a linguagem de programação *Python*[6](Versão 3.10.12), por ser uma linguagem de código aberto com uma variedade de bibliotecas disponíveis, e muito difundida no desenvolvimento de ML.

Para manipulação dos dados, a biblioteca *Pandas* [7] foi ultilizada para leitura, limpeza, transformação, normalização e análises dos atributos.

A biblioteca de código aberto *scikit-learn*[8] é abrangente e oferece uma variedade de funcionalidades para diversas tarefas de aprendizado de máquina. E por isso, foi empregada no desenvolvimento dos modelos de classificação presentes neste projeto.

Finalmente, o ambiente de desenvolvimento escolhido foi o Google Colaboratory[9].

---

[2] ***Raspberry Pi*** *is a tiny, powerful computer*. Disponível em: <https://www.raspberrypi.com/for-home/>. Acesso em: 04 jun. 2024

[3] Lâmpada inteligente **Philips HUE** Disponível em: <https://www.philips-hue.com/pt-br>. Acesso em: 4 junho 2024.

[4] Assistente pessoal inteligente **Amazon Echo** Disponível em: <https://www.amazon.com.br/Echo-Dot-2Âł-GeraÃğÃčo-Black/dp/B01DFKC2SO>. Acesso em: 4 junho 2024.

[5] Fechadura inteligente **Somfy.** Disponível em: <https://www.somfy.pt/produtos/fechaduras-inteligentes>. Acesso em: 4 junho 2024.

[6] **Python**. Disponível em: <https://www.python.org/>. Acesso em: 4 junho 2024.

[7] **Pandas**. Disponível em: <https://pandas.pydata.org>. Acesso em: 4 junho 2024

[8] **scikit-learn**. Disponível em: <https://scikit-learn.org/stable/>. Acesso em: 4 junho 2024

[9] **Google Colaboratory**. Disponível em: <https://colab.research.google.com/>. Acesso em: 4 de junho 2024.

Tabela 2 – Descrição dos atributos da Base de Dados IoT-23

Nome	Descrição
ts	The timestamp of the connection event.
uid	A unique identifier for the connection.
id.orig_h	The source IP address.
id.orig_p	The source port.
id.resp_h	The destination IP address.
id.resp_p	The destination port.
proto	The network protocol used (e.g., "tcp").
service	The service associated with the connection.
duration	The duration of the connection.
orig_bytes	The number of bytes sent from the source to the destination.
resp_bytes	The number of bytes sent from the destination to the source.
conn_state	The state of the connection.
local_orig	Indicate whether the connection is considered local or not.
local_resp	Indicate whether the connection is considered local or not.
missed_byte	The number of missed bytes in the connection.
history	A history of connection states.
orig_pkts	The number of packets sent from the source to the destination.
orig_ip_bytes	The number of IP bytes sent from the source to the destination.
resp_pkts	The number of packets sent from the destination to the source.
resp_ip_bytes	The number of IP bytes sent from the destination to the source.
tunnel_parents	Indicates if this connection is part of a tunnel.
label	A label associated with the connection (e.g., "Malicious"or "Benign").
detailed-label	A more detailed description or label for the connection.

–

Fonte: Adapatado de (GARCIA *et al.*, 2021)

A plataforma fornece *notebooks* hospedados no Jupyter[10], permitindo o desenvolvimento e execução de scripts *Python*, com acesso a recursos computacionais como GPUs. Os *notebooks* são armazenados no Google Drive[11], um serviço de armazenamento, compartilhamento e sincronização de arquivos. Mais detalhes sobre o ambiente de execução estão disponíveis na Tabela 4.

3.4 Limpeza e Transformação dos Dados

3.4.1 Limpeza dos Dados

Nessa etapa, a limpeza nos dados foi executada antes de alimentá-los nos classificadores propostos. As colunas vazias `"local_orig"` e `"local_resp"` foram removidas e registros duplicados foram deletados. Em seguida, os valores ausentes e nulos foram preenchidos com zero. Como um algoritmo de aprendizado supervisionado será aplicado, a coluna `"Label"` foi

[10] **Jupyter**. Disponível em: <https://jupyter.org/>. Acesso em: 4 junho 2024.
[11] **Google Drive**. Disponível em: <https://www.google.com/drive/>. Acesso em: 4 junho 2024.

Tabela 3 – Ambiente de Execução Google Colaboratory: TPU v2, RAM alta.

Resourse	***Description***
Operational System	Ubuntu 22.04.4 LTS, Release: 22.04, Codename: jammy
RAM	334.6 GB
Memory Disk	225.3 GB
CPUs	96
Vendor ID:	GenuineIntel
Model name:	Intel(R) Xeon(R) CPU @ 2.00GHz
CPU family:	6
Model:	85
Thread(s) per core:	2
Core(s) per socket:	24
Socket(s):	2

Fonte: Elaborado pela autor.

modificada para especificar fluxos de *malware* e benignos com 1 (um) e 0 (zero), respectivamente. Após essa etapa, o conjunto de dados resultante apresenta 17 atributos e uma coluna que indica a classe (0,1).

3.4.2 Transformação dos Dados

Há várias características não numéricas no conjunto de dados IoT-23 e os valores de entrada dos classificadores devem ser vetores numéricos. Para garantir a conformidade, os endereços IP de origem(‘‘`id.orig_h`’’) e destino(‘‘`id.resp_h`’’) são convertidos para o tipo numérico float64 [12]. Geralmente, um endereço *Internet Protocol* (IP) contém dígitos e é formatado em quatro partes de 8 bits separadas por um ponto. Além disso, as variáveis categóricas, como ‘‘`proto`’’ (que é usada para o protocolo de rede) e ‘‘`id.orig_p`’’‘‘`id.resp_p`’’ que identificam as portas do endereço IP, são convertidas para variáveis inteiras, dessa forma, há um representante inteiro para cada categoria em cada um desses atributos.

3.4.3 Normalização dos Dados

Para normalizar cada valor no conjunto de dados, o Z-score é usado de modo que a média de todos os valores seja 0 e o desvio padrão seja 1. O Z-score é defindo da seguinte forma:

$$z = \frac{x - \mu}{\sigma} \tag{3.1}$$

Onde x é uma amostra em um conjunto de dados, μ é a média dos dados e σ é o

[12] **float64** é um tipo de dado de ponto flutuante de 64 bits. Disponível em: <https://numpy.org/doc/stable/user/basics.types.html>. Acesso em: 10 julho 2024.

desvio padrão das amostras. Para calcular o valor z, a função `StandardScaler` da biblioteca scikit-learn é usada.

3.4.4 Seleção de Atributos

A fim de obter um melhor desempenho, utilizou-se a *Principal Component Analysis* (PCA)[13] para identificar os componentes principais que capturam a maior parte da variância nos dados.

Para os dados sem normalização, a variância explicada por cada componente principal indica a seleção de 15 componentes principais que explicam pelo menos 99% da variância, listados a seguir: [‘id.orig_h’, ‘id.orig_p’, ‘id.resp_h’, ‘id.resp_p’, ‘proto’, ‘service’, ‘duration’, ‘orig_bytes’, ‘resp_bytes’, ‘conn_state’, ‘missed_bytes’, ‘orig_pkts’, ‘orig_ip_bytes’, ‘resp_pkts’, ‘resp_ip_bytes’]. Além da PCA, foi elaborada uma matriz de correlação dos dados para auxiliar na seleção. Observa-se na Figura 3 que os atributos “`history`” e “`conn_state`” apresentam uma correlação de 0,99. Por isso, um deles foi removido. Entretanto, “history” não está no grupo de componentes principais da PCA, então já estava previamente excluído do conjunto de dados. Logo, os atributos selecionados para os dados não normalizados são: [‘id.orig_h’, ‘id.orig_p’, ‘id.resp_h’, ‘id.resp_p’, ‘proto’, ‘service’, ‘duration’, ‘orig_bytes’, ‘resp_bytes’, ‘conn_state’, ‘missed_bytes’, ‘orig_pkts’, ‘orig_ip_bytes’, ‘resp_pkts’, ‘resp_ip_bytes’].

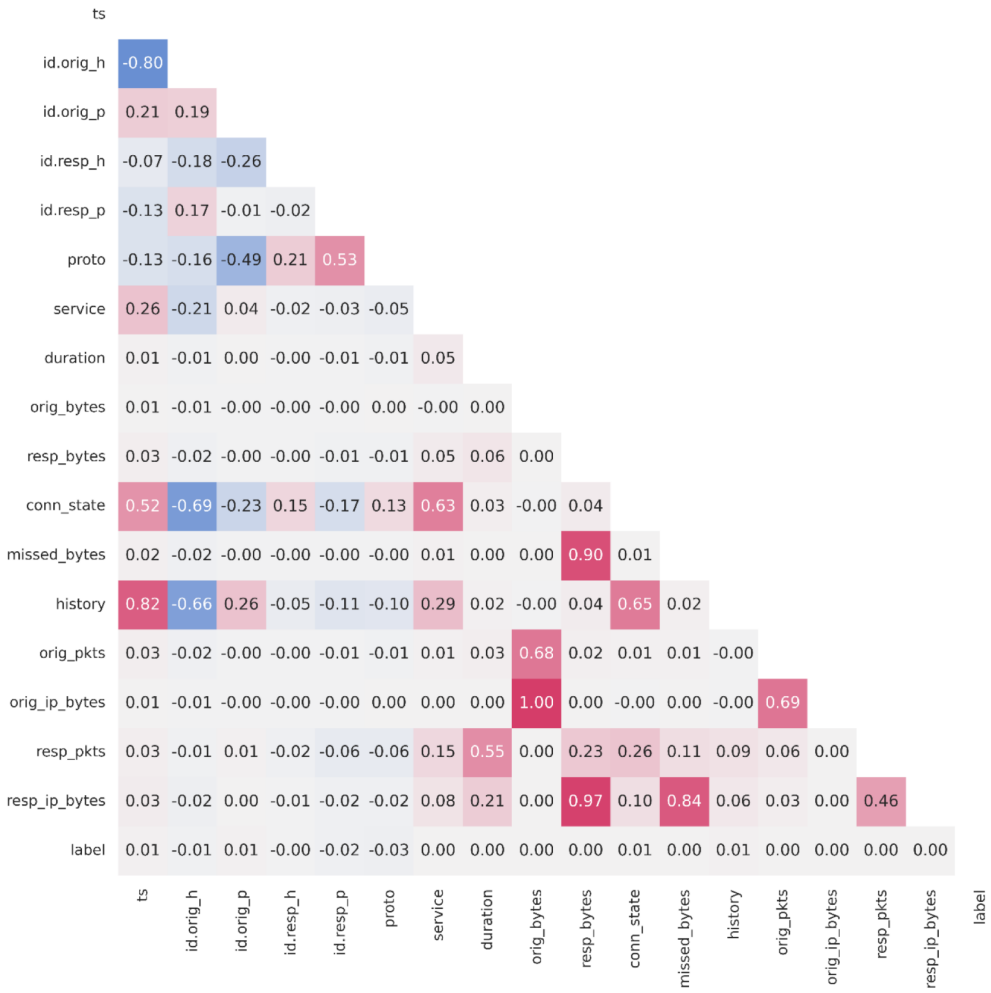
Para os dados normalizados, a variância explicada por cada componente principal indica a seleção de 13 componentes principais que explicam pelo menos 99% da variância, listados a seguir: [‘id.orig_p’, ‘id.resp_h’, ‘id.resp_p’, ‘proto’, ‘service’, ‘duration’, ‘orig_bytes’, ‘resp_bytes’, ‘history’, ‘orig_pkts’, ‘orig_ip_bytes’, ‘resp_pkts’, ‘resp_ip_bytes’]. Além da PCA, foi elaborada uma matriz de correlação dos dados para auxiliar na seleção. Observa-se na Figura 2 que os atributos “`orig_ip_bytes`” e "‘‘`orig_bytes`’’ apresentam uma correlação de 1,0, e os atributos “`resp_ip_bytes`’’ e “`resp_bytes`” apresentam correlação de 0,97. Por isso, um deles deve ser excluído de cada par. Então, as colunas “`orig_ip_bytes`” e “`resp_ip_bytes`” são deletadas do conjunto de dados. Assim, os atributos selecionados para os dados normalizados são: [‘id.orig_p’, ‘id.resp_h’, ‘id.resp_p’, ‘proto’, ‘service’, ‘duration’, ‘orig_bytes’, ‘resp_bytes’, ‘history’, ‘orig_pkts’, ‘resp_pkts’].

[13] A ***Principal Component Analysis*** reduz o número de dimensões em grandes conjuntos de dados para componentes principais que retêm a maior parte das informações originais. Ela faz isso transformando variáveis potencialmente correlacionadas em um conjunto menor de variáveis, chamadas de componentes principais. Disponível em: <https://www.ibm.com/topics/principal-component-analysis>. Acesso em: 4 junho 2024.

Dessa forma, após a normalização e seleção de atributos, há 4 conjuntos de dados, listados a seguir:

- Dados não normalizados e todos os atributos;
- Dados não normalizados e Atributos Selecionados - AS;
- Dados normalizados - DN e todos os atributos;
- Dados não normalizados - DN e Atributos Selecionados - AS;

Figura 2 – Matriz de Correlação dos Atributos Normalizados.

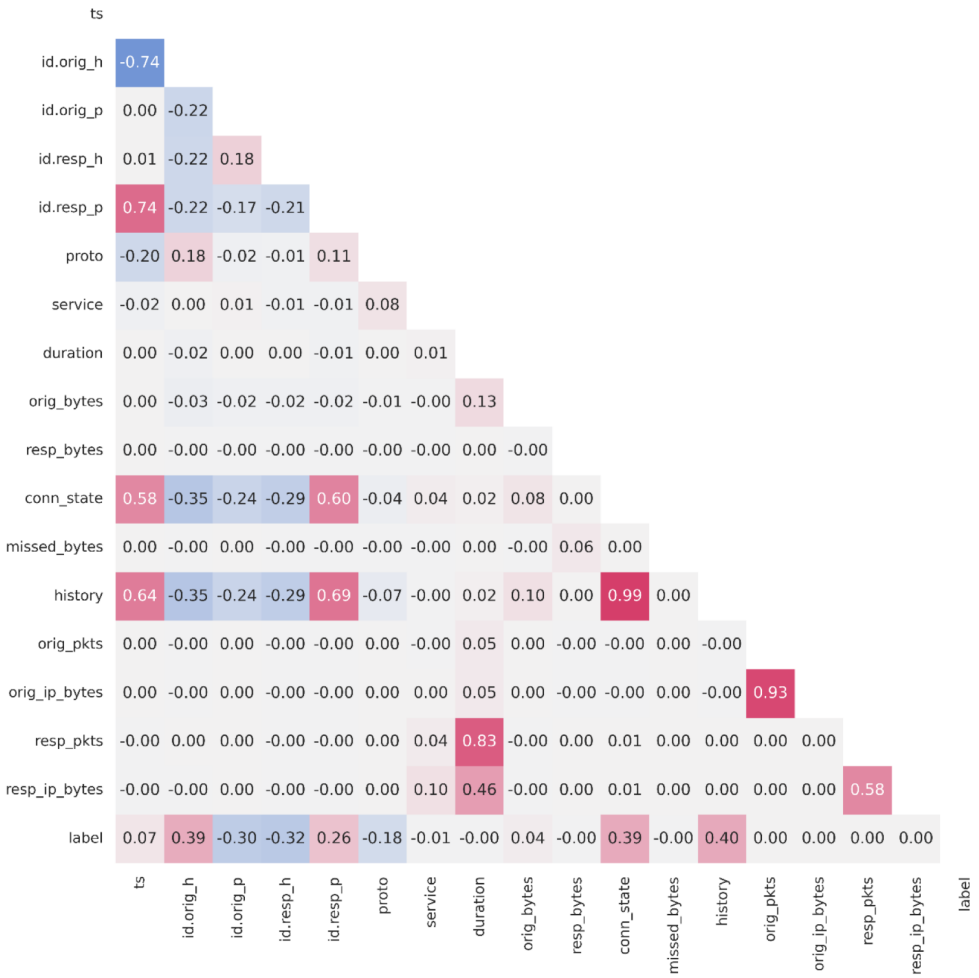


Fonte: Elaborado pelo autor.

3.5 Classificação

Para a detecção de *malware*, realizou-se o processo de classificação por meio de 5 técnicas: *Decision Tree* 2.5, *Random Forest* 2.6, *Extra Trees* 2.7, *K-Nearest Neighbors* 2.8 e *Gaussian Naive Bayes* 2.9. Na implementação dos classificadores foram utilizados os recursos da biblioteca *scikit-learn* da linguagem Python.

Figura 3 – Matriz de Correlação dos Atributos não Normalizados.



Fonte: Elaborado pelo autor.

Tudo foi executado em um ambiente no *Google Colaboratory Pro*, que fornece uma infraestrutura para desenvolvimento robusta o suficiente. Mais detalhes sobre o ambiente de execução estão disponíveis na Tabela 4.

Tabela 4 – Ambiente de Execução Google Colaboratory: TPU v2, RAM alta.

Resourse	***Description***
Operational System	Ubuntu 22.04.4 LTS, Release: 22.04, Codename: jammy
RAM	334.6 GB
Memory Disk	225.3 GB
CPUs	96
Vendor ID:	GenuineIntel
Model name:	Intel(R) Xeon(R) CPU @ 2.00GHz
CPU family:	6
Model:	85
Thread(s) per core:	2
Core(s) per socket:	24
Socket(s):	2

Fonte: Elaborado pela autor.

Foram selecionados, de cada um dos conjuntos de dados, de forma aleatória, 1 milhão de registros de não infectados por *malware* (*label=Benign*) e 1 milhão de registros infectados por *malware* (*label=Malicious*) para uso nos algoritmos de ML. Dessa forma, os dados para uso nos classificadores são balanceados.

Além disso, foi definida uma proporção de 1/5 para validação, dessa forma, a base de dados foi dividida, sendo 80% da base (1.600.000 registros) destinados ao treinamento dos modelos e 20% (400.000 registros) destinados aos testes dos modelos treinados. Seguindo a mesma proporção o K-Fold foi executado com k=5, ou seja, 5 segmentos.

3.5.1 Criação dos Modelos de Classificação

Para a criação dos modelos de classificação, foram utilizadas as classes correpondentes da biblioteca *scikit-learn*, definindo os hiperparâmetros, quando necessário.

Para cada conjunto de dados, foi executado o treinamento do modelo, e para teste, a classificação de todos os dados do conjunto destinado aos testes, e por fim, calculadas as métricas de avaliação. Além das métricas definidas na Seção 2.10, há mais três métricas adicionais para avaliar o desempenho observando o tempo de execução, são elas:

- ***Training Time***: Tempo de execução do treinamento do modelo com 80% da base de dados;
- ***Predict Time***: Tempo de execução da classificação da amostra de teste, equivalente a 20% da base de dados;
- ***Predict Unit Time***: Tempo de execução da classificação de uma amostra de teste, calculada com o *Predict Time* e a quantidade de registros na amostra de teste, seguindo o cálculo:

$$Predict\ Unit\ Time = \frac{Predict\ Time}{400.000} \tag{3.2}$$

- ***Decision Tree Classifier***: Esse classificador foi implementado usando a classe `DecisionTreeClassif` utilizando os hiperparâmetros *default*.
- ***Random Forest Classifier***: Esse classificador foi implementado usando a classe `RandomForestClassi` utilizando os hiperparâmetros *default*.

[14] **DecisionTreeClassifier**. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>. Acesso em: 8 junho 2024.

[15] **RandomForestClassifier**. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>. Acesso em: 8 junho 2024.

- ***Extra Trees Classifier***: Esse classificador foi implementado usando a classe `ExtraTreesClassifier`[16] utilizando os hiperparâmetros *default*.
- ***Gaussian Naive Bayes Classifier***: Esse classificador foi implementado usando a classe `GaussianNB`[17], utilizando os hiperparâmetros *default*.
- ***KNearest Neighbors Classifier***: Esse classificador foi implementado usando a classe `KNeighborsClassifier`[18], utilizando os hiperparâmetros *default*, exceto 'n_neighbors'=2, que define a quantidade de vizinhos próximos.

3.6 Validação Cruzada

Nesta etapa, aplica-se a validação cruzada k-fold com $k = 5$, usando a classe `KFold`[19], e a função `cross_val_score`[20]. Divide-se o conjunto de dados aleatoriamente em 5 partes. Em cada iteração, uma parte é usada para validação e as outras para treinamento. O processo é repetido até que todas as partes tenham sido utilizadas para teste. Ao final, avaliamos o desempenho do modelo pela média das acurácias obtidas.

3.7 Análise dos resultados

Para cada caso, foram construídas matrizes de confusão para apresentar de forma visual o desempenho dos modelos nos segmentos de teste, evidenciando os erros e acertos das previsões em contraste com as classes esperadas. Portanto, a matriz de confusão definida neste trabalho é uma matriz bidimensional indexada pela classe verdadeira (linhas) e pela classe predita (colunas) em cada dimensão, referenciando os rótulos de classe "*Benign*" e "*Malicious*".

Além disso, foram calculadas e agrupadas em tabelas as métricas de acurácia, *Training Time*, *Predict Time* e *Predict Unit Time* propostas para avaliar o desempenho de cada implementação.

---

[16] **ExtraTreesClassifier**. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html>. Acesso em: 8 junho 2024.

[17] **GaussianNB**. Disponível em: <https://scikit-learn.org/stable/modules/naive_bayes.html>. Acesso em: 8 junho 2024.

[18] **KNeighborsClassifier**. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>. Acesso em: 8 junho 2024.

[19] **KFold**. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html>. Acesso em: 8 junho 2024.

[20] **cross_val_score**. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html>. Acesso em: 8 junho 2024.

4 RESULTADOS

Nesta seção, apresentaremos os resultados dos experimentos realizados neste trabalho.

4.1 Classificação

4.1.1 Decision Tree

O classificador *Decision Tree* foi implementado e aplicado nos quatro cenários de conjunto de dados, dados não normalizados e todos os atributos, dados não normalizados e AS, DN e todos os atributos, e DN com AS. As Figuras 4, 5, 6 e 7 apresentam as matrizes de confusão para os cenários de dados não normalizados e todos os atributos, dados não normalizados e AS, DN e todos os atributos, e DN com AS, respectivamente. Elas mostram a quantidade de acertos que os classificadores alcançaram para as classes "*Benign*" e "*Malicious*".

A Tabela 5 apresenta os resultados de acurácia dos testes em cada *Fold* da validação cruzada, a média das cinco execuções e o desvio padrão. Além disso, a Tabela 6 mostra os resultados com as métricas de *Training Time*, *Predict time* e *Predict Unit Time* para todos os cenários.

Ademais, quando os dados não estão normalizados e não há seleção de atributos, a classificação se mostra mais assertiva, com um erro de apenas 2 casos dentre a amostra para teste, como mostra a Figura 4. Além disso, o tempo de treinamento do modelo é alguns segundos, o tempo de teste alguns décimos de segundo e o tempo de teste unitário uma fração de segundo na oitava casa decimal.

Tabela 5 – *Decision Tree*: Resultados K-Fold ($k = 5$) observando Acurácia.

	DN	**AS**	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Média**	**Desvio Padrão**
Decision Tree			**99,999**	**99,998**	**99,996**	**99,997**	**99,997**	**99,997**	**9.30** e^{-6}
Decision Tree		x	99,997	99,996	99,994	99,994	99,996	99,995	1.22 e^{-5}
Decision Tree	x	x	49,505	49,582	49,508	49,607	49,508	49,542	0,0004
Decision Tree	x		49,055	48,995	48,900	49,030	48,965	48,989	0.0005

Fonte: Elaborado pela autor.

Figura 4 – Decision Tree.

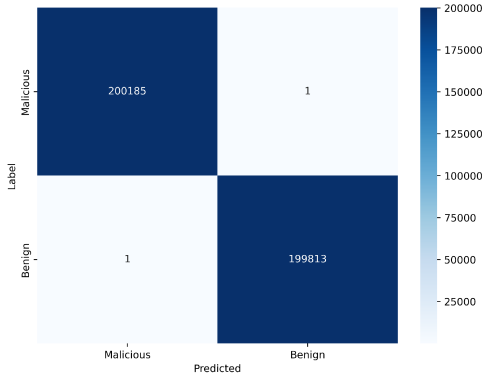


Figura 5 – Decision Tree: DN.

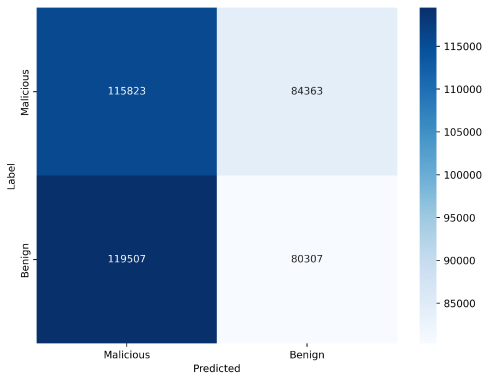


Figura 6 – Decision Tree: AS.

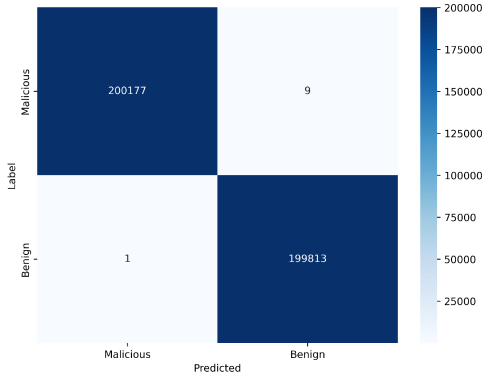
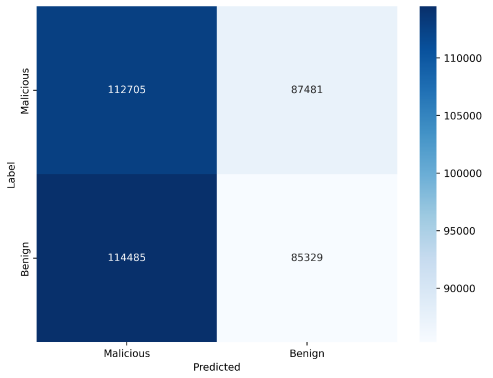


Figura 7 – Decision Tree: AS e DN.



Fonte: Elaborado pelo autor.

Tabela 6 – *Decision Tree*: Tempo de Execução.

	DN	AS	Training time	Predict time	Predict Unit Time
Decision Tree			**6,20 s**	**0,0311 s**	**7,78** $e^{-8}s$
Decision Tree		x	5,56 s	0,0314 s	7,85 $e^{-8}s$
Decision Tree	x		22,61 s	0,358 s	8,96 $e^{-7}s$
Decision Tree	x	x	3,03 s	0,0640 s	6,40 $e^{-7}s$

Fonte: Elaborado pela autor.

4.1.2 Random Forest

O classificador *Random Forest* foi implementado e aplicado nos quatro cenários de conjunto de dados, dados não normalizados e todos os atributos, dados não normalizados e AS, DN e todos os atributos, e DN com AS. As Figuras 8, 9, 10 e 11 apresentam as matrizes de confusão para os cenários de dados não normalizados e todos os atributos, dados não normalizados e AS, DN e todos os atributos, e DN com AS, respectivamente. Elas mostram a quantidade de acertos que os classificadores alcançaram para as classes "*Benign*" e "*Malicious*".

A Tabela 7 apresenta os resultados de acurácia dos testes em cada *Fold* da validação cruzada, a média das cinco execuções e o desvio padrão. Além disso, a Tabela 8 mostra os resultados com as métricas de *Training Time*, *Predict time* e *Predict Unit Time* para todos os cenários.

Ademais, quando os dados não estão normalizados e não há seleção de atributos, a classificação se mostra mais assertiva, com acurácia superior a 99,99%. Ademais, o tempo de treinamento do modelo foi mais de dois minutos, o tempo de teste custou alguns segundos e o tempo de teste unitário uma fração de segundo na sexta casa decimal.

Figura 8 – *Random Forest.*

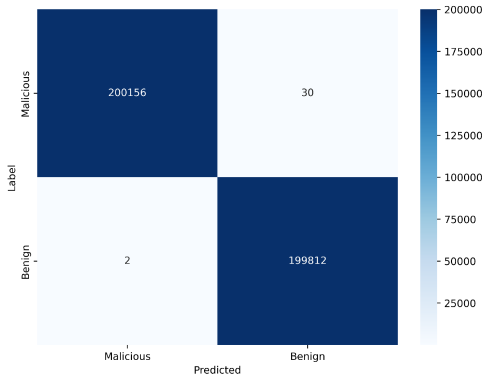


Figura 9 – *Random Forest*: DN.

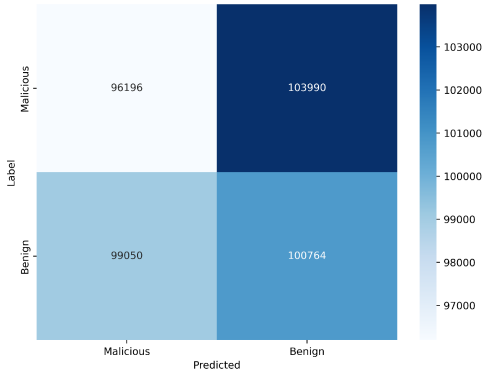


Figura 10 – *Random Forest*: AS.

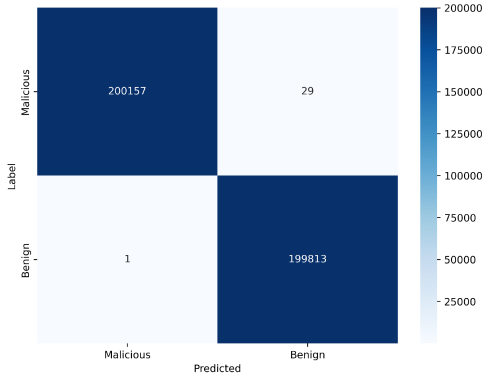
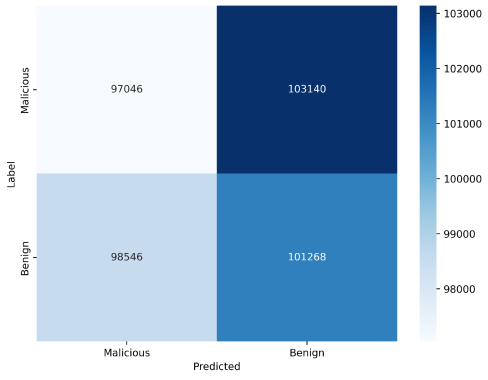


Figura 11 – *Random Forest*: AS e DN.



Fonte: Elaborado pelo autor.

Tabela 7 – *Random Forest*: Resultados K-Fold ($k = 5$) observando Acurácia.

	DN	AS	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Média	Desvio Padrão
Random Forest			**99,992**	**99,994**	**99,989**	**99,993**	**99,993**	**99,992**	**1,52** e^{-5}
Random Forest		x	99,993	99,993	99,989	99,991	99,992	99,991	1,56 e^{-5}
Random Forest	x		49,226	49,224	49,121	49,164	49,088	49,165	0,0005
Random Forest	x	x	49,585	49,591	49,546	49,621	49,563	49,581	0,0688

Fonte: Elaborado pela autor.

Tabela 8 – *Random Forest*: Tempo de Execução.

	DN	AS	Training time	Predict time	Predict Unit Time
Random Forest			**124.54 s**	**2.42 s**	**6,06** e^{-6} **s**
Random Forest		x	118,36 s	2.76 s	6,90 e^{-6} s
Random Forest	x		563,20 s	30,31 s	7,57 e^{-5} s
Random Forest	x	x	657,71 s	23,01 s	5,75 e^{-5} s

Fonte: Elaborado pela autor.

4.1.3 Extra Trees

O classificador *Extra Trees* foi implementado e aplicado nos quatro cenários de conjunto de dados, dados não normalizados e todos os atributos, dados não normalizados e AS, DN e todos os atributos, e DN com AS. As Figuras 12, 13, 14 e 15 apresentam as matrizes de confusão para os cenários de dados não normalizados e todos os atributos, dados não normalizados e AS, DN e todos os atributos, e DN com AS, respectivamente. Elas mostram a quantidade de acertos que os classificadores alcançaram para as classes "*Benign*" e "*Malicious*".

A Tabela 9 apresenta os resultados de acurácia dos testes em cada *Fold* da validação cruzada, a média das cinco execuções e o desvio padrão. Além disso, a Tabela 10 mostra os resultados com as métricas de *Training Time*, *Predict time* e *Predict Unit Time* para todos os cenários.

Ademais, quando os dados não estão normalizados, a classificação se mostra mais assertiva, independente da seleção dos atributps, com acurácia superior a 99,9%, como mostra as Figuraa 12 e 10. Ademais, os tempos de execução foram similares. Entretanto, os dados normalizados se mostraram com desempenho muito inferior em relação aos outros cenários.

Tabela 9 – *Extra Trees*: Resultados K-Fold ($k = 5$) observando Acurácia.

	DN	AS	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Média	Desvio Padrão
Extra Trees			99,981	99,980	99,976	99,979	99,972	99,978	3,08 e^{-5}
Extra Trees		x	**99,981**	**99,981**	**99,977**	**99,980**	**99,974**	**99,978**	**2,78** e^{-5}
Extra Trees	x		49,118	48,061	48,970	49,070	49,021	49,048	0,0004
Extra Trees	x	x	49,505	49,531	49,510	49,613	49,473	49,526	0,0004

Fonte: Elaborado pela autor.

Figura 12 – *Extra Trees.*

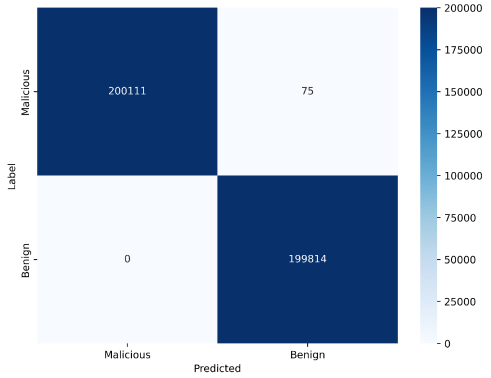


Figura 13 – *Extra Trees*: DN.

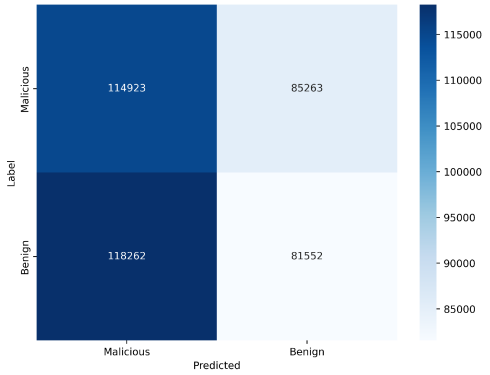


Figura 14 – *Extra Trees*: AS.

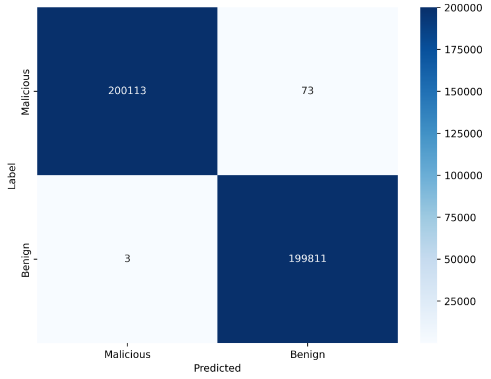
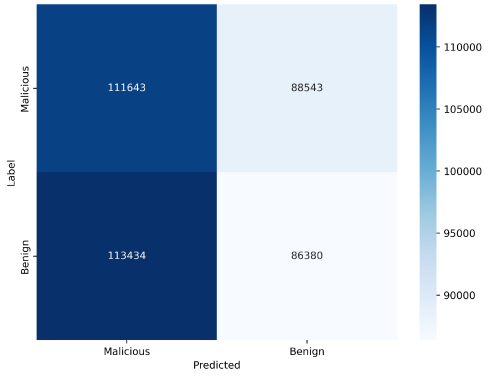


Figura 15 – *Extra Trees*: AS e AS.



Fonte: Elaborado pelo autor.

Tabela 10 – *Extra Trees*: Tempo de Execução.

	DN	AS	Training time	Predict time	Predict Unit Time
Extra Trees			**71,74 s**	**3,41 s**	**8,54** $e^{-6}s$
Extra Trees		x	71,73 s	4,11 s	1,02 $e^{-5}s$
Extra Trees	x		201,79 s	28,03 s	7,00 $e^{-5}s$
Extra Trees	x	x	165,42 s	22,74 s	5,68 $e^{-5}s$

Fonte: Elaborado pela autor.

4.1.4 K-Nearest Neighbors

O classificador KNN foi implementado e aplicado nos quatro cenários de conjunto de dados, dados não normalizados e todos os atributos, dados não normalizados e AS, DN e todos os atributos, e DN com AS. As Figuras 16, 17, 18 e 19 apresentam as matrizes de confusão para os cenários de dados não normalizados e todos os atributos, dados não normalizados e AS, DN e todos os atributos, e DN com AS, respectivamente. Elas mostram a quantidade de acertos

que os classificadores alcançaram para as classes “*Benign*” e “*Malicious*”.

A Tabela 11 apresenta os resultados de acurácia dos testes em cada *Fold* da validação cruzada, a média das cinco execuções e o desvio padrão. Além disso, a Tabela 12 mostra os resultados com as métricas de *Training Time*, *Predict time* e *Predict Unit Time* para todos os cenários.

Ademais, quando os dados não estão normalizados e não há seleção de atributos, a classificação se mostra mais assertiva, com acurácia superior a 99,99%, como mostra a Figura 16. Ademais, o tempo de treinamento do modelo foi 0,14 minutos, entretanto, o tempo de teste custou alguns minutos e o tempo de teste unitário uma fração de segundo na quarta casa decimal.

Figura 16 – *KNN*.

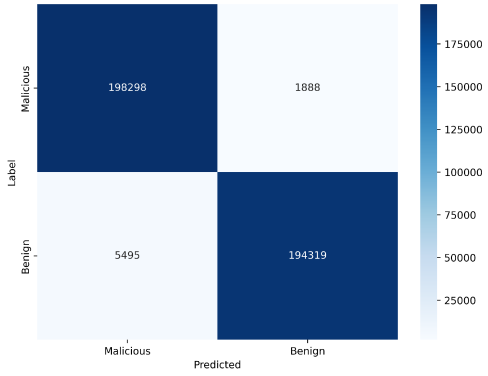


Figura 17 – *KNN*: DN.

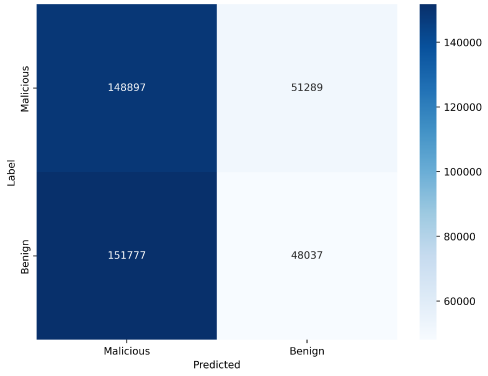


Figura 18 – *KNN*: AS.

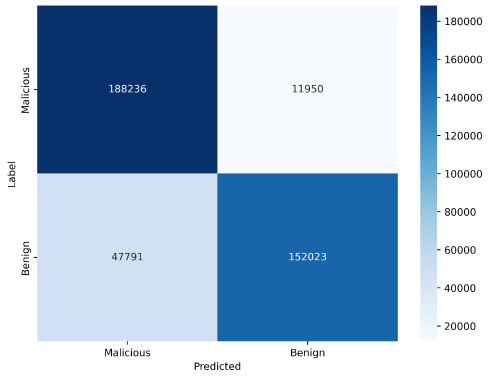
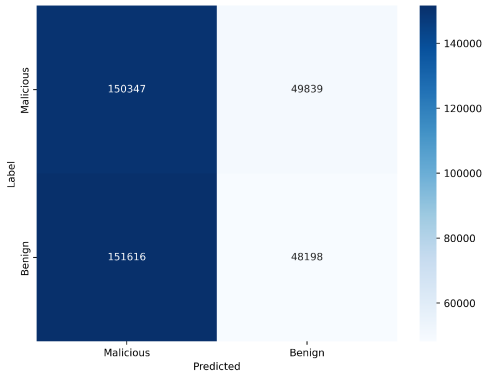


Figura 19 – *KNN*: AS e AS.



Fonte: Elaborado pelo autor.

Tabela 11 – *KNN*: Resultados K-Fold ($k = 5$) observando Acurácia.

	DN	AS	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Média	Desvio Padrão
K-Nearest Neighbors			**98,154**	**98,163**	**98,153**	**98,171**	**98,168**	**98,162**	**7,22** e^{-5}
K-Nearest Neighbors		x	85,064	85,023	85,010	85,045	85,076	85,044	0,0002
K-Nearest Neighbors	x	x	49,606	49,741	49,764	49,765	49,609	49,697	0,0007
K-Nearest Neighbors	x		49,494	49,424	49,432	49,467	49,343	49,432	0,0005

Fonte: Elaborado pela autor.

Tabela 12 – *KNN*: Tempo de Execução.

	DN	AS	Training time	Predict time	Predict Unit Time
K-Nearest Neighbors			0.14 s	372,69 s	9,31 $e^{-4}s$
K-Nearest Neighbors		x	**6,69 s**	**66,29 s**	**1,65** $e^{-4}s$
K-Nearest Neighbors	x	x	3,92 s	780,54 s	1,95 $e^{-3}s$
K-Nearest Neighbors	x		0,14 s	375,45 s	9,38 $e^{-4}s$

Fonte: Elaborado pela autor.

4.1.5 Gaussian Naive Bayes

O classificador *Gaussian Naive Bayes* foi implementado e aplicado nos quatro cenários de conjunto de dados, dados não normalizados e todos os atributos, dados não normalizados e AS, DN e todos os atributos, e DN com AS. As Figuras 20, 21, 22 e 23 apresentam as matrizes de confusão para os cenários de dados não normalizados e todos os atributos, dados não normalizados e AS, DN e todos os atributos, e DN com AS, respectivamente. Elas mostram a quantidade de acertos que os classificadores alcançaram para as classes "*Benign*" e "*Malicious*".

A Tabela 13 apresenta os resultados de acurácia dos testes em cada *Fold* da validação cruzada, a média das cinco execuções e o desvio padrão. Além disso, a Tabela 14 mostra os resultados com as métricas de *Training Time*, *Predict time* e *Predict Unit Time* para todos os cenários.

Ademais, quando os dados não estão normalizados e não há seleção de atributos, a classificação se mostra mais assertiva, entretanto, inferior aos resultados obtidos por outros classificadores nesse trabalho. Ademais, o tempo de treinamento do modelo foi de meio segundo, o tempo de teste custou alguns décimos de segundo e o tempo de teste unitário uma fração de segundo na sétima casa decimal.

Figura 20 – *Gaussian Naive Bayes.*

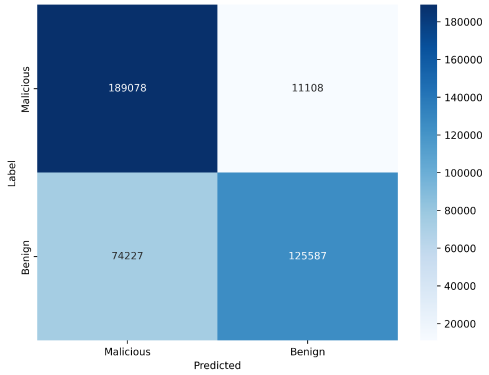


Figura 21 – *Gaussian Naive Bayes*: DN.

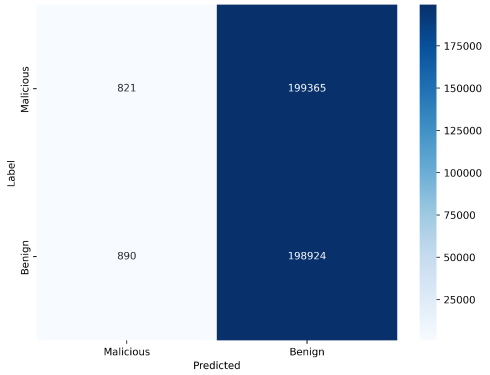


Figura 22 – *Gaussian Naive Bayes*: AS.

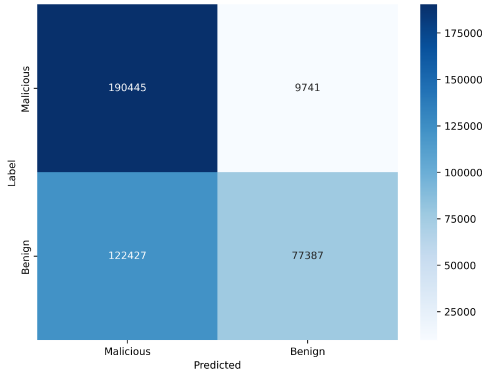
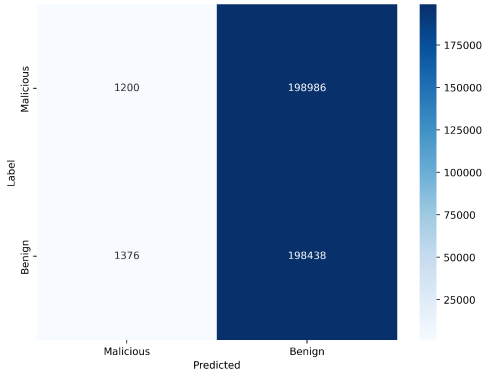


Figura 23 – *Gaussian Naive Bayes*: AS e AS.



Fonte: Elaborado pelo autor.

Tabela 13 – *Gaussian Naive Bayes*: Resultados K-Fold ($k = 5$) observando Acurácia.

	DN	AS	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Média	Desvio Padrão
Gaussian Naive Bayes			**78,666**	**78,657**	**78,614**	**78,757**	**78,797**	**78,698**	**0,0006**
Gaussian Naive Bayes		x	66,958	66,777	66,843	67,009	66,942	66,906	0,0008
Gaussian Naive Bayes	x	x	49,909	50,081	51,157	49,924	50,848	50,384	0,0051
Gaussian Naive Bayes	x		49,936	50,085	50,103	49,927	50,011	50,012	0,0007

Fonte: Elaborado pela autor.

Tabela 14 – *Gaussian Naive Bayes*: Tempo de Execução.

	DN	AS	Training time	Predict time	Predict Unit Time
Gaussian Naive Bayes			0,51 s	0,08 s	2,09 $e^{-7}s$
Gaussian Naive Bayes		x	**0,46 s**	**0,07 s**	**1,82** $e^{-7}s$
Gaussian Naive Bayes	x	x	0,41 s	0,07 s	1,88 $e^{-7}s$
Gaussian Naive Bayes	x		0,52 s	0,08 s	2,07 $e^{-7}s$

Fonte: Elaborado pela autor.

4.2 Discussão

A Tabela 15 agrupa os melhores resultados de acurácia em cada técnica implementada, e mostra que o algoritmo de *Decision Tree* se destacou pelos melhores resultados em acurácia média de validação quando aplicado ao conjunto de dados não normalizado com todos os atributos. Entretanto, ao utilizar conjuntos de dados normalizados, observou-se uma queda significativa na acurácia média de validação e desempenho inferior em comparado com os resultados obtidos em conjuntos não normalizados. Além disso, os conjuntos de dados com atributos selecionados não obtiveram ganhos consideráveis de desempenho e, na maioria dos casos, apresentaram resultados inferiores aos conjuntos sem seleção.

Quanto à detecção de *Malware*, o algoritmo *Decision Tree* se mostrou mais eficaz e assertivo, com apenas um caso de falha na identificação de *Malware* e um único falso positivo, como mostra a Figura 4.

Além do mais, em termos de tempo de execução, a Tabela 16 concatena os melhores resultados de cada técnica, e indica que a *Decision Tree* novamente se destacou, apresentando o menor tempo de classificação unitária entre todas as implementações, com o tempo de treinamento do modelo inferior a 10 segundos.

Tabela 15 – Resultados K-Fold ($k = 5$) observando Acurácia.

	DN	AS	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Média	Desvio Padrão
Decision Tree			**99,999**	**99,998**	**99,996**	**99,997**	**99,997**	**99,997**	**9.30** e^{-6}
Random Forest			99,992	99,994	99,989	99,993	99,993	99,992	1,52 e^{-5}
Extra Trees		x	99,981	99,981	99,977	99,980	99,974	99,978	2,78 e^{-5}
K-Nearest Neighbors			98,154	98,163	98,153	98,171	98,168	98,162	7,22 e^{-5}
Gaussian Naive Bayes			78,666	78,657	78,614	78,757	78,797	78,698	0,0006

Fonte: Elaborado pela autor.

Tabela 16 – Tempos de Execução.

	DN	AS	Training time	Predict time	Predict Unit Time
Decision Tree			**6,20 s**	**0,0311 s**	**7,78** $e^{-8}s$
Gaussian Naive Bayes		x	0,46 s	0,07 s	1,82 $e^{-7}s$
Random Forest			124.54 s	2.42 s	6,06 $e^{-6}s$
Extra Trees			71,74 s	3,41 s	8,54 $e^{-6}s$
K-Nearest Neighbors		x	6,69 s	66,29 s	1,65 $e^{-4}s$

Fonte: Elaborado pela autor.

5 CONSIDERAÇÕES FINAIS

Neste trabalho, foi realizada a análise do desempenho de diferentes algoritmos de aprendizado de máquina na detecção de *malware* em fluxos de tráfego de redes no contexto IoT, utilizando a base de dados IoT-23. No desenvolvimento, foram implementados e avaliados cinco algoritmos: *Decision Tree*, *Random Forest*, *Extra Trees*, *K-Nearest Neighbors* (KNN), e *Gaussian Naive Bayes*. Esses modelos foram aplicados para classificar o tráfego de rede como *Malicious* ou *Benign*.

Para o desenvolvimento deste estudo, foram utilizadas diversas etapas metodológicas, incluindo a limpeza e transformação dos dados, normalização, seleção de atributos e validação cruzada *k*-fold. Os resultados obtidos foram analisados para verificar a acurácia, o tempo de execução dos modelos e os erros de classificação, proporcionando uma comparação abrangente entre as técnicas utilizadas e os diversos conjuntos de dados.

A análise dos resultados revelou que o algoritmo *Decision Tree* apresentou o melhor desempenho geral, com alta acurácia, erro mínimo, além de um tempo de execução razoável. Essa abordagem alcançou uma acurácia média de validação superior às demais técnicas, destacando-se como uma abordagem robusta para a detecção de *malware* em redes IoT. Em contrapartida, o algoritmo *Gaussian Naive Bayes* apresentou limitações em termos de precisão, sugerindo que sua aplicação pode ser menos eficaz no contexto de dados de tráfego de rede IoT.

Além disso, foi verificado que a normalização dos dados e a seleção adequada de atributos impactam o desempenho dos classificadores. As técnicas de normalização e a aplicação da PCA contribuíram significativamente para a redução da dimensionalidade dos dados, mas neste estudo, com a base de dados IoT-23, e as transformações aplicadas, não contribuíram para ganho de eficácia dos algoritmos de aprendizado de máquina.

Por fim, este estudo destaca a relevância da utilização de técnicas de aprendizado de máquina para enfrentar os desafios de segurança em ambientes IoT, e além disso, contribui para o desenvolvimento de métodos mais eficazes de proteção de dispositivos IoT e a segurança das redes em que estão inseridos.

5.1 Trabalhos Futuros

Para futuras pesquisas, sugere-se explorar algoritmos de aprendizado profundo, como *Convolutional Neural Networks* (CNNs) e *Recurrent Neural Networks* (RNNs), que

podem oferecer melhorias em precisão e robustez na detecção de *Malware*. Aplicar técnicas avançadas de aumento de dados, como o uso de *Generative Adversarial Networks* (GANs) para gerar amostras sintéticas e técnicas de oversampling para balancear conjuntos de dados desbalanceados, pode melhorar a generalização dos modelos. Desenvolver modelos de *Ensemble* mais complexos, combinando múltiplos algoritmos de aprendizado de máquina e *deep learning*, pode criar sistemas de detecção mais robustos.

Ademais, expandir a análise para diversos contextos de IoT, como saúde e cidades inteligentes, pode abordar desafios únicos de segurança. Integrar os modelos com sistemas de detecção existentes pode melhorar a eficácia geral. Aliás, coletar e disponibilizar novas bases de dados de tráfego de rede IoT, incluindo cenários de ataque emergentes, é fundamental para o desenvolvimento contínuo.

REFERÊNCIAS

ABOAOJA, F. A.; ZAINAL, A.; GHALEB, F. A.; AL-RIMY, B. A. S.; EISA, T. A. E.; ELNOUR, A. A. H. **Malware Detection Issues, Challenges, and Future Directions: A Survey**. [S.l.]: MDPI, 2022.

ALJABRI, M.; ALJAMEEL, S. S.; MOHAMMAD, R. M. A.; ALMOTIRI, S. H.; MIRZA, S.; ANIS, F. M.; ABOULNOUR, M.; ALOMARI, D. M.; ALHAMED, D. H.; ALTAMIMI, H. S. **Intelligent techniques for detecting network attacks: Review and research directions**. [S.l.]: MDPI, 2021.

ALZUBAIDI, L.; ZHANG, J.; HUMAIDI, A. J.; AL-DUJAILI, A.; DUAN, Y.; AL-SHAMMA, O.; SANTAMARíA, J.; FADHEL, M. A.; AL-AMIDIE, M.; FARHAN, L. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. **Journal of Big Data**, v. 8, 2021. ISSN 21961115.

BERRAR, D. Cross-validation. In: ____. [S.l.: s.n.], 2018. ISBN 9780128096338.

BOVENZI, G.; ACETO, G.; CIUONZO, D.; MONTIERI, A.; PERSICO, V.; PESCAPé, A. Network anomaly detection methods in iot environments via deep learning: A fair comparison of performance and robustness. **Computers and Security**, Elsevier Ltd, v. 128, 5 2023. ISSN 01674048.

DOUIBA, M.; BENKIRANE, S.; GUEZZAZ, A.; AZROUR, M. An improved anomaly detection model for iot security using decision tree and gradient boosting. **Journal of Supercomputing**, Springer, v. 79, p. 3392–3411, 2 2023. ISSN 15730484.

FERENCZ, K.; DOMOKOS, J.; KOVACS, L. Review of industry 4.0 security challenges. In: . [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2021. p. 245–248. ISBN 9781728195445.

FILHO, O. G. **Inteligência artificial e aprendizagem de máquina: aspectos teóricos e aplicações**. Editora Edgard Blücher Ltda., 2023. ISBN 9786555066203. Disponível em: <www.blucher.com.br>.

GARCIA, S.; PARMISANO, A.; ERQUIAGA, M. J. **IoT-23: A labeled dataset with malicious and benign IoT network traffic**. Zenodo, 2021. Disponível em: <https://doi.org/10.5281/zenodo.4743746>.

GEURTS, P.; ERNST, D.; WEHENKEL, L. Extremely randomized trees. **Machine Learning**, v. 63, p. 3–42, 4 2006. ISSN 08856125.

JANIESCH, C.; ZSCHECH, P.; HEINRICH, K. Machine learning and deep learning. **Electronic Markets**, v. 31, 2021. ISSN 14228890.

KAUR, B.; DADKHAH, S.; SHOELEH, F.; NETO, E. C. P.; XIONG, P.; IQBAL, S.; LAMONTAGNE, P.; RAY, S.; GHORBANI, A. A. **Internet of Things (IoT) security dataset evolution: Challenges and future directions**. 2023.

LORENA, A.; FACELI, K.; ALMEIDA, T.; CARVALHO, A. de; GAMA, J. **Inteligência Artificial: uma abordagem de Aprendizado de Máquina (2a edição)**. [S.l.: s.n.], 2021. ISBN 9788521637493.

LUTHRA, S.; MANGLA, S. K. Evaluating challenges to industry 4.0 initiatives for supply chain sustainability in emerging economies. **Process Safety and Environmental Protection**, v. 117, 2018. ISSN 09575820.

MARTíNEZ, A. L.; PéREZ, M. G.; RUIZ-MARTíNEZ, A. A comprehensive review of the state-of-the-art on security and privacy issues in healthcare. **ACM Computing Surveys**, Association for Computing Machinery, v. 55, 3 2023. ISSN 15577341.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. In: **Sistemas Inteligentes Fundamentos e Aplicações**. 1. ed. Barueri-SP: Manole Ltda, 2003. p. 89–114. ISBN 85-204-168.

MONTAVON, G.; SAMEK, W.; MüLLER, K. R. **Methods for interpreting and understanding deep neural networks**. 2018.

NASER, M. Z.; ALAVI, A. H. Error metrics and performance fitness indicators for artificial intelligence and machine learning in engineering and sciences. **Architecture, Structures and Construction**, Springer Science and Business Media LLC, v. 3, n. 4, p. 499–517, nov. 2021. ISSN 2730-9894. Disponível em: <http://dx.doi.org/10.1007/s44150-021-00015-8>.

NOBAKHT, M.; JAVIDAN, R.; POUREBRAHIMI, A. Demd-iot: a deep ensemble model for iot malware detection using cnns and network traffic. **Evolving Systems**, Institute for Ionics, v. 14, p. 461–477, 6 2023. ISSN 18686486.

PATTERN Recognition and Machine Learning. **Journal of Electronic Imaging**, v. 16, 2007. ISSN 1017-9909.

QUINLAN, J. R. Induction of decision trees. **Machine Learning**, v. 1, 1986. ISSN 15730565.

RASCHKA, S. **STAT 479: Machine Learning Lecture Notes**. 2018. Disponível em: <http://stat.wisc.edu/âĹijsraschka/teaching/stat479-fs2018/>.

RAY, P. P. **A survey on Internet‘ of Things architectures**. [S.l.]: King Saud bin Abdulaziz University, 2018. 291-319 p.

REY, V.; SáNCHEZ, P. M. S.; CELDRáN, A. H.; BOVET, G. Federated learning for malware detection in iot devices. **Computer Networks**, Elsevier B.V., v. 204, 2 2022. ISSN 13891286.

SADHU, P. K.; YANAMBAKA, V. P.; ABDELGAWAD, A. **Internet of Things: Security and Solutions Survey**. 2022.

SAMMUT, C.; WEBB, G. I. **Encyclopedia of Machine Learning**. Springer Science+Business Media LLC, 2010. 209-209 p. ISBN 978-0-387-34558-1. Disponível em: <https://doi.org/10.1007/978-0-387-30164-8_157>.

SCHONLAU, M.; ZOU, R. Y. The random forest algorithm for statistical learning. **Stata Journal**, SAGE Publications Inc., v. 20, p. 3–29, 3 2020. ISSN 15368734.

SCIKIT-LEARN. **3.1. Cross-validation: evaluating estimator performance**. 2007. 3.1. Cross-validation: evaluating estimator performance. Disponível em: <https://scikit-learn.org/stable/modules/cross_validation.html>. Acesso em: 04 jun. 2024.

VILLAMIL, S.; HERNáNDEZ, C.; TARAZONA, G. An overview of internet of things. **Telkomnika (Telecommunication Computing Electronics and Control)**, Universitas Ahmad Dahlan, v. 18, p. 2320–2327, 10 2020. ISSN 23029293.

WEINBERGER, K. **Bayes Classifier and Naive Bayes**. 2018. CORNELL CS4780 "Machine Learning for Intelligent Systems". Disponível em: <http://www.inpe.br/webelat/ABNT_NBR5419_Ng/>. Acesso em: 04 jun. 2024.

YEGNANARAYANA, B. Artificial neural networks for pattern recognition. **Sadhana**, v. 19, 1994. ISSN 02562499.