Universidade Estadual de Londrina

ISABELA HARA BANDO

ALGORITMOS ONE-CLASS PARA FLUXOS CONTÍNUOS DE DADOS E A DETECÇÃO DE ATAQUES EM INTERNET DAS COISAS

LONDRINA

2024

ISABELA HARA BANDO

ALGORITMOS ONE-CLASS PARA FLUXOS CONTÍNUOS DE DADOS E A DETECÇÃO DE ATAQUES EM INTERNET DAS COISAS

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Bruno Bogaz Zarpelão

LONDRINA
2024

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

B214a Bando, Isabela Hara.
Algoritmos one-class para fluxos contínuos de dados e a detecção de ataques em internet das coisas / Isabela Hara Bando. - Londrina, 2024.
60 f. : il.

Orientador: Bruno Bogaz Zarpelão.
Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Graduação em Ciência da Computação, 2024.
Inclui bibliografia.

1. Fluxos contínuos de dados - TCC. 2. Sistema de Detecção de Intrusão - TCC. 3. Aprendizado de Máquina - TCC. 4. Inteligência Computacional - TCC. I. Zarpelão, Bruno Bogaz. II. Universidade Estadual de Londrina. Centro de Ciências Exatas. Graduação em Ciência da Computação. III. Título.

CDU 519

ISABELA HARA BANDO

ALGORITMOS ONE-CLASS PARA FLUXOS CONTÍNUOS DE DADOS E A DETECÇÃO DE ATAQUES EM INTERNET DAS COISAS

Trabalho de Conclusão de Curso apresentado ao curso de Bacharelado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Dr. Bruno Bogaz Zarpelão
Universidade Estadual de Londrina

Prof. Dr. Gilberto Fernandes Junior
Universidade Estadual de Londrina – UEL

Prof. Blenda Oliveira Mazetto
Universidade Estadual de Londrina – UEL

Londrina, 10 de maio de 2024.

Este trabalho é dedicado a todos aqueles que têm a ousadia de perseguir os seus sonhos e inspiram outros a fazerem o mesmo.

AGRADECIMENTOS

Os agradecimentos iniciais são dedicados a Deus, meus pais, Sérgio e Edna, e a minha irmã, Juliana. Vocês são a minha base, obrigada por me acompanharem nessa jornada, por todo apoio e incentivo em todos os meus passos e decisões. Agradeço também a toda a minha família e amigos mais próximos, que, independentemente de estarem perto ou longe de mim, sempre torceram pela realização dos meus sonhos. Em especial, a minha prima, Letícia Hara, que viveu toda essa fase universitária junto comigo e está sempre presente nos momentos mais importantes da minha vida.

Em seguida, gostaria de expressar o quanto sou grata pela oportunidade de ter conhecido tantas pessoas incríveis ao longo desse curso. Aos meus amigos de computação, agradeço por todos os momentos, conversas e ajuda que me forneceram durante esses últimos anos. Em especial, a Jennifer do Prado da Silva e ao Pedro Eduardo Garbossa de Almeida, vocês são os melhores amigos que eu poderia ter encontrado. Obrigada por estarem ao meu lado nos momentos mais difíceis e nos mais felizes também, não consigo imaginar como teria sido a faculdade sem vocês.

Além disso, agradeço também ao grupo Ryukyu Koku Matsuri Daiko - Filial Londrina e a todos os amigos que fiz através dele. Durante os 10 anos em que venho sendo parte do grupo, vocês não só me ensinaram muito, mas também sempre incentivaram o meu crescimento, tanto pessoal quanto acadêmico/profissional.

Por fim, mas não menos importante, gostaria de agradecer à Universidade Estadual de Londrina e aos professores do curso de Ciência da Computação, por todo conhecimento e ensinamentos que nos proporcionaram, tanto dentro quanto fora das salas de aula. Um destaque ao meu professor e orientador Bruno Bogaz Zarpelão. Obrigada por acreditar no meu potencial, por todo ensinamento, pela paciência e auxílio durante o desenvolvimento deste trabalho.

"Nós só podemos ver um pouco do futuro, mas o suficiente para perceber que há muito a fazer."
(Alan Turing)

BANDO, I. H.. **Algoritmos one-class para fluxos contínuos de dados e a detecção de ataques em Internet das Coisas**. 2024. 60f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2024.

RESUMO

O desenvolvimento da tecnologia nos últimos anos aumentou significativamente o uso de dispositivos inteligentes interconectados, isto é, a Internet das Coisas (*Internet of Things* - IoT). Devido ao grande fluxo de informações compartilhadas e armazenadas, garantir a segurança das redes IoT é essencial, e também um enorme desafio, tendo em vista os diversos tipos de ciberataques que ameaçam essas redes atualmente. Grande parte das soluções estudadas para detecção de ataques cibernéticos utilizam algoritmos de aprendizado em lote, cujo treinamento é feito a partir de uma base de dados estática, levando à perda de eficácia com o surgimento de novos comportamentos na rede. Os dados monitorados passam, constantemente, por muitas mudanças naturais e por isso, é necessário que os algoritmos sejam capazes de se adaptar com uma maior facilidade. Considerando esse contexto, este trabalho estuda a viabilidade do emprego do algoritmo *One-Class Support Vector Machines* (OCSVM) na detecção de ataques em redes IoT, especialmente em cenários de Fluxos Contínuos de Dados (FCDs). A análise incluiu uma comparação de desempenho entre o OCSVM e outros algoritmos convencionais, tanto em abordagens em lote quanto incrementais. Os conjuntos de dados públicos TON_IoT e CIC-IoT 2023 foram utilizados para realizar os testes e avaliar a eficácia dos algoritmos em questão.

Palavras-chave: Inteligência Computacional. Fluxos contínuos de dados. Sistema de Detecção de Intrusão. Aprendizado de Máquina.

BANDO, I. H.. **One-class algorithms for continuous data streams and attack detection in the Internet of Things**. 2024. 60p. Final Project (Bachelor of Science in Computer Science) – State University of Londrina, Londrina, 2024.

ABSTRACT

The development of technology in recent years has significantly increased the use of interconnected smart devices, that is, the Internet of Things (IoT). Due to the large flow of shared and stored information, ensuring the security of IoT networks is essential, and also a huge challenge, given the several forms of cyberattacks that threaten these networks today. Most of the solutions studied for detecting cyber attacks use batch learning algorithms, which are trained from a static database, which leads to loss of effectiveness as new behaviors emerge in the network. The monitored data is constantly undergoing many natural changes and therefore it is necessary that the algorithms are able to adapt more easily. In this context, this paper studies the feasibility of using the One-Class Support Vector Machines (OCSVM) algorithm to detect attacks on IoT networks, especially in continuous data stream scenarios. The analysis included a performance comparison between the OCSVM and other conventional algorithms, both in batch and incremental approaches. The TON_IoT and CIC IoT 2023 public datasets were used to conduct the tests and evaluate the effectiveness of these algorithms.

Keywords: Computational Intelligence. Continuous Data Streams. Intrusion Detection System. Machine Learning.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

FCDs	Fluxos Contínuos de Dados
IoT	Internet of Things
IIoT	Industrial Internet of Things
IDS	Intrusion Detection System
VMs	Virtual Machines
SVM	Support Vector Machine
OCSVM	One-class Support Vector Machine
RF	Random Forest
HTC	Hoeffding Tree Classifier
PCAP	Packet Capture
DoS	Denial of Service
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
PCA	Principal Component Analisys

SUMÁRIO

1 INTRODUÇÃO

Com o avanço tecnológico e o aumento constante da conectividade, a Internet das Coisas (*Internet of Things* - IoT) emergiu como um campo de grande potencial de desenvolvimento. Além disso, a IoT está se tornando cada vez mais integrada à vida cotidiana das pessoas, estando presente em uma ampla variedade de dispositivos inteligentes, desde eletrodomésticos e sistemas de segurança até veículos autônomos e equipamentos médicos. Portanto, garantir a segurança das redes IoT e dos sistemas é um desafio relevante, que tem sido objeto de estudos ao longo dos anos [4, 5, 6, 7].

Devido à diversidade de ameaças existentes, à falta de conscientização dos usuários e à ausência de atualizações regulares de software, torna-se importante o desenvolvimento de uma camada de proteção extra, ou seja, algoritmos capazes de se adaptar às mudanças e depender cada vez menos da intervenção humana.

Embora os algoritmos de aprendizado em lote (*batch*) tenham sido amplamente utilizados na detecção de ataques em redes [8, 9, 10], eles possuem uma limitação significativa: são treinados com um conjunto de dados estático e têm dificuldade em lidar com mudanças na rede, como mudanças de conceito (*concept drift*), exigindo a atualização ou retreinamento do modelo [11, 12]. Como as redes de computadores estão em constante evolução e os padrões de tráfego podem variar com o tempo, modelos treinados com dados do passado podem perder sua eficácia rapidamente, resultando em taxas de detecção de ataques mais baixas.

Dada essa limitação, outra abordagem que pode ser explorada é o uso de algoritmos de aprendizado em fluxo contínuo de dados. Esses algoritmos são projetados para lidar com situações que exigem aprendizado incremental, permitindo que o modelo seja ajustado à medida que novos dados são recebidos em um fluxo contínuo, sem a necessidade de processar novamente todo o conjunto de dados. Estudos recentes já começaram a investigar essa abordagem [13, 14, 15], entretanto envolvem apenas uma parte dos algoritmos potenciais que podem ser utilizados, indicando que ainda há muito a ser explorado nesse campo.

Neste projeto, será estudado e avaliado o potencial de um algoritmo *One-Class* para fluxos contínuos de dados na detecção de ataques em redes IoT, comparando-o com outros algoritmos de aprendizado. Vale destacar que uma das principais vantagens do *One-Class*, em relação ao aprendizado supervisionado tradicional, é que seu treinamento não exige a presença das duas classes de dados (benignos e maliciosos), sendo necessário acesso apenas a exemplos da classe majoritária ou benigna.

O primeiro passo será realizar um levantamento de conjuntos de dados públicos que contenham tráfego de rede IoT e uma diversidade de ataques, para compreender melhor o potencial de detecção frente a diferentes ameaças. Em seguida, os pacotes coletados da rede serão transformados em fluxos IP, que formarão os cenários de ataque. Por fim, cada um desses cenários será testado nos algoritmos *Random Forest, Hoeffding Tree Classifier, One-Class SVM batch* e por fim, o *One-Class SVM stream*. Em cada execução, métricas de desemprenho serão coletadas, para avaliar o desempenho preditivo de cada um dos modelos e comparar as performances do *One-Class* com os demais paradigmas.

No Capítulo 2, são apresentadas a fundamentação teórica e o estado da arte, abrangendo aspectos fundamentais da Internet das Coisas (IoT), destacando suas características e desafios associados à segurança. Além disso, são abordados tópicos como sistemas de detecção de intrusão e mineração de fluxos contínuos de dados, incluindo a abordagem baseada em *One-Class SVM*. No Capítulo 3, é detalhada a metodologia adotada neste estudo, bem como a implementação de cada algoritmo selecionado. O Capítulo 4 é dedicado à apresentação e discussão dos resultados obtidos por meio da aplicação dos métodos e técnicas descritos anteriormente. Por fim, no Capítulo 5, são apresentadas as conclusões deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICO-METODOLÓGICA E ESTADO DA ARTE

2.1 Internet das Coisas

O termo “Internet das Coisas” foi popularizado pelo britânico Kevin Ashton, em 1999 [16], e desde então seu uso vem crescendo cada vez mais. Atualmente, a IoT é um paradigma tecnológico que se refere à interconexão de dispositivos do nosso cotidiano por meio da Internet. Esses dispositivos estão equipados com sensores, atuadores e tecnologia de comunicação que permitem a coleta, troca e análise de dados, possibilitando assim, a automação de tarefas e a tomada de decisões em tempo real [17].

Para entender a melhor a respeito dessa rede interconectada, é imperativo compreender sua arquitetura, que é fundamentada em camadas ou fases, que podem variar em nome e quantidade dependendo da abordagem ou modelo específico, mas uma arquitetura típica pode incluir três camadas principais, conforme a apresentada na Figura 1.

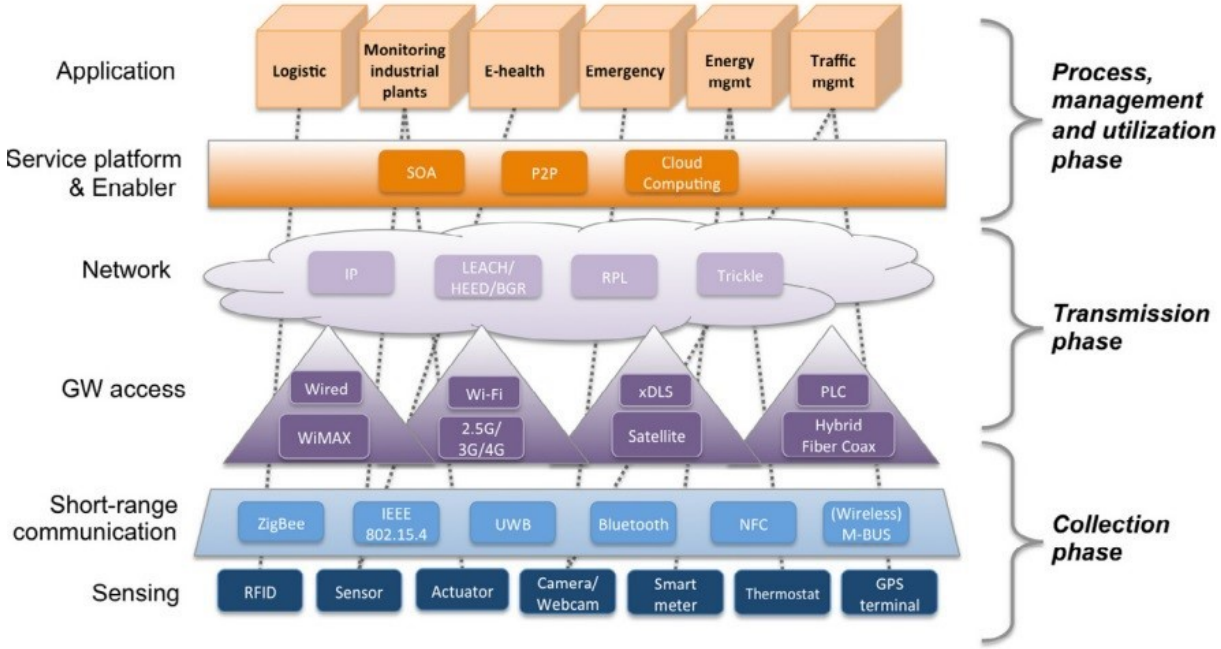

Figura 1 – Representação horizontal para aplicações de IoT. Extraída de [1]

As camadas são divididas da seguinte forma:

- Camada de Coleta (ou Camada de Percepção): composta pelos dispositivos sensores e atuadores distribuídos fisicamente no ambiente. Os sensores são responsáveis por captar dados essenciais, como luz, temperatura, umidade e outros, em tempo real,

enquanto os atuadores executam ações em resposta a essas informações, gerando uma perspectiva geral do ambiente. Essa camada serve como ponto de entrada para os dados, representando o elo entre o mundo físico e o ambiente digital.

- Camada de Transmissão (ou Camada de Rede): responsável pela comunicação e transmissão eficiente dos dados coletados pelos sensores para a camada superior. Estão inclusas nessa camada algumas tecnologias heterogêneas, que implementam métodos de endereçamento, roteamento e permitem o acesso à rede e a entrega dos dados a aplicações e servidores externos.

- Camada de Aplicação (ou Camada de gerenciamento e utilização): onde ocorre o processamento, análise e aplicação dos dados em soluções específicas, adaptadas conforme as diferentes necessidades. Esta camada também representa a interface com os usuários e as operações que se beneficiam com as informações provenientes dos dispositivos IoT.

2.1.1 Segurança em Redes IoT

Atualmente, o número de dispositivos que utilizam as redes IoT tem crescido cada vez mais e assegurar a proteção dessas redes tem se tornado mais difícil, considerando que existem muitos fatores que podem contribuir com a violação de suas seguranças.

Quando comparada às redes tradicionais, a natureza heterogênea e distribuída dos dispositivos IoT, muitas vezes com recursos limitados de hardware e software, intensifica as vulnerabilidades e dificulta a implementação de medidas de segurança robustas [17]. Além disso, a diversidade de padrões de comunicação e protocolos em ambientes IoT contribui para a complexidade, tornando a interoperabilidade entre dispositivos e a aplicação de políticas de segurança coesas uma tarefa desafiadora [18, 19]. A vasta quantidade de dados gerados por dispositivos IoT também amplia os riscos de privacidade e exposição a ameaças cibernéticas, demandando estratégias eficientes de gerenciamento e proteção [20].

De acordo com a *Open Worldwide Application Security Project* (OWASP) [21], os dez principais problemas de segurança, em IoT, que podemos citar são:

1. Senhas fracas, adivinháveis ou codificadas: senhas padrões e comuns, disponíveis publicamente e imutáveis, que podem facilmente ser descobertas por meio do uso da força bruta, ou seja, algoritmos que testam todas as senhas até encontrar uma correspondência;

2. Serviços de rede inseguros: serviços de rede que operam no dispositivo e são desnecessários ou vulneráveis, especialmente aqueles que estão conectados à Internet,

podendo comprometer a confidencialidade, integridade/autenticidade ou disponibilidade das informações;

3. Interfaces inseguras do ecossistema: interfaces Web, API de *backend*, nuvem ou interfaces móveis inseguras no ecossistema externo ao dispositivo podem resultar na vulnerabilidade do dispositivo, ou de seus componentes associados. Normalmente, os problemas estão associados a ausência de autenticação/autorização, criptografia ausente ou fraca, e a falta de filtragem de entrada e saída;

4. Falta de mecanismo de atualização seguro: abrange a ausência da validação do firmware no próprio dispositivo, da entrega segura (sem criptografia durante a transmissão), a inexistência de mecanismos anti-reversão e de notificações sobre alterações de segurança decorrentes de atualizações;

5. Uso de componentes inseguros ou desatualizados: utilização de componentes ou bibliotecas de software desatualizados, ou vulneráveis representa um risco significativo para a segurança do dispositivo. Isso engloba a personalização insegura de plataformas de sistema operacional, assim como a adoção de software de terceiros ou componentes de hardware provenientes de uma cadeia de suprimentos comprometida;

6. Proteção insuficiente da privacidade: a manipulação da informação pessoal do usuário, armazenada no dispositivo ou no ecossistema, ocorre de maneira insegura, inadequada ou sem consentimento;

7. Transferência e armazenamento inseguro de dados: ausência de medidas de criptografia ou de controle de acesso para dados confidenciais em qualquer ponto do sistema, seja em repouso, em trânsito ou durante o processamento;

8. Falta de gerenciamento de dispositivos: a ausência de suporte de segurança em dispositivos implementados na produção, abrangendo áreas como gerenciamento de ativos, atualizações, desativação segura, monitoramento de sistemas e capacidades de resposta;

9. Configurações padrão inseguras: dispositivos ou sistemas são entregues com configurações padrão inseguras, ou sem a capacidade de serem aprimorados em termos de segurança, uma vez que a modificação das configurações pelos operadores é restringida;

10. Falta de proteção física: ausência de salvaguardas físicas possibilita que potenciais invasores obtenham informações confidenciais, o que pode facilitar futuros ataques remotos ou a tomada de controle local do dispositivo.

2.2 Sistemas de Detecção de Intrusão

Os sistemas de detecção de intrusão (*Intrusion Detection Systems* - IDS) são ferramentas essenciais na segurança da informação, projetadas para identificar atividades suspeitas ou maliciosas em redes, ou sistemas. Eles podem ser predominantemente um software ou um hardware, ou uma combinação de ambos e desempenham um grande papel na detecção e prevenção de intrusões, auxiliando na proteção, manutenção da integridade, confidencialidade e disponibilidade dos sistemas [22].

A arquitetura geral de um IDS inclui os seguintes componentes [22, 23]:

- Sensores: responsáveis pela coleta de dados sobre o sistema monitorado e que, posteriormente, serão utilizados para a detecção de eventos de segurança. Como exemplo tem-se os dados sobre o tráfego de rede.
- Mecanismo de Análise: processa os dados coletados pelos sensores e identifica padrões suspeitos ou maliciosos.
- Base de Conhecimento: contém informações sobre padrões de ataques conhecidos e perfis de comportamento típicos. Essa base é usada para comparar os padrões identificados durante a análise.
- Resposta a Incidentes: após a detecção de um ataque, o sistema pode desencadear uma resposta, como bloquear o acesso do usuário ou realizar o envio de alertas.

Um IDS pode ser classificado em *Network-based Intrusion Detection Systems* (NIDS) ou em *Host-based Intrusion Detection Systems* (HIDS). No primeiro, a implantação ocorre no perímetro da rede e os pacotes que a atravessam são examinados em tempo real, em busca de atividades maliciosas. Já no segundo, a instalação é feita diretamente nos hospedeiros (computadores ou servidores) e monitoram as atividades e eventos que ocorrem no próprio sistema operacional, incluindo chamadas de sistema, processos em execução, entre outros [23].

Para determinar se um ataque está de fato ocorrendo, os padrões de tráfego, atividade ou código podem ser comparados com uma base de conhecimento que contém assinaturas conhecidas de ataques previamente identificados. Esse método é conhecido como baseado em assinatura (*signature-based*) e se mostra altamente eficaz para detectar ataques bem conhecidos, visto que ao encontrar uma correspondência com a base, o evento é considerado uma intrusão [24, 23].

Em contrapartida, outra possível abordagem é o método baseado em anomalias (*anomaly-based*), mais eficiente na identificação de ataques menos conhecidos, uma vez que envolve a criação de um perfil do comportamento normal do sistema ou usuário.

Qualquer desvio significativo desse padrão é considerado uma anomalia e pode indicar uma possível atividade maliciosa [24, 23].

2.3 Mineração de Fluxos Contínuos de Dados

A mineração de dados é um campo da ciência de dados que lida com a análise e extração de padrões relevantes a partir de um grande conjunto de dados. Este campo multidisciplinar integra técnicas de estatística, aprendizado de máquina e inteligência artificial para explorar e revelar informações ocultas nos dados [25].

O processo de mineração de dados compreende diversas etapas, começando com a compreensão do domínio e a seleção dos dados relevantes, passando pelo pré-processamento para tratar ruídos e valores ausentes, até a aplicação de algoritmos de mineração para identificar padrões e estruturas nos dados [25, 26].

Normalmente são utilizados algoritmos de aprendizado em lote, ou *Batch Learning Model* (BLM), que se refere a uma abordagem de treinamento no qual os dados são estáticos, coletados ao longo do tempo e periodicamente utilizados em lotes para treinar o modelo. O processo de treinamento em lote não permite uma aprendizagem incremental, pois o sistema é treinado offline e, portanto, em caso de novos dados, é necessário retreiná-lo. Essa técnica demanda uma quantidade significativa de tempo e recursos, incluindo CPU, RAM e espaço em disco, devido ao volume acumulado de dados, sendo menos adequado para sistemas que precisam responder a dados sujeitos a mudanças rápidas, como em detecção de intrusões [27].

A mineração de dados, tradicionalmente centrada em conjuntos de dados estáticos, trabalha com dados em diferentes formatos, como texto, imagem, som, e mais recentemente, fluxos contínuos de dados em tempo real. Sendo assim, enfrenta novos desafios quando aplicada a dados não estáticos, ou seja, os fluxos contínuos, exigindo adaptações e inovações nas técnicas existentes.

Os fluxos contínuos de dados representam uma classe especial de dados caracterizada por sua natureza dinâmica e características únicas, portanto os algoritmos devem ser capazes de lidar com dados que estão em constante evolução, chegando em abundantemente, em alta velocidade e de forma contínua [28, 29, 30].

Diferentemente de abordagens tradicionais, a mineração de fluxos contínuos de dados opera adaptando-se à dinâmica rápida e contínua dos dados. Esses fluxos, provenientes de diversas fontes heterogêneas, desafiam as técnicas convencionais, requerendo algoritmos sofisticados para lidar com a diversidade de tipos de dados e também com a análise, processamento e extração de informações úteis em tempo real, permitindo a compreensão de padrões subjacentes e a tomada de decisões rápidas e eficazes [27].

Nesse cenário, os modelos e algoritmos são projetados para apresentar algumas características mais específicas como [31]:

- Aprendizado Incremental: cada nova amostra é processada incrementalmente e, em seguida, é realizado o aprendizado e a atualização de suas estatísticas e parâmetros;
- Descarte ou Retenção Limitada: considerando o alto e potencialmente infinito volume de dados, após o processamento de uma amostra, dependendo da aplicação, a amostra pode ser descartada para economizar recursos ou pode ser armazenada em um histórico limitado para análises futuras ou para a atualização de modelos;
- Adaptabilidade e Mudança de Conceito: deve ser capaz de se adaptar a mudanças nas características dos dados ao longo do tempo, conhecidas como mudanças de conceito.

Considerando essas características, os algoritmos normalmente utilizados trabalham com três principais tipos de aprendizado de máquina: o supervisionado, o não-supervisionado e o semi-supervisionado.

No aprendizado supervisionado, os modelos são treinados utilizando um conjunto de dados rotulados, onde as entradas e suas correspondentes saídas desejadas são fornecidas. Essa técnica é frequentemente empregada em tarefas de classificação, na qual exemplos de todas as classes que fazem parte do problema em questão devem ser apresentadas ao algoritmo durante a fase de treinamento, permitindo assim categorizar novos dados em classes predefinida. Quando aplicada à regressão, possibilita prever respostas contínuas e valores numéricos [32, 33].

Por outro lado, o aprendizado não supervisionado é uma técnica no qual o modelo é treinado em um conjunto de dados desprovido de rótulos associados [32]. O modelo identifica padrões e estruturas nos dados por conta própria, sem orientação externa, buscando agrupar instâncias semelhantes ou reduzir a dimensionalidade dos dados [33]. Sendo assim, apresenta grande valor quando utilizado em abordagens de agrupamento (ou *clustering*), no qual os dados são agrupados em conjuntos distintos, e os membros de cada grupo compartilham características comuns.

Por fim, no aprendizado semi-supervisionado, os modelos são treinados em um conjunto de dados que combina exemplos rotulados e não rotulados. Esta abordagem visa aproveitar ao máximo a informação disponível, especialmente quando a obtenção de rótulos para todos os dados é custosa ou impraticável [34]. Uma abordagem que se destaca é o algoritmo *One-Class*, projetado para lidar com conjuntos de dados em que apenas uma classe, geralmente a classe normal, está disponível durante o treinamento. É capaz de modelar a distribuição dos dados normais e identificar anomalias como instâncias que se desviam significativamente desse padrão.

2.3.1 *One-Class Support Vector Machine*

Uma abordagem muito comum para a detecção de anomalias é o algoritmo *One-class Support Vector Machine* (OCSVM), proposto pela primeira vez em 1999 por Scholkopf [35]. O OCSVM é uma extensão do algoritmo SVM, de aprendizado supervisionado, proposto em 1995 por Vapnik [36], no qual é realizado o uso de funções discriminantes que serão responsáveis por separar os dados em duas classes distintas considerando o conjunto de dados de treinamento fornecido [37].

No entanto, o SVM apresenta algumas limitações quando aplicado a situações que representam o mundo real. O algoritmo é notavelmente eficaz em cenários de classificação binária convencional, mas sua aplicação em conjuntos de dados não convencionais, onde as classes são desproporcionalmente representadas, pode resultar em modelos direcionados à classe majoritária [38]. Além disso, por ser um algoritmo de classificação binária, o mesmo assume que durante a fase de treinamento estarão disponíveis exemplos de ambas as classes, quando na realidade, isso nem sempre irá ocorrer.

Já o OCSVM utiliza o hiperplano, uma superfície de decisão no espaço de características, cuja função é encontrar uma fronteira de decisão que envolva a maior parte dos dados normais [39]. Sendo assim, durante o treinamento, é utilizada apenas a classe com o comportamento esperado, ou seja, o algoritmo aprende e define situações normais, e tudo aquilo que não estiver na área delimitada pela fronteira, é considerado um comportamento anômalo.

O emprego do OCSVM apresenta flexibilidade ao se adaptar a diferentes modos de processamento de dados: em lote (*batch*) e em fluxo (*stream*). Estes dois cenários refletem distintas abordagens no treinamento e na aplicação do modelo e a escolha da melhor opção depende da natureza temporal dos dados disponíveis e dos requisitos de atualização do modelo, respondendo assim, às necessidades específicas de ambientes estáticos e dinâmicos.

O processamento em lote é empregado quando o conjunto de dados está completamente disponível no início do treinamento e avaliação do modelo. Nesse contexto, todo esse conjunto de dados é utilizado para treinar o modelo, permitindo que o OCSVM identifique padrões representativos da classe normal. O treinamento em lote é adequado para cenários onde os dados são estáticos, não sofrendo alterações frequentes, e onde a atualização do modelo pode ser realizada de forma assíncrona ao processamento dos dados.

Em contrapartida, a aplicação do OCSVM em fluxo é apropriada para cenários dinâmicos nos quais os dados são recebidos sequencialmente ao longo do tempo, exigindo que o modelo seja adaptável a mudanças graduais ou abruptas nos padrões dos dados. Essa abordagem pode contribuir para situações em que a atualização contínua do modelo é necessária para manter a relevância em um ambiente em constante evolução.

Sendo assim, o OCSVM emerge como uma alternativa promissora para a detecção de intrusões, principalmente devido à sua capacidade de modelar padrões em conjuntos de dados desbalanceados, sua eficácia na identificação de instâncias anômalas e sua capacidade de criar um limite de decisão em torno da classe normal, tornando-o resiliente a ataques adversários e a variações nas características das instâncias anômalas.

2.4 Trabalhos Correlatos

A detecção de ataques em ambientes computacionais é uma interessante área de pesquisa à medida que a complexidade e a sofisticação das ameaças cibernéticas continuam a evoluir. Por isso, na literatura, são apresentadas diversas técnicas para realizar a detecção de ataques e anomalias em redes.

Vu et al. [40] propuseram o uso de *Deep Transfer Learning* em um modelo baseado em *Autoencoders* para transferir conhecimento entre domínios de dados, permitindo a detecção eficaz de ataques em domínios sem informações de rótulo. Os experimentos realizados demonstraram que o modelo proposto, denominado MMD-AE, superou outros métodos de detecção de ataques, melhorando significativamente a Área Sob a Curva (AUC) na detecção de ataques em redes IoT.

Abordando a crescente complexidade dos modelos de IoT, Hasan et al. [41] utilizaram técnicas de aprendizado de máquina, como Regressão Logística, *Support Vector Machine* e Árvore de Decisão, para prever com precisão ataques e anomalias em sistemas de IoT. Eles compararam o desempenho desses algoritmos e demonstraram que modelos como Árvore de Decisão e *Random Forest* apresentaram maior precisão na detecção de anomalias.

Utilizando o aprendizado federado e redes neurais recorrentes (LSTM e GRU) para modelar o comportamento normal dos dispositivos IoT e detectar anomalias em tempo real, Mothukuri et al. [42] visaram também preservar a privacidade dos dados dos dispositivos IoT. Assim, os modelos foram treinados localmente em cada dispositivo e, em seguida, agregados em um modelo global, sem compartilhar os dados brutos. Os resultados experimentais mostraram que a abordagem proposta superou os métodos de detecção de anomalias centralizadas em termos de precisão e minimização de alarmes falsos. Além disso, a integração do aprendizado federado com um *ensemble* de modelos melhorou ainda mais a precisão da detecção de anomalias.

Já Bhunia et al. [43] apresentam o framework *SoftThings*, que utiliza técnicas de SDN para detectar e mitigar ameaças de segurança em dispositivos IoT de forma dinâmica e adaptativa. O método envolve o uso de algoritmos de aprendizado de máquina para monitorar e aprender o comportamento dos dispositivos IoT ao longo do tempo, permitindo a detecção precoce de tráfego anômalo e a mitigação de ataques. Os resultados

preliminares dos experimentos realizados no emulador Mininet mostram que o *SoftThings* é capaz de detectar ataques com cerca de 98% de precisão e mitigar os fluxos subsequentes, bloqueando ou limitando a taxa em poucos segundos.

Outro framework, trazido por Rathore et al. [44], envolve a detecção de ataques distribuídos baseada em aprendizado semi-supervisionado para IoT. O método proposto utiliza um algoritmo *Semi-Supervised Fuzzy C-Means* que combina o algoritmo *Extreme Learning Machine* com o algoritmo *Fuzzy C-Means* para detecção em tempo real. Os resultados da avaliação experimental no conjunto de dados NSL KDD demonstram que o framework alcançou um bom desempenho, com um tempo de detecção de ataque de 11 ms e uma taxa de precisão de 86,53%.

Em um contexto mais geral, isto é, não apenas voltado para a IoT, o trabalho apresentado por Jha e Ragha [45] aborda a utilização de *Support Vector Machine* em Sistemas de Detecção de Intrusão. O estudo propõe um modelo de Aprendizado de Máquina que combina os benefícios de aprendizado supervisionado e não supervisionado, utilizando uma versão modificada do SVM. Além disso, é fornecido um processo preliminar de seleção de características usando Algoritmos Genéticos para selecionar campos de pacotes mais apropriados. O método proposto foi testado no conjunto de dados KDD 99, demonstrando a eficácia na detecção de anomalias de rede.

Parveen et al. [46] apresentam um algoritmo de mineração de fluxo baseado em conjunto para a detecção de ameaças. O algoritmo aborda o desafio de identificar anomalias raras em contextos nos quais comportamentos em evolução tendem a mascará-las. Para isso, são utilizados métodos de aprendizado supervisionado, incluindo o *One-Class Support Vector Machine* e a abordagem de *ensemble*. Os resultados obtidos demonstram que o classificador desenvolvido exibe uma precisão de classificação substancialmente aumentada para fluxos reais de ameaças internas em comparação com abordagens tradicionais de aprendizado supervisionado e outros métodos de modelo único, destacando-se pela sua capacidade de lidar com comportamentos evolutivos e anomalias raras.

Ainda nesse contexto, com estratégias inovadoras de aprendizado ativo para lidar com a detecção de mudanças de conceito em fluxos de dados, Krawczyk et al. [47], abordam a necessidade de adaptação rápida à natureza evolutiva desses fluxos. O estudo apresentou um framework para aprendizado ativo em cenários de aprendizado online a partir de fluxos de dados com mudanças de conceito, utilizando o detector de mudanças ADWIN2 devido à sua eficiência com baixa complexidade computacional. Além disso, foram propostas três estratégias de aprendizado ativo, baseadas na incerteza do modelo, na alocação dinâmica de recursos ao longo do tempo e na randomização do espaço de busca. Os resultados obtidos demonstraram a eficácia dessas estratégias, mostrando que elas conseguiram obter rótulos para objetos provenientes de distribuições evoluídas, resultando em uma classificação precisa do fluxo de dados.

Por fim, Winter et al. [48] propuseram um sistema de detecção de intrusão em redes que opera com fluxos de rede e utiliza o *One-Class SVM*. Em contraste com os sistemas tradicionais de detecção de anomalias, o sistema é treinado apenas com dados maliciosos, visando reconhecer ataques previamente aprendidos, incluindo variações de ataques, em vez de detectar anomalias. O estudo realizou uma avaliação que resultou em uma taxa de alarmes falsos de 0% e uma taxa de erro de detecção de 2%. Esses resultados indicam que a abordagem proposta é promissora para pesquisas futuras e complementa os sistemas tradicionais de detecção de intrusões baseados em assinaturas.

3 MATERIAIS E MÉTODOS

Nesta seção, serão apresentados os procedimentos adotados para avaliar o emprego de algoritmos de aprendizado em fluxo contínuo visando à detecção de ataques em tempo real. O objetivo principal deste estudo é investigar a eficácia desses algoritmos na identificação de anomalias em fluxos de dados contínuos, com foco na redução da dependência de exemplos rotulados e, por conseguinte, da intervenção humana no processo de detecção.

Para alcançar os objetivos, uma abordagem sistemática foi adotada e pode ser resumida da seguinte forma: 1) busca e seleção dos conjuntos de dados, 2) transformação e pré-processamento dos dados, 3) implementação dos algoritmos e 4) realização dos testes e coleta de métricas de desempenho. Uma visão geral dessa abordagem pode ser visualizada na Figura 2.

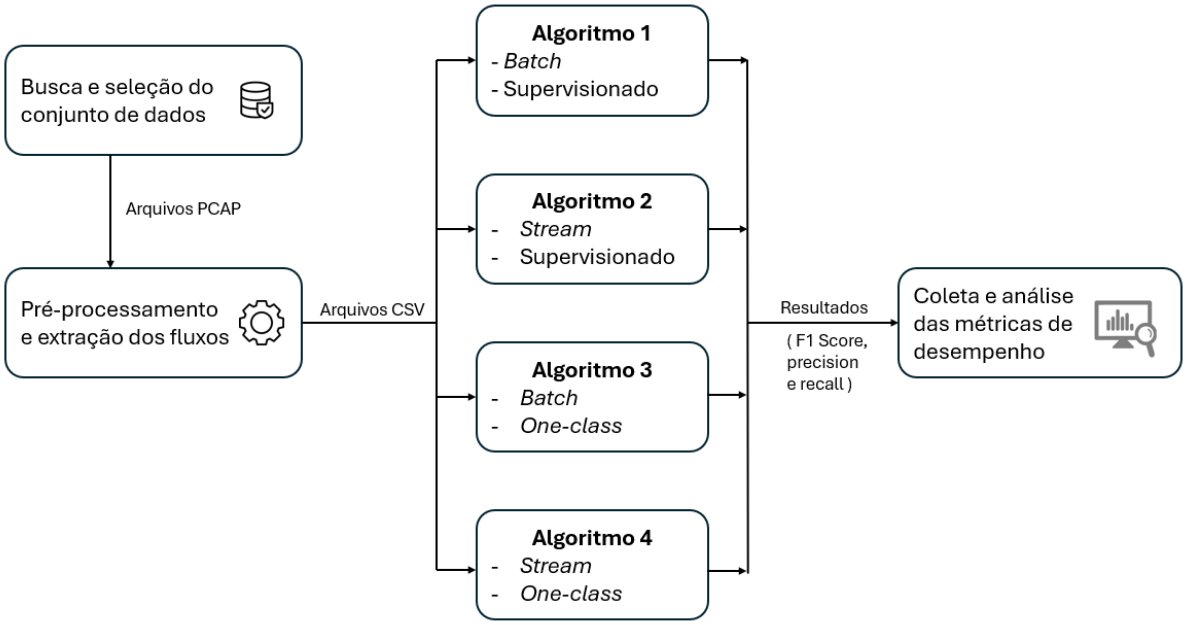

Figura 2 – Diagrama da visão geral da abordagem proposta

A escolha dos quatro algoritmos implementados foi realizada com base em alguns critérios e objetivos. O primeiro critério foi a escolha de um algoritmo de aprendizado supervisionado e em *batch*, uma vez que seu uso já é bem conhecido e muito utilizado na detecção de ataques, apresentando, normalmente, ótimos resultados, como visto na seção de trabalhos correlatos. Para isso, o algoritmo *Random Forest* foi o selecionado.

Em seguida, o algoritmo *Hoeffding Tree Classifier*, de aprendizado supervisionado e em *stream*, foi escolhido como linha de base para comparação. Este algoritmo foi selecionado por sua capacidade de lidar com a natureza dinâmica e contínua dos dados.

Como etapa subsequente, uma linha de base de *One-Class* foi implementada. Para isso, foi usado o OCSVM com aprendizado em *batch* (lote), visando verificar se a redução de exemplos rotulados é viável, e avaliar seus resultados quando comparado com um algoritmo supervisionado.

Por fim, o alvo principal da investigação é o OCSVM *stream* que combina técnicas de aprendizado em *stream* com a abordagem de *One-Class Learning*. Esta etapa final teve como objetivo principal investigar a eficácia do modelo proposto em detectar anomalias em fluxos contínuos de dados, com especial ênfase em situações nas quais o algoritmo conhece apenas o comportamento normal das redes.

3.1 Conjunto de Dados

O primeiro passo foi encontrar conjuntos de dados públicos que fossem pertinentes e representativos ao experimento, ou seja, deveriam simular um comportamento que condiz com a realidade à qual as redes IoT estão expostas. Para isso, os conjuntos devem apresentar diferentes tipos de ataques conhecidos, proporcionando um desafio real para os algoritmos selecionados.

Essa etapa influencia a avaliação da capacidade dos algoritmos de adaptação a situações diversas e em constante evolução. Assim, foram escolhidos os conjuntos de dados TON_IoT [49] e CIC IoT 2023 [50].

3.1.1 TON_IoT

O conjunto de dados TON_IoT foi criado por Nour Moustafa e a motivação para o seu desenvolvimento foi não somente a exigência de grandes conjuntos de dados on-line ou off-line para determinar a fidelidade dos modelos de IA desenvolvidos, como também a falta de bases de dados que sejam heterogêneas e representativas. Para que isso ocorra, a arquitetura do banco de testes deve abranger os serviços de IoT e suas comunicações de rede nas camadas *edge*, *fog* e *cloud*, e os dados devem envolver cenários normais e de ameaças realistas que simulem o comportamento das redes [2, 3].

O conjunto de dados apresenta nove tipos diferentes de ataques, são eles: *Injection*, *Man in the Middle* (MITM), *Backdoor Malware*, *Distributed Denial of Service* (DDoS), *Denial of Service* (DoS), *Runsomware*, *Scanning*, *Cross-site Scripting* (XSS) e *Password Cracking*. Eles incluem fontes de dados coletadas a partir de sensores IoT e *Industrial Internet of Things* (IIoT), de sistemas operacionais Windows 7 e 10, Ubuntu 14 e 18 TLS, e tráfego de rede.

Para essa coleta, foi projetada uma rede de testes realista e de grande escala no Laboratório de IoT da University of New South Wales Canberra Cyber. Na figura 3, é possível entender o seu desenvolvimento [2, 3].

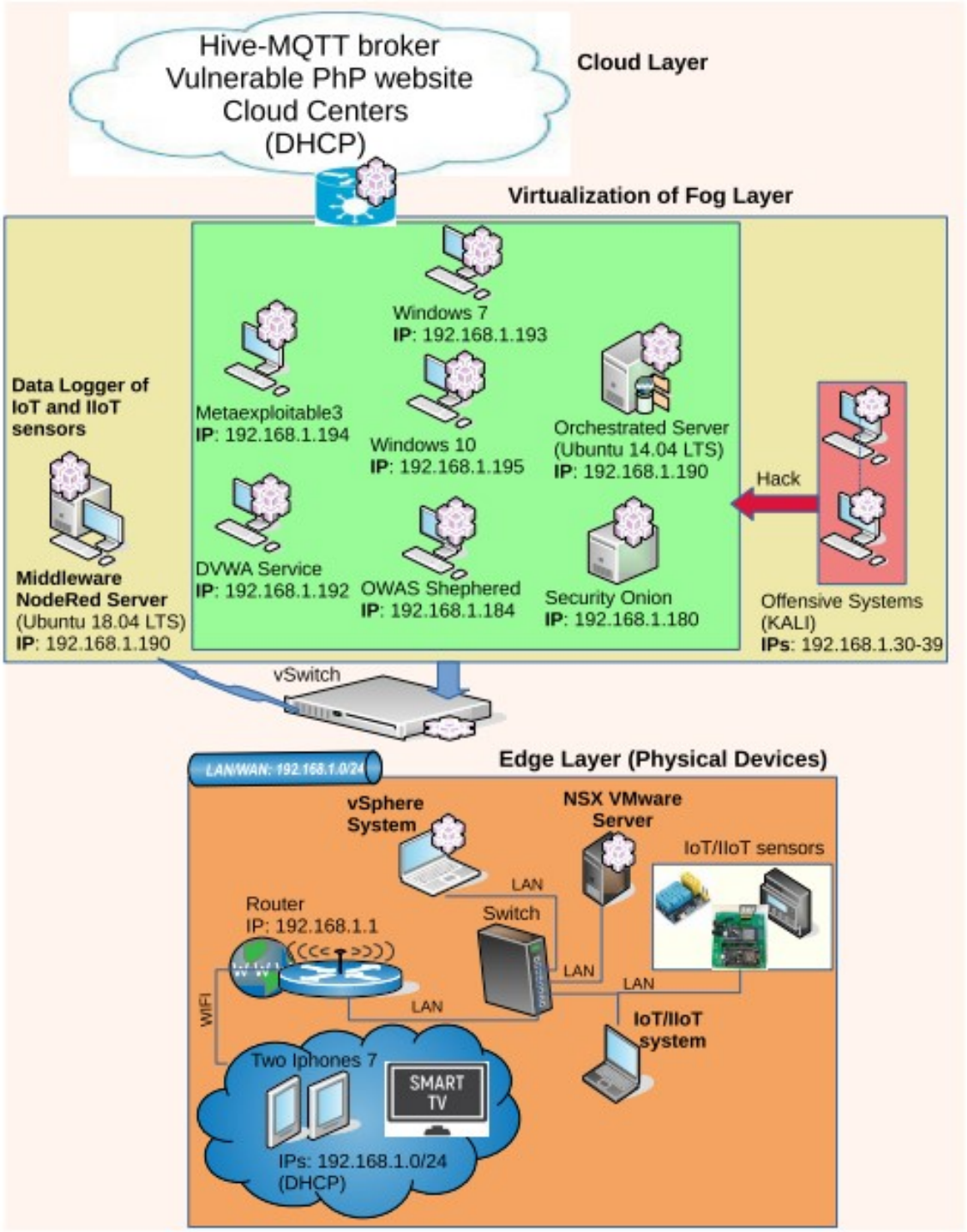

Figura 3 – Configuração da rede de testes do TON_IoT. Extraída de [2, 3]

A primeira camada, *edge*, é composta pelos dispositivos físicos, um servidor e equipamentos de rede como switches e roteadores, todos conectados a diferentes interfaces de LAN, e também, vinculados à camada *fog* por meio do vSwitch. Essa camada é a fundamental para as aplicações de IoT/IIoT, uma vez é responsável por adquirir dados medindo

diretamente as condições físicas do mundo real e posteriormente, enviar as informações para as demais camadas.

Na camada *fog*, na área em verde, estão representadas as máquinas virtuais (VMs) que seriam expostas aos ataques. Já a área em vermelho corresponde às máquinas virtuais que atuaram como atacantes, dez VMs com sistema Kali Linux, endereços IP estáticos (192.168.1.30-39) e vários *scripts bash* e Python de cenários de ataques. Também está presente nessa camada, um servidor virtualizado que executa serviços de IoT/IIoT e possui um registrador para armazenar os dados de telemetria.

Por fim, na camada *cloud* estão contidos vários serviços de nuvem, como um *Hive-MQTT broker*, utilizado para receber os dados de IoT da camada inferior para uma análise posterior, e um site PHP público usado para lançar eventos de ataques de injeção contra sites.

3.1.2 CIC IoT 2023

O conjunto de dados CIC IoT 2023, foi desenvolvido no laboratório de IoT do Canadian Institute for Cybersecurity (CIC) com a proposta de ser um banco de dados referência para ataques em larga escala associados a IoT. Sendo assim, sua criação engloba dados totalmente novos e muito extensos, envolvendo diferentes tipos de ataques que não são facilmente encontrados em outros conjuntos de dados e possibilitando o desenvolvimento de novas soluções voltadas à cibersegurança [51].

Para isso, foram utilizados 105 dispositivos, sendo 67 deles equipamentos de IoT e 33 dispositivos Zigbee e Z-Wave que estavam conectados a cinco hubs. Outra novidade neste conjunto, além dos novos ataques, é que os dispositivos de IoT também foram utilizados como agentes maliciosos e foram responsáveis por executar parte dos ataques [51].

A captura de dados ocorreu em duas etapas. Na primeira, durante 16 horas foram coletados os pacotes referentes ao tráfego benigno, ou seja, normal e sem ataques. Já na segunda etapa, os diversos ataques foram executados e, em seguida, para cada tipo de ataque, todo o tráfego capturado foi rotulado como pertencente àquele tipo de ameaça. No total foram executados 33 ataques diferenciados, divididos nas seguintes categorias: DDoS, DoS, *Reconnaissance, Web-based, Brute Force, Spoofing* e *Mirai*. Cada um dos ataques foi executado distintamente, atingindo todos os aparelhos cabíveis à situação.

3.2 Pré-processamento dos Dados

Os dados disponibilizados pelos dois conjuntos de dados se encontravam no formato de arquivos *packet capture* (PCAP), sendo um formato padrão utilizado para armazenar os dados de pacotes de rede capturados durante o período monitorado. Esses arquivos

contêm informações detalhadas sobre as comunicações de rede, incluindo cabeçalhos de pacotes, dados de *payload* e metadados associados. Os pacotes fornecem análises sob uma perspectiva mais restritiva

Por isso, o primeiro passo foi realizar a extração dos fluxos dos arquivos PCAP, que permite uma melhor análise das estatísticas e outras características de sessões de comunicação entre os nós da rede. Essa etapa envolve a identificação e a organização dos pacotes de rede em fluxos distintos, considerando as similaridades e também alguns critérios como endereços IP, portas, protocolos de comunicação, e outros. Para isso, foi utilizada a ferramenta NFstream [52], na versão 6.5.3, uma estrutura Python multiplataforma capaz de coletar os pacotes e, em seguida, extrair os fluxos e suas características, conforme a escolha do usuário.

A geração de cada um dos fluxos é feita através da agregação de pacotes que compartilham uma chave comum. Atualmente, essa chave é definida por uma tupla de sete elementos: endereço IP de origem, endereço IP de destino, porta de origem, porta de destino, protocolo, VLAN e identificadores de túnel. Caso a decodificação de túnel esteja desativada, a chave será formada somente pelos seis elementos restantes. Os cálculos dos atributos do fluxo envolvem métodos centrados em análises, como a determinação do resumo estatístico (mínimo, máximo, média e desvio padrão) [53].

Para esse estudo, foram selecionadas as características principais e as características de *post-mortem*, de cada fluxo. Características de *post-mortem* são análises referentes ao exame de fluxos de dados após terem sido capturados e armazenados, ou seja, busca entender os eventos e comportamentos que ocorreram na rede antes do término de um fluxo. As descrições de cada uma delas, fornecidas na própria documentação do NFstream, serão apresentadas nas Tabelas 1 e 2. Cada um dos arquivos PCAP disponíveis passou por esse tratamento e as características foram então armazenadas em arquivos no formato CSV.

Tabela 1 – Descrição das características principais

Nome	**Descrição**
`id`	Identificador do fluxo
`expiration_id`	Identificador do gatilho de expiração de fluxo. Pode ser 0 para idle_timeout, 1 para active_timeout ou -1 para expiração personalizada.
`src_ip`	Representação da *string* do endereço IP de origem.
`src_mac`	Representação de *string* de endereço MAC de origem.
`src_oui`	Representação de cadeia de caracteres do identificador organizacional exclusivo de origem.
`src_port`	Porta de origem da camada de transporte.

dst_ip	Representação da cadeia de caracteres do endereço IP de destino.
dst_mac	Representação de *string* de endereço MAC de destino.
dst_oui	Representação de cadeia de caracteres do identificador organizacional exclusivo de destino.
dst_port	Porta de destino da camada de transporte.
protocol	Protocolo da camada de transporte.
ip_version	Versão IP.
vlan_id	Identificador de LAN virtual.
bidirectional_first_seen_ms	*Timestamp* em milissegundos no primeiro pacote bidirecional de fluxo.
bidirectional_last_seen_ms	*Timestamp* em milissegundos no último pacote bidirecional de fluxo.
bidirectional_duration_ms	Duração bidirecional do fluxo em milissegundos.
bidirectional_packets	Acumulador de pacotes bidirecionais de fluxo.
bidirectional_bytes	Acumulador de bytes bidirecionais de fluxo (depende do accounting_mode).
src2dst_first_seen_ms	*Timestamp* em milissegundos no primeiro pacote src2dst de fluxo.
src2dst_last_seen_ms	*Timestamp* em milissegundos no último pacote src2dst de fluxo.
src2dst_duration_ms	Duração do fluxo src2dst em milissegundos.
src2dst_packets	Acumulador de pacotes src2dst de fluxo.
src2dst_bytes	Acumulador de bytes de fluxo src2dst (depende de accounting_mode).
dst2src_first_seen_ms	*Timestamp* em milissegundos no primeiro pacote dst2src de fluxo.
dst2src_last_seen_ms	*Timestamp* em milissegundos no último pacote dst2src de fluxo.
dst2src_duration_ms	Duração do fluxo dst2src em milissegundos.
dst2src_packets	Acumulador de pacotes dst2src de fluxo.
dst2src_bytes	Acumulador de bytes dst2src de fluxo (depende do Accounting_mode).

Tabela 2 – Descrição das características de *post-mortem*

Nome	**Descrição**
bidirectional_min_ps	Tamanho mínimo do pacote bidirecional de fluxo.
bidirectional_mean_ps	Tamanho médio do pacote bidirecional de fluxo.
bidirectional_stddev_ps	Desvio padrão da amostra do tamanho do pacote bidirecional de fluxo.
bidirectional_max_ps	Tamanho máximo do pacote bidirecional de fluxo.
src2dst_min_ps	Tamanho mínimo do pacote src2dst de fluxo.
src2dst_mean_ps	Fluxo src2dst tamanho médio do pacote.
src2dst_stddev_ps	Desvio padrão da amostra do tamanho do pacote src2dst do fluxo.
src2dst_max_ps	Tamanho máximo do pacote src2dst do fluxo.
dst2src_min_ps	Fluxo de tamanho mínimo do pacote dst2src.
dst2src_mean_ps	Fluxo dst2src tamanho médio do pacote.
dst2src_stddev_ps	Desvio padrão da amostra do tamanho do pacote dst2src do fluxo.
dst2src_max_ps	Fluxo de tamanho máximo do pacote dst2src.
bidirectional_min_piat_ms	Fluxo mínimo bidirecional de pacotes entre o tempo de chegada.
bidirectional_mean_piat_ms	Fluxo bidirecional de pacotes médios entre tempos de chegada.
bidirectional_stddev_piat_ms	Fluxo de pacote bidirecional entre desvio padrão da amostra de tempo de chegada.
bidirectional_max_piat_ms	Fluxo bidirecional máximo de pacotes entre tempo de chegada.
src2dst_min_piat_ms	Fluxo src2dst tempo mínimo de chegada do pacote.
src2dst_mean_piat_ms	Fluxo src2dst significa tempo de chegada entre pacotes.
src2dst_stddev_piat_ms	Fluxo do pacote src2dst entre desvio padrão da amostra do tempo de chegada.
src2dst_max_piat_ms	Fluxo src2dst tempo máximo de chegada do pacote.
dst2src_min_piat_ms	Fluxo dst2src tempo mínimo de chegada do pacote.
dst2src_mean_piat_ms	Fluxo dst2src significa tempo de chegada entre pacotes.
dst2src_stddev_piat_ms	Fluxo do pacote dst2src entre tempo de chegada e desvio padrão da amostra.
dst2src_max_piat_ms	Fluxo dst2src tempo máximo de chegada do pacote.
bidirectional_syn_packets	Acumuladores de pacotes syn bidirecionais de fluxo.

bidirectional_cwr_packets	Acumuladores de pacotes cwr bidirecionais de fluxo.
bidirectional_ece_packets	Fluxo de acumuladores de pacotes ece bidirecionais.
bidirectional_urg_packets	Fluxo de acumuladores de pacotes urgentes bidirecionais.
bidirectional_ack_packets	Fluxo de acumuladores de pacotes de confirmação bidirecionais.
bidirectional_psh_packets	Fluxo de acumuladores de pacotes psh bidirecionais.
bidirectional_rst_packets	Fluxo de acumuladores de primeiros pacotes bidirecionais.
bidirectional_fin_packets	Acumuladores de pacotes de aletas bidirecionais de fluxo.
src2dst_syn_packets	Fluxo de acumuladores de pacotes syn src2dst.
src2dst_cwr_packets	Fluxo de acumuladores de pacotes cwr src2dst.
src2dst_ece_packets	Fluxo de acumuladores de pacotes src2dst ece.
src2dst_urg_packets	Fluxo de acumuladores de pacotes urg src2dst.
src2dst_ack_packets	Fluxo de acumuladores de pacotes de confirmação src2dst.
src2dst_psh_packets	Fluxo de acumuladores de pacotes psh src2dst.
src2dst_rst_packets	Fluxo dos primeiros acumuladores de pacotes src2dst.
src2dst_fin_packets	Fluxo de acumuladores de pacotes fin src2dst.
dst2src_syn_packets	Fluxo de acumuladores de pacotes syn dst2src.
dst2src_cwr_packets	Fluxo de acumuladores de pacotes cwr dst2src.
dst2src_ece_packets	Fluxo de acumuladores de pacotes dst2src ece.
dst2src_urg_packets	Fluxo de acumuladores de pacotes urg dst2src.
dst2src_ack_packets	Fluxo de acumuladores de pacotes de confirmação dst2src.
dst2src_psh_packets	Fluxo de acumuladores de pacotes psh dst2src.
dst2src_rst_packets	Fluxo dos primeiros acumuladores de pacotes dst2src.
dst2src_fin_packets	Fluxo de acumuladores de pacotes fin dst2src.

Com os arquivos CSV prontos, o próximo passo foi rotular cada um dos fluxos. Como cada arquivo tinha suas particularidades, o processo de rotulagem foi um pouco diferente.

Para o TON_IoT, como mencionado na seção 3.1.1, os atacantes eram representados pelos IPs 192.168.1.{30, 31, 32, 33, 34, 35, 36, 37, 38 e 39}. Sendo assim, todos os fluxos que tinham como origem um desses IPs eram rotulados com "ataque" e os demais fluxos foram rotulados como "normais". Após esse processo, os cenários já estavam prontos para serem utilizados nos testes dos algoritmos.

Já para o CIC IoT, diferentemente do outro conjunto de dados, os arquivos foram organizados em várias pastas, conforme o ataque executado, porém essas pastas continham apenas os fluxos correspondentes ao ataque, e por isso, todos os fluxos foram rotulados com "ataque". Os fluxos correspondentes ao tráfego normal se encontravam isolados, em outra pasta e receberam todos a atribuição de "normal". Nesse caso, com os fluxos em locais separados, o cenário ainda não estava pronto para uso.

Para a montagem dos cenários do CIC IoT alguns passos extras foram necessários. Primeiramente, considerando que os fluxos dos diversos dispositivos utilizados estavam todos misturados e armazenados em um mesmo arquivo, foi preciso realizar a separação dos fluxos pertencentes a cada um dos dispositivos. Esse processo foi realizado por meio de uma filtragem baseada nos endereços MAC, aplicada tanto nos arquivos de fluxos normais quanto nos de ataque.

Com o objetivo de criar cenários que representem o comportamento de um dispositivo de rede durante um ataque, foi preciso juntar os dois tráfegos (normal e de ataque). Para isso, cada arquivo teve seus fluxos ordenados segundo o *timestamp* da coluna "bidirectional_last_seen_ms". Em seguida, uma nova coluna foi criada e o valor foi preenchido com a diferença entre os *timestamps* de cada fluxo em relação ao *timestamp* do primeiro fluxo presente no arquivo. Após esse processo, os tráfegos normais e de ataque, de cada dispositivo, foram mesclados com base na ordem crescente dos valores dessa nessa nova coluna.

Ainda no CIC IoT, a proporção entre dados normais e de ataques eram muito diferentes e a partir de um certo ponto, os ataques acabavam e o fluxo normal ainda continuava presente em grande quantidade, por isso, para balancear melhor os dados e facilitar a execução dos algoritmos, os cenários foram limitados a 1000 fluxos por arquivo CSV.

Após todo esse processo, a seleção dos ataques e dos dispositivos que compõem os cenários desenvolvidos foi ponderada. Além de levar em conta a disponibilidade e a quantidade de informações sobre cada tipo de ataque, foi dada uma atenção especial à inclusão daqueles que estavam presentes em ambos os conjuntos de dados selecionados.

Os escolhidos foram: DoS, *injection* e *backdoor malware*.

O ataque de DoS visa tornar um sistema ou rede indisponível para seus usuários, geralmente sobrecarregando alvo com um volume excessivo de tráfego ou solicitando recursos que esgotam a capacidade do sistema. Ataques de *injection* envolvem a inserção maliciosa de código ou comandos em um sistema através de pontos de entrada vulneráveis para executar ações não autorizadas. Por fim, o *Backdoor Malware* é um tipo de software malicioso que cria uma porta dos fundos em um sistema, permitindo que atacantes obtenham acesso remoto não autorizado.

Já entre os dispositivos, alguns não apresentavam dados suficientes para o experimento, portanto foram selecionados, para o CIC IoT, os dispositivos com um maior quantidade de fluxos. A definição dos cenários criados com o CIC IoT 2023 podem ser vistos na Tabela 3.

Tabela 3 – Dispositivos usados em cada cenário do CIC IoT 2023

Ataque	**Dispositivo analisado**	**Categoria**	**Endereço MAC**
DoS	*Fibaro Home Center Lite*	Hub	ac:17:02:05:34:27
Injection	*Wyze Camera*	Câmera	7c:78:b2:86:0d:81
Backdoor Malware	*Google Nest Mini Speaker*	Áudio	cc:f4:11:9c:d0:00

3.3 Implementação dos Algoritmos

Nesta seção, estão descritos os processos de implementação dos algoritmos de aprendizado de máquina para a detecção de ataques. Todos os algoritmos utilizados foram desenvolvidos utilizando a linguagem de programação Python e as bibliotecas Scikit-learn (versão 1.2.2) e River (versão 0.15.0).

Inicialmente, cada arquivo de teste foi lido e, em seguida, foram selecionadas para uso apenas as colunas que apresentavam informações relevantes. Considerando que a função dos algoritmos é generalizar padrões e aprender com eles, colunas que contenham informações únicas ou mais específicas, especialmente aquelas altamente dependentes do contexto em que ocorreram, não contribuem para a capacidade do modelo de generalização.

Além disso, informações temporais, como datas e horários exatos de ocorrência de eventos, podem ser removidas para evitar que o modelo se torne excessivamente dependente desses fatores temporais e, assim, não consiga generalizar para novos conjuntos de dados. Sendo assim, foram removidas as seguintes colunas: "“id”, “expiration_id”, “src_ip”, “src_mac”, “ src_oui”, “dst_ip”, “dst_mac”, “dst_oui”, “vlan_id”, “bidirectional_first_seen_ms”, “bidirectional_last_seen_ms”, “src2dst_first_seen_ms”, “src2dst_

last_seen_ms", "dst2src_first_seen_ms" e "dst2src_last_seen_ms".

Os conjuntos de dados resultantes foram formados por 61 colunas, que seriam as *features* utilizadas para fazer as classificações, e mais a coluna "label", que contém a resposta, ou seja, se o fluxo é normal ou de ataque.

Após a seleção das colunas, nos algoritmos em que se é recomendado, os dados foram normalizados para garantir que todas as características contribuíssem igualmente para o modelo. A importância desta etapa reside no fato de que muitos algoritmos de aprendizado de máquina assumem que os dados estão normalmente distribuídos e têm uma média e variância consistentes. Entretanto, se as variáveis tiverem escalas ou distribuições muito diferentes, alguns algoritmos podem ser influenciados inadequadamente, dando mais peso às variáveis com escalas maiores, fazendo com que dominem o processo de treinamento do modelo. Essa etapa foi aplicada aos algoritmos *Random Forest*, OCSVM *batch* e OCSVM *stream*.

Com os dados preparados, foi realizada a criação dos modelos de detecção de ataques. Os algoritmos selecionados foram implementados em Python e utilizaram algumas bibliotecas de aprendizado de máquina disponíveis. Detalhes específicos sobre a implementação de cada algoritmo serão discutidos em suas respectivas seções.

Para os modelos de *One-Class*, existe um parâmetro "nu" que representa a fração esperada de anomalias. Para cada um dos cenários, foi necessário ajustar esse parâmetro, considerando a porcentagem de fluxos de ataque presentes em cada arquivo, apresentados na Tabela 4. Sendo assim, o valor desse parâmetro deve estar entre 0 e 1, ou seja, se a quantidade de fluxos de ataque representa 10% do conjunto de dados, o "nu" será "0,1", e assim por diante.

Tabela 4 – Porcentagem de fluxos de ataque por cenário

Conjunto de dados	**Cenário**	**Porcentagem de fluxos de ataque**
TON_IoT	*DoS*	52%
	Injection	94%
	Backdoor Malware	26%
CIC IoT	*Dos*	50%
	Injection	17,6%
	Backdoor Malware	6,3%

3.3.1 *Random Forest*

O *Random Forest* (RF) é um método de aprendizado *ensemble* que combina múltiplas árvores de decisão para realizar tarefas de classificação ou regressão. Durante a fase de classificação, cada árvore emite uma previsão e a classe mais frequente é escolhida como a previsão final. O RF tem sido amplamente adotado devido à sua capacidade de lidar com conjuntos de dados de alta dimensionalidade e à sua robustez contra *overfitting*.

Nessa implementação, para a normalização dos dados foi utilizado o *StandardScaler* e para a criação do modelo de *Random Forest*, o *RandomForestClassifier*, ambos da biblioteca *scikit-learn*.

Após realizar toda a leitura e normalização dos dados, por ser um algoritmo de *batch*, é necessário separar o conjunto de dados em treinamento e teste e, nesse caso, foram separados igualmente, isto é, 50% para cada um. Em seguida, o modelo foi implementado, utilizando as configurações padrão, disponibilizadas na documentação do Scikit-learn [54].

Por fim, é realizado o treinamento do modelo, com os dados definidos anteriormente para tal fim e, em seguida, as predições são realizadas a partir dos dados de teste. Para verificar a eficácia, os resultados inferidos pelo algoritmo são comparados com os valores reais.

3.3.2 *Hoeffding Tree Classifier*

O *Hoeffding Tree Classifier* (HTC) é um algoritmo de árvore de decisão que se destaca pela sua capacidade de aprender incrementalmente com fluxos de dados em tempo real. Ao contrário dos métodos tradicionais, ele não exige o acesso a todo o conjunto de dados de treinamento e sua árvore é construída e atualizada à medida que novas amostras são apresentadas. Para evitar o reprocessamento dos dados, o HTC utiliza o Teste de Hoeffding para decidir quando uma decisão sobre um nó da árvore pode ser feita com confiança estatística suficiente. Essa abordagem o torna particularmente adequado para aplicações nas quais os dados estão sujeitos a alterações contínuas ao longo do tempo.

Devido a forma como sua árvore é criada, esse algoritmo não requer normalização de dados e consegue lidar muito bem com diferentes escalas de características. Para a implementação do modelo, foi utilizada a biblioteca *River*.

Nesse estudo foram implementadas três versões dessa árvore, *Hoeffding Tree Classifier, Hoeffding Adaptive Tree Classifier* e a *Extremely Fast Decision Tree Classifier* (EFDT). Após alguns testes, foi possível verificar que os resultados obtidos eram extremamente similares, porém o tempo de execução variava bastante. Exclusivamente por isso, foi escolhida a *Hoeffding Tree Classifier*, que se apresentou como a mais rápida entre as três, enquanto a EFDT foi a mais lenta.

Os parâmetros utilizados no modelo desenvolvido foram os padrões, disponibili-

zados na documentação do River [55], exceto pelo “grace_period” e pelo “delta”, cujos valores usados foram, respectivamente, “100” e “1e-4”.

Diferentemente dos algoritmos de *batch*, no *stream* não é utilizada a metodologia de *hold-out* para validação dos dados. O modelo segue o método *prequential*, na qual a série de dados é lida sequencialmente de acordo com os *timestamps* e o modelo é avaliado continuamente à medida que novas instâncias de dados chegam.

3.3.3 OCSVM - aprendizado em *batch*

O *One-Class SVM* é uma variante do algoritmo SVM, projetada para a detecção de anomalias. Seu objetivo é aprender a fronteira de decisão que separa a classe de interesse (maligna) das amostras normais (benigna) no espaço de características. Durante o treinamento, o OCSVM ajusta um hiperplano que encapsula a maioria das amostras normais. Então, na fase de teste, as amostras que se encontram fora da margem do hiperplano são consideradas anomalias.

Para a normalização dos dados e a criação do modelo, foram usados o *StandardScaler* e o *OneClassSVM*, pertencentes à biblioteca *scikit-learn*.

Durante a fase de treinamento do algoritmo, apenas dados da classe normal são necessários para o treinamento adequado do modelo. Para atender a essa demanda, foi realizada a separação dos fluxos normais e dos de ataques. Nesse processo, 50% dos fluxos normais foram reservados para o treinamento do modelo, enquanto o restante foi reintegrado ao conjunto de dados de ataques, compondo assim os dados de teste.

A implementação do modelo utilizou os parâmetros padrões, disponíveis na documentação do OCSVM do *scikit-learn* [56], com exceção do parâmetro “nu” que, como explicado anteriormente, varia de acordo com a quantidade de dados disponíveis. Dessa forma, para cada cenário testado, foi necessário alterar o “nu”, seguindo os valores apresentados na Tabela 4.

3.3.4 OCSVM - aprendizado em *stream*

O *One-Class SVM* com aprendizado em *stream*, funciona de forma similar ao OCSVM de *batch*, e seu princípio subjacente é identificar uma hiper superfície que envolva a maioria dos dados normais, enquanto minimiza a quantidade de pontos de dados anômalos incluídos. Durante o treinamento incremental do modelo, o OCSVM ajusta essa hiper superfície, identificando regiões de baixa densidade de dados, as quais são mais propensas a conter anomalias. Isso permite que o modelo seja capaz de generalizar para novos dados e identificar padrões anômalos com base nas características aprendidas durante o treinamento.

O modelo de OCSVM utilizado trabalha com a geração de uma pontuação de anomalia, cujo cálculo é feito considerando as características das amostras com relação ao que o modelo considera "normal" durante o treinamento. Logo, para determinar se uma amostra é anômala ou não de acordo com essa pontuação, é necessário um componente adicional que fará essa classificação com base em um limiar definido, que pode ser ajustado de acordo com a situação. Nesse trabalho, o responsável por essa função foi o *QuantileFilter*.

A normalização dos dados utilizou o *StandardScaler*, e o modelo recorreu ao *OneClassSVM*, juntamente com o *QuantileFilter*, todos da biblioteca *River*.

Considerando o modo de aprendizado, esse algoritmo possui algumas peculiaridades em relação aos demais. Como o modelo busca aprender com os dados normais e também segue o método *prequential* no qual os dados são lidos de forma sequencial, é necessário garantir que no início de cada um dos arquivos de teste, esteja presente uma quantidade razoável de dados normais, uma vez que o modelo começará o treinamento incremental a partir desses dados. Nesse caso, foram utilizados, em média, 200 fluxos normais no início de cada teste.

Para esse modelo, o parâmetro "nu" segue a mesma ideia do OCSVM de *batch* e os valores para cada cenário de ataque podem ser consultados na Tabela 4. O restante dos parâmetros utilizados foram padrões, definidos na documentação do *One-Class SVM* do River [57].

Com relação ao *QuantileFilter*, os parâmetros "q" e "protect_anomaly_detector" foram estabelecidos os valores "0.8" e "false", respectivamente. O parâmetro "q" representa o nível de quantil acima do qual se deve classificar uma pontuação como anômala. Para a escolha de seu valor, foram testados, manualmente, diferentes valores e escolheu-se aquele que apresentava, em média, os melhores resultados para todos os cenários simulados. Já o "protect_anomaly_detector" indica se o detector deve ser atualizado quando a pontuação é anômala, algo que não é muito recomendado em casos de dados que contém anomalias esporádicas e, por isso, foi desativado para esse trabalho.

3.4 Testes e Métricas de Desempenho

A última etapa proposta foi testar cada um dos seis cenários selecionados, para isso, cada um dos arquivos CSV foram fornecidos como entrada e executados nos quatro algoritmos implementados.

Para avaliar a eficácia de cada um dos algoritmos, foram coletadas métricas de desempenho baseadas em matrizes de confusão. Para isso, foram contabilizados quatro valores, sendo eles: *true positive* (TP), *true negative* (TN), *false positive* (FP), *false negative* (FN). A compreensão de cada uma delas fornece um melhor entendimento das

próximas métricas que serão apresentadas:

1. *True positive* ou verdadeiro positivo: contagem de instâncias em que o modelo previu corretamente a ocorrência de um ataque, ou seja, estava de fato ocorrendo um ataque e ele foi detectado;

2. *True negative* ou verdadeiro negativo: quantidade de vezes em que o modelo indicou a ausência de um evento, nesse caso, um ataque, e acertou na previsão;

3. *False positive* ou falso positivo: contagem que acontece quando a ocorrência de um evento é detectada e, na verdade, ela não está ocorrendo;

4. *False negative* ou falso negativo: indica os casos em que uma intrusão está acontecendo e o modelo falha em identificá-la.

Todas essas medidas explicadas anteriormente oferecem uma base para o cálculo das métricas de desempenho utilizadas para julgar a capacidade de cada um dos algoritmos na detecção de ataques, *precision*, *recall* e *F1 score* e, as fórmulas de cada uma delas podem ser observadas nas equações 3.1, 3.2 e 3.3, respectivamente.

$$Precision = \frac{TP}{TP + FP} \tag{3.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{3.2}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall} \tag{3.3}$$

A *precision*, ou precisão, é a proporção de verdadeiros positivos (TP) em relação ao total de instâncias positivas previstas pelo modelo (TP + FP), em outras palavras, ela indica a precisão do modelo em classificar corretamente as instâncias positivas. Já o *recall*, também conhecido como sensibilidade, é a proporção de verdadeiros positivos (TP) em relação ao total de instâncias positivas reais na amostra (TP + FN), ou seja, mede a capacidade do modelo de identificar corretamente todas as instâncias de uma classe positiva.

Por fim, o *F1 score* é a média harmônica entre *precision* e *recall*, sendo particularmente útil em situações em que há um desequilíbrio entre as classes positivas e negativas, uma vez que considera tanto a capacidade de classificação correta quanto a de recuperação das instâncias relevantes.

4 RESULTADOS

Neste capítulo, serão apresentados os resultados obtidos a partir da aplicação dos diferentes algoritmos e da coleta das métricas de desempenho. Uma análise detalhada dos resultados das métricas será conduzida, buscando entender não apenas a eficácia de cada algoritmo isoladamente, mas também suas capacidades em diferentes cenários e contextos. Serão exploradas as razões por trás das variações nos resultados e como essas variações podem impactar a aplicabilidade de cada algoritmo em situações reais.

Em seguida, os resultados das Análises de Componentes Principais (PCA) serão apresentados e discutidos. Essa abordagem foi adotada visando investigar a estrutura subjacente dos dados disponíveis, buscando identificar padrões latentes e entender melhor a complexidade do conjunto de dados.

4.1 Análise da Capacidade Preditiva

Esta análise permitirá uma comparação direta entre as abordagens utilizadas, apresentando os desempenhos individuais de cada algoritmo para cada um dos ataques executados. Para facilitar as comparações, os resultados serão analisados por tipo de ataque, considerando os dois conjuntos de dados.

Serão apresentados os valores de *F1 score*, *precision* e *recall* resultantes de cada cenário, por modelos desenvolvidos. Para uma melhor visualização e comparação entre os algoritmos, os resultados foram plotados em gráficos, divididos por ataque e por conjunto de dados.

Tendo em vista que os conjuntos de dados utilizados para o treinamento dos algoritmos de *batch* são selecionados de forma aleatória e podem gerar resultados diferentes, para esses algoritmos, os testes foram executados dez vezes, para cada um dos cenários. Os resultados apresentados são as médias dessas dez execuções e os resultados individuais de cada execução podem ser conferidos nos apêndices de A a D, no final deste trabalho.

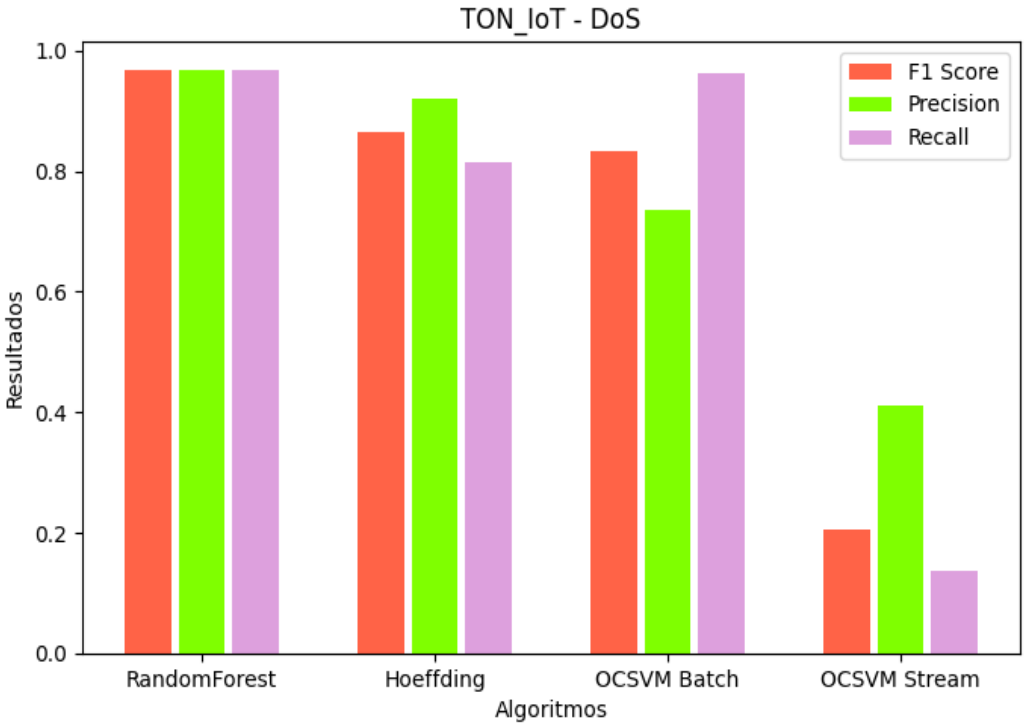

Figura 4 – Resultados para o cenário TON_IoT ataque de DoS

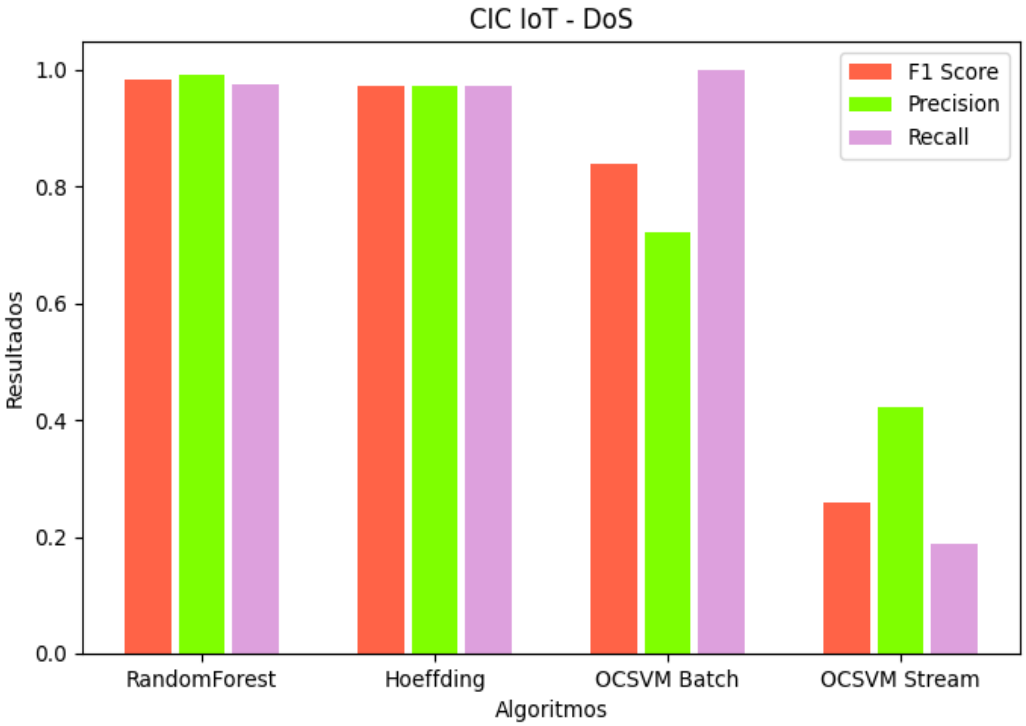

Figura 5 – Resultados para o cenário CIC IoT 2023 ataque de DoS

Conforme observado nas Figuras 4 e 5, para o ataque de DoS, todos os algoritmos, exceto o *One-Class* de *stream*, obtiveram bons resultados, todos com um *F1 score* acima de 0,8. Já o OCSVM de aprendizado em *stream*, apresentou métricas bem abaixo dos demais algoritmos, mostrando que nesse caso, a combinação do *stream* com o *One-Class* não funcionou muito bem.

Analisando os gráficos é possível notar que ambos possuem uma aparência muito similar, uma vez que quando observamos o mesmo algoritmo em cada um dos conjuntos de dados, eles apresentam resultados muito próximos e isso acontece em todos os quatro algoritmos testados. Isso pode ser um indicativo de que os dados gerados pelo ataque de DoS possuem características claras, ou seja, que os padrões dos ataques de DoS são de natureza semelhante em conjuntos de dados diferentes e, por isso, as métricas ficaram parecidas.

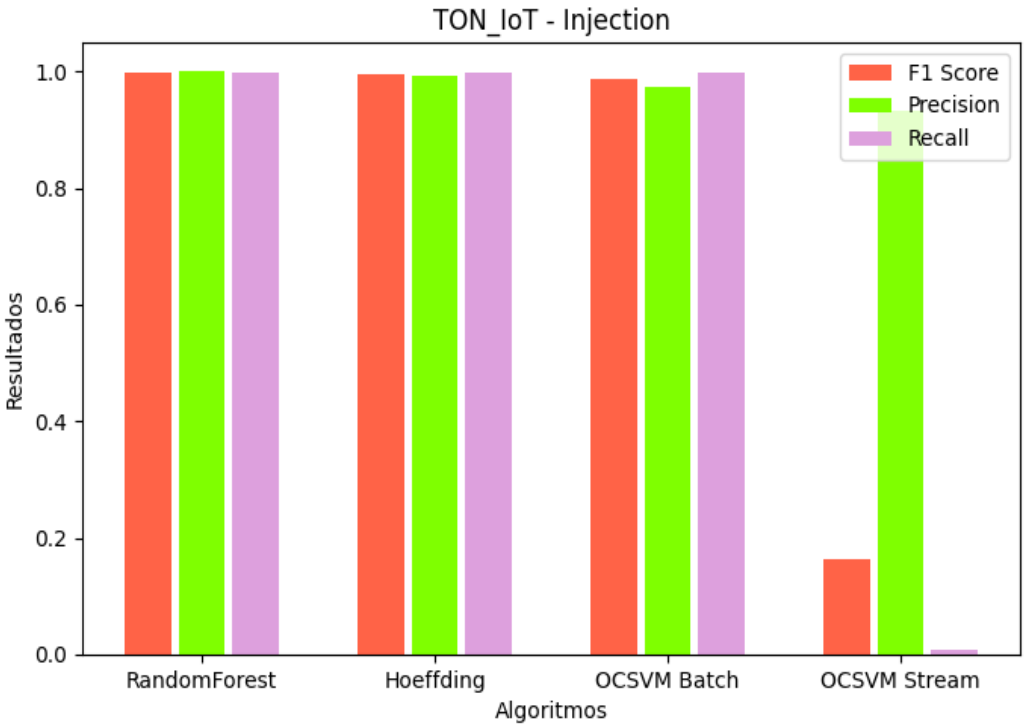

Figura 6 – Resultados para o cenário TON_IoT ataque de injection

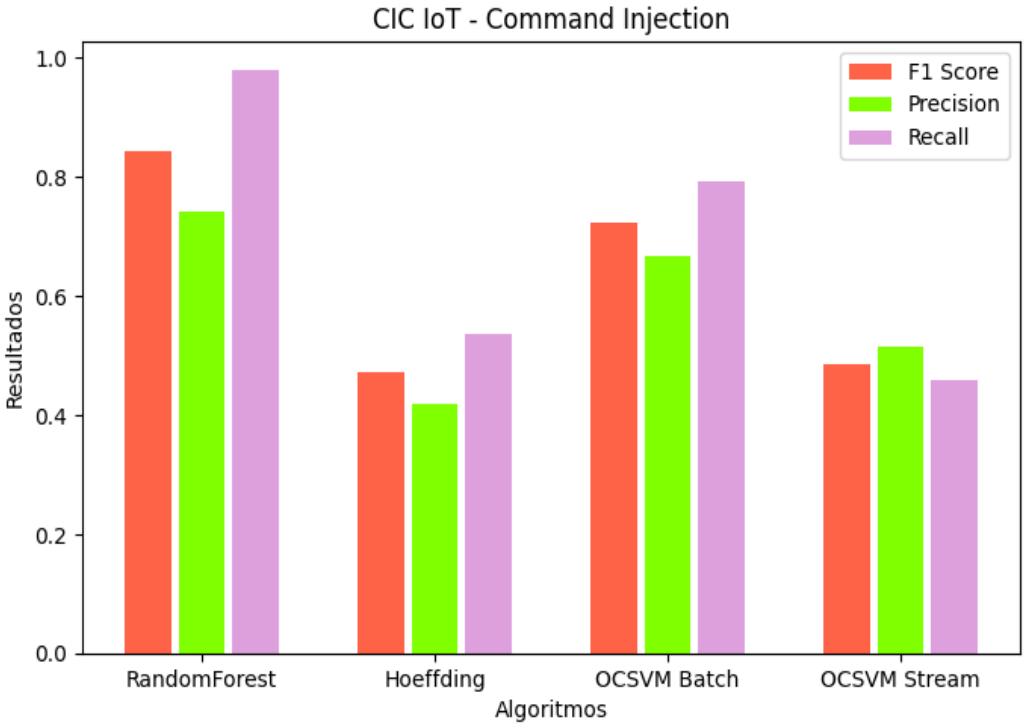

Figura 7 – Resultados para o cenário CIC IoT 2023 ataque de command injection

Para o cenário de injection do TON_IoT, na figura 6, os três primeiros algoritmos se mostraram satisfatórios, com resultados próximos do valor máximo possível de ser atingido. Em contrapartida, o OCSVM de *stream* novamente não apresentou um resultado muito satisfatório, mas é possível observar que, nesse caso, a *precision* obteve resultados altos, diferentemente do *recall* que se mostrou péssimo, falhando na identificação da maioria dos ataques, sendo assim, a causa do baixo valor de *F1 score*.

Já conforme a figura 7, no CIC IoT, enquanto o OCSVM *stream* teve uma melhora significativa, apresentando valores em um patamar mais intermediário. Os outros três algoritmos demonstraram uma queda na capacidade preditiva. Essa queda pode estar relacionada ao tipo de ataque ou também aos dados fornecidos, uma vez que todos os algoritmos apresentaram um comportamento inferior ao observado no conjunto de dados do TON_IoT.

Ainda é possível notar que diferentemente dos outros cenários, a *Hoeffding Tree* apresentou valores abaixo da média, evidenciando que os algoritmos de aprendizado em lote obtiveram uma pequena vantagem em relação aos de aprendizado em fluxo.

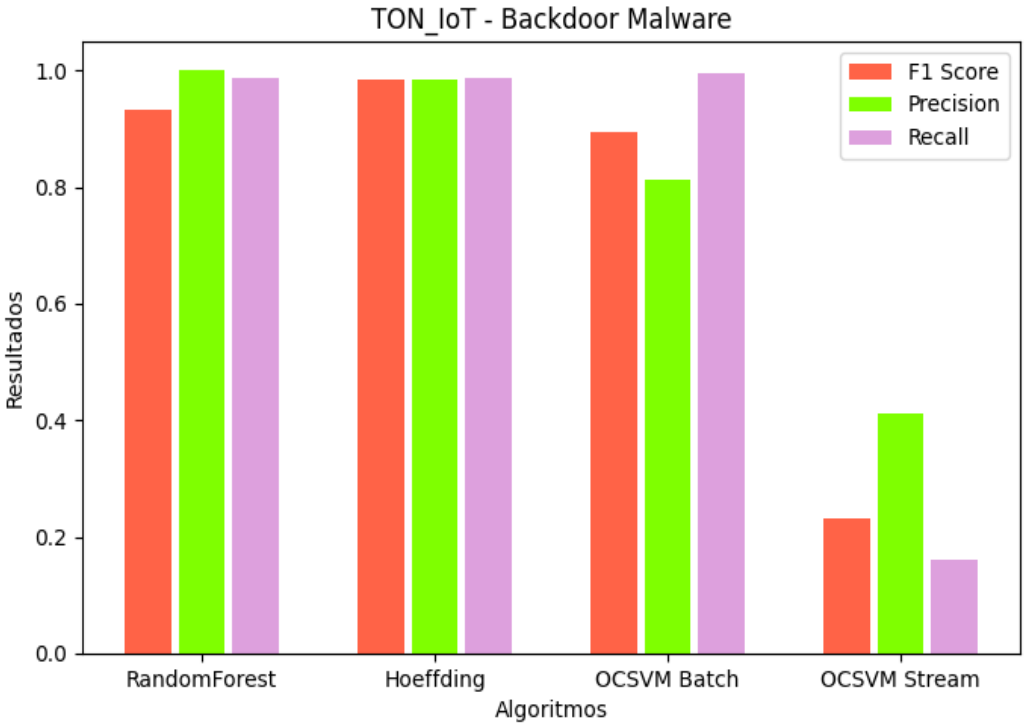

Figura 8 – Resultados para o cenário TON_IoT ataque de backdoor malware

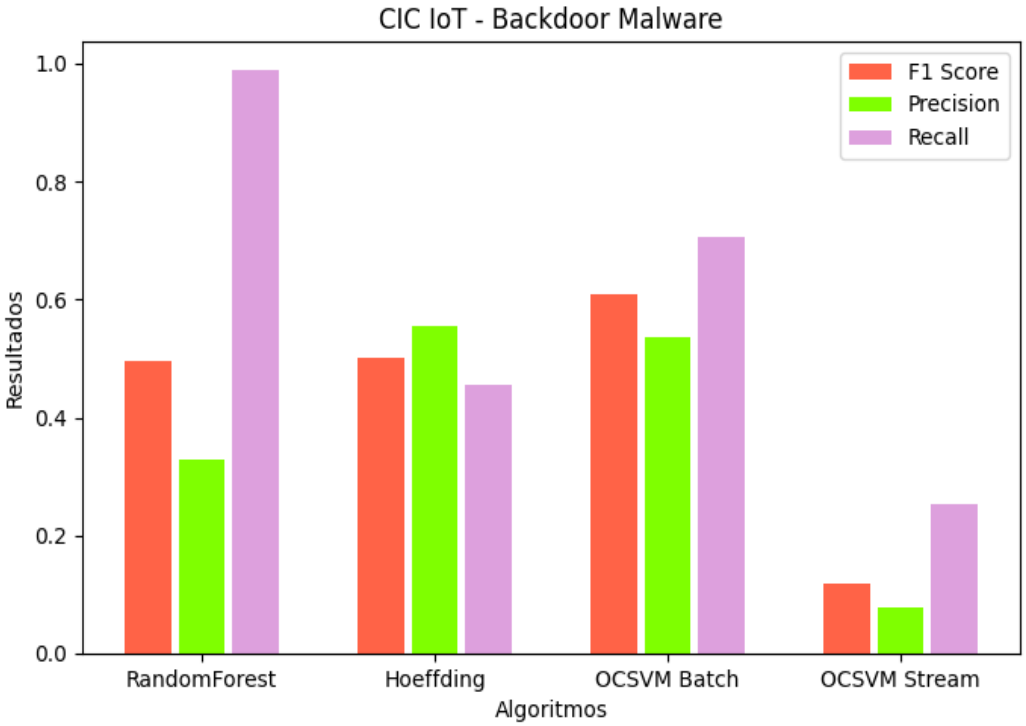

Figura 9 – Resultados para o cenário CIC IoT 2023 ataque de backdoor malware

No TON_IoT, Figura 8, o ataque de *Backdoor Malware* apresentou características muito aproximadas dos dois gráficos gerados pelo DoS, com resultados ótimos para todos os algoritmos, exceto o OCSVM de *stream*. Por outro lado, para o CIC-IoT, Figura 9, os resultados alcançados foram totalmente diferentes dos demais, considerando que até mesmo a *Random Forest*, que normalmente apresenta resultados bem consistentes, manifestou alto valor de *recall*, mas valores baixos para a *precision* e para o *F1-Score*. Apenas nesse cenário, todos os algoritmos resultaram em um *F1 score* abaixo de 0,6.

Uma das possíveis causas é que os dados disponíveis não são adequados para a tarefa em questão. Isso pode ocorrer se os dados forem de baixa qualidade, estiverem incompletos ou não representarem adequadamente a complexidade do problema. Além disso, a situação pode ser exacerbada se estivermos lidando com um problema desafiador, como um ataque projetado para confundir os algoritmos. Em tais casos, os algoritmos podem falhar em capturar os padrões relevantes nos dados, resultando em um desempenho consistentemente ruim, como visto na Figura 9.

Tabela 5 – Resultados gerais dos testes executados.

Conjunto de dados	**Ataque**	**Métrica**	**Random Forest**	**Hoeffding Tree**	**OCSVM *batch***	**OCSVM *stream***
TON_IoT	DoS	F1 score	0,967	0,865	0,833	0,206
		Precision	0,967	0,921	0,735	0,412
		Recall	0,967	0,815	0,962	0,137
	Injection	F1 score	0,999	0,996	0,987	0,165
		Precision	1,000	0,994	0,974	0,932
		Recall	0,999	0,998	0,999	0,090
	Backdoor	F1 score	0,933	0,985	0,895	0,233
		Precision	1,000	0,984	0,814	0,412
		Recall	0,987	0,987	0,995	0,162
CIC IoT	DoS	F1 score	0,984	0,973	0,839	0,259
		Precision	0,993	0,972	0,723	0,423
		Recall	0,975	0,974	0,999	0,187
	Injection	F1 score	0,843	0,471	0,723	0,486
		Precision	0,741	0,420	0,666	0,516
		Recall	0,978	0,536	0,792	0,460
	Backdoor	F1 score	0,495	0,500	0,608	0,118
		Precision	0,330	0,556	0,535	0,077
		Recall	0,998	0,455	0,705	0,254

Após uma análise geral dos resultados, apresentados na Tabela 5, nota-se que os resultados do OCSVM de *stream* não foram satisfatórios para nenhum dos cenários de ataque. No entanto, pode-se ressaltar que o problema não parece estar relacionado ao aprendizado em fluxo, uma vez que o HTC apresentou um desempenho notável. Além disso, também não parece ser uma questão exclusiva do *One-Class*, já que o OCSVM de lote também obteve resultados satisfatórios.

4.2 Análise de Componentes Principais

A análise de componentes principais, do inglês, *principal component analysis*, é uma técnica de análise multivariada cujo objetivo é simplificar a complexidade de conjuntos de dados, reduzindo o número de variáveis enquanto mantém o máximo de informações possível, ou seja, ele identifica os componentes principais que melhor representam os dados originais. Esses componentes principais são uma combinação linear das variáveis originais e são organizados de forma que o primeiro componente capture a maior parte da variação nos dados, o segundo capture a segunda maior parte, e assim por diante.

O PCA opera reduzindo a dimensionalidade dos conjuntos de dados, transformando o conjunto de dados original em um novo conjunto de dados, onde as variáveis não são correlacionadas entre si. No caso desse trabalho, ele transforma as 61 colunas de dados que estavam sendo utilizadas em apenas duas colunas, buscando facilitar a interpretação dos resultados.

As figuras 10 a 15 representam os resultados da análise de componentes principais dos seis cenários testados:

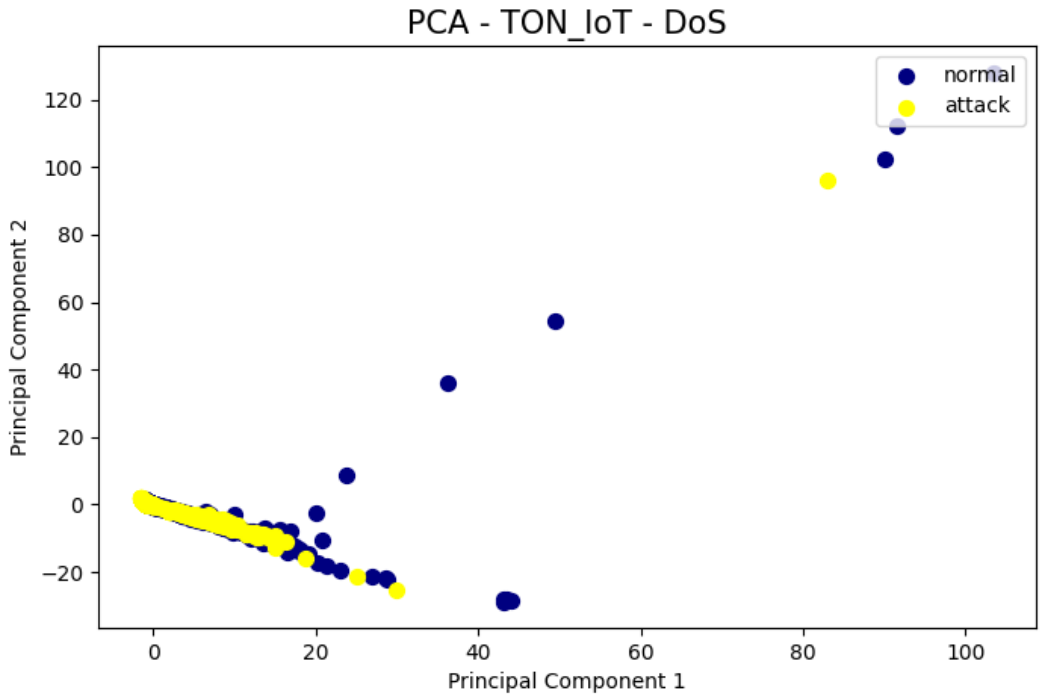

Figura 10 – PCA TON_IoT - DoS

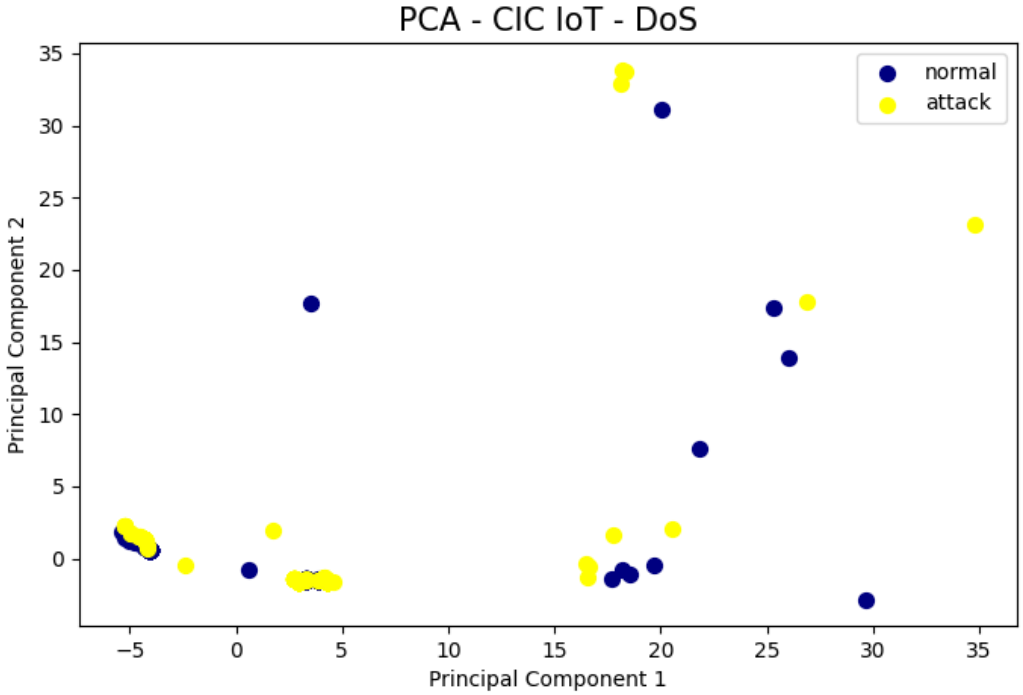

Figura 11 – PCA CIC IoT 2023 - DoS

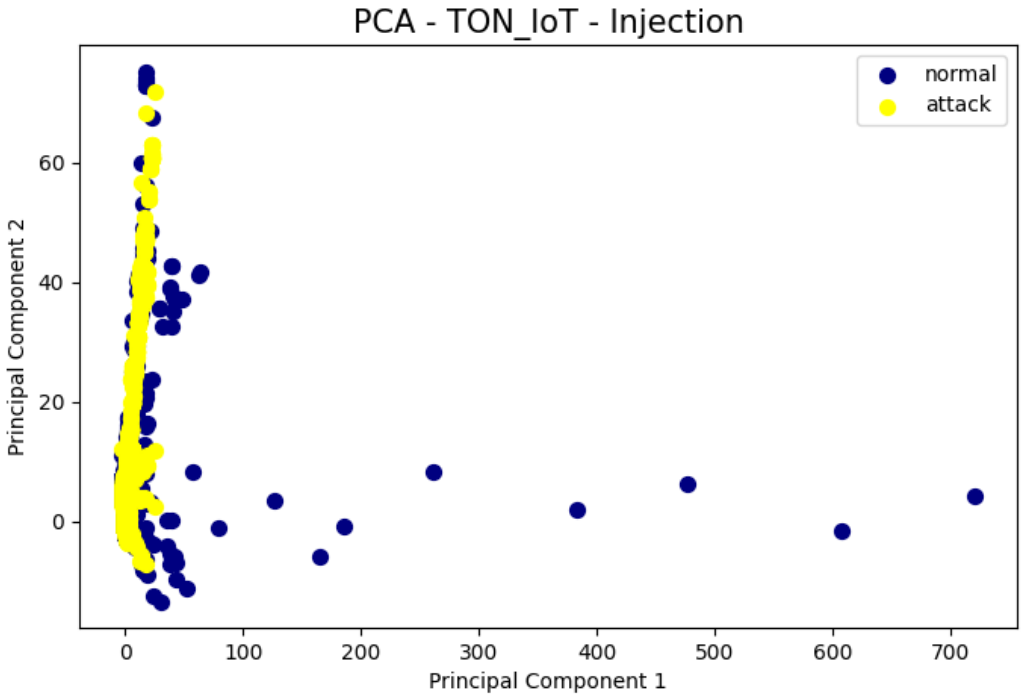

Figura 12 – PCA TON_IoT - injection

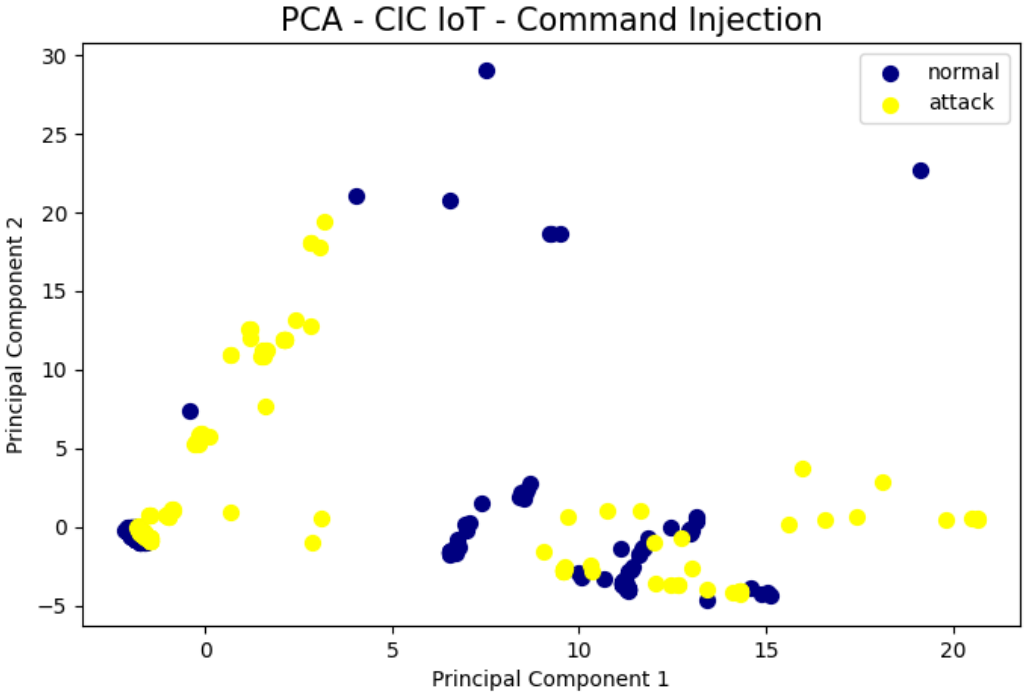

Figura 13 – PCA CIC IoT 2023 - command injection

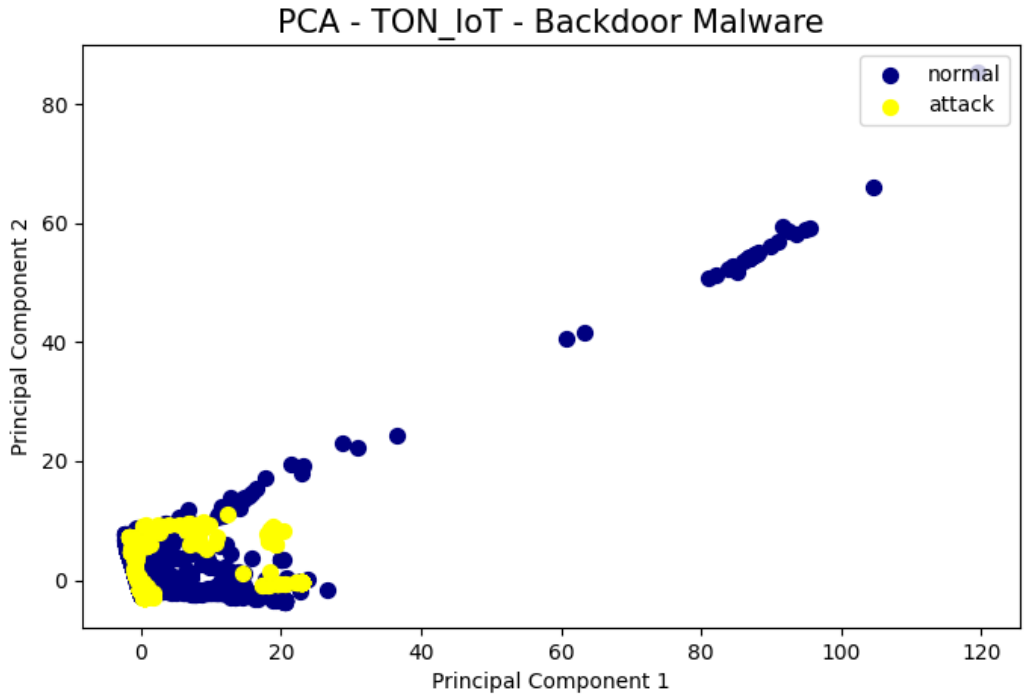

Figura 14 – PCA TON_IoT - backdoor malware

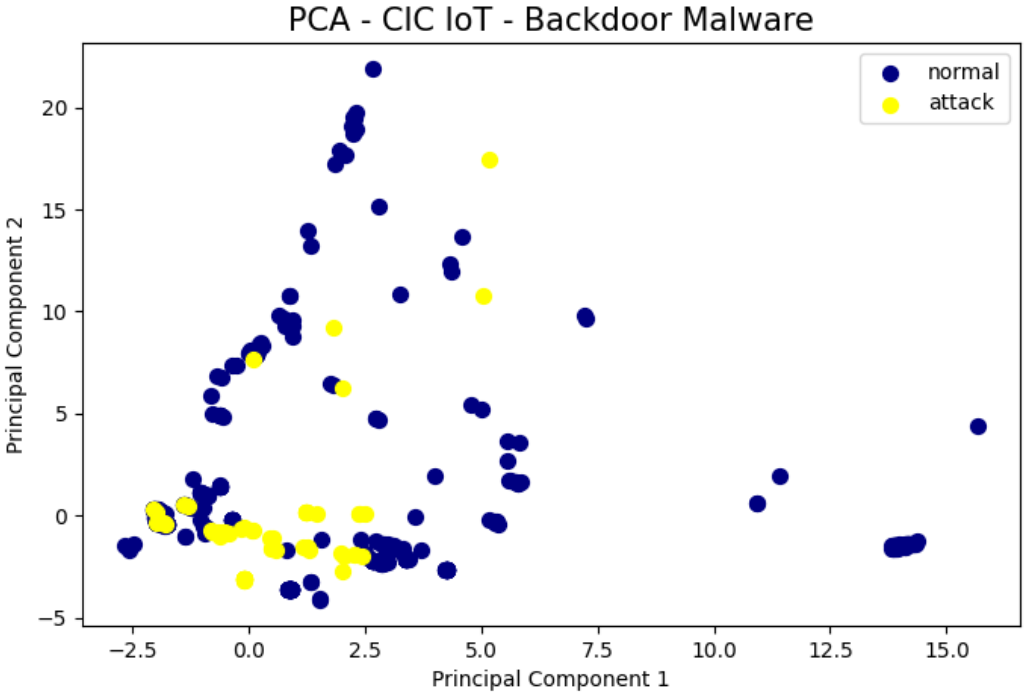

Figura 15 – PCA CIC IoT 2023 - backdoor malware

A partir dos resultados do PCA de cada cenário, observa-se que os gráficos das análises não mostraram diferenças evidentes, indicando que as características dos dados originais não são distintas o suficiente para permitir uma separação clara entre as classes. Isso pode acontecer em situações em que a variabilidade nos dados é insuficiente. Outra possível explicação é que ao fazer a redução de dimensionalidade, o PCA visa preservar a maior quantidade possível de variabilidade dos dados, mas pode acontecer de alguma informação discriminativa ser perdida durante o processo de redução.

Na tentativa de encontrar uma explicação para esses resultados obtidos, foi executado também o PCA para três componentes principais. Esse procedimento foi executado para todos os seis cenários e com os novos gráficos gerados, foi possível observar que o PCA com dois componentes principais realmente estava lidando com perda de informações durante a redução de dimensões, uma vez que o PCA de três componentes apresentou uma melhor separação entre as classes benigna e maliciosa.

Na Figura 16, está representado um exemplo do resultado de uma execução do PCA de três componentes principais para o cenário de ataque DoS, do conjunto de dados do CIC IoT 2023, juntamente com a delimitação de duas áreas que auxiliam para um melhor entendimento dos resultados. Na Figura 17, é destacada a área 1 na qual estão contidos os maiores volumes de dados e constata-se a divisão existente entre as duas classes apresentadas.

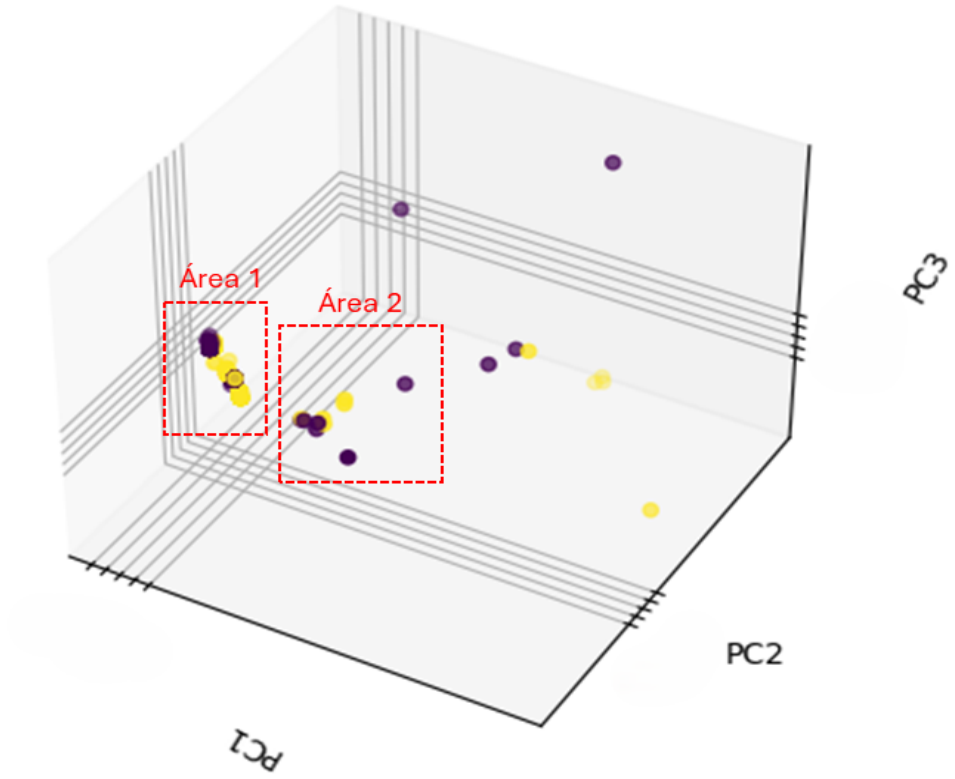

Figura 16 – Exemplo de PCA com 3 componentes principais - CIC IoT 2023 ataque DoS

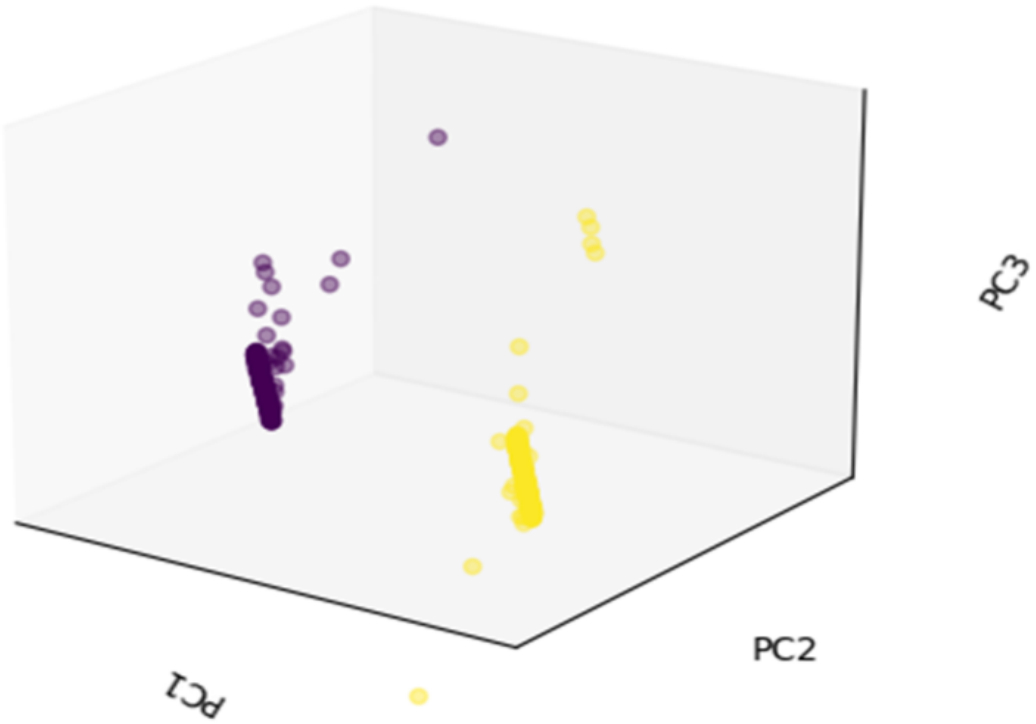

Figura 17 – Zoom aplicado na área 1

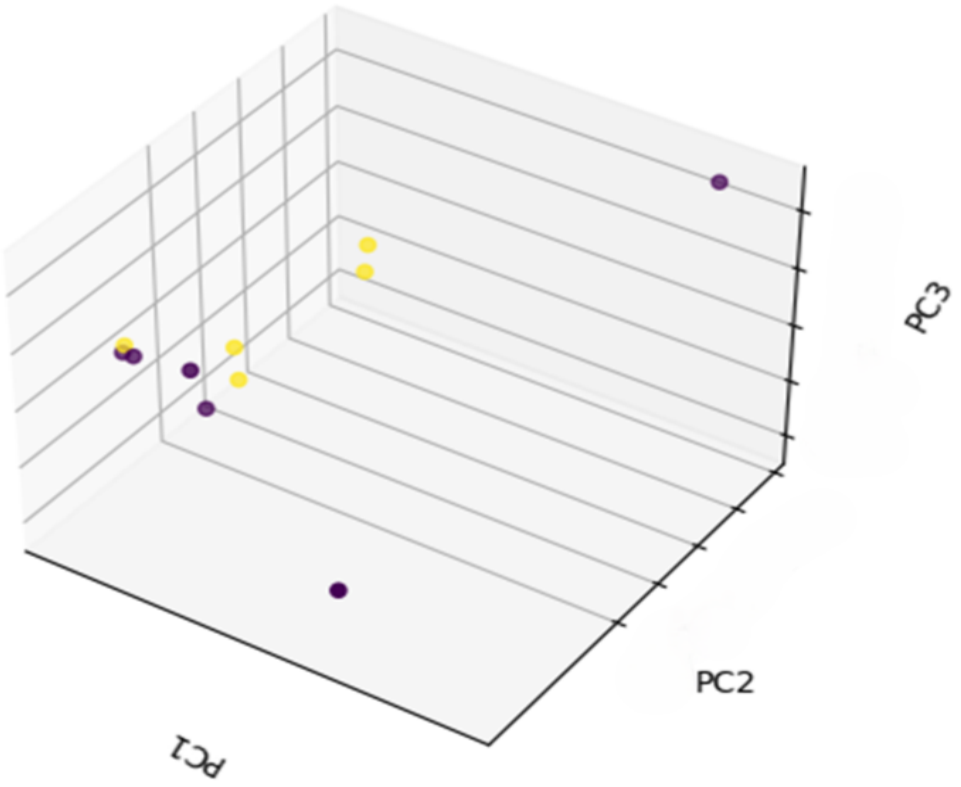

Figura 18 – Zoom aplicado na área 2

Por fim, a Figura 18 foca na área 2 do gráfico, revelando a presença de alguns fluxos que se encontram distantes das áreas delimitadas por cada classe. Estes pontos, apesar de representarem exceções, estão presentes em um volume relativamente pequeno quando comparados aos dados que apresentaram uma separação clara entre as classes. Embora esses pontos influenciem a visualização dos dados, sua influência sobre os resultados tende a ser mitigada devido à sua quantidade limitada.

5 CONCLUSÃO

A crescente expansão de dispositivos de IoT trouxe consigo não apenas uma série de benefícios inegáveis para o cotidiano das pessoas, mas também desafios significativos em termos de segurança cibernética. À medida que avançamos para um futuro mais conectado, o investimento em soluções robustas de detecção de ataques em redes IoT torna-se cada vez mais necessário, garantindo um aproveitamento dessa tecnologia de maneira segura e confiável. Sendo assim, a detecção de ataques em redes IoT emergiu como uma área de pesquisa crítica para reduzir os riscos associados à exploração de vulnerabilidades nesses ambientes altamente distribuídos e heterogêneos.

Com o objetivo de verificar a eficácia de se utilizar o aprendizado de *stream* juntamente com a diminuição da necessidade de exemplos rotulados, quando comparados a outros métodos de aprendizado, foram implementados quatro algoritmos com diferentes meios de aprendizagem.

Essas diferenças de aprendizado contribuíram para a formação de algumas conclusões. Primeiramente, os algoritmos de aprendizado supervisionado, independente de trabalharem com *batch* ou *stream*, no geral, apresentaram resultados satisfatórios. Sendo assim, a *Hoeffding Tree Classifier*, que opera com aprendizado *stream*, reforça que esse aprendizado também não é um empecilho na detecção de anomalias. Por fim, o *One-Class* aplicado a *batch*, mostrou que o aprendizado semi-supervisionado, pode ser aplicado com sucesso na diminuição dos rótulos.

O grande problema surgiu com a junção dos aprendizados de *stream* e do *One-Class*, cujas métricas resultantes foram bem baixas em relação aos outros modelos testados. Nesse caso, o modelo não consegue identificar de maneira independente quais fluxos são normais e por isso, acaba não percebendo e classificando os ataques equivocadamente.

Por fim, este trabalho mostrou que a diminuição da intervenção humana na rotulação de dados é possível de ser implementada quando se trata da detecção de ataques conhecidos, uma vez que o aprendizado de *batch* se mostrou eficiente. Entretanto, para aplicá-lo também a realidade atual das redes IoT, na qual os dados chegam de maneira rápida e estão em constante evolução, ainda são necessários muitos estudos para compreender e poder melhorar ou adaptar as implementações dos algoritmos existentes.

Para trabalhos futuros, pode-se destacar a necessidade de entender melhor o funcionamento de algumas etapas mais internas aos algoritmos, como a geração da pontuação de anomalia, e o impacto causado pela mudança de determinados parâmetros. Além disso, também podem ser exploradas outras técnicas de pré-processamento e tratamento de dados, com o objetivo de melhorar o desempenho do OCSVM.

REFERÊNCIAS

[1] BORGIA, E. The internet of things vision: Key features, applications and open issues. *Computer Communications*, v. 54, p. 1–31, 2014. ISSN 0140-3664. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0140366414003168>.

[2] MOUSTAFA, N. A new distributed architecture for evaluating ai-based security systems at the edge: Network ton_iot datasets. *Sustainable Cities and Society*, Elsevier, v. 72, p. 102994, 2021.

[3] ALSAEDI, A. et al. Ton_iot telemetry dataset: A new generation dataset of iot and iiot for data-driven intrusion detection systems. *IEEE Access*, v. 8, p. 165130–165150, 2020.

[4] ABOMHARA, M.; KøIEN, G. M. Security and privacy in the internet of things: Current status and open issues. In: *2014 International Conference on Privacy and Security in Mobile Systems (PRISMS)*. [S.l.: s.n.], 2014. p. 1–8.

[5] FRUSTACI, M. et al. Evaluating critical security issues of the iot world: Present and future challenges. *IEEE Internet of Things Journal*, v. 5, n. 4, p. 2483–2495, 2018.

[6] MAHMOUD, R. et al. Internet of things (iot) security: Current status, challenges and prospective measures. In: *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*. [S.l.: s.n.], 2015. p. 336–341.

[7] ZHANG, Z.-K. et al. Iot security: Ongoing challenges and research opportunities. In: *2014 IEEE 7th International Conference on Service-Oriented Computing and Applications*. [S.l.: s.n.], 2014. p. 230–234.

[8] LIU, Z. et al. Anomaly detection on iot network intrusion using machine learning. In: *2020 International Conference on Artificial Intelligence, Big Data, Computing and Data Communication Systems (icABCD)*. [S.l.: s.n.], 2020. p. 1–5.

[9] MANIRIHO, P. et al. Anomaly-based intrusion detection approach for iot networks using machine learning. In: *2020 International Conference on Computer Engineering, Network, and Intelligent Multimedia (CENIM)*. [S.l.: s.n.], 2020. p. 303–308.

[10] SUSILO, B.; SARI, R. F. Intrusion detection in software defined network using deep learning approach. In: *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*. [S.l.: s.n.], 2021. p. 0807–0812.

[11] DITZLER, G. et al. Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, v. 10, n. 4, p. 12–25, 2015.

[12] L'HEUREUX, A. et al. Machine learning with big data: Challenges and approaches. *IEEE Access*, v. 5, p. 7776–7797, 2017.

[13] ARBEX, G. V. et al. Iot ddos detection based on stream learning. In: *2021 12th International Conference on Network of the Future (NoF)*. [S.l.: s.n.], 2021. p. 1–8.

[14] AZUMAH, S. W. et al. A deep lstm based approach for intrusion detection iot devices network in smart home. In: *2021 IEEE 7th World Forum on Internet of Things (WF-IoT)*. [S.l.: s.n.], 2021. p. 836–841.

[15] NAKAGAWA, F. H. Y.; JUNIOR, S. B.; ZARPELãO, B. B. Attack detection in smart home iot networks using clustream and page-hinkley test. In: *2021 IEEE Latin-American Conference on Communications (LATINCOM)*. [S.l.: s.n.], 2021. p. 1–6.

[16] ASHTON, K. et al. That 'internet of things' thing. *RFID journal*, Hauppauge, New York, v. 22, n. 7, p. 97–114, 2009.

[17] ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer Networks*, v. 54, n. 15, p. 2787–2805, 2010. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128610001568>.

[18] ROMAN, R. et al. Key management systems for sensor networks in the context of the internet of things. *Computers & Electrical Engineering*, Elsevier, v. 37, n. 2, p. 147–159, 2011.

[19] VERMESAN, O.; FRIESS, P. *Internet of things applications-from research and innovation to market deployment*. [S.l.]: Taylor & Francis, 2014.

[20] FERNÁNDEZ-CARAMÉS, T. M.; FRAGA-LAMAS, P. A review on the use of blockchain for the internet of things. *Ieee Access*, IEEE, v. 6, p. 32979–33001, 2018.

[21] OWASP Foundation. *OWASP IoT Top 10 2018*. 2018. Disponível em: <https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf>.

[22] LAZAREVIC, A.; KUMAR, V.; SRIVASTAVA, J. Intrusion detection: A survey. In: _____. [S.l.: s.n.], 2005. v. 5, p. 19–78. ISBN 0-387-24226-0.

[23] ZARPELãO, B. B. et al. A survey of intrusion detection in internet of things. *Journal of Network and Computer Applications*, v. 84, p. 25–37, 2017. ISSN 1084-8045. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1084804517300802>.

[24] AXELSSON, S. Intrusion detection systems: A survey and taxonomy. Citeseer, 2000.

[25] HAN, J.; KAMBER, M.; PEI, J. Data mining concepts and techniques third edition. *University of Illinois at Urbana-Champaign Micheline Kamber Jian Pei Simon Fraser University*, 2012.

[26] FAYYAD, U.; PIATETSKY-SHAPIRO, G.; SMYTH, P. From data mining to knowledge discovery in databases. *AI magazine*, v. 17, n. 3, p. 37–37, 1996.

[27] ADEWOLE, K. S. et al. Empirical analysis of data streaming and batch learning models for network intrusion detection. *Electronics*, MDPI, v. 11, n. 19, p. 3109, 2022.

[28] HULTEN, G.; SPENCER, L.; DOMINGOS, P. Mining time-changing data streams. In: . New York, NY, USA: Association for Computing Machinery, 2001. ISBN 158113391X. Disponível em: <https://doi.org/10.1145/502512.502529>.

[29] GABER, M. M.; ZASLAVSKY, A.; KRISHNASWAMY, S. Mining data streams: A review. Association for Computing Machinery, New York, NY, USA, v. 34, n. 2, 2005. ISSN 0163-5808. Disponível em: <https://doi.org/10.1145/1083784.1083789>.

[30] LESKOVEC ANAND RAJARAMAN, J. U. J. *Mining of Massive Datasets*. [S.l.]: Cambridge University Press, 2021.

[31] DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. [S.l.: s.n.], 2000. p. 71–80.

[32] BISHOP, C. M.; NASRABADI, N. M. *Pattern recognition and machine learning*. [S.l.]: Springer, 2006. v. 4.

[33] HASTIE, T. et al. *The elements of statistical learning: data mining, inference, and prediction*. [S.l.]: Springer, 2009. v. 2.

[34] ENGELEN, J. E. V.; HOOS, H. H. A survey on semi-supervised learning. *Machine learning*, Springer, v. 109, n. 2, p. 373–440, 2020.

[35] SCHöLKOPF, B. et al. Support vector method for novelty detection. In: SOLLA, S.; LEEN, T.; MüLLER, K. (Ed.). *Advances in Neural Information Processing Systems*. MIT Press, 1999. v. 12. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/1999/file/8725fb777f25776ffa9076e44fcfd776-Paper.pdf>.

[36] VAPNIK, V. *The nature of statistical learning theory*. [S.l.]: Springer science & business media, 1999.

[37] MAMMONE, A.; TURCHI, M.; CRISTIANINI, N. Support vector machines. *WIREs Computational Statistics*, v. 1, n. 3, p. 283–289, 2009. Disponível em: <https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wics.49>.

[38] HE, H.; GARCIA, E. A. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, v. 21, n. 9, p. 1263–1284, 2009.

[39] BEZERRA, V. H. Botnet detection in internet of things devices using one-class classification. 2019.

[40] VU, L. et al. Deep transfer learning for iot attack detection. *IEEE Access*, IEEE, v. 8, p. 107335–107344, 2020.

[41] HASAN, M. et al. Attack and anomaly detection in iot sensors in iot sites using machine learning approaches. *Internet of Things*, Elsevier, v. 7, p. 100059, 2019.

[42] MOTHUKURI, V. et al. Federated-learning-based anomaly detection for iot security attacks. *IEEE Internet of Things Journal*, IEEE, v. 9, n. 4, p. 2545–2554, 2021.

[43] BHUNIA, S. S.; GURUSAMY, M. Dynamic attack detection and mitigation in iot using sdn. In: IEEE. *2017 27th International telecommunication networks and applications conference (ITNAC)*. [S.l.], 2017. p. 1–6.

[44] RATHORE, S.; PARK, J. H. Semi-supervised learning based distributed attack detection framework for iot. *Applied Soft Computing*, Elsevier, v. 72, p. 79–89, 2018.

[45] JHA, J.; RAGHA, L. Intrusion detection system using support vector machine. *International Journal of Applied Information Systems (IJAIS)*, v. 3, p. 25–30, 2013.

[46] PARVEEN, P. et al. Supervised learning for insider threat detection using stream mining. In: IEEE. *2011 IEEE 23rd international conference on tools with artificial intelligence*. [S.l.], 2011. p. 1032–1039.

[47] KRAWCZYK, B.; PFAHRINGER, B.; WOŹNIAK, M. Combining active learning with concept drift detection for data stream mining. In: IEEE. *2018 IEEE International Conference on Big Data (Big Data)*. [S.l.], 2018. p. 2239–2244.

[48] WINTER, P.; HERMANN, E.; ZEILINGER, M. Inductive intrusion detection in flow-based network data using one-class support vector machines. In: IEEE. *2011 4th IFIP international conference on new technologies, mobility and security*. [S.l.], 2011. p. 1–5.

[49] MOUSTAFA, N. *ToN_IoT Datasets*. 2021. Accessed: 2024-05-24. Disponível em: <https://research.unsw.edu.au/projects/toniot-datasets>.

[50] Canadian Institute for Cybersecurity. *IoT Dataset 2023*. 2023. Accessed: 2024-05-24. Disponível em: <https://www.unb.ca/cic/datasets/iotdataset-2023.html>.

[51] NETO, E. C. P. et al. Ciciot2023: A real-time dataset and benchmark for large-scale attacks in iot environment. *Sensors*, v. 23, n. 13, 2023. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/23/13/5941>.

[52] nfstream Contributors. *nfstream Documentation*. 2024. Disponível em: <https://www.nfstream.org/docs/api>.

[53] AOUINI, Z.; PEKAR, A. Nfstream: A flexible network data analysis framework. *Computer Networks*, v. 204, p. 108719, 2022. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128621005739>.

[54] scikit-learn Contributors. *scikit-learn Documentation: RandomForestClassifier*. 2024. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>.

[55] River Contributors. *River: Hoeffding Tree Classifier*. 2024. Disponível em: <https://riverml.xyz/dev/api/tree/HoeffdingTreeClassifier/>.

[56] scikit-learn Contributors. *scikit-learn Documentation: OneClassSVM*. 2024. Acessado em: 24 de abril de 2024. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html>.

[57] River Contributors. *River: OneClass SVM*. 2024. Disponível em: <https://riverml.xyz/dev/api/anomaly/OneClassSVM/>.

A RESULTADOS RANDOM FOREST - TON_IOT

Tabela 6 – Resultados testes executados para o conjunto de dados TON_IoT

Ataque	**F1 Score**	**Precision**	**Recall**
DoS	0,966	0,967	0,966
	0,966	0,965	0,968
	0,969	0,969	0,968
	0,968	0,968	0,967
	0,967	0,967	0,967
	0,967	0,967	0,967
	0,966	0,966	0,967
	0,967	0,968	0,966
	0,968	0,968	0,969
	0,967	0,966	0,969
Injection	0,999	1,000	0,999
	0,999	1,000	0,999
	0,999	1,000	0,999
	0,999	1,000	0,999
	0,999	1,000	0,999
	0,999	1,000	0,999
	0,999	1,000	0,999
	0,999	1,000	0,999
	0,999	1,000	0,999
	0,999	1,000	0,999
Backdoor Malware	0,993	1,000	0,986
	0,993	1,000	0,987
	0,994	1,000	0,988
	0,993	1,000	0,986
	0,993	1,000	0,987
	0,993	1,000	0,987
	0,993	1,000	0,987
	0,993	1,000	0,986
	0,994	1,000	0,988
	0,993	1,000	0,986

B RESULTADOS - RANDOM FOREST - CIC IOT 2023

Tabela 7 – Resultados testes executados para o conjunto de dados CIC IoT

Ataque	**F1 Score**	**Precision**	**Recall**
DoS	0,984	0,992	0,975
	0,984	0,992	0,975
	0,985	0,995	0,975
	0,984	0,992	0,975
	0,985	0,995	0,975
	0,986	0,992	0,980
	0,985	0,995	0,975
	0,984	0,992	0,975
	0,984	0,992	0,975
	0,985	0,995	0,975
Injection	0,848	0,747	0,980
	0,845	0,746	0,973
	0,848	0,747	0,980
	0,849	0,745	0,986
	0,839	0,730	0,986
	0,841	0,734	0,986
	0,834	0,733	0,966
	0,840	0,738	0,973
	0,841	0,745	0,966
	0,848	0,747	0,980
Backdoor Malware	0,478	0,320	0,941
	0,515	0,347	1,000
	0,507	0,340	1,000
	0,493	0,327	1,000
	0,515	0,347	1,000
	0,500	0,340	0,941
	0,442	0,283	1,000
	0,515	0,347	1,000
	0,523	0,354	1,000
	0,459	0,298	1,000

C RESULTADOS - OCSVM *BATCH* - TON_IOT

Tabela 8 – Resultados testes executados para o conjunto de dados TON_IoT

Ataque	**F1 Score**	**Precision**	**Recall**
DoS	0,832	0,733	0,964
	0,833	0,735	0,962
	0,832	0,732	0,963
	0,832	0,734	0,961
	0,833	0,735	0,963
	0,834	0,737	0,961
	0,831	0,731	0,962
	0,832	0,734	0,960
	0,832	0,732	0,965
	0,833	0,735	0,961
Injection	0,986	0,974	0,999
	0,987	0,974	0,999
	0,986	0,974	0,999
	0,986	0,974	0,999
	0,987	0,974	0,999
	0,987	0,974	0,999
	0,987	0,974	0,999
	0,987	0,975	0,999
	0,987	0,974	0,999
	0,987	0,975	0,999
Backdoor Malware	0,897	0,816	0,995
	0,895	0,814	0,995
	0,897	0,816	0,995
	0,896	0,814	0,995
	0,894	0,811	0,995
	0,895	0,813	0,995
	0,894	0,812	0,995
	0,895	0,814	0,995
	0,897	0,817	0,995
	0,895	0,813	0,995

D RESULTADOS - OCSVM *BATCH* - CIC IOT 2023

Tabela 9 – Resultados testes executados para o conjunto de dados CIC IoT

Ataque	**F1 Score**	**Precision**	**Recall**
DoS	0,831	0,712	0,998
	0,840	0,724	0,998
	0,826	0,704	0,998
	0,837	0,720	0,998
	0,835	0,717	1,000
	0,837	0,720	1,000
	0,843	0,728	1,000
	0,849	0,738	1,000
	0,847	0,735	1,000
	0,844	0,730	1,000
Injection	0,705	0,629	0,801
	0,734	0,690	0,784
	0,711	0,647	0,790
	0,742	0,704	0,784
	0,729	0,658	0,818
	0,704	0,622	0,813
	0,693	0,618	0,790
	0,707	0,665	0,756
	0,731	0,661	0,818
	0,769	0,771	0,767
Backdoor Malware	0,624	0,564	0,698
	0,571	0,484	0,698
	0,629	0,571	0,698
	0,607	0,537	0,698
	0,644	0,566	0,746
	0,620	0,557	0,698
	0,577	0,484	0,714
	0,611	0,543	0,698
	0,624	0,564	0,698
	0,571	0,484	0,698