UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ
CAMPUS DE FOZ DO IGUAÇU
PROGRAMA DE PÓS-GRADUAÇÃO EM
ENGENHARIA ELÉTRICA E COMPUTAÇÃO

DISSERTAÇÃO DE MESTRADO

DETECÇÃO DE INTRUSÃO APLICANDO RANDOM FORESTS EM AMBIENTE FEDERATED LEARNING

MARCIO FERNANDES DA COSTA

FOZ DO IGUAÇU
2025

Marcio Fernandes da Costa

Detecção de Intrusão Aplicando Random Forests em Ambiente Federated Learning

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica e Computação. Área de concentração: Sistemas Elétricos e Computação.

Orientador: Renato Bobsin Machado

Foz do Iguaçu
2025

Ficha de identificação da obra elaborada através do Formulário de Geração Automática do Sistema de Bibliotecas da Unioeste.

Costa, Marcio Fernandes da
Detecção de Intrusão Aplicando Random Forests em Ambiente Federated Learning / Marcio Fernandes da Costa; orientador Renato Bobsin Machado. -- Foz do Iguaçu, 2025.
138 p.

Dissertação (Mestrado Profissional Campus de Foz do Iguaçu) -- Universidade Estadual do Oeste do Paraná, Centro de Engenharias e Ciências Exatas, Programa de Pós-Graduação em Engenharia Elétrica e Computação, 2025.

1. Aprendizado Federado. 2. Dispositivos IoT. 3. Detecção de Intrusão. 4. Florestas Aleatórias. I. Machado, Renato Bobsin, orient. II. Título.

Detecção de Intrusão Aplicando Random Forests em Ambiente Federated Learning

Marcio Fernandes da Costa

Esta Dissertação de Mestrado foi apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Computação e aprovada pela Banca Examinadora assim constituída:

Prof. Dr. **Renato Bobsin Machado** - (Orientador)
Universidade Estadual do Oeste do Paraná – UNIOESTE

Prof. Dr. **Joylan Nunes Maciel**
Universidade Federal da Integração Latino Americana – UNILA

Prof. Dr. **Willian Zalewski**
Universidade Estadual do Oeste do Paraná – UNIOESTE

Data da defesa: 17/07/2025

Resumo

A crescente expansão dos dispositivos da Internet das Coisas tem transformado diversos setores, impulsionando ganhos em eficiência e automação. No entanto, essa evolução também amplia a vulnerabilidade a ataques cibernéticos, gerando riscos financeiros e danos à reputação. Diante desse cenário de crescente preocupação com a segurança, métodos eficazes de detecção de intrusões tornam-se indispensáveis. O aprendizado federado surge como uma abordagem promissora para enfrentar esse desafio, pois possibilita o treinamento de modelos globais sem comprometer a privacidade dos dados. Com base nessa perspectiva, este trabalho propõe um método de detecção de intrusão que utiliza florestas aleatórias como modelo de classificação, aplicado ao conjunto de dados IoTID20 em um ambiente federado. A escolha das florestas aleatórias justifica-se por sua robustez na classificação de padrões, tornando-as adequadas para cenários de segurança cibernética. Para otimizar o desempenho do modelo, foram realizadas etapas de pré-processamento no conjunto de dados, proporcionando maior eficiência no treinamento. Para mitigar o desbalanceamento dos dados, aplicou-se a técnica de *oversampling*, contribuindo para a melhoria dos resultados e, adicionalmente, a utilização de um *ensemble* para a seleção dos atributos permitiu a criação de vários conjuntos de dados, possibilitando a avaliação do desempenho do modelo em diferentes cenários. Os experimentos demonstraram que o modelo federado apresentou desempenho superior em relação ao modelo centralizado, mesmo na ausência da otimização de hiperparâmetros em ambos modelos. O modelo federado obteve médias de acurácia de 98,51%, F1-score de 98,60% e Recall de 98,43%, enquanto o modelo centralizado alcançou 97,49% de acurácia, 97,61% de F1-score e 96,67% de Recall, posicionando estes resultados de forma competitiva em relação a outros trabalhos na literatura que abordam a detecção de intrusão em ambientes federados.

Palavras-chave: Aprendizado Federado. Dispositivos IoT. Detecção de Intrusão. Florestas aleatórias.

Abstract

The growing expansion of Internet of Things devices has transformed several sectors, driving gains in efficiency and automation. However, this evolution also increases vulnerability to cyber attacks, generating financial risks and damage to reputation. Given this scenario of growing concern about security, effective intrusion detection methods become indispensable. Federated learning emerges as a promising approach to address this challenge, as it enables the training of global models without compromising data privacy. Based on this perspective, this work proposes an intrusion detection method that uses random forests as a classification model, applied to the IoTID20 dataset in a federated environment. The choice of random forests is justified by their robustness in pattern classification, making them suitable for cybersecurity scenarios. To optimize the model's performance, pre-processing steps were performed on the dataset, providing greater efficiency in training. To mitigate data imbalance, the oversampling technique was applied, contributing to improved results. Additionally, the use of an ensemble for attribute selection allowed the creation of several datasets, enabling the evaluation of model performance in different scenarios. The experiments demonstrated that the federated model performed better than the centralized model, even in the absence of hyperparameter optimization in both models. The federated model achieved average accuracy of 98.51%, F1-score of 98.60%, and Recall of 98.43%, while the centralized model achieved 97.49% accuracy, 97.61% F1-score, and 96.67% Recall, positioning these results competitively in relation to other works in the literature that address intrusion detection in federated environments.

Keywords: Federated Learning. IoT Devices. Intrusion Detection. Random Forests.

Sozinhos vamos mais rápido.
Juntos vamos mais longe.

Agradecimentos

Gostaria de expressar meus sinceros agradecimentos a todos que, direta ou indiretamente, contribuíram para a realização deste trabalho de mestrado. Esta caminhada foi marcada por aprendizados, desafios e conquistas, e cada etapa só foi possível graças ao apoio generoso de pessoas que, de diferentes formas, fizeram parte dessa jornada.

Em primeiro lugar, agradeço ao meu orientador, Prof. Dr. Renato Bobsin Machado, por sua orientação, paciência e constante apoio ao longo de todo o processo. Seus conselhos, críticas construtivas e incentivo permanente foram fundamentais para o desenvolvimento desta pesquisa e para o meu crescimento acadêmico.

Estendo meus sinceros agradecimentos aos Prof. Dr. Joylan Nunes Maciel e Prof. Dr. Willian Zalewski, pela generosidade em participar da banca examinadora. Suas observações contribuíram de forma significativa para o aprimoramento e a qualidade final deste estudo.

Agradeço igualmente à Universidade Estadual do Oeste do Paraná – UNIOESTE, por oferecer um ensino público, gratuito e de qualidade, e por manter um ambiente acadêmico pautado na ética, na ciência e na formação crítica. A infraestrutura disponibilizada, o corpo docente qualificado e o incentivo à pesquisa foram essenciais para a realização deste mestrado.

Manifesto minha profunda gratidão à minha família, pela presença constante, pelo apoio moral, amor incondicional e encorajamento ao longo de toda a trajetória acadêmica. Sem o suporte, a compreensão e a paciência de vocês, este trabalho não teria sido possível.

Por fim, deixo meu agradecimento a todos que, de alguma forma, contribuíram para esta jornada com palavras de incentivo, colaborações técnicas ou apoio nos momentos em que os caminhos pareciam incertos. Cada gesto teve uma importância única e inesquecível neste processo.

A todos, o meu mais sincero obrigado.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Abreviaturas

AAL	Ambient assisted living
AC	Aprendizado centralizado
AD	Árvore de decisão
ADASYN	Adaptive synthetic sampling
AF	Aprendizado federado
AM	Aprendizado de máquina
ANOVA	Analysis of variance
APT	Advanced persistent threat
CatBoost	Categorial boosting
CNN	Convolutional neural network
CS	Chi-squared
DDoS	Distributed denial-of-service
DNN	Deep neural network
DoS	Denial-of-service
FA	Florestas aleatórias
GR	Gain ratio
GRU	Gated recurrent units
HIDS	Host intrusion detection system
IA	Inteligência artificial
IDS	Intrusion detection system
IG	Info gain
IID	Independent and identically distributed
IOT	Internet of things
IP	Internet protocol
KDD	Knowledge discovery in databases
KNN	K-nearest neighbors
LaPSeC	Laboratório de Pesquisa em Segurança Computacional
LightGBM	Light gradient-boosting machine
LSTM	Long short-term memory

MLP	Multi layer perceptron
NBA	Network behavior analysis
NIDS	Network intrusion detection system
NIST	National Institute of Standards and Technology
non-IID	non-Independent and identically distributed
OSINT	Open source intelligence
P2P	Peer-to-peer
PGEEC	Programa de Pós-Graduação em Engenharia Elétrica e Computação
RFC	Request for comments
RNA	Rede neural artificial
RNN	Recurrent neural network
ROS	Random oversampling
RUS	Random undersampling
SIEM	Security information and event management
SMOTE	Synthetic minority oversampling technique
SNMP	Simple network management protocol
TCP	Transmission control protocol
UDP	User Datagram Protocol
UKC	Unified kill chain
XGBoost	Extreme gradient boosting

Lista de Símbolos

σ	Desvio padrão
μ	Média
$\Uparrow$	Indica o maior resultado do teste
$\Downarrow$	Indica o menor resultado do teste

Capítulo 1

Introdução

Nos últimos anos, o número de dispositivos *Internet of Things* (IoT), conectados pelo mundo, tem crescido fortemente. Segundo estimativas, em 2024 o total de equipamentos atingiu cerca de 18,8 bilhões, com possibilidade de chegar a 21,5 bilhões no ano de 2025, representando um crescimento de aproximadamente 15% (Iot-Analytics, 2024). Essas projeções podem ser vistas na Figura 1.1.

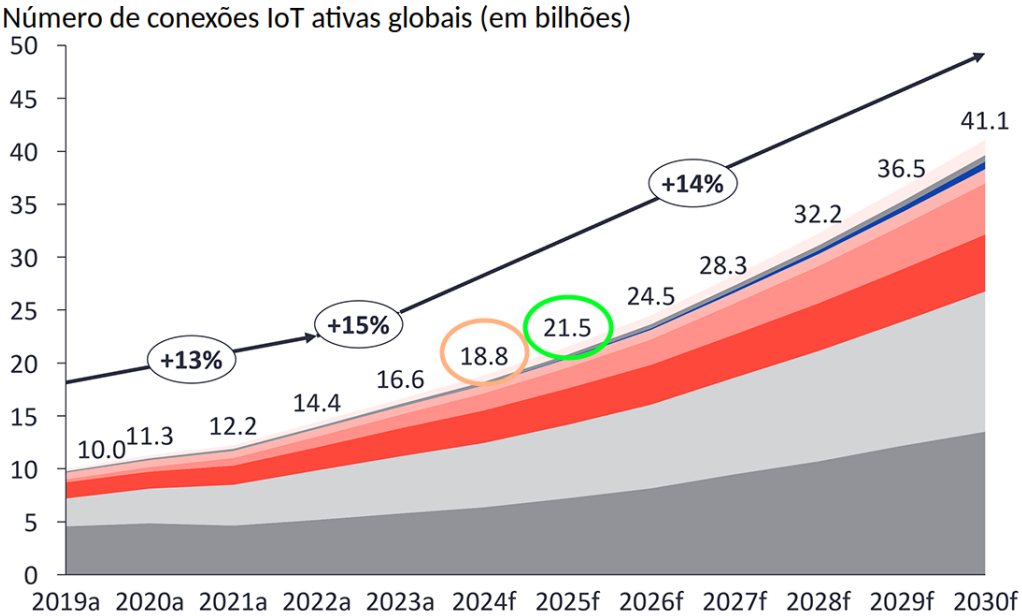

Figura 1.1: Número de dispositivos IoT conectados.
Fonte: (Iot-Analytics, 2024).

Essa classe de dispositivos pode ser descrita como um sistema interligado, com vários graus de capacidades de processamento e de armazenamento, e que compartilham a propriedade de se comunicar através da Internet (Hussein, 2019). Alguns exemplos desses equipamentos são *smartwatches*, câmeras de segurança, carros autônomos e sensores de *healthcare*, entre outros.

O rápido aumento de dispositivos IoT interconectados cria várias possibilidades de utilização, sejam elas para usuários comuns ou para grandes empresas. Entretanto, também introduzem riscos significativos de segurança cibernética. Segundo (Williams, Dutta, Daoud and Bayoumi, 2022), essas vulnerabilidades podem ser causadas por restrições relacionadas ao *hardware* (poder computacional, armazenamento, energia e memória), ao *software* (*software* embarcado) por vezes mal testado e a comunicação (conexões lentas ou intermitentes) devido

ao uso de rádios de baixa potência.

Em pesquisa divulgada pela *International Business Machines* - (IBM, 2024), estima-se que o custo médio de uma violação de dados em 2024 é de US$ 4,88 milhões de dólares. Esses valores envolvem danos à reputação, aquisições, contratações e o tempo de indisponibilidade dos serviços, entre outros custos. A Figura 1.2 exibe essas estimativas entre os anos de 2018 e 2024.

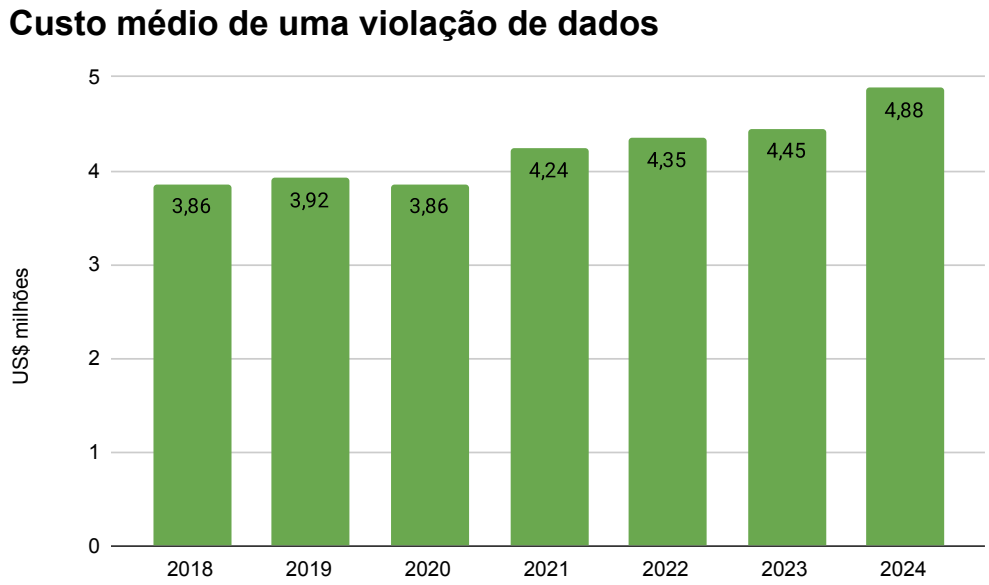

Figura 1.2: Custos de uma violação de dados 2018 - 2024.
Fonte: (IBM, 2024).

Considerando esse cenário, uma importante ferramenta em segurança computacional são os Sistemas de Detecção de Intrusão (IDS), capazes de monitorar e analisar o tráfego em uma rede de dados ou *host*, na busca por atividades indicativas de intrusões, as quais podem comprometer dispositivos, serviços ou a própria rede, emitindo alertas para os operadores realizarem o tratamento adequado (Scarfone and Mell, 2007).

Os IDS têm seu funcionamento principal baseado em assinaturas, que são os padrões ou características exclusivas que identificam um tipo específico de ameaça ou comportamento. Apesar de ser uma boa forma de identificar e prevenir vários tipos de *malwares*, vírus e comportamentos de risco, tem o inconveniente de dependerem de atualizações das bases de conhecimentos para a detecção de novas ameaças (Scarfone and Mell, 2007).

Diante dessas limitações, especialmente em um contexto de ameaças cada vez mais sofisticadas e dinâmicas, surgem alternativas baseadas em técnicas mais flexíveis e autônomas. Nesse sentido, destaca-se o uso da Inteligência Artificial (IA) (Russell and Norvig, 2010), que oferece uma abordagem promissora para a detecção de intrusões ao permitir a identificação de comportamentos anômalos, mesmo que não correspondam a assinaturas previamente conhecidas.

Nos últimos tempos, houve uma evolução significativa na área de IA, que passou a utilizar técnicas de *machine learning* (Alpaydin, 2021) e de *deep learning* (Goodfellow, Bengio and Courville, 2016), capazes de aprender e se adaptar a partir da experiência sem serem explicitamente programadas (Russell and Norvig, 2010). Essas técnicas constituem importantes alternativas para explorar soluções em segurança computacional, e sobretudo no âmbito da construção de métodos de IDS.

Normalmente, essas técnicas de IA usam de processamento centralizado para o treinamento de um modelo computacional, o que torna necessário o envio de todos os dados para tratamento pela entidade ou servidor principal. De acordo com (McMahan, Moore, Ramage, Hampson and y Arcas, 2016), ocorre que tratar grandes volumes de dados, em um ponto centralizado, pode ser caro do ponto de vista computacional, ou ainda, pode criar riscos de segurança envolvidos na transferência dos dados entre as partes.

Segundo (Ferrag, Friha, Maglaras, Janicke and Shu, 2021), um dos principais desafios enfrentados é a preservação da privacidade dos dados. A transferência de informações sensíveis dos usuários para servidores centrais pode infringir diretrizes de proteção de dados, como as estabelecidas por legislações. Além disso, outro obstáculo relevante é a latência: o tempo necessário para enviar os dados ao servidor, processá-los e retornar os resultados pode ser elevado, o que compromete o desempenho de aplicações que demandam respostas em tempo real.

Em 2016, foi apresentada uma técnica de treinamento distribuído chamada de *Federated Learning* (McMahan et al., 2016). Essa abordagem cria um modelo base, por exemplo, uma rede neural (Aggarwal, 2023) ou uma floresta aleatória (Breiman, 2001), a partir do processamento centralizado e o compartilha com os dispositivos clientes, os quais utilizarão seus dados locais e de sua capacidade de processamento para o treinamento e avaliação do modelo. Como apenas os eventos referentes aos modelos são compartilhados entre os membros, essa abordagem está alinhada com a crescente preocupação em torno da proteção da privacidade dos dados dos usuários (Kairouz, McMahan, Avent, Bellet, Bennis, Bhagoji, Bonawitz, Charles, Cormode, Cummings, D'Oliveira, Eichner, Rouayheb, Evans, Gardner, Garrett, Gascón, Ghazi, Gibbons, Gruteser, Harchaoui, He, He, Huo, Hutchinson, Hsu, Jaggi, Javidi, Joshi, Khodak, Konečný, Korolova, Koushanfar, Koyejo, Lepoint, Liu, Mittal, Mohri, Nock, Özgür, Pagh, Raykova, Qi, Ramage, Raskar, Song, Song, Stich, Sun, Suresh, Tramèr, Vepakomma, Wang, Xiong, Xu, Yang, Yu, Yu and Zhao, 2021). Uma comparação entre o treinamento centralizado e o federado pode ser visto na Figura 1.3.

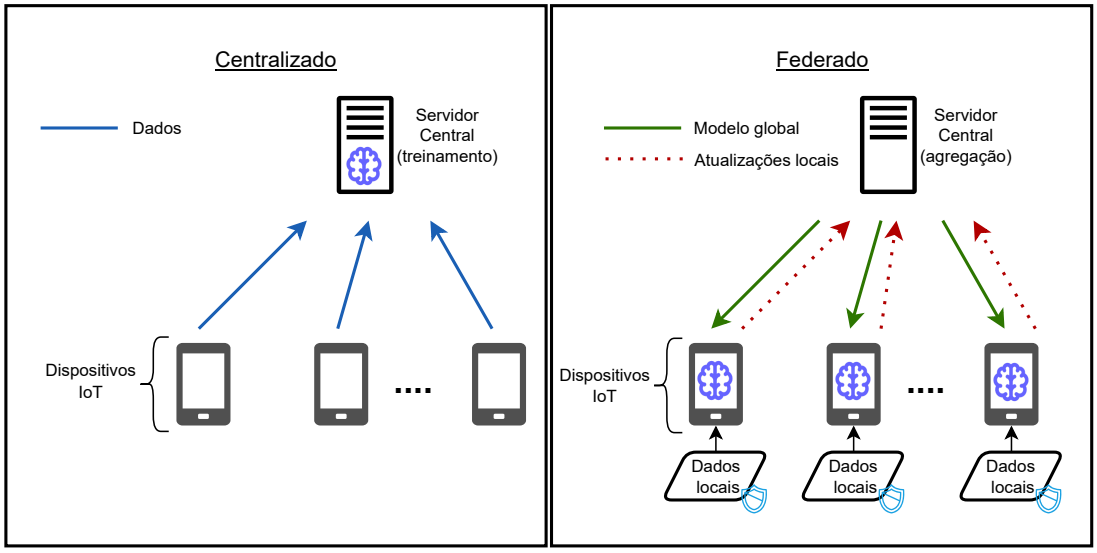

Figura 1.3: Comparação entre o treinamento centralizado e o federado.
Fonte: O autor.

Considerando que as técnicas tradicionais de IDS frequentemente dependem da atualização das bases de eventos computacionais para detectar novas ameaças, o que pode levar tempo e resultar em uma eficiência reduzida diante do crescente número de ataques e invasões, identifica-se o seguinte problema: considerando os dispêndios ocasionados por uma violação de dados e a otimização da detecção de intrusão por meio de IDS, quais técnicas de IA poderiam ser utilizadas para melhorar esses processos?

1.1 Linha de pesquisa em segurança computacional

O Programa de Pós-Graduação em Engenharia Elétrica e Computação (PGEEC) destaca-se pela linha de pesquisa em "Controle, Automação e Inteligência Computacional". Dentro dessa área, o Laboratório de Pesquisa em Segurança Computacional (LaPSeC) dedica-se ao estudo e desenvolvimento de novos métodos para segurança computacional e as suas linhas de pesquisa incluem:

- Prevenção e detecção de intrusão.
- Métodos criptográficos.
- Inteligência artificial aplicada à segurança de redes e sistemas.
- Segurança de dispositivos móveis e ubíquos.
- Forense computacional e auditoria de segurança.

Dentre as pesquisas publicadas pelo LaPSeC, algumas se destacam na área de segurança computacional e são exibidas na Tabela 1.1:

Tabela 1.1: Pesquisas publicadas pelo LaPSeC do PGEEC.

Autor	Título
(de Souza, 2018)	Método híbrido de detecção de intrusão aplicando inteligência artificial.
(de Oliveira, 2020)	Combinação autoajustada de modelos de aprendizagem de máquina com otimização de hiperparâmetros para detecção de intrusos em redes de computadores.
(Alves, 2021)	Detecção de intrusão em nós sensores de redes de sensores sem fio.
(Valencio, 2021)	Abordagem de detecção de intrusão em ambientes *fog computing* e *internet of things*.
(Ribera, 2024)	Detecção de intrusão em dispositivos de internet das coisas com uma abordagem de aprendizado federado.
(Silva, 2024)	Abordagens para o problema do desbalanceamento em detecção de intrusão - um estudo de caso aplicando CIC-IDS2018.

Fonte: Biblioteca Digital de Teses e Dissertações - https://tede.unioeste.br/.

1.2 Proposta

O presente trabalho se enquadra na linha de pesquisa em segurança computacional do PGEEC, constituindo uma abordagem para detecção de intrusões baseada em florestas aleatórias, utilizando um ambiente de aprendizado federado, avaliando o desempenho de classificação do modelo em diferentes cenários e volumes de dados. A hipótese baseia-se no fato de que florestas aleatórias podem alcançar resultados tão eficazes quanto outros modelos de aprendizado de máquina, como redes neurais, utilizados na classificação de intrusões em um ambiente de aprendizado federado. Além disso, como objetivos específicos estão:

- Implementar um modelo de detecção de intrusões baseado em florestas aleatórias no contexto de aprendizado federado, utilizando dispositivos IoT como clientes locais para o treinamento distribuído, verificando a viabilidade técnica da integração entre florestas aleatórias e aprendizado federado em um cenário de dispositivos IoT.
- Comparar, baseado na literatura associada, o desempenho do modelo de florestas aleatórias com outros modelos de aprendizado de máquina em termos de métricas de classificação, como acurácia, no ambiente de aprendizado federado, analisando se o modelo baseado em florestas aleatórias é competitivo a outras abordagens da literatura.
- Avaliar o impacto de diferentes configurações de dados, como distribuições IID (*Independent and Identically Distributed*) e non-IID (*non-Independent and Identically Distributed*) sobre o desempenho do modelo de florestas aleatórias no cenário federado, de modo a compreender como a heterogeneidade dos dados afeta o modelo.

A escolha pelo uso de florestas aleatórias é justificada com base em estudos comparativos que demonstraram seu alto desempenho na detecção de ataques:

O experimento foi realizado com diferentes algoritmos de classificação para o conjunto de dados, com e sem redução de atributos, e ficou evidente que as florestas aleatórias apresentam uma alta acurácia nos testes em comparação com todos os outros algoritmos em ambos os casos (Revathi and Malathi, 2013).

Este artigo trata do algoritmo de florestas aleatórias para detectar quatro tipos de ataques como DOS, probe, U2R e R2L. A abordagem proposta foi avaliada usando o conjunto de dados NSL-KDD. O resultado prova que a precisão, a taxa de detecção e o coeficiente de correlação de Matthews para os quatro tipos de ataques aumentam com o método proposto (Farnaaz and Jabbar, 2016).

Foram utilizados quatro diferentes conjuntos de dados de referência para IDS (KDD99, NSL-KDD, UNSW-NB15 e CIC-IDS2017), para avaliar o desempenho dos algoritmos de aprendizado de máquina selecionados tanto para detecção de intrusões quanto para classificação de ataques. Os resultados demonstraram que as florestas aleatórias é o algoritmo mais adequado em termos de acurácia do modelo e tempo de execução (Leon, Markovic and Punnekkat, 2022).

Neste trabalho foram propostos modelos robustos de aprendizado de máquina e aprendizado profundo para a detecção de intrusões e classificação de tipos de ataques. Os métodos foram testados no conjunto de dados UNSW-NB15 e as florestas aleatórias chegaram a uma acurácia de 98,96% (Dhanya, Vajipayajula, Srinivasan, Tibrewal, Kumar and Kumar, 2023).

1.3 Estrutura do trabalho

No Capítulo 2 são apresentados conceitos sobre segurança computacional e cibernética, detecção de intrusão, aprendizado de máquina e avaliação de modelos. O Capítulo 3 oferece uma revisão da literatura disponível. O Capítulo 4 descreve os materiais e métodos empregados na pesquisa. No Capítulo 5, realiza-se a análise dos resultados e, por fim, o Capítulo 6 apresenta as conclusões e contribuições obtidas ao longo do estudo.

Capítulo 2

Fundamentação Teórica

2.1 Considerações iniciais

Este capítulo destina-se a abordar temas que fundamentam o método proposto. Para isso, apresentam-se conceitos e fundamentos relacionados às áreas de segurança computacional e cibernética, aos sistemas de detecção de intrusão e à inteligência artificial, todos direcionados às técnicas exploradas neste trabalho.

2.2 Segurança computacional e cibernética

Os conceitos acerca do tema da segurança computacional, que serão apresentados ao longo desta seção, foram elaborados de acordo com (Stallings, 2017).

Sistemas computacionais e outros dispositivos conectados à Internet enfrentam uma ampla gama de ataques devido à exploração de diversas vulnerabilidades, abrangendo desde tentativas de quebra de controles de acesso por usuários internos até complexos ataques utilizando recursos distribuídos.

A segurança de computadores é "a proteção oferecida a um sistema de informação automatizado para alcançar os objetivos de preservar a integridade, a disponibilidade e a confidencialidade dos recursos do sistema de informação (incluindo *hardware*, *software*, *firmware*, informações/dados e telecomunicações)".

Desta definição, o mesmo autor destaca os três objetivos principais que estão no cerne da segurança computacional:

- Confidencialidade: garantir que informações não estejam disponíveis nem sejam reveladas a indivíduos não autorizados (confidencialidade de dados). Além disso, assegura que os envolvidos controlem quais informações relacionadas a eles podem ser acessadas, armazenadas e distribuídas (privacidade).
- Integridade: assegurar que as informações e programas sejam modificados apenas de maneira autorizada (integridade dos dados). Também garante que um sistema execute suas funcionalidades de forma íntegra e livre de manipulações (integridade do sistema).
- Disponibilidade: garantir que os sistemas operem prontamente e não fiquem indisponíveis

para usuários autorizados.

Ainda no campo da segurança percebe-se que conceitos adicionais são necessários para apresentar um quadro completo. A seguir definem-se duas importantes premissas:

- Autenticidade: propriedade de ser genuíno e capaz de ser verificado e confiável. Isso significa verificar que os usuários são quem dizem ser e, além disso, que cada entrada no sistema vem de uma fonte confiável.
- Responsabilização: a meta de segurança que gera o requisito para que ações de uma entidade sejam atribuídas exclusivamente a ela. Isso provê irretratabilidade, isolamento de falhas, detecção e prevenção de intrusão, além de ações legais. Os sistemas precisam manter registros de suas atividades a fim de permitir posterior análise forense de modo a rastrear as violações de segurança.

Esses pilares formam a base da segurança de sistemas computacionais e de dispositivos, que de algum modo estão conectados à Internet, sendo essencial que recebam uma maior atenção para a proteção eficaz contra ameaças e ataques.

Alguns conceitos extraídos da RFC (*Request for comments*) 4949 - Glossário de Segurança na Internet [1], e que são importantes para a compreensão dos assuntos relacionados à área de segurança são:

- Vulnerabilidade: uma falha ou fraqueza no projeto, implementação ou operação e gerenciamento de um sistema que possa ser explorada para violar a política de segurança do sistema.
- Ameaça: uma potencial violação de segurança que existe quando há uma circunstância, capacidade, ação ou evento que pode violar a segurança e causar danos. Ou seja, uma ameaça é um perigo iminente que uma vulnerabilidade ou falha seja explorada por um atacante.
- Ataque: uma investida contra a segurança do sistema que deriva de uma ameaça. Ou seja, um ato deliberado de burlar serviços de segurança e violar a política de segurança de um sistema.
- Política de segurança: um objetivo, curso ou método de ação definido para orientar e determinar decisões presentes e futuras relativas à segurança em um sistema.

A segurança cibernética é uma área ampla, focada na proteção de tudo o que está conectado à Internet, incluindo dados, redes, dispositivos e sistemas. Um estudo de (Schiliro, 2023) propõe uma definição mais abrangente de segurança cibernética para os dias atuais, entendida como:

[1] https://www.rfc-editor.org/info/rfc4949. Acesso em junho/2024.

> "O conjunto e a coordenação de recursos, incluindo pessoal, infraestrutura, estruturas e processos, destinados a proteger redes e sistemas computacionais habilitados para o ciberespaço contra eventos que comprometem sua integridade e interferem nos direitos de propriedade, resultando em algum grau de perda."

Complementarmente, o *National Institute of Standards and Technology* (NIST), em seu glossário[2], define os seguintes termos:

- Segurança cibernética (*cyber security*): a capacidade de proteger ou defender o uso do ciberespaço contra ataques cibernéticos.
- Ciberespaço (*cyberspace*): a rede interdependente de infraestruturas de tecnologia da informação, que inclui a Internet, redes de telecomunicações, computadores, sistemas de informação, sistemas de controle industrial, redes e processadores, bem como controladores embarcados.
- Ataque cibernético (*cyber attack*): as ações realizadas por meio do uso de redes de computadores com o objetivo de interromper, negar, degradar ou destruir informações armazenadas em computadores e redes, ou ainda comprometer o funcionamento dessas infraestruturas.
- Ataque de dia zero (*zero day attack*): o ataque que explora uma vulnerabilidade de *hardware*, *firmware* ou *software* previamente desconhecida. Segundo (Guo, 2023), a origem do termo "dia zero"se refere justamente por ser um ataque desconhecido do público e da comunidade de segurança.

A segurança cibernética é essencialmente uma batalha de inteligência entre um criminoso que tenta encontrar falhas e o administrador que tenta corrigi-las. A vantagem que o atacante tem é que ele só precisa encontrar uma única fraqueza para explorá-la, enquanto o administrador deve encontrar e eliminar várias para alcançar um nível elevado de segurança (Stallings, 2017).

2.3 Ataques

No contexto da segurança da informação, a compreensão dos tipos de ataques e de seus respectivos funcionamentos constitui um elemento fundamental para a proteção de infraestruturas modernas. Nesse sentido, as seções a seguir têm como objetivo descrever de maneira breve alguns dos principais tipos de ataques e as suas etapas.

2.3.1 Tipos de ataques

Os tipos de ataques referem-se às diversas estratégias utilizadas por atacantes para comprometer a segurança de sistemas, redes ou dados. Esses ataques podem visar à violação da confidencialidade, integridade ou disponibilidade das informações, princípios fundamentais da segurança da informação (Li and Liu, 2021). Dentre os principais tipos de ataques estão:

[2]https://csrc.nist.gov/glossary. Acesso em abril/2025.

- *Man-in-the-Middle* (MITM): O ataque MITM, segundo (Morsy and Nashat, 2022), é um tipo de ciberataque no qual um terceiro não autorizado intercepta secretamente a comunicação entre dois *hosts*, com o objetivo de ler e/ou modificar os dados transferidos entre eles. Inicialmente, o MITM compromete a confidencialidade dos dados, ao realizar escuta clandestina das informações trocadas entre os *hosts*. Em seguida, compromete a integridade dos dados, ao interceptar e alterar os dados transmitidos. Por fim, o ataque pode afetar a disponibilidade dos dados, ao forçar um dos participantes a interromper a comunicação com o outro, modificando ou destruindo os dados transferidos.
- *Denial of Service* (DoS): O ataque de negação de serviço tem como principal objetivo tornar temporariamente indisponível um equipamento ou recurso de rede. Isso ocorre, geralmente, por meio da sobrecarga do sistema alvo, que é inundado com um grande volume de requisições, impedindo-o de processar solicitações legítimas (Gupta and Badve, 2017).

  Uma variação desse ataque é o *Distributed Denial of Service* (DDoS), que envolve múltiplos sistemas comprometidos, como uma *botnet*, rede de dispositivos comprometidos, atuando de forma coordenada para gerar tráfego massivo. Devido à sua escala, esse tipo de ataque é capaz de afetar sistemas e serviços com maior capacidade de processamento, tornando-os igualmente indisponíveis.
- Mirai: O Mirai é uma *botnet*, rede de dispositivos infectados por *malware* e controlados por cibercriminosos, composta principalmente por dispositivos IoT (Antonakakis, April, Bailey, Bernhard, Arbor, Bursztein, Cochran, Durumeric, Halderman, Arbor, Invernizzi, Kallitsis, Kumar, Lever, Ma, Mason, Menscher, Seaman, Sullivan, Thomas and Zhou, 2017). O Mirai é notável por sua rápida propagação, infectando dispositivos vulneráveis por meio da exploração de credenciais fracas e o dispositivo, uma vez comprometido, se torna parte da *botnet*, capaz de executar comandos de ataques coordenados e gerar grandes volumes de tráfego malicioso (DDoS), com o objetivo de tornar indisponíveis, dispositivos ou sistemas e serviços online, para usuários legítimos.
- *Scan*: Identificar portas abertas para determinar os serviços em execução em um dispositivo ou sistema é uma tarefa muito importante, pois qualquer porta desnecessária que permaneça aberta adiciona vulnerabilidades e, portanto, torna-se uma potencial ameaça à segurança do dispositivo ou sistema em questão. A varredura de portas é um método no qual um dispositivo é examinado quanto à presença de portas abertas dos protocolos *Transmission Control Protocol* (TCP) e *User Datagram Protocol* (UDP) (Kumar and Sudarsan, 2014).

  Essa técnica é amplamente utilizada durante auditorias de segurança, avaliações de vulnerabilidades e testes de intrusão em servidores ou dispositivos. Durante os testes, uma porta ativa para comunicação é denominada porta aberta, uma porta que está inativa é chamada de porta fechada e todas as portas cujo estado não pôde ser determinado, devido à ausência de resposta válida do equipamento em teste, são consideradas portas filtradas.

2.3.2 Etapas de um ataque

Uma vez que um ataque é compreendido e desmembrado em fases menores, permite que os defensores mapeiem contramedidas considerando cada uma dessas fases, visto que ataques cibernéticos normalmente se estendem além de explorar apenas uma vulnerabilidade em um sistema conectado à Internet. As definições desta seção foram extraídas de (Pol, 2017).

Um *framework* relacionado à segurança computacional que contribui para a compreensão das etapas usadas em ataque cibernético ou invasão é o *Unified Kill Chain* (UKC). O UKC é um modelo que descreve as fases consecutivas de um ataque, sendo que o termo "kill chain" descreve o processo ponta a ponta, ou toda a cadeia de eventos necessária para realizar um ataque bem-sucedido.

De modo geral, inicialmente o invasor obtém impressões digitais do sistema alvo para coletar informações relevantes. Esta etapa é conhecida como reconhecimento. Esse estágio não envolve nenhum contato com o sistema alvo e nenhum alerta é produzido.

Posteriormente, o sistema de destino é verificado para determinar o status de conexões e identificar serviços e protocolos em execução. Então, a verificação de vulnerabilidades é realizada para descobrir quaisquer pontos fracos ou um ponto para obter acesso.

O ataque real é executado durante o estágio de exploração e é baseado nas varreduras feitas nas etapas anteriores. Um código malicioso pode ser entregue e instalado e, neste ponto, a máquina alvo fica comprometida. Depois disso, as atividades são realizadas durante o estágio de movimento, como extrair dados ou fazer ataques distribuídos de negação de serviço (DDoS).

A Figura 2.1 apresenta as etapas do UKC, as quais serão detalhadas na sequência.

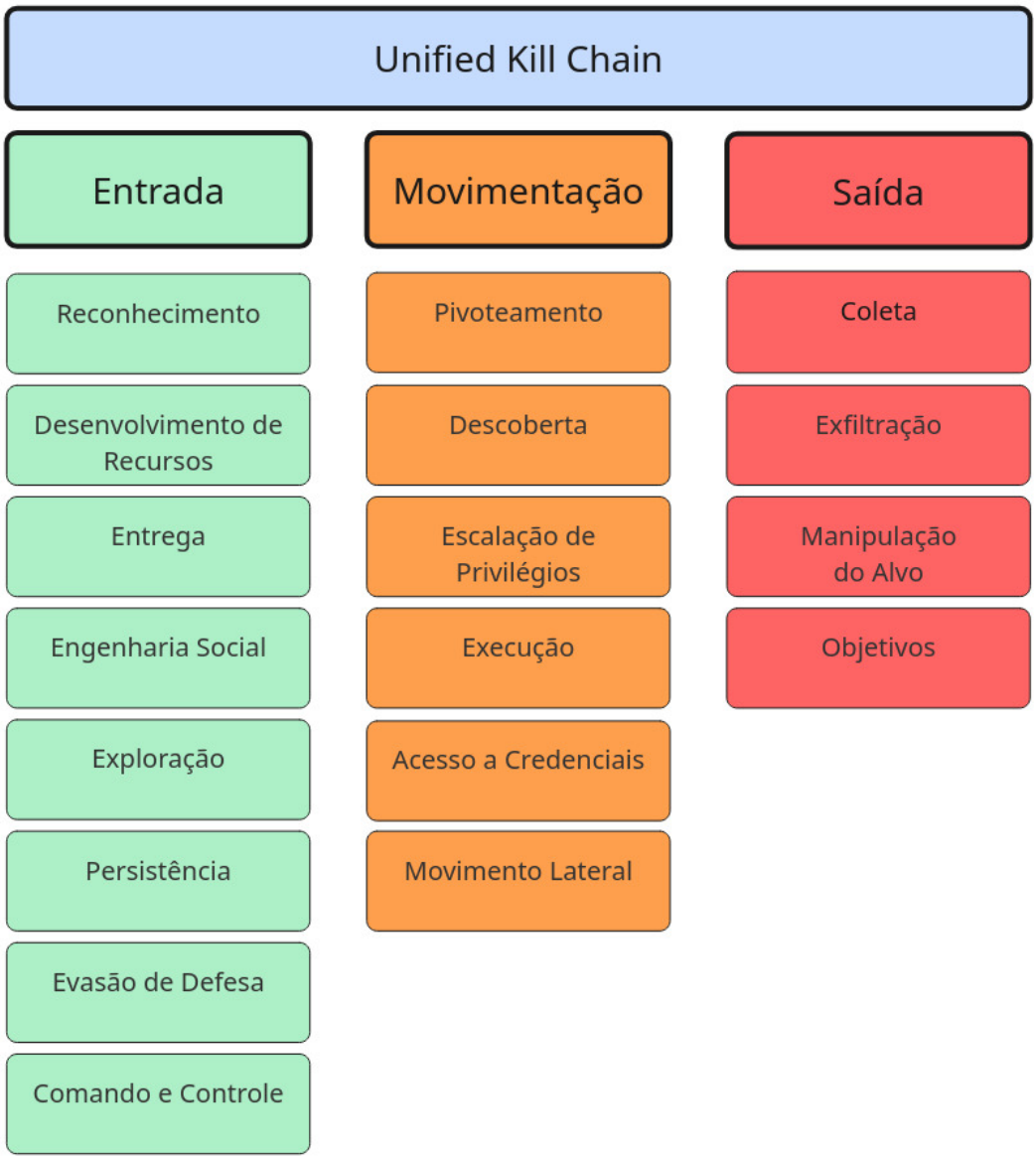

Figura 2.1: *Unified Kill Chain*.
Fonte: Adaptado de (Pol, 2017).

As etapas descritas pelo UKC são:

1. Reconhecimento (*Reconnaissance*): tarefa de pesquisar, identificar e selecionar alvos, por meio de reconhecimento ativo ou passivo. Ferramentas *Open source intelligence* (OSINT) [3] encontradas na Internet podem fornecer vários *insights* sobre o alvo.

2. Desenvolvimento de Recursos (*Resource Development*): execução de atividades preparatórias, destinadas a criar a infraestrutura necessária ao ataque. O registro de um domínio com o nome semelhante ao do alvo, apenas alterando algumas letras ou removendo o sufixo do país de .com.br para somente .com é um exemplo de atividade executada nesta etapa.

[3] Softwares para coletar e analisar informações de fontes disponíveis publicamente, como sites e mídias sociais (https://osintframework.com/).

3. Entrega (*Delivery*): técnicas que resultam na transmissão de um objeto armado para o ambiente alvo. Como exemplo, um arquivo aparentemente legítimo, mas que quando é aberto carrega um código malicioso para ser executado.

4. Engenharia Social (*Social Engineering*): técnicas que visam a manipulação de pessoas para a realização de ações inseguras. O uso de *phishing* (envio de e-mails falsos solicitando o fornecimento de dados pessoais) é um dos meios mais explorados nesta etapa.

5. Exploração (*Exploitation*): técnicas para explorar vulnerabilidades em sistemas que podem, entre outras, resultar na execução de código malicioso. O uso de HIDS podem ser contramedidas preventivas nesta etapa.

6. Persistência (*Persistence*): qualquer acesso, ação ou alteração que proporcione a presença persistente de um invasor no sistema. O carregamento de código malicioso durante a inicialização de um sistema é um exemplo.

7. Evasão de Defesa (*Defense Evasion*): técnicas que um invasor pode usar especificamente para evitar a detecção ou evitar outras defesas. O uso dos mesmos nomes de processos ou de tarefas legítimas e conhecidas do sistema operacional é uma forma de manter-se oculto.

8. Comando e Controle (*Command & Control*): técnicas que permitem que invasores se comuniquem com sistemas controlados dentro de uma rede alvo. Os NIDS podem detectar essa atividade.

9. Pivoteamento (*Pivoting*): tunelamento do tráfego através de um sistema controlado para outros sistemas que não são diretamente acessíveis. Por questões de desenho da rede ou para limitar ainda mais o acesso, podem ser acessíveis somente por meio de determinadas redes ou equipamentos, situação pensada para aumentar ainda mais a segurança.

10. Descoberta (*Discovery*): métodos que permitem a um invasor obter conhecimento sobre um sistema e seu ambiente de rede. Informações sobre grupos e usuários, softwares instalados e suas versões e se *patches* de segurança estão aplicados ou não são obtidas nessa etapa.

11. Escalação de Privilégio (*Privilege Escalation*): o resultado de técnicas que fornecem ao invasor permissões mais altas em um sistema ou rede. Alguns comandos são executados somente por um superusuário: root (Linux) e Administrador (Windows)™.

12. Execução (*Execution*): processos que resultam na execução de código controlado pelo invasor em um sistema local ou remoto. Serviços ou aplicativos podem ser iniciados sem o conhecimento do usuário regular e permitem o acesso remoto ao equipamento pelos invasores.

13. Acesso a Credenciais (*Credential Access*): técnicas que resultam no acesso ou controle de credenciais de sistema, serviço ou domínio. O uso de ferramentas de força bruta pode recuperar as senhas dos usuários que estão gravadas de maneira criptografada, revelando credenciais que podem ser usadas para acessar outros sistemas.

14. Movimento Lateral (*Lateral Movement*): técnicas que permitem que um adversário acesse e controle horizontalmente outros sistemas remotos. O uso do mesmo par usuário/senha do sistema comprometido pode permitir que esses sejam facilmente acessados.

15. Coleta (*Collection*): ferramentas usadas para identificar e coletar dados de uma rede alvo antes da exfiltração. O uso de *keyloggers* (ferramentas que capturam o que é digitado no teclado pelo usuário), *screenshots* e leitura de e-mails permitem que os invasores tenham acessos a mais dados e que podem ser usados para novos acessos.

16. Exfiltração (*Exfiltration*): métodos que resultam ou ajudam um invasor a remover dados de uma rede alvo. Os dados são copiados de forma lenta e gradual, de maneira que essa atividade não seja identificada por sistemas de prevenção. De posse dos dados, os invasores podem vendê-los a grupos específicos ou anunciá-los na *dark web* (rede que precisa de *softwares* específicos para acessá-la).

17. Manipulação do Alvo (*Target Manipulation*): técnicas destinadas a manipular, interromper ou destruir dados ou o sistema alvo. Nessa fase, o atacante investe diretamente contra a tríade CIA (confidencialidade, integridade e disponibilidade) do alvo, causando a interrupção ou degradação de serviços, a modificação de dados e a publicação de informações reservadas.

18. Objetivos (*Objectives*): objetivos sociotécnicos de um ataque que visam atingir de modo estratégico o sistema ou rede. Por exemplo, acessos remotos em infraestruturas críticas podem ser definidos como objetivos, mas nenhuma ação adicional é realizada na infraestrutura a fim de evitar perturbações não intencionais. Atacantes as vezes querem apenas aparecer na mídia para demonstrar seus conhecimentos.

Como detalhado, em algumas situações a utilização de IDS contribuem para o incremento da segurança de dispositivos ou redes, fato que faz com que tais sistemas devem ser constantemente melhorados a fim de manter sua eficiência ao longo do tempo.

2.4 Sistemas de detecção de intrusão

Detecção de intrusão é o processo de monitorar os eventos que ocorrem em sistemas e redes computacionais, analisando-os em busca de sinais de possíveis incidentes, que podem ser violações ou ameaças iminentes de violação de políticas de segurança ou de uso, ou de práticas de segurança (Scarfone and Mell, 2007).

Segundo os mesmos autores, dentre as principais funções dos sistemas de detecção de intrusão (IDS) estão:

- Registrar informações relacionadas aos eventos observados: os dados geralmente são registrados localmente, mas também podem ser enviados para outros sistemas.
- Notificar administradores de segurança sobre eventos observados: esse alerta pode ser enviado de várias maneiras, como e-mail, páginas web, *traps* SNMP ou programas e *scripts* definidos pelo usuário.
- Interromper o ataque (reatividade): encerrar a conexão de rede ou a sessão do usuário usada para o ataque, bloquear os acessos do atacante a partir de seu endereço IP ou bloquear os acessos aos recursos do alvo do ataque.
- Alterar a configuração do ambiente de segurança: reconfigurar outros dispositivos, como um *firewall* ou um *switch* para interromper o ataque.
- Alterar o conteúdo do ataque: algumas tecnologias podem remover ou substituir partes de um ataque, por exemplo, removendo um arquivo infectado do anexo de um e-mail.

Os IDS podem ser classificados com base em diferentes critérios, tais como:

- Local de instalação
  - Baseados em rede (*Network Intrusion Detection System* - NIDS).
  - Baseados em host (*Host-based Intrusion Detection System* - HIDS).
  - Wireless.
  - Análise do comportamento da rede (*Network Behavior Analysis* - NBA).
- Métodos de detecção
  - Baseada em assinaturas.
  - Baseada em anomalias.
  - Análise de protocolo *stateful*.
- Abordagens de detecção
  - Base estatística.
  - Baseada em regras.
  - Baseada em heurísticas.
  - Baseada em padrões.
  - Baseada em nuvem.
  - *Machine Learning*.

– *Deep Learning*.

A Figura 2.2 apresenta um panorama geral dessas classificações, as quais serão exploradas em maior detalhe nas seções seguintes.

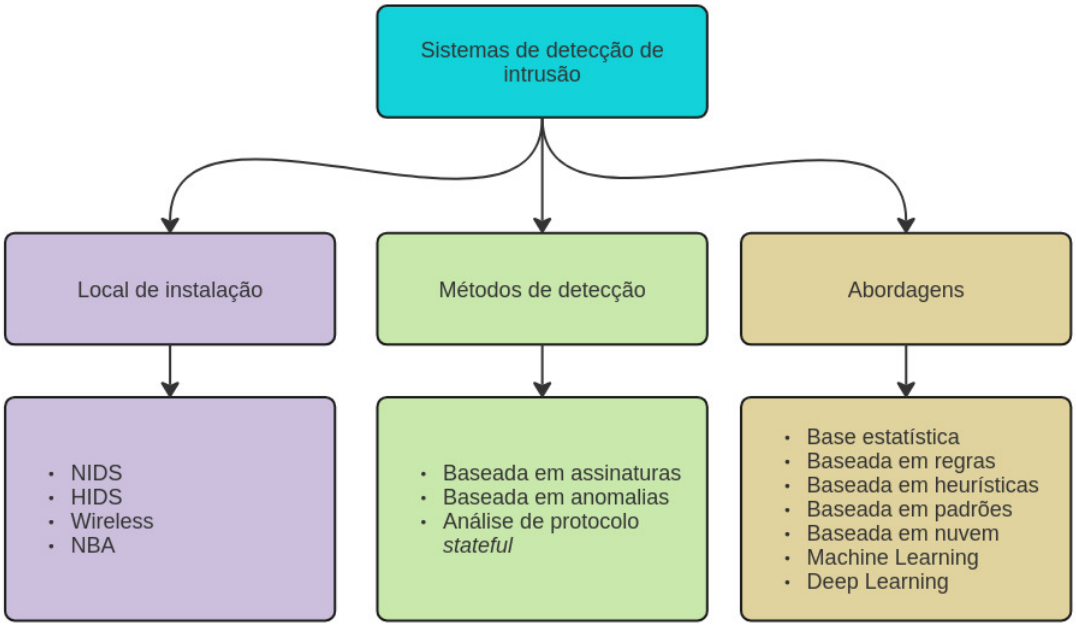

Figura 2.2: Classificações dos sistemas de detecção de intrusão.
Fonte: Adaptado de (Samet, Aslan, Gupta and Ozkan-Okay, 2021).

2.4.1 Classificação dos IDS conforme o local de instalação

Segundo (Scarfone and Mell, 2007), os IDS podem ser classificados conforme o seu local de instalação:

I - Baseados em rede (*Network Intrusion Detection System* - NIDS): monitoram o tráfego de rede para determinados segmentos ou dispositivos e analisam a atividade dos protocolos e de aplicações para identificar atividades suspeitas. É comumente implantado na fronteira entre redes, próximos a *firewalls*, roteadores de borda e a redes sem fio.

II - Baseados em host (*Host-based Intrusion Detection System* - HIDS): monitoram as atividades de um único *host* e os eventos suspeitos que ocorrem dentro desse *host*. Exemplos são o monitoramento do tráfego de rede, *logs*, processos em execução, acesso e modificação de arquivos e alterações nas configurações do sistema. São usualmente implantados em *hosts* críticos, como servidores acessíveis publicamente.

III - Wireless: monitoram o tráfego da rede sem fio e analisam seus protocolos para identificar atividades suspeitas. É normalmente implantado dentro do alcance da rede sem fio, mas também pode ser em locais onde redes não autorizadas podem estar ativas.

IV - Análise do comportamento da rede (*Network Behavior Analysis* - NBA): examinam o tráfego de rede para identificar ameaças que geram fluxos de tráfego incomuns, como ataques

distribuídos de negação de serviço (DDoS), certas formas de *malware* (por ex. *worms* e *backdoors*) e violações de políticas (por ex. um sistema cliente fornecendo serviços de rede para outros sistemas). Os sistemas NBA são mais frequentemente implantados para monitorar fluxos internos de uma organização, mas às vezes também são implantados onde podem monitorar fluxos entre parceiros de negócios.

2.4.2 Métodos para a detecção de intrusão dos IDS

Os IDS podem utilizar múltiplos métodos para a detecção de uma intrusão, seja de uma forma combinada ou individual dos seguintes modelos (Scarfone and Mell, 2007):

I - Detecção baseada em assinaturas: a assinatura é um padrão que corresponde a uma ameaça conhecida. Este método consiste em comparar assinaturas com eventos observados para identificar possíveis incidentes. É uma técnica rápida e eficaz na detecção de ameaças conhecidas, mas ineficaz na detecção de ameaças desconhecidas (ataques de dia zero), ou disfarçadas pelo uso de técnicas de evasão ou ainda por variantes de ameaças conhecidas.

Esses métodos não conseguem rastrear e compreender o estado de comunicações complexas e de lembrar de solicitações anteriores ao processar a solicitação atual, de tal modo que esta limitação impede que sejam detectados ataques que compreendem vários eventos se nenhum dos eventos conter uma indicação clara de um ataque.

II - Detecção baseada em anomalias: um IDS usando detecção baseada em anomalias cria perfis da atividade de usuários, *hosts*, rede ou aplicações, por meio do monitoramento das atividades durante um período de treinamento, que pode levar dias ou semanas, e depois utiliza esses perfis para fazer a comparação com os eventos observados, com a finalidade de identificar desvios significativos entre eles.

Um dos principais benefícios dos métodos de detecção baseados em anomalias é que os mesmos podem ser muito eficazes na detecção de ameaças anteriormente desconhecidas, pois, se ocorrer um aumento repentino no processamento de um equipamento devido a um ataque ou infecção por vírus ou *malwares*, isso pode gerar um alerta porque o comportamento observado está fora do perfil normal de uso.

Um problema com a construção de perfis é que pode ser muito desafiador em alguns casos torná-los precisos, uma vez que a atividade computacional pode ser muito complexa devido a sazonalidade na utilização dos recursos, a qual pode não ter sido observada durante o período de treinamento.

III - Análise de protocolo *stateful*: a análise de protocolo *stateful* é o processo de comparação de perfis de uso predeterminados de protocolos em relação a eventos observados para identificar desvios. A análise de protocolo depende de padrões desenvolvidos pelos fornecedores ou que constam em RFC, e que especificam como determinados protocolos devem e como não devem ser usados. O termo *stateful* significa que o IDS é capaz de compreender e rastrear

o estado dos protocolos.

Uma característica dessa análise é a capacidade de identificar sequências inesperadas de comandos, como emitir o mesmo repetidamente ou realizar um comando sem efetivar um outro do qual depende, ou ainda, validar o tamanho mínimo e o máximo dos argumentos enviados. Como desvantagem, consomem muitos recursos devido à complexidade da análise e a sobrecarga envolvida na execução do rastreamento do estado para muitas sessões simultâneas. Outro problema é que os métodos de análise de protocolo com estado não podem detectar ataques que não violem as características do comportamento padrão do protocolo, como realizar muitas ações benignas em um curto intervalo de tempo para causar uma negação de serviço (DoS).

2.4.3 Abordagens para a detecção de intrusão dos IDS

Os IDS podem adotar diferentes técnicas para a análise dos eventos, as quais estão intrinsecamente relacionadas aos métodos de detecção, conforme descrito por (Samet et al., 2021):

I - Base estatística: monitora as transações normais para criar um perfil de uso e identifica quando os eventos observados desviam do padrão normal (legítimo), indicando um possível ataque. Os IDS baseados em estatística podem detectar ataques de dia zero já que não dependem de assinaturas específicas de ataques.

II - Baseada em regras: detecta possíveis intrusões a partir de regras (padrões) predefinidas. A partir de uma única regra, muitos ataques podem ser reconhecidos. Em comparação com a abordagem baseada em assinaturas, que requer milhares de assinaturas para reconhecer os mesmos tipos e quantidade de ataques, a abordagem baseada em regras precisa apenas de algumas entradas.

III - Baseada em heurísticas: constrói um modelo de detecção que define os comportamentos aceitáveis e identifica qualquer desvio desses padrões como suspeito. Para distinguir entre comportamentos normais e ataques, essa abordagem requer conhecimento e experiência. Durante a análise, as atividades são examinadas em busca de comportamentos suspeitos, gerando alertas se desvios nos padrões forem detectados.

IV - Baseada em padrões (assinaturas): se concentra na identificação de sequências de instruções maliciosas nos ataques, conhecidas como assinaturas. Envolve a identificação de caracteres e strings específicos para extrair padrões significativos nos dados coletados, permitindo assim a detecção de ataques que se baseiam nesses padrões. Embora essa abordagem seja capaz de detectar rapidamente ataques conhecidos de forma eficiente, não consegue identificar a maioria dos ataques de dia zero, pois suas assinaturas ainda não são conhecidas.

V - Baseada em nuvem: a utilização de nuvens públicas e privadas oferece a oportunidade de detectar vários tipos de ataques simultaneamente e com alto desempenho. O IDS baseado em nuvem compreende três componentes distintos: (1) o coletor de dados do usuário que é um servidor independente responsável por coletar, filtrar, padronizar e enviar pacotes para o serviço

de nuvem, (2) o serviço de nuvem que analisa e valida os eventos recebidos do coletor de dados do usuário, traduzindo-os para um formato comum para o componente de detecção de intrusão na nuvem e (3) a detecção de intrusão na nuvem é a parte central do sistema, responsável por receber os dados do serviço de nuvem, analisar os pacotes e identificar possíveis intrusões.

VI - *Machine Learning*: emprega uma variedade de técnicas de aprendizado, cada uma com suas características específicas. O método supervisionado aplica conjuntos de dados rotulados, permitindo que o algoritmo aprenda a associar entradas a saídas corretas. Já o não supervisionado lida com conjuntos de dados não rotulados, onde o algoritmo busca identificar padrões ou estruturas intrínsecas nos dados, sem a necessidade de rótulos prévios. Além disso, o semissupervisionado combina elementos dos dois anteriores para treinar o modelo.

As vantagens dessa abordagem na detecção de intrusões são diversas: sua adaptabilidade permite que os modelos se ajustem dinamicamente às mudanças nas ameaças e no ambiente de rede. O alto desempenho desses algoritmos, especialmente quando aplicados a conjuntos de dados grandes e complexos, garante uma detecção eficiente e em tempo real de atividades suspeitas. Além disso, a flexibilidade dos modelos permite a incorporação de novos dados e atualizações de forma contínua, garantindo que o sistema permaneça eficaz ao longo do tempo. Uma das vantagens mais significativas é a capacidade de detectar novos tipos de ataques, mesmo aqueles para os quais não há assinaturas pré-existentes, permitindo uma defesa proativa contra ameaças emergentes.

VII - *Deep Learning*: é um subcampo do aprendizado de máquina e que tem ganhado popularidade devido à sua capacidade de aprender com exemplos, e de eliminar a necessidade de engenharia manual de atributos. O cerne da aprendizagem profunda reside em sua arquitetura de redes neurais, que consistem em várias camadas de neurônios interconectados. Cada camada recebe os dados de entrada da camada anterior e extrai características progressivamente mais complexas e abstratas à medida que os dados fluem pela rede. Essa hierarquia de representações permite que o modelo aprenda a partir dos dados de forma automatizada e autônoma, sem a necessidade de intervenção humana para selecionar e extrair manualmente atributos relevantes.

2.5 *Knowledge Discovery in Databases*

O processo de *Knowledge Discovery in Databases* (KDD) compreende uma sequência estruturada de etapas destinadas à descoberta de conhecimento útil e compreensível a partir de grandes volumes de dados. As definições desta seção que trata do KDD foram obtidas a partir de (Fayyad, Piatetsky-Shapiro and Smyth, 1996).

Segundo os autores, o KDD é um processo interativo e iterativo, que envolve decisões em diversos níveis por parte do usuário. Essas etapas podem ser organizadas em três fases principais: pré-processamento, processamento e pós-processamento, conforme descrito a seguir.

2.5.1 Pré-processamento

A fase de pré-processamento visa a preparar os dados para a etapa de processamento. Essa fase abrange:

- Compreensão do domínio e definição dos objetivos: consiste na análise do contexto da aplicação, do conhecimento prévio disponível e na formulação dos objetivos do processo KDD, a partir da perspectiva do usuário final.
- Seleção do conjunto de dados de interesse: define-se o subconjunto de variáveis ou amostras de dados que serão analisadas.
- Limpeza: inclui atividades como a remoção de ruídos, tratamento de valores ausentes, consolidação de informações temporais e normalização.
- Redução e projeção dos dados: envolve a transformação de variáveis e a redução de dimensionalidade, facilitando a representação dos dados conforme o objetivo.

2.5.1.1 Normalização

A normalização é uma das abordagens de pré-processamento em que os dados são dimensionados ou transformados para que cada atributo contribua de forma equilibrada para o modelo. Segundo (Faceli, Lorena, Gama, de Almeida and Carvalho, 2021), geralmente é preferível aplicar a padronização em vez da reescala, uma vez que a padronização lida melhor com a presença de *outliers* nos conjuntos de dados.

As definições desta seção foram obtidas a partir de (Singh and Singh, 2020):

O *Z-score* (Equação (2.1)) é uma técnica de padronização baseada em média e desvio padrão, na qual essas medidas são usadas para redimensionar os dados de forma que as características resultantes tenham média zero e variância unitária. Em outras palavras, cada instância $x_{i,n}$ dos dados é transformada em $x'_{i,n}$, onde μ e σ são, respectivamente, a média e o desvio padrão dos dados em análise.

$$x'_{i,n} = \frac{x_{i,n} - \mu_i}{\sigma_i} \tag{2.1}$$

Já a técnica de reescala *Min-Max Scaling* (Equação (2.2)) transforma os dados para um intervalo específico, geralmente $[0, 1]$. Nesse processo, o valor mínimo (x_{min}) e o máximo (x_{max}) são extraídos diretamente do conjunto de dados. Entretanto, o *Min-Max* pode ser adaptado para mapear os dados para qualquer intervalo definido pelo usuário, como $[-1, 1]$ ou [a, b]. Nesse caso, *a* e *b* representam, respectivamente, o novo valor mínimo e o novo valor máximo desejados.

$$x' = a + \left(\frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \right) (b - a) \tag{2.2}$$

O sucesso dos algoritmos de aprendizado de máquina (AM) depende da qualidade dos dados para obter um modelo preditivo generalizado do problema de classificação. A normalização acelera o processo de treinamento ao colocar as variáveis em uma escala similar, facilitando a convergência dos algoritmos de AM. Ela também evita que variáveis com escalas muito grandes dominem o modelo, pois, ao colocar todas as variáveis na mesma escala, garante que cada característica tenha uma influência proporcional sobre o resultado.

2.5.1.2 Seleção de atributos

Segundo (ZorarpacI and Ozel, 2016), a seleção de atributos, também conhecida como redução de dimensionalidade, é o método utilizado para selecionar um subconjunto ótimo de características relevantes que representa o conjunto original com o menor erro possível no aprendizado de um modelo de classificação.

Executadas durante o pré-processamento, as técnicas de seleção trazem benefícios como melhor interpretabilidade do modelo, tempos de treinamento mais curtos e uma maior capacidade de generalização. De maneira geral, as abordagens mais empregadas para a avaliação de subconjuntos de atributos são *filter* e *wrapper*.

Na abordagem *filter*, a seleção de atributos é feita de forma independente do algoritmo de AM, utilizando exclusivamente propriedades estatísticas ou métricas intrínsecas dos dados para determinar a relevância dos atributos (Leevy, Hancock, Zuech and Khoshgoftaar, 2021). Entre os exemplos mais comuns dessa abordagem estão os cálculos de *Information Gain* (Quinlan, 1986), *Gain Ratio* (Witten, Frank, Hall and Pal, 2017) e *Chi-Squared* (Turhan, 2020).

Por outro lado, quando um classificador é utilizado para avaliar os subconjuntos de características gerados, o método de seleção é denominado *wrapper*. Essa abordagem realiza a avaliação de diferentes subconjuntos de atributos por meio da aplicação direta no algoritmo de AM, selecionando o subconjunto que apresentar o melhor desempenho, considerando métricas como a menor taxa de erro e a maior redução no número de atributos (ZorarpacI and Ozel, 2016).

2.5.1.3 Técnicas de balanceamento de dados

O desbalanceamento de dados, também conhecido como *Data Imbalance* em AM, refere-se a uma distribuição desigual de classes dentro de um conjunto de dados. Esse problema é particularmente comum em tarefas de classificação, nas quais a distribuição de classes ou rótulos não é uniforme (Mohammed, Jordan, Rawashdeh and Abdullah, 2020).

Uma das abordagens mais utilizadas para mitigar esse problema é o método de reamostragem, que consiste em adicionar registros à classe minoritária ou remover registros da classe majoritária, com o objetivo de equilibrar o conjunto de dados. Os métodos comumente utilizados são o *oversampling* e o *undersampling*:

O *oversampling* ou sobreamostragem é aplicado quando uma ou mais classes possuem significativamente menos exemplos do que outras. Existem diversas estratégias de *oversampling*, dentre as quais se destacam:

- *Random Oversampling* (ROS)[4]: técnica simples que replica aleatoriamente exemplos da classe minoritária até atingir o equilíbrio entre as classes. Embora esse método ajude o modelo a aprender melhor os padrões da classe minoritária, ele pode aumentar o risco de *overfitting*[5], uma vez que apenas repete exemplos existentes, sem adicionar novas informações (Khan, Chaudhari and Chandra, 2023).
- *Adaptive Synthetic Sampling Approach for Imbalanced Learning* (ADASYN)[4]: a ideia central é utilizar uma distribuição ponderada para gerar exemplos sintéticos com base no nível de dificuldade de aprendizagem de cada instância minoritária. Dessa forma, são gerados mais dados sintéticos para exemplos mais difíceis de serem aprendidos e menos para os exemplos mais fáceis (Haibo, Yang, Garcia and Shutao, 2008).
- *Synthetic Minority Oversampling Technique* (SMOTE)[4]: propõe uma abordagem mais sofisticada, utilizando o algoritmo *K-Nearest Neighbors* (KNN)[6] para gerar exemplos sintéticos. Segundo (Chawla, Bowyer, Hall and Kegelmeyer, 2002), para cada amostra da classe minoritária, identificam-se seus k vizinhos mais próximos pertencentes à mesma classe. Novos exemplos são então criados ao interpolar aleatoriamente pontos ao longo das linhas que conectam a amostra original a seus vizinhos. Esse processo gera amostras artificiais que representam combinações lineares de exemplos reais, ampliando de forma mais realista o espaço da classe minoritária e reduzindo o risco de *overfitting* associado ao simples aumento de cópias.

Por outro lado, o *undersampling* ou subamostragem é utilizado para lidar com dados desbalanceados em que a quantidade de exemplos da classe majoritária é elevada. Entre as abordagens de *undersampling*, destacam-se:

- *Random Undersampling* (RUS)[4]: uma técnica simples que remove instâncias da classe majoritária de forma aleatória e sem reposição. Ou seja, uma vez removida, a instância não é substituída (Zuech, Hancock and Khoshgoftaar, 2021).
- *Cluster Centroids*[4]: de acordo com (Nápoles and Grau, 2023), essa técnica aplica o agrupamento *k-means*[7] para identificar grupos de instâncias da classe majoritária. Cada grupo é substituído por seu centroide, reduzindo a quantidade de dados da classe majoritária e mantendo a representatividade dos dados.

[4]https://imbalanced-learn.org/stable/references. Acesso em abril/2025.

[5]*Overfitting* ocorre quando o modelo aprende demais os dados de treino e perde a capacidade de generalização (Fayyad et al., 1996). *Underfitting* acontece quando o modelo não aprende o suficiente, tendo desempenho ruim em treino e teste (Hastie, Tibshirani and Friedman, 2017).

[6]https://scikit-learn.org/stable/modules/neighbors.html. Acesso em abril/2025.

[7]https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html. Acesso em abril/2025.

- *Tomek Links*[4]: segundo (Nápoles and Grau, 2023), os *Tomek Links* são pares de instâncias de classes opostas que estão muito próximas entre si. A remoção da instância majoritária nesse par reduz a sobreposição entre as classes, contribuindo para um conjunto de dados mais limpo e balanceado.

2.5.2 Processamento

Essa etapa representa o núcleo do processo de KDD, em que técnicas de mineração de dados são aplicadas para extrair padrões e relações significativas nos conjuntos. Envolve:

- Mapeamento entre objetivos e métodos de mineração de dados: nesta fase, os objetivos previamente definidos são associados a técnicas específicas, como classificação, regressão, agrupamento e sumarização.
- Eleição de modelos e hipóteses: selecionam-se os algoritmos, definem-se os parâmetros e modelos adequados aos dados e aos propósitos do estudo.
- Execução da mineração de dados: consiste na aplicação iterativa dos algoritmos de mineração escolhidos para identificação de padrões de interesse, como regras, árvores de decisão ou agrupamentos. Essa etapa envolve as atividades de treinamento e teste de modelos e, assim, a mineração de dados deixa de ser apenas um mecanismo de descoberta de padrões e passa a incorporar práticas modernas que tornam os sistemas capazes de realizar predições com alto grau de assertividade.

2.5.3 Pós-processamento

A fase final do processo KDD está relacionada à interpretação dos resultados, avaliação do conhecimento extraído e sua aplicação prática. Abrange:

- Interpretação dos padrões extraídos: envolve a análise crítica dos modelos gerados, sua visualização e, quando necessário, o retorno às etapas anteriores para ajustes e repetições.
- Ação baseada no conhecimento descoberto: inclui a utilização direta dos resultados, sua incorporação em sistemas decisórios ou a documentação dos achados para comunicação com as partes interessadas. Métricas de desempenho, testes estatísticos, bem como técnicas de visualização da solução podem ser utilizadas para a análise dos resultados.

2.6 Inteligência artificial

O termo Inteligência Artificial (IA) ou *Artificial Intelligence* foi cunhado em 1956 por John McCarthy, professor do Instituto de Tecnologia de Massachusetts, para uma conferência que ele organizou, conhecida como a Conferência de Dartmouth (Ertel, 2017).

Conforme o mesmo autor, nesse evento foram estabelecidos os principais objetivos da IA: compreender e modelar o pensamento humano, além de projetar máquinas capazes de reproduzir esse comportamento. A ideia fundamental da IA é desenvolver sistemas que possam realizar tarefas de forma autônoma, simulando a capacidade humana de raciocínio, aprendizado e adaptação.

Outros autores definem a IA como a combinação de sistemas sofisticados de *hardware* e de *software*, juntamente com bancos de dados e modelos de processamento baseados em conhecimento, para demonstrar características da capacidade de decisão humana (Shubhendu and Vijay, 2013). A Figura 2.3 exibe a IA e os subcampos que a compõem, os quais serão descritos nas próximas seções.

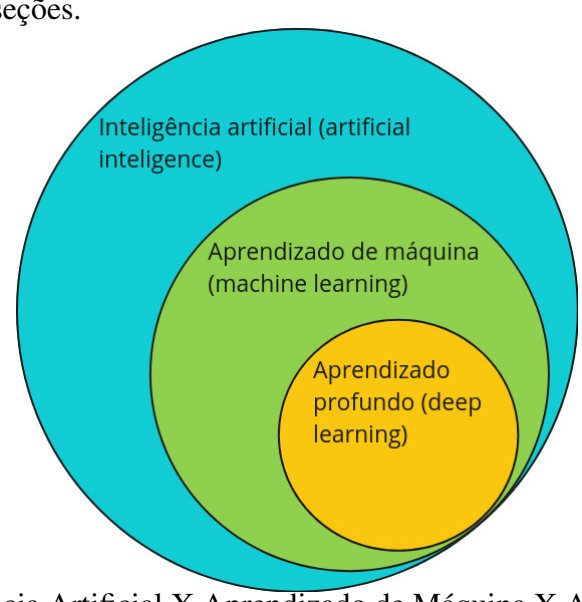

Figura 2.3: Inteligência Artificial X Aprendizado de Máquina X Aprendizado Profundo.
Fonte: O autor.

2.6.1 Aprendizado de máquina

De acordo com (Tiwari, Tiwari and Tiwari, 2018), o Aprendizado de Máquina (AM) ou *Machine Learning* é um ramo da IA que permite que sistemas de computadores aprendam diretamente com exemplos, dados e experiências. Ao invés de seguir as abordagens tradicionais de programação baseadas em regras codificadas passo a passo, os sistemas de AM aprendem a executar tarefas específicas de forma inteligente, analisando grandes conjuntos de dados para identificar padrões e aperfeiçoar suas saídas.

Consoante com (Saranya, Sridevi, Deisy, Chung and Khan, 2020), esses algoritmos podem ser classificados como supervisionados, não supervisionados ou semissupervisionados, dependendo da natureza dos dados e dos problemas que estão sendo resolvidos:

- O supervisionado lida com dados rotulados em classes e encontra o relacionamento entre os atributos de entrada e a sua classe.

- O não supervisionado trata de descobrir padrões quando não há rótulos ou categorias predefinidas nos exemplos de entrada.
- O semissupervisionado usa uma combinação de uma pequena quantidade de dados rotulados e uma grande quantidade de dados não rotulados, para treinar os modelos.

O autor (Ertel, 2017) adiciona ainda mais um tipo de aprendizado:

- Por reforço, que envolve treinar um agente para tomar sequências de decisões, recebendo recompensas ou penalidades como retorno. O objetivo é que o agente aprenda a escolher as melhores ações para maximizar as recompensas ao longo do tempo. Esse tipo de aprendizado é amplamente aplicado em áreas como robótica e jogos, onde o agente deve aprender comportamentos por meio da interação contínua com o ambiente.

2.6.2 Aprendizado profundo

Conforme (Wei and Shangguan, 2023), o Aprendizado Profundo (*Deep Learning*), é uma abordagem de AM no campo da IA que utiliza grandes volumes de dados e arquiteturas de redes neurais profundas, que consistem de várias camadas de unidades de processamento interconectadas, conhecidas como neurônios artificiais.

Esta abordagem é amplamente utilizada em áreas como detecção de intrusão e demonstram também poderosas capacidades analíticas em outros campos, como processamento de imagens, bioinformática, processamento de linguagem natural, análises de *big data*, processamento de vídeo e de fala.

2.6.3 Tarefas de classificação

De acordo com (Farah, 2020), a classificação refere-se à tarefa de atribuir rótulos a exemplos com o auxílio de algoritmos de AM, sendo o valor a ser previsto denominado rótulo de classe ou alvo. O autor ainda apresenta as seguintes definições:

- Classificação binária: trata-se da tarefa de classificar variáveis em duas categorias ou classes de destino. No contexto deste trabalho, os eventos do conjunto de dados são categorizados em dois tipos:
  - Anomalia: indica a presença de um ataque à rede.
  - Normal: indica a ausência de ataques à rede.
- Classificação multiclasses: envolve a tarefa de mapear variáveis para mais de duas classes de destino. Por exemplo, os rótulos de classe poderiam ser definidos da seguinte forma:
  - Anomalia/*Scan*: indica a ocorrência de um ataque à rede, do tipo varredura.
  - Anomalia/*DoS*: indica uma ocorrência de um ataque à rede, do tipo negação de serviço.

2.6.4 *Ensemble learning*

O aprendizado baseado em conjuntos, ou *ensemble learning*, consiste na combinação de predições de diversos modelos treinados nos mesmos conjuntos de dados, com o objetivo de maximizar sua performance. Segundo (Silva, 2024), as técnicas com mais ampla utilização são *bagging*, *boosting* e *stacking*, onde o conceito central é que ao combinar as previsões de diversos modelos (chamados de modelos fracos), melhora-se o resultado final, especialmente em problemas complexos.

2.6.4.1 *Bagging*

Conforme discutido por (Hastie et al., 2017), o *bagging* ou *bootstrap aggregation* é uma técnica projetada para procedimentos caracterizados por alta variância (modelos sensíveis aos dados de treinamento) e baixo viés (modelos que aprendem bem os padrões reais dos dados), como é o caso das árvores de decisão.

O *bagging* envolve a criação de múltiplos subconjuntos de dados, selecionados aleatoriamente a partir da base original, com reposição. Cada subconjunto é então utilizado para construir um modelo distinto, empregando sempre o mesmo algoritmo de AM e, ao término do processo, o resultado final é apanhado por meio de um mecanismo de votação para tarefas de classificação, ou por meio da média para tarefas de regressão. A Figura 2.4 apresenta esta técnica.

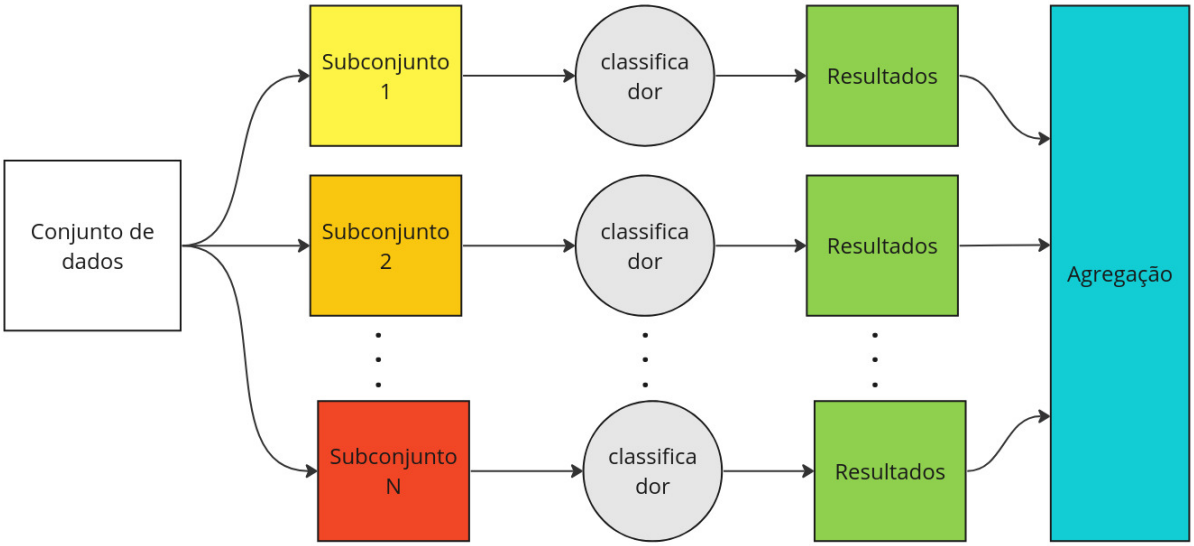

Figura 2.4: Técnica *ensemble learning bagging*.
Fonte: O autor.

2.6.4.2 *Boosting*

O *boosting* busca combinar vários modelos fracos, como árvores de decisão simples, para formar um modelo forte que seja capaz de realizar previsões mais precisas (Witten et al., 2017).

O conceito baseia-se na construção sequencial de modelos que sejam complementares, por meio de um processo iterativo: cada novo modelo é treinado com foco nas instâncias que foram mal classificadas pelos modelos anteriores. Durante o treinamento, são atribuídos pesos às instâncias, dando maior importância àquelas que tiveram erros de classificação. Dessa forma, os novos modelos são incentivados a se tornarem especialistas nos casos tratados incorretamente. Assim como no *bagging*, o *boosting* utiliza sempre o mesmo algoritmo de AM para a construção de todos os modelos ao longo do processo. A Figura 2.5 ilustra a técnica de *boosting*.

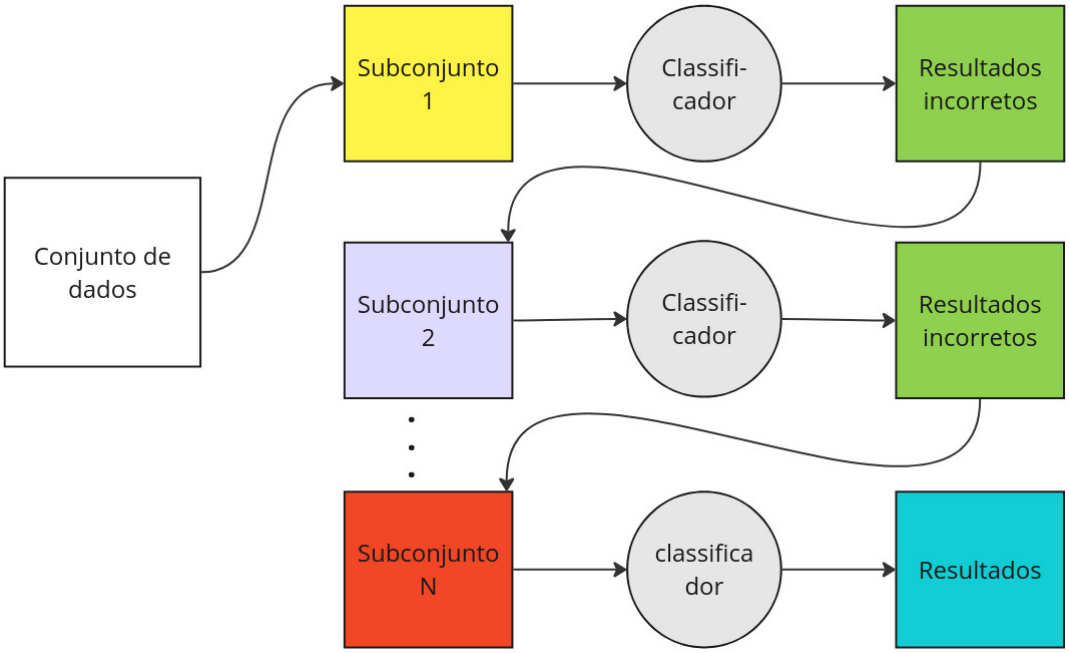

Figura 2.5: Técnica *ensemble learning boosting*.
Fonte: O autor.

2.6.4.3 *Stacking*

O *stacking* é uma abordagem que combina algoritmos distintos de AM, ao contrário de técnicas como *bagging* e *boosting* que utilizam sempre os mesmos durante o processamento. No *stacking*, diferentes algoritmos, como árvores de decisão, redes neurais e outros são treinados simultaneamente e seus resultados são integrados por um modelo conhecido como meta-aprendiz (Witten et al., 2017).

O meta-aprendiz analisa o desempenho dos modelos individuais e, diferente de usar uma votação simples como no *bagging*, atribui maior peso aos classificadores mais confiáveis para cada situação. Isso resulta em uma abordagem mais sofisticada de combinação de modelos, potencializando a capacidade preditiva geral. A arquitetura dessa técnica é apresentada na Figura 2.6.

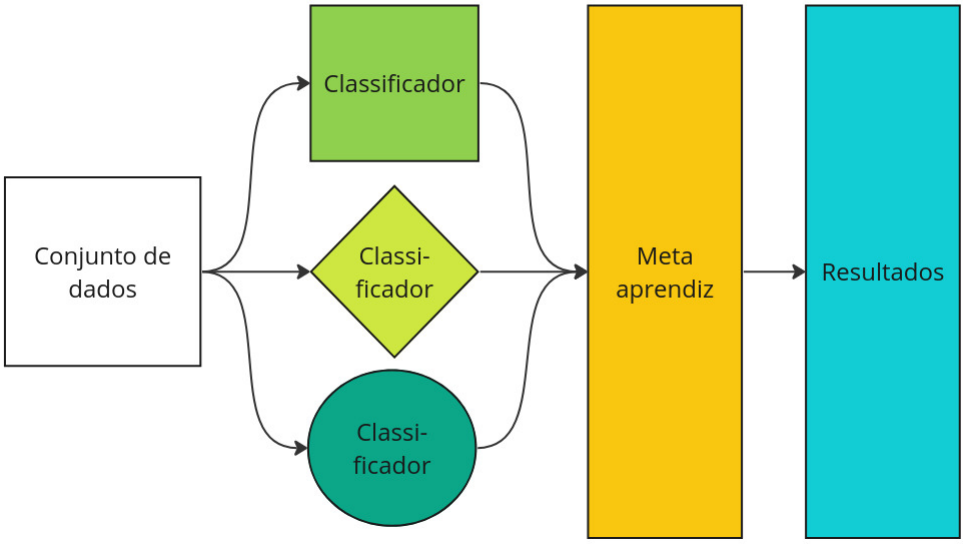

Figura 2.6: Técnica *ensemble learning stacking*.
Fonte: O autor.

Segundo o mesmo autor, embora o *stacking* tenha sido desenvolvido há alguns anos, é menos mencionado na literatura sobre AM do que o *bagging* e o *boosting*, em parte, porque é difícil de analisar teoricamente e em parte porque não há uma maneira melhor de fazê-lo geralmente aceita.

2.6.5 Algoritmos baseados em árvores

Nesta seção são apresentados os principais algoritmos de AM que exploram o uso de árvores de decisão como base para a construção de modelos preditivos.

2.6.5.1 Árvore de decisão

Uma árvore de decisão (AD) é uma estrutura hierárquica usada para tomar decisões ou fazer previsões, partindo de um nó raiz e se ramificando por meio de decisões internas até atingir os nós folha. Cada divisão busca separar os dados de forma a maximizar a pureza dos subconjuntos formados, ou seja, agrupar os exemplos de maneira que pertençam à mesma classe (em problemas de classificação) ou apresentem valores similares (em regressão) (Rokach and Maimon, 2006).

O processo de construção de uma AD geralmente segue os seguintes passos (Schonlau and Zou, 2020):

1. Seleção do atributo de divisão: em cada nó, o algoritmo avalia todos os atributos disponíveis e escolhe aquele que melhor separa os dados, de acordo com um critério como ganho de informação, razão de ganho ou índice de Gini.

2. Divisão dos dados: com base no atributo escolhido, os dados são divididos em subconjun-

tos. Para atributos categóricos, essa divisão ocorre para cada valor distinto; para atributos numéricos, determina-se um ponto de corte (por exemplo, maior ou menor que um valor).

3. Repetição recursiva: o processo é repetido recursivamente para cada novo subconjunto, criando nós sucessivos com novos testes até que se atinja um critério de parada.

4. Critérios de parada: a construção é encerrada quando todos os exemplos em um nó pertencem à mesma classe, quando não há mais atributos a serem divididos ou quando limites como a profundidade máxima ou o número mínimo de exemplos são alcançados.

5. Formação de nós folha: quando a recursão termina, os nós finais (folhas) representam a classe predita (no caso de classificação) ou o valor estimado (no caso de regressão).

De acordo com (Panda and Sagar, 2022), a definição de conceitos adicionais é necessária para uma compreensão completa do funcionamento das AD e de seus componentes:

- Nó Raiz (*Root Node*): é o primeiro nó da árvore, representando o conjunto completo. A partir dele, os dados são divididos com base no atributo mais relevante. O nó raiz é o ancestral de todos os outros nós da árvore.
- Nó Interno (*Internal Node*): nós intermediários que realizam testes ou decisões com base em atributos. Cada nó interno divide os dados em subconjuntos e possui nós filhos, que podem ser outros nós internos ou nós folha.
- Nó Terminal ou Folha (*Leaf Node*): nó que não realiza novas divisões. Representa o resultado de uma sequência de decisões, normalmente uma classe predita ou um valor numérico. Não possui filhos.
- Subárvore: estrutura hierárquica derivada de um nó interno, que pode ser tratada como uma nova árvore, contendo parte dos dados e decisões originais.
- Profundidade (*Depth*): número de níveis da árvore, ou seja, a distância máxima entre o nó raiz e qualquer nó folha. Reflete o grau de detalhamento da árvore.
- Divisão (*Split*): ato de particionar o conjunto de dados com base em um atributo. Cada divisão ocorre em um nó interno e gera ramos que levam a novos nós, refinando a separação entre as classes.
- Poda (*Pruning*): processo de remoção seletiva de nós internos ou subárvores que não contribuem significativamente à árvore. Visa a melhorar a generalização do modelo.

Na Figura 2.7 é demonstrada uma AD com seus respectivos componentes, utilizada para avaliar se um evento hipotético deve ser classificado como tráfego normal ou como uma tentativa de invasão, com base nos resultados obtidos nos nós de decisão. Essa representação pode, por exemplo, corresponder a um conjunto de regras do tipo "se-então"implementadas em um

firewall, no qual apenas determinados serviços estão ativos e autorizados a receber conexões oriundas da Internet.

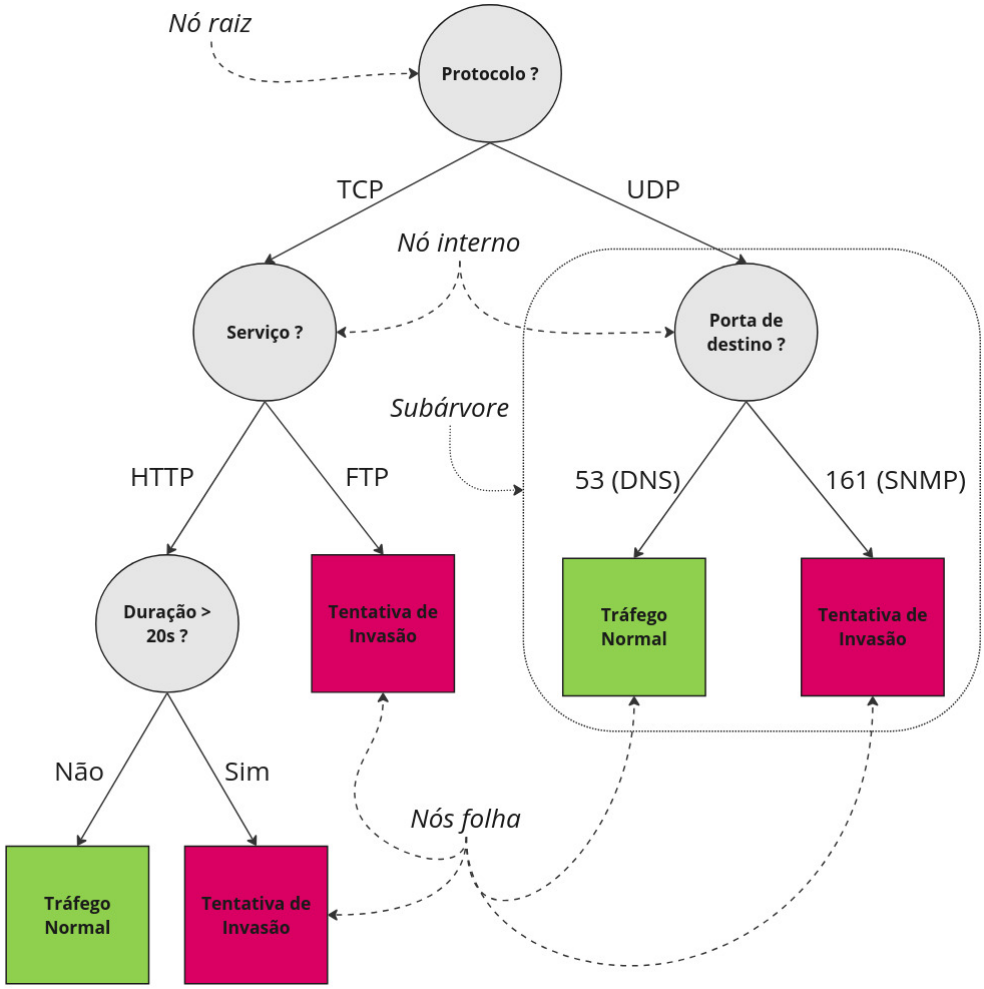

Figura 2.7: Representação de uma árvore de decisão.
Fonte: O autor.

Conforme (Breiman, Friedman, Olshen and Stone, 1984), o problema na construção de árvores é como determinar as divisões do conjunto de dados em pedaços cada vez menores. A ideia é selecionar cada divisão de um subconjunto, de modo que os dados em cada um dos subconjuntos descendentes criados sejam "mais puros"do que os do subconjunto pai e repetir esses passos até finalizar a construção da árvore.

Para tomar a decisão sobre qual atributo será escolhido para fazer a divisão dos nós de uma árvore, todos os atributos são avaliados com base em critérios estatísticos, sendo que os mais utilizados são:

- Índice de Gini (*Gini index*): consiste em uma medida de dispersão estatística destinada a representar a desigualdade numa distribuição (Breiman et al., 1984). Quando considerado no ambiente de ciência de dados, mede a probabilidade de classificar erroneamente um elemento aleatório do conjunto, se ele for rotulado de acordo com a distribuição de classes existente no nó. A sua fórmula é exibida na Equação (2.3):

$$\textit{Gini}(t) = 1 - \sum_{i=1}^{C} p_i^2 \tag{2.3}$$

Onde:

- *Gini*(t): Representa o índice de Gini do nó t, ou seja, a medida de impureza ou pureza do nó t. Quanto menor o valor do índice de Gini, mais "puro"é o nó, ou seja, os elementos desse nó pertencem a uma única classe.
- C: Número total de classes no conjunto de dados. Em um problema de classificação binária, $C = 2$, mas em problemas com múltiplas classes, C pode ser maior.
- p_i: Proporção de elementos pertencentes à classe i dentro do nó t. Essa proporção é calculada dividindo o número de elementos da classe i pelo total de elementos no nó t.

- Ganho de Informação (*Information Gain*): o ganho de informação é uma métrica baseada na entropia (medição da desordem da informação em um conjunto de dados) e utilizada para medir o nível de eficiência com que um atributo separa os dados (Quinlan, 1986). O método quantifica a redução da impureza ao dividir um conjunto de dados com base em um atributo específico, e sua fórmula geral é dada pela Equação (2.4):

$$\textit{Information Gain}(S, A) = \text{Entropia}(S) - \sum_{v \in \text{Valores}(A)} \frac{|S_v|}{|S|} \cdot \text{Entropia}(S_v) \tag{2.4}$$

Onde:

- *Information Gain*(S, A): Representa o ganho de informação ao dividir o conjunto S com base no atributo A. Quanto maior o ganho, mais efetivamente o atributo separa as classes.
- Entropia(S): Entropia do conjunto original de dados S, antes da divisão. Mede a desordem ou impureza do conjunto como um todo.
- $\sum_{v \in \text{Valores}(A)}$: O somatório é feito sobre todos os valores possíveis v do atributo A. Cada valor define uma partição diferente do conjunto de dados.
- $|S_v|$: Número de elementos no subconjunto S_v, onde o atributo A assume o valor v.
- $|S|$: Número total de elementos no conjunto original S.
- $\frac{|S_v|}{|S|}$: Proporção de exemplos que pertencem ao subconjunto S_v, em relação ao conjunto total S. Essa proporção pondera a contribuição de cada subdivisão na soma total.
- Entropia(S_v): Entropia do subconjunto S_v, ou seja, o nível de impureza do subconjunto gerado ao considerar o valor v do atributo A.

- Razão de Ganho (*Gain Ratio*): métrica derivada do *Information Gain*, proposta por (Quinlan, 1993), com o objetivo de corrigir o viés do ganho de informação, que tende a favorecer atributos com grande quantidade de valores distintos, como identificadores ou códigos. Para isso, o ganho de informação é normalizado pela entropia intrínseca do atributo, resultando na razão entre a informação efetivamente ganha e o potencial de divisão do atributo. Com isso, priorizam-se divisões que geram partições informativas, mas evitam fragmentações excessivas. A fórmula é dada pela Equação (2.5):

$$\textit{Gain Ratio}(S, A) = \frac{\text{Ganho}(S, A)}{\textit{Split Information}(S, A)} \tag{2.5}$$

Onde:

  - *Gain Ratio*(S, A): Representa a razão entre o ganho de informação e a informação intrínseca do atributo A. O atributo selecionado para a divisão será aquele que apresentar o maior ganho de informação.
  - Ganho(S, A): Ganho de informação obtido ao dividir o conjunto S com base no atributo A. Mede a redução de entropia causada pela divisão.
  - *Split Information*(S, A): Informação intrínseca do atributo A, calculada com base na distribuição dos exemplos entre os valores do atributo. É definida pela Equação (2.6) como:

$$\textit{Split Information}(S, A) = -\sum_{v \in \text{Valores}(A)} \frac{|S_v|}{|S|} \cdot \log_2 \left(\frac{|S_v|}{|S|} \right) \tag{2.6}$$

    Onde:

  - $|S_v|$: Número de exemplos no subconjunto S_v, correspondente ao valor v do atributo A.
  - $|S|$: Número total de exemplos no conjunto S.
  - $\frac{|S_v|}{|S|}$: Proporção de elementos do subconjunto S_v em relação ao conjunto total S, usada para ponderar a importância de cada divisão.

Conforme (Patel and Rana, 2014), as AD são aplicadas em várias áreas:

- Negócios: são utilizadas no gerenciamento do relacionamento com o cliente e pontuação de crédito para usuários de cartão de crédito.
- Detecção de intrusão: são usadas para gerar as regras para sistemas de detecção de intrusão.
- Medicina: são úteis no diagnóstico de doenças e de sons cardíacos.
- Veículos inteligentes: no desenvolvimento de veículos inteligentes são utilizadas para encontrar o limite da faixa de rodagem.

- Marketing: são empregadas para a segmentação de clientes e análise de comportamento de compras.
- Educação: são aplicadas para a previsão do desempenho e do risco de evasão de alunos.

Os autores (Rokach and Maimon, 2006) apontam vantagens e desvantagens para o uso de AD como ferramenta de classificação:

✓ São autoexplicativas e quando são compactas são fáceis de seguir, em outras palavras, se a AD tiver um número razoável de folhas, pode ser interpretada por usuários leigos. Além disso, podem ser convertidas em um conjunto de regras do tipo se-então.

✓ Podem lidar com atributos de entrada nominais e numéricos, tornando a tarefa de tratamento inicial dos dados menos complexa.

✓ A apresentação é rica para representar qualquer classificação de valores discretos.

✓ São capazes de lidar com conjuntos de dados que podem conter erros ou valores faltantes.

✓ São consideradas um método não paramétrico, isso significa que as AD não têm suposições sobre a distribuição do espaço e a estrutura do classificador.

✗ Usam o método "dividir para conquistar", tendendo a funcionar bem se existirem alguns atributos altamente relevantes, mas nem tanto se existirem muitas interações complexas. Deste modo, outros classificadores podem descrever de forma compacta, o que seria muito complexo de ser representado por meio de uma AD.

✗ Tendência gananciosa, onde o algoritmo tende a fazer escolhas que parecem ser as melhores no momento, sem considerar o resultado.

✗ Sensibilidade ao conjunto de treinamento, onde pequenas mudanças nos dados de treinamento podem levar a grandes mudanças na estrutura da árvore e nas decisões elencadas.

✗ Sensibilidade a atributos irrelevantes, em outras palavras, as AD podem ser influenciadas por atributos que não têm relevância para a tarefa em questão, levando a árvores muito complexas e difíceis de interpretar, além de serem menos eficientes em fazer previsões corretas.

✗ Árvores muito grandes podem ajustar demais os dados (*overfitting*), enquanto uma árvore pequena pode não capturar uma estrutura importante (*underfitting*).

De outro lado, para (Hastie et al., 2017), uma forma de contornar as desvantagens do uso de AD individuais é utilizar métodos *ensemble*, como florestas aleatórias, que combinam os resultados de várias AD para melhorar o desempenho do modelo.

2.6.5.2 Florestas aleatórias

As Florestas Aleatórias (FA), ou *Random Forests*, segundo (Hastie et al., 2017), são um método de aprendizagem *ensemble*, formado essencialmente por uma coleção de AD, onde cada árvore faz a sua própria previsão. Seu funcionamento se baseia em dois princípios: *bagging* e aleatoriedade na seleção de atributos.

O processo de criação de uma FA é iniciado com a geração de vários subconjuntos de dados a partir do conjunto original, por meio de amostragem com reposição (*bagging*). Para cada subconjunto criado, é treinada uma AD independente (Breiman, 1996).

Durante o treinamento de cada AD, em vez de considerar todos os atributos disponíveis em cada ponto de divisão, é selecionado aleatoriamente um número limitado de atributos, normalmente $\sqrt{n}$ (n = número de atributos). Essa aleatoriedade na seleção dos atributos introduz diversidade entre as AD, o que contribui para a redução da correlação entre elas e melhora o desempenho da floresta (Farnaaz and Jabbar, 2016).

Após o treinamento de todas as AD, a predição é realizada por meio da agregação dos resultados individuais, segundo (Breiman, 2001):

- Classificação: utiliza a votação majoritária, em que a classe mais prevista pelas AD é selecionada como resultado final.
- Regressão: calcula a média aritmética das predições fornecidas por todas as AD da floresta.

Para (Speiser, Miller, Tooze and Ip, 2019), as FA proporcionam melhores resultados em comparação com um único modelo de AD, mantendo algumas das qualidades benéficas dos modelos de árvore, por exemplo, a capacidade de interpretar as relações entre os preditores e o resultado, além de oferecem melhores resultados de previsão em comparação com outros modelos no contexto da classificação.

O desempenho das FA pode ser ajustado por meio da configuração dos seus hiperparâmetros[8], sendo que os principais incluem:

- *n_estimators*: define o número de AD independentes que compõem a FA.
- *max_features*: determina o número máximo de atributos a serem considerados na escolha do melhor ponto de divisão em cada nó da AD.
- *max_depth*: define a profundidade máxima para cada AD da floresta, controlando o crescimento das árvores para evitar que fiquem excessivamente complexas e se ajustem demais aos dados de treinamento (*overfitting*).

[8]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. Acesso em abril/2025.

- *criterion*: especifica a função utilizada para medir a qualidade de uma divisão durante o crescimento das AD. O critério mais comum é o Índice de Gini.

De acordo com (Shah, Patel, Sanghvi and Shah, 2020), as FA ainda contam com as seguintes características:

- Lidam com um grande número de variáveis de entrada.
- As AD nas florestas podem ser criadas em paralelo, porque os conjuntos de dados utilizados são independentes entre cada árvore da floresta.
- Podem ser salvas em disco e carregadas para uso futuro.
- Suportam grandes conjuntos de dados.
- Podem ser criadas localmente por entidades que possuem silos de dados, e a previsão para uma nova instância é obtida por meio da média aritmética das previsões realizadas por todas as FA locais.

Na Figura 2.8 está um exemplo de FA, a qual recebe um conjunto de dados e o distribui entre todas as *N* árvores que a compõem. Após o processamento dos dados pelas árvores individuais, os resultados parciais são compilados e o resultado final é apresentado.

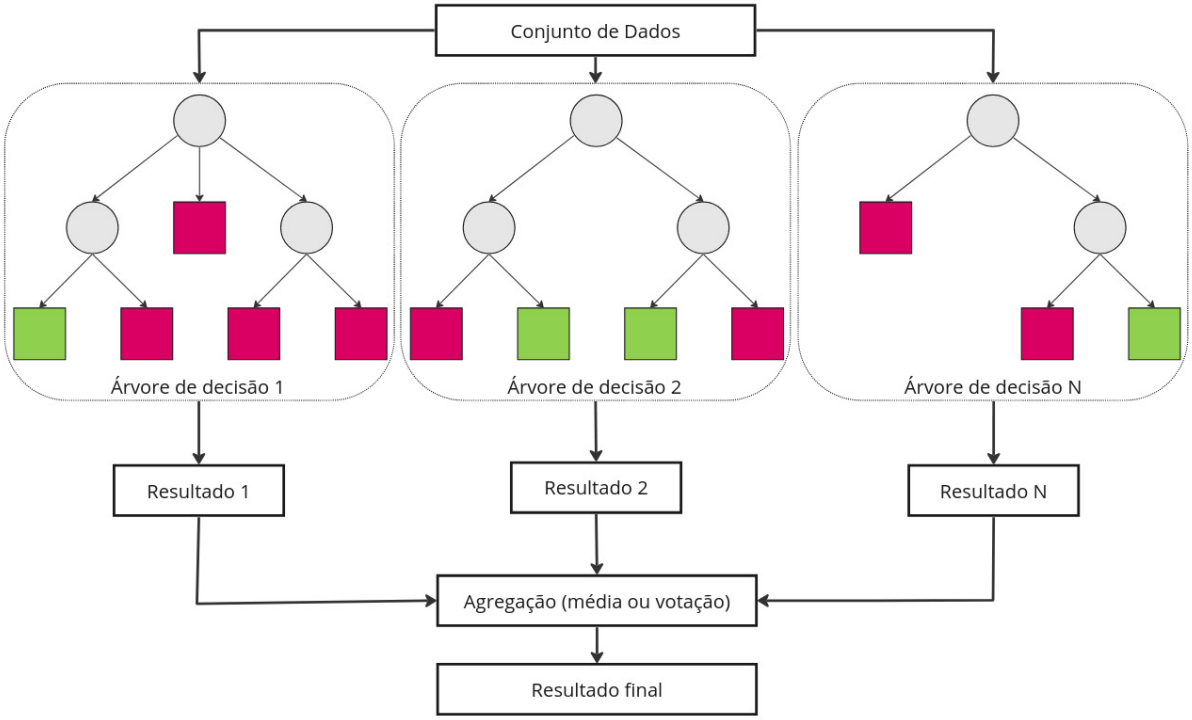

Figura 2.8: Representação de uma floresta aleatória.
Fonte: O autor.

2.6.5.3 *Gradient Boosting Decision Trees*

Segundo (Silva, 2024), o *Gradient Boosting Decision Trees* (GBDT) é um método de AM que combina diversas árvores de decisão fracas (*weak learners*) de forma sequencial. A construção do modelo ocorre de maneira aditiva, com o acréscimo progressivo de árvores, cada uma buscando melhorar o desempenho preditivo ao corrigir os erros das anteriores.

Em cada iteração, calcula-se a função de perda entre as previsões atuais e os valores reais. A partir disso, obtêm-se os gradientes negativos, chamados de pseudo-residuais, que representam a direção e a intensidade dos ajustes necessários. Esses pseudo-residuais são então utilizados como novos rótulos para treinar a próxima árvore da sequência, refinando o modelo passo a passo.

Esse ciclo se repete até que um critério de parada seja atingido, o que pode incluir um número máximo de iterações, ausência de melhoria no desempenho por várias rodadas consecutivas ou outros critérios definidos previamente.

2.6.5.4 *Light Gradient-Boosting Machine*

O *Light Gradient-Boosting Machine* (LightGBM) é um algoritmo baseado no método de GBDT, projetado para ser eficiente em termos de tempo e memória, permitindo o treinamento rápido de modelos mesmo em grandes volumes de dados, com alto desempenho preditivo e o suporte a execução paralela e distribuída.

De acordo com (Ke, Meng, Finley, Wang, Chen, Ma, Ye and Liu, 2017), ao contrário de outras implementações de GBDT que crescem as árvores de forma nivelada (*level-wise*), o LightGBM adota uma abordagem chamada *leaf-wise*, onde a árvore cresce escolhendo, a cada iteração, a folha com o maior ganho de informação. Essa estratégia permite uma redução mais agressiva do erro, o que pode resultar em árvores mais profundas e, consequentemente, modelos mais precisos. Outra característica importante do LightGBM é o uso de técnicas de otimização como:

- *Gradient-based One-Side Sampling* (GOSS): prioriza instâncias com grandes gradientes, que contribuem mais para o cálculo do ganho de informação. Ao manter essas instâncias e amostrar aleatoriamente aquelas com pequenos gradientes, o GOSS reduz o custo computacional sem comprometer significativamente a exatidão do modelo.
- *Exclusive Feature Bundling* (EFB): agrupa os atributos mutuamente exclusivos (aqueles que raramente assumem valores diferentes de zero simultaneamente) em um único atributo. Isso reduz a dimensionalidade do conjunto de dados, tornando o treinamento mais eficiente.

2.6.5.5 *eXtreme Gradient Boosting*

O *eXtreme Gradient Boosting* (XGBoost) é uma implementação otimizada do método de GBDT, projetada para ser altamente eficiente, escalável e preciso, e segue o princípio de *boosting*, onde os modelos são treinados sequencialmente. Segundo seus autores (Chen and Guestrin, 2016), as principais vantagens estão:

- Sistema de boosting regularizado: introduz uma função objetivo regularizada que penaliza a complexidade das árvores (controle do *overfitting*) e melhora a generalização do modelo.
- Divisão eficiente de atributos: utiliza um método de histogramas aproximados para encontrar os melhores pontos de divisão, otimizando o tempo de busca.
- Suporte a paralelismo: implementa paralelismo em nível de divisão de características (*feature split*), permitindo o treinamento em múltiplos núcleos simultaneamente. Suporta também a execução distribuída em *clusters*, o que o torna adequado para *big data* (Sagiroglu and Sinanc, 2013).
- Suporte a dados esparsos: trata nativamente valores ausentes e dados esparsos (dados que têm muitos elementos vazios ou irrelevantes), o que é essencial para aplicações do mundo real, como recomendação e texto.
- Cache e otimizações de baixo nível: usa otimizações como *cache awareness*[9] e estruturas de dados otimizadas para reduzir o custo de computação e uso de memória.

2.6.5.6 *Categorial Boosting*

De acordo com (Prokhorenkova, Gusev, Vorobev, Dorogush and Gulin, 2017), o *Categorial Boosting* (CatBoost) é uma biblioteca de AM baseada no método de GBDT e o seu diferencial em relação a outras bibliotecas de *boosting*, como o XGBoost e o LightGBM é o tratamento nativo de variáveis categóricas, eliminando a necessidade de transformações como *one-hot encoding*[10] ou *label encoding*[11], que podem introduzir viés ou perder informação semântica.

No CatBoost, as árvores de decisão são treinadas sequencialmente, onde cada nova árvore busca corrigir os erros cometidos pelas anteriores. Ele utiliza inovações para resolver problemas comuns em *boosting*, especialmente:

- *Ordered Boosting*: a previsão para uma amostra é feita apenas com base nas amostras anteriores, preservando a independência estatística e evitando vazamento de informação.

[9] Maximiza o uso do cache mais rápido do processador e minimiza o acesso à memória RAM.
[10] Cria uma nova variável binária para cada categoria, com valor 1 para a categoria presente e 0 para as demais.
[11] Converte cada categoria de uma variável em um número inteiro único.

- *Ordered Target Statistics*: converte variáveis categóricas para valores numéricos usando estatísticas baseadas na média da variável alvo, mas faz isso de forma ordenada, ou seja, a codificação de uma instância depende apenas de instâncias anteriores no conjunto de treino.

2.6.6 Algoritmos de avaliação de atributos

Nesta seção são apresentados os principais algoritmos de avaliação de atributos, que têm como objetivo mensurar a relevância de cada atributo de um conjunto de dados, com foco na seleção das características mais informativas.

2.6.6.1 *Information Gain*

O *Information Gain* (IG) é uma métrica utilizada em algoritmos de árvores de decisão para avaliar a relevância de atributos na separação de classes. Ela tem como base o conceito de entropia, introduzido por (Shannon, 1948) em sua obra que quantifica o grau de incerteza ou desordem presente em um conjunto de dados. Esta métrica foi previamente detalhada na Seção 2.6.5.

2.6.6.2 *Gain Ratio*

O *Gain Ratio* (GR) é uma métrica utilizada para selecionar atributos durante a construção de árvores de decisão e é uma evolução do IG, com um ajuste: penaliza atributos com muitos valores únicos (Witten et al., 2017). Detalhes sobre essa métrica podem ser encontrados na Seção 2.6.5.

2.6.6.3 *Chi-Squared*

O teste Qui-Quadrado ou *Chi-Squared* (CS), é uma técnica estatística utilizada para avaliar a dependência entre variáveis. Essa métrica é especialmente útil na seleção de atributos em conjuntos de dados com atributos categóricos, pois permite priorizar aqueles que possuem maior correlação com a classe alvo, contribuindo para a melhoria da performance do modelo.

Segundo (Turhan, 2020), o teste verifica se existe uma associação estatisticamente significativa entre um atributo e a variável de classe, por meio da formulação e avaliação de uma hipótese nula (H_0), que assume que o atributo e a classe são estatisticamente independentes. Se os resultados do teste indicarem que essa hipótese deve ser rejeitada, conclui-se que há evidência de dependência entre o atributo e a classe.

O valor do teste é calculado comparando-se as frequências observadas com as frequências esperadas em uma tabela de contingência. Quanto maior a discrepância entre esses valores, maior será o valor do CS, indicando maior grau de dependência.

2.6.7 Aprendizado centralizado

O Aprendizado Centralizado (AC) é um paradigma em que os dados coletados por dispositivos locais, como sensores, *smartphones* ou outros dispositivos, são transferidos para um servidor central ou *data center*. Nesse ambiente centralizado, ocorre a agregação das bases dos clientes, o processamento das informações e o treinamento dos modelos de AM (Zhu, Xu, Liu and Jin, 2021). Essa abordagem, representada na Figura 2.9, embora tradicionalmente eficaz em diversos contextos, impõe altas exigências em termos de armazenamento e capacidade computacional do servidor, especialmente quando o volume de dados envolvidos é muito grande.

Conforme destacado por (Ferrag et al., 2021), o AC enfrenta desafios, particularmente em contextos envolvendo a Internet das Coisas (IoT). Um dos principais problemas é a questão da privacidade dos dados, uma vez que transferir informações sensíveis de usuários para servidores centrais pode violar diretrizes de proteção de dados. Outro obstáculo é a latência, onde o tempo necessário para transmitir os dados até o servidor, processá-los e retornar os resultados pode ser elevado, o que compromete aplicações que exigem respostas em tempo real.

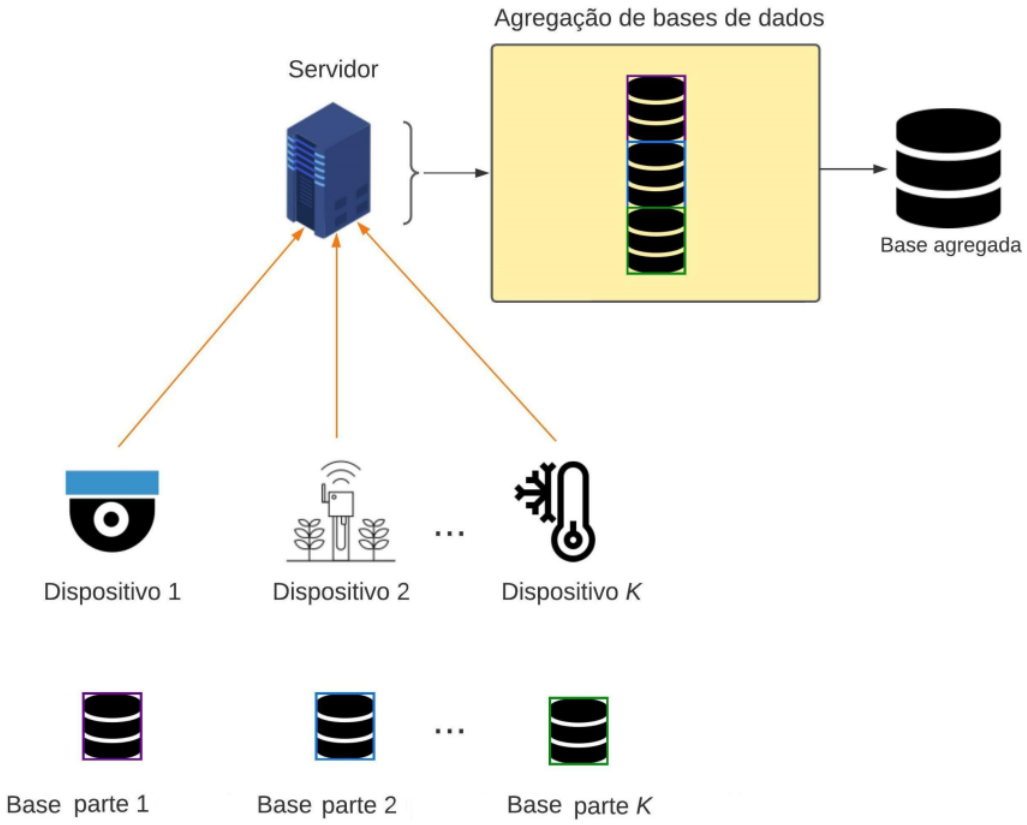

Figura 2.9: Aprendizado centralizado.
Fonte: Adaptado de (Ribera, 2024).

2.6.8 Aprendizado federado

O Aprendizado Federado (AF) ou *Federated Learning* (McMahan et al., 2016) é uma abordagem distribuída de AM que treina modelos em conjuntos de dados descentralizados,

mantidos em dispositivos locais, chamados clientes, sem a necessidade de transferir esses dados para um local centralizado.

O processo de treinamento, representado na Figura 2.10, ocorre localmente em cada dispositivo e apenas os parâmetros dos modelos são retornados para um servidor central, o qual é encarregado da atualização do modelo global. Essa abordagem é útil em cenários em que as informações são sensíveis, como no domínio das áreas médica ou financeira, onde os usuários podem ser relutantes em compartilhá-las devido a preocupações com privacidade, segurança ou atendimento a legislações (McMahan et al., 2016).

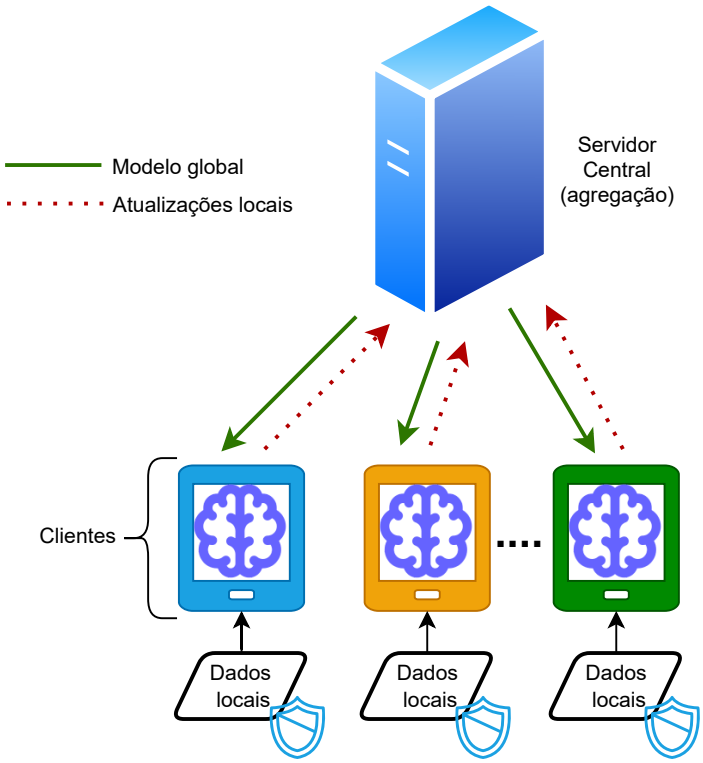

Figura 2.10: Aprendizado federado.
Fonte: O autor.

Um dos exemplos mais conhecidos foi desenvolvido por (Yang, Andrew, Eichner, Sun, Li, Kong, Ramage and Beaufays, 2018), no qual foi utilizado o AF para melhorar os sistemas oferecidos ao usuário pelo Google Keyboard (GBoard). O GBoard [12] é um teclado virtual para dispositivos móveis que apresenta correção automática de texto, digitação por gestos e voz, sugestão e conclusão de palavras, entre outras funcionalidades. O trabalho descreve a experiência de treinamento federado e compara os resultados da implantação do modelo treinado em usuários "ao vivo", demonstrando a capacidade de desenvolver modelos eficazes de maneira vantajosa.

Segundo (McMahan et al., 2016), o funcionamento geral do AF ocorre da seguinte ma-

[12] https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin

neira: há um conjunto fixo de k clientes, cada um com um conjunto local de dados. No início de cada rodada, uma fração aleatória c de clientes é selecionada e o servidor envia o estado atual do modelo global para cada um desses clientes. Cada cliente selecionado executa então a computação local com base no modelo recebido e no seu conjunto local de dados, e envia uma atualização para o servidor. O servidor então aplica essas atualizações ao seu modelo global e o processo se repete.

Além de preservar a privacidade dos dados, o AF também pode ser mais eficiente em termos de largura de banda, pois reduz a necessidade de transferir grandes volumes de dados pela rede. Isso é especialmente útil em cenários onde a conectividade de rede pode ser cara ou limitada, como em dispositivos IoT. Além disso, é possível aumentar o poder de processamento adicionando mais clientes que trabalham de forma independente entre cada rodada de comunicação ou com o aumento da carga de processamento em cada cliente, onde em vez de realizar apenas alguns cálculos, cada cliente realiza operações mais complexas entre cada rodada de comunicação (McMahan et al., 2016).

2.6.8.1 Paradigma centralizado e descentralizado

De acordo com (Kairouz et al., 2021), o AF pode ser estruturado em dois paradigmas, cada um com características próprias de coordenação e comunicação entre os participantes do processo de treinamento. Esses paradigmas definem a forma como os modelos são atualizados e sincronizados entre os clientes, além de implicarem diferentes requisitos em termos de infraestrutura e controle:

- Centralizado: nessa configuração, um servidor central coordena o treinamento, agregando as contribuições de diversos clientes. Embora essa abordagem seja mais adotada devido à sua eficiência e simplicidade, ela apresenta um ponto único de falha e exige a presença de uma autoridade central confiável.
- Descentralizado: nessa organização, a coordenação central é substituída pela comunicação ponto a ponto entre os clientes. Cada cliente realiza atualizações locais e interage apenas com seus vizinhos. Embora promova autonomia aos participantes, o ambiente descentralizado ainda pode demandar certo grau de centralização, especialmente na definição das tarefas, na escolha de algoritmos e na resolução de falhas.

2.6.8.2 Aprendizado federado horizontal e vertical

De acordo com os autores (Zhu et al., 2021), o AF pode ser categorizado conforme a distribuição dos dados entre os clientes em:

- Horizontal: neste contexto, os dados de treinamento dos clientes compartilham o mesmo espaço de atributos, mas têm espaços amostrais diferentes. No exemplo mostrado na

Figura 2.11, o cliente 1 e o cliente 2 contêm diferentes registros, porém, todos têm os mesmos atributos (*features*).

Características (features)

registros

# Evento	Flow_Byts/s	Fwd_Header_Len	Flow_Duration	Rótulo	
1	31	10	31	Normal	cliente 1
2	5	26	5	Anormal	
3	24	24	24	Normal	
4	10	36	10	Anormal	cliente 2
5	22	15	22	Normal	
6	2	10	2	Anormal	

Figura 2.11: Aprendizado federado horizontal.
Fonte: Adaptado de (Zhu et al., 2021).

- Vertical: os dados de treinamento dos usuários compartilham o mesmo espaço amostral (registros), mas têm atributos (*features*) diferentes. Conforme mostrado na Figura 2.12, o cliente 1 e o cliente 2 têm as mesmas amostras de dados, porém com atributos diferentes.

  Ao contrário do AF horizontal, onde todos os clientes têm seus próprios rótulos de dados, no AF vertical é frequentemente assumido que apenas um cliente armazena todos os rótulos de dados. Os clientes com rótulos de dados são chamados de *guest* ou parte passiva, enquanto os clientes sem rótulos de dados são chamados de *host* ou parte ativa (Zhu et al., 2021).

Características (features)

registros

# Evento	Flow_Byts/s	Fwd_Header_Len	Rótulo	# Evento	Flow_Duration
1	31	10	Normal	1	31
2	5	26	Anormal	2	5
3	24	24	Normal	3	24
4	10	36	Anormal	4	10
5	22	15	Normal	5	22
6	2	10	Anormal	6	2
cliente 1				cliente 2	

Figura 2.12: Aprendizado federado vertical.
Fonte: Adaptado de (Zhu et al., 2021).

Além das distribuições dos dados do cliente, segundo (Zhu et al., 2021) existem diferenças entre o AF horizontal e o AF vertical:

- No AF horizontal, é comum a presença de um servidor central, responsável por coordenar

e agregar os modelos treinados localmente por cada cliente, formando um modelo global. No AF vertical, embora a presença de um servidor não seja obrigatória, a coordenação geralmente é realizada pelo cliente designado *guest*, que colabora com os demais *hosts* para construir um modelo conjunto. Nesse caso, a agregação ocorre de forma distribuída, e o modelo global resulta da combinação de saídas parciais dos dados.

- No AF horizontal, os parâmetros dos modelos treinados localmente são compartilhados com o servidor central, que os agrega e distribui um modelo atualizado aos clientes. No AF vertical, por outro lado, os modelos locais são treinados sobre diferentes subconjuntos de atributos. Assim, o cliente *guest* recebe as saídas parciais dos *hosts*, processa essas informações e as utiliza para atualizar o modelo colaborativo.
- Em uma rodada de comunicação de AF horizontal, normalmente há apenas uma interação entre servidor e clientes: os modelos são enviados ao servidor, agregados e o modelo global atualizado é redistribuído. No AF vertical, o processo exige múltiplas interações entre o *guest* e os *hosts* dentro de uma única rodada, devido à necessidade de sincronização das partes do modelo, o que torna a comunicação mais intensa.

2.6.8.3 Dados IID e non-IID

A apresentação dos conceitos ao longo desta seção é baseada no material de (Neto, Dusparic, Mattos and Fernandes, 2022).

Um desafio fundamental na concepção de IDS federados é que os dados usados para treinar o modelo global é altamente não independente e não distribuído de forma idêntica (non-IID).

Dados coletados por cada dispositivo IoT podem ser tendenciosos para o ambiente desse dispositivo. Por exemplo, se uma empresa de segurança coletar dados de organizações em diferentes regiões geográficas, os padrões de tráfego podem variar significativamente: empresas em uma região com alto índice de ataques cibernéticos podem ter padrões de tráfego diferentes daquelas em áreas com baixa incidência de ataques. Esta característica é denominada **não independente e não distribuída de forma idêntica** (non-IID).

Em contraste ao non-IID, os dados **independentes e distribuídos de forma idêntica** (IID) possuem a mesma distribuição de probabilidade e os mesmos são independentes uns dos outros, ou seja, são homogêneos. A comparação entre essas formas de distribuição dos dados pode ser vista nos exemplos da Figura 2.13, onde, no lado esquerdo, o conjunto de dados está distribuído de forma equilibrada, e, no lado direito, há uma desordem na distribuição dos dados.

Dados non-IID representam desafios para a otimização da convergência do modelo treinado, porque atrasam o aprendizado de novos padrões de ataques. Além disso, não são todos os participantes que ajudam na formação do modelo global, no entanto, a seleção dos participantes não é trivial, uma vez que os dados sempre permanecem privados para cada participante.

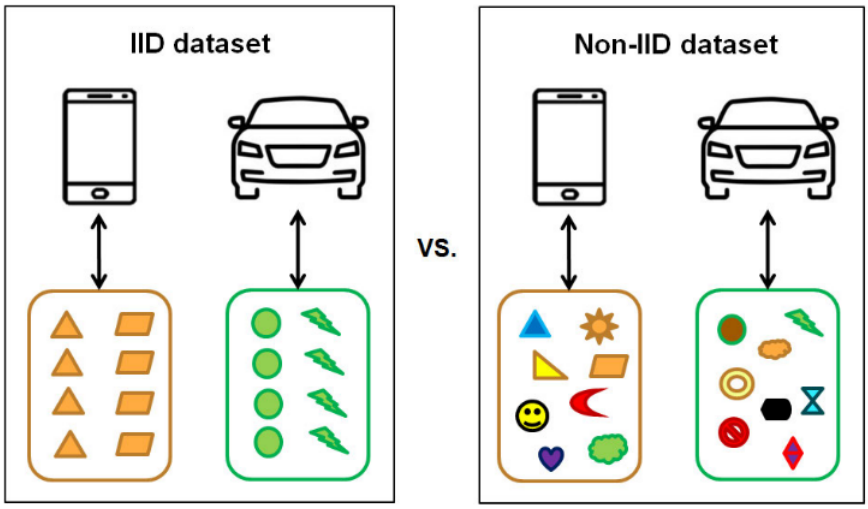

Figura 2.13: Distribuição de dados IID e non-IID.
Fonte: Adaptado de (Orlandi, Anjos, Leithardt, Santana and Geyer, 2023).

2.6.9 Avaliação de modelos

Medir os resultados dos algoritmos de AM é crucial por vários motivos, entre eles a identificação de problemas e assegurar a qualidade e a exatidão dos modelos antes de serem implementados em ambientes de produção. Nas próximas seções serão apresentadas as principais abordagens utilizadas para a avaliação de modelos de AM.

2.6.9.1 *k-fold Cross-validation*

Segundo (Ertel, 2017), o processo de *k-fold Cross-validation* consiste na divisão aleatória do conjunto de dados em *k folds* (*k* subconjuntos de dados), com aproximadamente o mesmo tamanho. Na sequência, são realizadas *k* iterações, utilizando *k*-1 *folds* para o treinamento do modelo de AM e o *fold* restante para teste. Deste modo, cada *fold* é utilizado tanto para treinamento quanto para teste e, ao final, os resultados das iterações são utilizados para calcular uma média do desempenho global do modelo. A Figura 2.14 ilustra este processo.

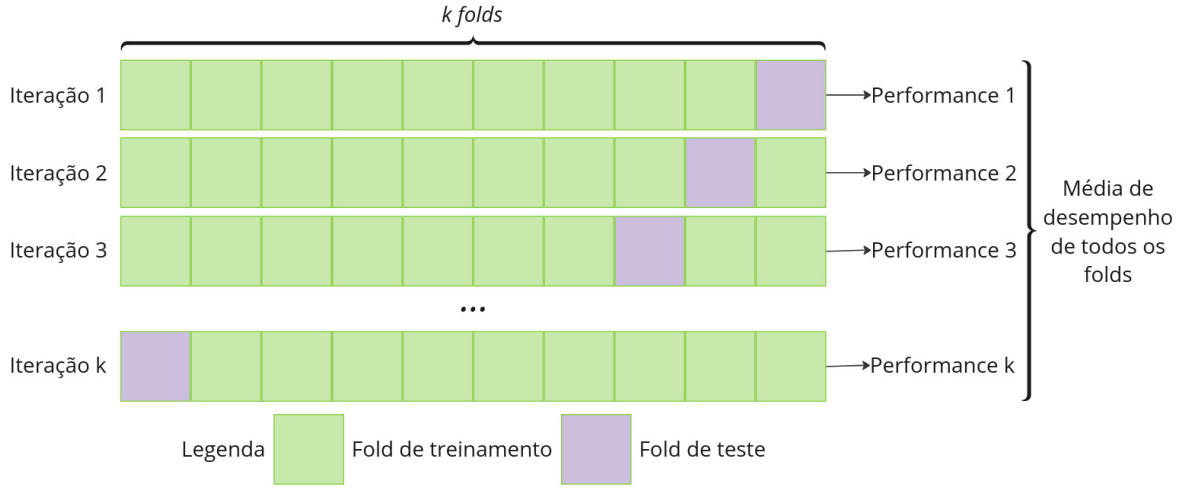

Figura 2.14: Validação cruzada (*Cross-validation*).
Fonte: O autor.

De acordo com (Witten et al., 2017), para evitar o problema de omitir todos os exemplos de uma determinada classe no treinamento, é fundamental garantir que a amostragem seja realizada de maneira que cada classe seja adequadamente representada tanto nos conjuntos de treinamento quanto nos de teste. A validação cruzada estratificada (*stratified k-fold cross-validation*) garante que cada *fold* mantenha a proporção de classes do conjunto original de dados, fundamental quando há classes desbalanceadas.

2.6.9.2 *Leave-One-Out Cross-Validation*

Conforme (Witten et al., 2017), o *Leave-One-Out Cross-Validation* (LOOCV) é uma forma específica de validação cruzada do tipo *n-folds*, em que *n* corresponde ao número total de instâncias no conjunto de dados. Nesse método, cada instância é sequencialmente excluída do conjunto de treinamento, e o modelo de aprendizado é treinado com todas as demais instâncias. A avaliação do modelo é realizada com base na instância que foi deixada de fora, sendo atribuída uma pontuação binária: 1 em caso de acerto e 0 em caso de erro. Esse processo é repetido para cada uma das *n* instâncias, e a média dos resultados obtidos constitui a estimativa final do erro do modelo.

Uma das principais vantagens do LOOCV é que ele utiliza quase todos os dados disponíveis para o treinamento em cada iteração, o que tende a produzir estimativas com baixo viés. No entanto, uma desvantagem relevante é o alto custo computacional, especialmente em conjuntos de dados grandes, uma vez que o modelo precisa ser treinado *n* vezes — uma para cada instância.

2.6.9.3 *Holdout*

Segundo (Tantithamthavorn, McIntosh, Hassan and Matsumoto, 2017), o método *Holdout* é uma das abordagens mais simples e amplamente utilizadas para avaliação de desempenho de

modelos de classificação. Consiste na divisão aleatória do conjunto de dados em dois subconjuntos mutuamente exclusivos: o conjunto de treinamento e o conjunto de teste, segundo uma proporção predefinida (por exemplo, 70% para treinamento e 30% para teste). O modelo é treinado exclusivamente com os dados de treinamento e, em seguida, avaliado com os dados de teste, que não participaram do processo de aprendizagem, a fim de estimar sua taxa de erro, acurácia e capacidade de generalização.

No contexto do método *Holdout*, quanto maior a quantidade de dados destinada ao treinamento, maior tende a ser a capacidade do modelo de aprender os padrões dos dados, o que pode resultar em uma performance aparentemente superior. No entanto, essa prática deve ser equilibrada com a necessidade de um conjunto de teste suficientemente representativo, a fim de garantir uma avaliação confiável da capacidade de generalização do modelo.

2.6.9.4 Métricas de desempenho

As definições apresentadas nesta seção têm como base a obra de (Kelleher, Namee and D'Arcy, 2015), que fornece uma abordagem sobre as métricas de desempenho e avaliação de algoritmos de AM.

A matriz de confusão é uma ferramenta de análise útil para capturar o que acontece em um teste de avaliação com um pouco mais de detalhes e é a base para o cálculo de outras medidas de desempenho. Nela são exibidas as frequências de cada possível resultado das previsões feitas por um modelo para um conjunto de dados de teste, a fim de mostrar em detalhes como o modelo está funcionando. Na Tabela 2.1 é demonstrado um exemplo de matriz de confusão.

Tabela 2.1: Matriz de confusão

		Classe Predita	
		Positiva	Negativa
Classe real	Positiva	VP	FN
	Negativa	FP	VN

Em uma matriz de confusão, os resultados das previsões para um modelo de classificação binária são comparados em quatro categorias diferentes:

- Verdadeiro Positivo (VP): são os casos em que o modelo previu corretamente que a classe é positiva (verdadeiro) e a classe real também é positiva.
- Falso Positivo (FP): são os casos em que o modelo previu incorretamente que a classe é positiva, quando a mesma é negativa.
- Verdadeiro Negativo (VN): são os casos em que o modelo previu corretamente que a classe é negativa (verdadeiro) e a classe real também é negativa.
- Falso Negativo (FN): são os casos em que o modelo previu incorretamente que a classe é negativa, quando na verdade é positiva.

Segundo os mesmos autores, os estatísticos muitas vezes se referem aos falsos positivos (FP) como erros do tipo I e aos falsos negativos (FN) como erros do tipo II.

Para calcular os resultados dos modelos e compará-los com diferentes técnicas, métricas de desempenho são usadas, sendo as mesmas derivadas da matriz de confusão. Conforme (Thakkar and Lohiya, 2022), algumas das métricas utilizadas para avaliação modelos de classificação são as seguintes:

- Acurácia (Equação (2.7)): pode ser definida como a taxa de classificação do modelo que é dada pela proporção de instâncias classificadas corretamente (VP + VN) em relação ao número total de instâncias no conjunto de dados (VP + VN + FP + FN).

$$\textit{Acurácia} = \frac{VP + VN}{VP + VN + FP + FN} \tag{2.7}$$

- Precisão (Equação (2.8)): é a medida dada pela razão de instâncias corretamente identificadas (VP) à soma das instâncias classificadas como corretas (VP + FP).

$$\textit{Precisão} = \frac{VP}{VP + FP} \tag{2.8}$$

- Taxa de falsos positivos (Equação (2.9)): é uma métrica de erro definida como a proporção do número de instâncias que são classificadas incorretamente (FP) para a soma de falsos positivos e verdadeiros negativos (FP + VN).

$$\textit{TFP} = \frac{FP}{FP + VN} \tag{2.9}$$

- Taxa de falsos negativos (Equação (2.10)): é uma métrica de erro definida como a proporção do número de instâncias que são classificadas incorretamente (FN) para a soma de falsos negativos e verdadeiros positivos (FN + VP).

$$\textit{TFN} = \frac{FN}{FN + VP} \tag{2.10}$$

- Sensibilidade/Recall (Equação (2.11)): consiste na taxa de verdadeiros positivos, e é definido como a proporção de classificações feitas corretamente (VP) para a soma de verdadeiros positivos e falsos negativos (VP + FN).

$$\textit{Sensibilidade/Recall} = \frac{VP}{VP + FN} \tag{2.11}$$

- Especificidade (Equação (2.12)): definida como a taxa de verdadeiros negativos (VN), considerando a proporção de amostras negativas classificadas corretamente (VN) pela soma de amostras de negativos verdadeiros e falsos positivos (VN + FP).

$$\textit{Especificidade} = \frac{VN}{VN + FP} \tag{2.12}$$

- F1-score (Equação (2.13)): é uma métrica de desempenho que representa a média harmônica entre precisão e sensibilidade. Deste modo, oferece uma avaliação geral do desempenho de um modelo de classificação, sendo útil quando se deseja equilibrar a importância de detectar corretamente exemplos positivos e minimizar falsos positivos, uma vez que a precisão mede a exatidão das previsões positivas e a sensibilidade mede a capacidade do modelo em encontrar todos os exemplos positivos.

$$\textit{F1-score} = 2\ X\ \frac{\textit{precisão} X\, sensibilidade}{\textit{precisão} + sensibilidade} \tag{2.13}$$

2.6.9.5 Testes estatísticos

Em uma avaliação estatística, primeiramente é necessário identificar se o conjunto de dados segue uma distribuição normal, sendo recomendado o uso do teste Shapiro-Wilk (Shapiro and Wilk, 1965). Em função do resultado de referido teste, identifica-se qual categoria de método de análise estatística aplicar, sendo utilizados testes paramétricos em caso de distribuição normal e testes não paramétricos no outro caso. De acordo com o método, utiliza-se a seguinte formulação de hipóteses:

- H_0 (hipótese nula): A amostra possui uma distribuição normal.
- H_1 (hipótese alternativa): A amostra não possui uma distribuição normal.

Neste trabalho foi aplicado um nível de significância do teste de 95%. Portanto, caso o *p-value* obtido pelo método seja:

- > 0,05: H_0 é aceita e pode-se afirmar com nível de significância de 95% que a amostra provém de uma distribuição normal.
- <= 0,05: H_0 é rejeitada e pode-se afirmar com nível de significância de 95% que a amostra não provém de uma distribuição normal.

Após a verificação da normalidade dos dados, deve-se aplicar testes estatísticos apropriados para avaliar a existência de diferenças significativas entre as amostras. Com base nos resultados da análise de normalidade, indicam-se os seguintes testes:

- Análise de Variância (ANOVA) (Fisher, 1992): Teste paramétrico utilizado para os grupos que sejam independentes, caso as amostras sigam uma distribuição normal;
- Teste de Kruskal-Wallis (Kruskal and Wallis, 1952): Aplicável caso os resultados não derivem de uma distribuição normal.

Assim como no teste estatístico de normalidade, os testes ANOVA e Kruskal-Wallis são testes de hipóteses, e considera-se que:

- H_0 (hipótese nula): As médias dos grupos ou as distribuições são iguais.
- H_1 (hipótese alternativa): Existe pelo menos uma das médias ou distribuições que é diferente.

Para a comparação das médias dos grupos também é utilizado o nível de significância do teste de 95%. Desta forma, caso o *p-value* obtido pelo método de comparação seja:

- > 0,05: H_0 é aceita e conclui-se com nível de significância de 95% que não há diferenças estatísticas significativas entre os grupos;
- <= 0,05: H_0 é rejeitada e pode-se afirmar com nível de significância de 95% que ao menos um dos grupos possui diferença estatisticamente relevante.

Quando a H_0 é rejeitada no teste ANOVA ou no teste Kruskal-Wallis, conclui-se que ao menos um dos grupos difere dos demais. Nesse contexto, recomenda-se a aplicação de um pós-teste para identificar quais grupos apresentam diferenças estatisticamente significativas entre si.

Para dados que atendem aos pressupostos de normalidade, ou seja, quando é utilizado o teste ANOVA, um pós-teste recomendado é a correção de Bonferroni (Armstrong, 2014), que ajusta os valores de probabilidade (p) devido ao aumento do risco de um erro do tipo I. Em situações em que os dados não seguem uma distribuição normal e foi utilizado o teste não paramétrico de Kruskal-Wallis, o pós-teste de Dunn (Everitt and Dunn, 2001) é o mais indicado, por ser adequado para comparações múltiplas entre medianas.

2.7 Considerações finais

Neste capítulo, abordou-se os fundamentos teóricos que sustentam o desenvolvimento do método para detecção de intrusão em ambientes IoT, com ênfase na aplicação integrada do aprendizado federado e florestas aleatórias. Foram discutidos os principais conceitos relacionados à segurança da informação, com destaque para os sistemas de detecção de intrusão e suas diferentes abordagens.

Adicionalmente, foi detalhado o processo de descoberta de conhecimento em bases de dados (KDD), fundamental para a preparação dos dados empregados em modelos de aprendizado de máquina. No âmbito da inteligência artificial, abordaram-se os paradigmas de aprendizado de máquina, as técnicas de *ensemble learning* e os principais algoritmos baseados em árvores de decisão. O capítulo também apresentou uma comparação entre os paradigmas de aprendizado centralizado e federado, destacando suas características e limitações no que tange à distribuição dos dados.

Dessa forma, os conceitos e técnicas aqui discutidos constituem o arcabouço teórico necessário para a compreensão e o desenvolvimento do método proposto, cuja implementação e avaliação serão aprofundadas nos capítulos subsequentes.

Capítulo 3

Trabalhos Relacionados

3.1 Considerações iniciais

Este capítulo apresenta trabalhos de estado da arte que exploram soluções baseadas em aprendizado de máquina (AM), aprendizado federado (AF) e computação distribuída, destacando as principais abordagens e contribuições para o avanço dessas áreas.

3.2 Estado da arte

Os autores (Liu, Liang, Liu, Liu, Meng, Zhang and Zheng, 2022) se concentraram no AF vertical, apresentando seu modelo denominado *Federated Forest* (FF), baseado no método de florestas aleatórias e que permite que diferentes clientes construam um modelo global sem trocar seus dados.

O servidor master seleciona aleatoriamente um conjunto de dados e de características (*features*), e os distribui entre os clientes, que podem variar entre um e oito. Cada cliente constrói e testa um modelo local baseado em árvores, utilizando o subconjunto de características recebido. Após o treinamento, os clientes enviam ao servidor os detalhes do melhor ponto de *split* (divisão) encontrado. O servidor, por sua vez, analisa e encontra o melhor *split* entre todos os recebidos, e o melhor é incorporado ao modelo global.

Os experimentos foram aplicados em problemas de classificação e de regressão utilizando conjuntos de dados da Universidade da Califórnia Irvine (UCI), além da utilização de dados reais que tiveram as informações confidenciais criptografadas antes do uso. A métrica utilizada para avaliação dos experimentos foi a acurácia, que chegou a 99,50%.

O modelo aplica métodos de criptografia para proteger a privacidade e a segurança dos dados durante a comunicação entre as partes, e foi considerado uma solução eficiente e robusta, quando comparada a uma versão não federada, com possibilidade de aplicabilidade em ambiente real.

Utilizando uma abordagem federada e descentralizada para a criação de florestas aleatórias, (Souza, Rebello, Camilo, Guimaraes and Duarte, 2020) apresentaram o *Decentralized Federated Forest* (DFedForest), que usa *blockchain* para compartilhar referências aos modelos de árvores de decisão criados localmente por cada domínio.

Os domínios (clientes) são as partes encarregadas de criar modelos de árvores por meio de *bootstrap* usando seus dados, compartilhar esses modelos por meio do *blockchain* e consultar novas árvores de decisão para aprimorar a floresta aleatória local. Antes de adicionar árvores à floresta local, o domínio usa parte de seu conjunto de dados para validar os novos modelos e, se o resultado da validação for maior que um limite predefinido, o domínio adiciona o novo modelo à floresta local. A floresta aleatória federada final em cada domínio inclui árvores de decisão criadas localmente e por outros domínios.

A implementação utilizou o *Hyperledger Fabric 2.0*[1], um *framework open source* para a construção de *blockchains*, e os experimentos realizados para detectar intrusões usaram o conjunto de dados CTU-13[2], que tem amostras de tráfego normal e malicioso gerado por *botnets*.

Os autores afirmam que o sistema garante a privacidade dos dados, a confiabilidade dos modelos e a detecção de comportamentos maliciosos, além de baixa complexidade computacional e alta exatidão, tendo chegado ao resultado de 98% para o F1-score.

Considerando a detecção de intrusão em dispositivos IoT, os autores (Mothukuri, Khare, Parizi, Pouriyeh, Dehghantanha and Srivastava, 2022) abordaram o AF para as tarefas de classificação de eventos intrusivos por meio de modelos *Long short-term memory* (LTSM) e *Gated recurrent units* (GRU), que são variações de redes neurais e, um método *ensemble* baseado em florestas aleatórias para agregar as atualizações dos modelos locais e otimizar o modelo global.

Nesse trabalho, os dispositivos IoT são simulados por instâncias construídas utilizando o *PySyft*[3], uma biblioteca *open source* que provê aprendizado seguro e privado, realizando os treinamentos e testes no modelo GRU local e periodicamente compartilhando os pesos da sua rede para a agregação pelo servidor central. O servidor central, após realizar a atualização dos pesos, envia novamente os modelos para cada dispositivo para uma nova rodada de treinamentos e de testes, sendo o resultado final criado por meio de votação usando o *ensemble* baseado em florestas aleatórias, que combina as probabilidades fornecidas como entrada para a função de predição.

Segundo os autores, o resultado de 90,25% para a acurácia demonstrou que a abordagem proposta supera a versão não federada dos algoritmos de detecção de intrusão, e o desempenho da abordagem é melhorado ainda mais com a combinação de previsões de diferentes camadas das GRUs, além de manter a privacidade dos dados.

Os autores (Wardana, Sukarno, Basuki and Utomo, 2024) investigaram o uso do conjunto de dados CIC-IoT2023 [4] e de florestas aleatórias federadas com técnicas de seleção de atributos para a detecção de intrusão em dispositivos IoT. O modelo, denominado *Federated Random Forest* (FRF), combina os princípios do AF com a abordagem de florestas aleatórias em fontes de dados descentralizadas.

[1] https://hyperledger-fabric.readthedocs.io. Acesso em abril/2025.
[2] https://www.stratosphereips.org/datasets-ctu13. Acesso em abril/2025.
[3] https://docs.openmined.org/en/latest/index.html. Acesso em abril/2025.
[4] https://www.unb.ca/cic/datasets/iotdataset-2023.html. Acesso em abril/2025.

No pré-processamento, valores ausentes e zeros foram removidos, seguidos da exclusão de atributos redundantes que não agregavam valor ao modelo. Em seguida, os rótulos dos trinta e três tipos de ataques foram agrupados em sete categorias. Para lidar com o grande volume de dados, foram utilizados apenas 30% dos eventos.

Um processo de seleção de atributos baseado em florestas aleatórias foi empregado, resultando na escolha de atributos com base em seus valores de importância. Esses atributos foram distribuídos uniformemente entre todos os clientes participantes, na proporção 80% para treinamentos e 20% para testes.

O estudo envolveu três clientes e cinco rodadas de treinamentos e de testes. Cada cliente treinou independentemente uma árvore de decisão com base em parâmetros fornecidos pelo servidor central. O nó central então agregou os modelos individuais dos clientes por meio da técnica FedAvg (McMahan et al., 2016), melhorando o modelo a cada rodada.

Os resultados experimentais mostraram que o modelo proposto atingiu uma acurácia de aproximadamente 99,68%, demonstrando um bom equilíbrio e confiabilidade na detecção de intrusões.

Para a detecção de malwares em dispositivos IoT, (Rey, Sánchez, Celdrán and Bovet, 2022) apresentaram sua pesquisa baseada em AF utilizando o conjunto de dados N-BaIoT[5] em um cenário para uso em redes celulares 5G, nuvem ou *Fog*[6].

O modelo usa dois tipos de modelos de AM: um classificador supervisionado que utiliza redes *multi layer perceptron* (MLP) e um não supervisionado baseado em *autoencoders*, ambos baseados em redes neurais e distribuídos pelos oito clientes que participam do processo. O *framework* estuda as questões de segurança inerentes ao AF, como a presença de participantes maliciosos que podem degradar o modelo federado.

Os resultados mostraram que o uso do AF permite detectar malware em dispositivos, preservando a privacidade dos participantes e com resultados similares aos de uma abordagem centralizada, chegando em alguns casos a 99,96% de acurácia. Também são feitos testes com algumas funções de agregação que melhoram a robustez do modelo federado.

Os pesquisadores (Campos, Saura, González-Vidal, Hernández-Ramos, Bernabé, Baldini and Skarmeta, 2022) apresentaram uma avaliação abrangente do uso de AF para IDS em dispositivos IoT. Considerando diferentes distribuições de dados e métodos de agregação, utilizaram o conjunto de dados ToN_IoT [7] no ambiente IBM *Federated Learning*[8], um *framework* para ambiente empresarial. A simulação considerou dez dispositivos IoT e um servidor central, ambos executando tarefas de AF de forma paralela por até trezentas repetições.

[5]https://doi.org/10.24432/C5RC8J. Acesso em abril/2025.

[6]Forma de computação distribuída que aproxima a computação e o armazenamento de dados da extremidade da rede.

[7]https://research.unsw.edu.au/projects/toniot-datasets. Acesso em abril/2025.

[8]https://www.ibm.com/docs/en/watsonx/saas?topic=models-federated-learning. Acesso em abril/2025.

Foram propostos três cenários de classificação multiclasses dos eventos intrusivos, sendo que o primeiro cenário, básico, é caracterizado por uma distribuição de dados non-IID e altamente distorcida, o segundo cenário, equilibrado, onde cada participante tem o mesmo número de exemplos de cada classe e o terceiro cenário, misto, gerado para alcançar um *tradeoff* entre os dois primeiros.

Os resultados demonstraram o impacto que distribuições de dados não idênticas e altamente distorcidas no desempenho do AF, o que afeta diretamente a eficácia da detecção de eventos intrusivos: dispositivos com dados desbalanceados podem ter alta acurácia, mas isso pode não ser representativo devido à predominância de uma classe majoritária. Um processo de seleção de instâncias baseado em entropia em cada conjunto de dados local conseguiu melhorar os resultados, chegando a uma acurácia de aproximadamente 91%, obtendo resultados semelhantes quando comparado a um cenário onde o conjunto de dados está equilibrado entre as partes.

A pesquisa de (Neto et al., 2022) propõe o uso de uma meta-heurística chamada *Federated Simulated Annealing* (FedSA) para otimizar, a cada rodada de execução, a seleção de participantes, a taxa de aprendizado e as atualizações locais em um ambiente de AF.

O *Simulated Annealing*[9] é um método probabilístico para encontrar o mínimo global de uma função, e no contexto do estudo é usado para selecionar um subconjunto ótimo de hiperparâmetros, visando acelerar a convergência do modelo global. A ideia é que, em vez de usar todos os participantes disponíveis para cada rodada de treinamento (o que pode ser computacionalmente caro e demorado), seja selecionado um subconjunto de participantes que provavelmente produzirão um modelo global de alta qualidade.

No trabalho foram simulados dois cenários: o primeiro com cem participantes, em que foram selecionados 50% do total e, no segundo cenário, o número de participantes é aumentado para cento e cinquenta, sendo quarenta participantes selecionados, aproximadamente 27%. Ambos cenários foram submetidos ao FedSA e ao FedAvg (método de agregação que calcula a média dos pesos) (McMahan et al., 2016).

Os resultados mostraram que o FedSA, selecionando 27% dos participantes, alcançou 96% de acurácia em três rodadas de agregação, enquanto o FedAvg, selecionando 50% dos participantes, executou oito rodadas de agregações. A solução proposta atinge os máximos resultados em cinco rodadas de agregação, 50% mais rápida que a tradicional.

Utilizando florestas aleatórias, os autores (Markovic, Leon, Buffoni and Punnekkat, 2022) consideraram os conjuntos de dados KDD[10], NSL-KDD[11], UNSW-NB15[12] e CIC-IDS2017[13] para a detecção de intrusão em um ambiente federado horizontal.

[9]https://www.mathworks.com/help/gads/what-is-simulated-annealing.html. Acesso em abril/2025.
[10]http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Acesso em abril/2025.
[11]https://www.unb.ca/cic/datasets/nsl.html. Acesso em abril/2025.
[12]https://research.unsw.edu.au/projects/unsw-nb15-dataset. Acesso em abril/2025.
[13]https://www.unb.ca/cic/datasets/ids-2017.html. Acesso em abril/2025.

Para cada conjunto de dados, uma característica (*feature*) foi escolhida como critério de divisão, sendo que esta é removida do conjunto de treinamento usado pelos modelos locais para a construção de sua floresta aleatória local.

Os conjuntos de dados foram divididos nas proporções de 70% para treinamentos, 10% para validações e 20% para testes, sendo que o conjunto de validação foi utilizado para avaliar a performance das florestas aleatórias criadas nos clientes por meio das métricas de acurácia e acurácia ponderada, sendo as melhores enviadas ao servidor central para agregação.

Diversas estratégias de agregação foram avaliadas para compor a floresta aleatória global, a partir das florestas aleatórias recebidas dos clientes, mas o melhor resultado foi alcançado ao ordenar, considerando a métrica acurácia, as árvores de decisão de cada floresta e selecionar as mais eficientes. Essa abordagem permitiu que a floresta aleatória global atingisse 93,51% de acurácia.

Em seu estudo, (Friha, Ferrag, Shu, Maglaras, Choo and Nafaa, 2022) apresentaram seu modelo chamado *Federated learning-based intrusion detection system for agricultural Internet of Things* (FELIDS), um sistema de detecção de intrusão baseado em AF para proteger infraestruturas de IoT agrícolas e projetado para ser usado na camada de borda da rede (*edge layer*), tendo utilizado os conjuntos de dados CIC-IDS2018[14], MQTTset[15] e InSDN[16].

O modelo utiliza (I) redes neurais profundas (DNN) variando entre 64~80 nós ocultos, com uma ou duas camadas ocultas, (II) redes neurais convolucionais (CNN) variando entre 120~130 nós ocultos, com duas ou três camadas convolucionais e (III) redes neurais recorrentes (RNN) variando entre 20~80 nós ocultos e duas camadas ocultas LTSM.

O número de clientes (chamados de *edge nodes*) utilizados no AF foram cinco, dez e quinze, além de cinquenta *rounds* para a execução federada. O funcionamento de forma ampla atende aos princípios do AF, onde um modelo genérico é inicialmente criado pelo servidor e enviado aos clientes para que seja treinado com os dados locais e, na sequência, um conjunto de novos pesos computados pelas redes neurais locais seja enviado ao servidor para o processo de melhoria do modelo global.

Para o conjunto de dados CIC-IDS2018, o FELIDS alcançou uma acurácia de 94,09%, valor próximo aos 94,24% do modelo central. No conjunto de dados InSDN, o modelo atingiu 99,71%, superior aos 97,71% do modelo centralizado e, no conjunto MQTTset, a acurácia foi de 89,56%, inferior aos 90,76% do modelo central.

Propondo uma arquitetura descentralizada para detecção de intrusão usando AF, os autores (Nakip, Gül and Gelenbe, 2023) apresentaram seu modelo denominado *Decentralized and Online Federated Learning Intrusion Detection* (DOF-ID), usando os conjuntos de dados Kit-

[14]https://www.unb.ca/cic/datasets/ids-2018.html. Acesso em abril/2025.
[15]https://www.mdpi.com/1424-8220/20/22/6578
[16]http://aseados.ucd.ie/datasets/SDN. Acesso em abril/2025.

sune[17] e Bot-IoT[18] com a utilização de três clientes.

No modelo foi utilizado *G-Networks* (uma generalização de redes de filas) para detectar ataques, incluindo ataques de dia zero, de forma colaborativa entre diferentes componentes de uma cadeia de suprimentos. A arquitetura é baseada na colaboração de *N* nós usando instâncias locais separadas do mesmo modelo, podendo ser considerado como um serviço compartilhado *Peer-to-Peer* (P2P), que distribui e recebe as atualizações de parâmetros de outros assinantes de maneira que cada nó (assinante) possa melhorar o seu modelo local e o global de todos os nós simultaneamente.

No contexto do trabalho, foram avaliados modos distintos para a atualização dos parâmetros dos nós, como a média entre todos os nós, a média com o nó mais próximo e a média com nó mais próximo por camada, de modo que os resultados mostram que a arquitetura DOF-ID melhora significativamente o desempenho na detecção de intrusões em todos os componentes colaborativos, chegando a uma acurácia de 98% e ainda mantendo um tempo de computação aceitável para o aprendizado *online*.

Os autores (Cholakoska, Gjoreski, Rakovic, Denkovski, Kalendar, Pfitzner and Arnrich, 2023) discutiram o uso do aprendizado federado para a detecção de intrusão em redes de ambientes de vida assistida (*ambient assisted living*).

Segundo (Vimarlund, Borycki, Kushniruk and Avenberg, 2021) um ambiente de vida assistida (AAL) é definido como o uso de tecnologias de informação e de comunicação (TIC), na vida diária de uma pessoa, para permitir que permaneçam ativas por mais tempo, socialmente conectadas e viverem de forma independente na velhice (livre tradução).

O estudo utilizou o conjunto de dados IoTID20[19] e redes neurais *feedforward* com oitenta e três neurônios, um neurônio para cada *feature* do conjunto de dados pré-processado, além de cinquenta clientes simulando ambientes AAL e que possuem diferentes porções do conjunto base. Foram executados no máximo trinta e cinco *rounds* para o treinamento do modelo federado.

Os resultados mostraram que quando foi utilizado todo o conjunto de dados, o modelo federado obteve uma acurácia de 84%, um pouco inferior ao modelo centralizado que obteve 86%. Quando foi feito o agrupamento de todos os tipos de ataque Mirai do conjunto de dados, o resultado do modelo federado chegou a 98,30%, enquanto o modelo centralizado foi de aproximadamente 97%.

Segundo os autores, ainda assim é vantajoso seu uso por preservar a privacidade do usuário, já que o modelo federado não compartilha os dados do AAL, apenas os pesos do modelo e, além disso, o modelo federado fornece melhor estabilidade e convergência mais rápida quando comparado ao modelo centralizado.

[17]https://doi.org/10.24432/C5D90Q. Acesso em abril 2025.
[18]https://research.unsw.edu.au/projects/bot-iot-dataset. Acesso em abril/2025.
[19]https://sites.google.com/view/iot-network-intrusion-dataset/home/. Acesso em abril/2025.

Em (Alkhpor and Alserhani, 2023), os pesquisadores propuseram um sistema de detecção de intrusão que utiliza modelos de AF para identificar cenários de ataques avançados, como *Advanced Persistent Threat* (APT), e para correlação de alertas, com a utilização do conjunto de dados UNSW-NB15[20].

Para o modelo centralizado foram utilizados os classificadores XGBoost, florestas aleatórias, CatBoost e um *ensemble* para combinar todos os resultados destes modelos e melhorar a performance da detecção de ataques. Além disso, um modelo de rede neural convolucional (CNN) centralizado foi construído para que seus resultados pudessem ser comparados aos resultados de um modelo CNN federado (CNN_FL), que foi treinado pelos quatro clientes que participaram do processo.

Os resultados mostraram que o modelo centralizado que obteve a maior acurácia foi o método *ensemble* com 88,15% de acertos, enquanto o modelo CNN_FL obteve uma acurácia de 90,18% na detecção de vários ataques.

No estudo de (Ribera, 2024), foram utilizadas redes neurais artificiais em um ambiente federado horizontal para a detecção de eventos intrusivos, utilizando o conjunto IoTID20[21].

Para a seleção de atributos, foram aplicadas as técnicas *Info Gain* e correlação de *Spearman*, resultando em diferentes conjuntos de dados. Tais conjuntos foram submetidos a redes neurais, e os resultados passaram por testes estatísticos para identificar o melhor conjunto a ser utilizado no AF.

O experimento envolveu cinco clientes no processo federado, cada um recebendo diferentes distribuições de dados para avaliar o desempenho do modelo em cenários IID e non-IID. O objetivo foi verificar se a distribuição dos dados entre os clientes influencia o desempenho do modelo federado.

Os resultados indicaram que, quando as proporções de classes e a quantidade de registros foram equilibradas entre os clientes, o modelo federado apresentou um desempenho comparável ao treinamento centralizado. O autor destaca que, com a seleção adequada do conjunto de dados, o treinamento centralizado alcançou uma acurácia média de 97,99%, enquanto o melhor modelo federado obteve 98,30%.

Com base na discussão apresentada neste capítulo, elaborou-se a Tabela 3.1, que resume os trabalhos citados e suas principais características.

[20]https://research.unsw.edu.au/projects/unsw-nb15-dataset. Acesso em abril/2025.
[21]https://sites.google.com/view/iot-network-intrusion-dataset/home/. Acesso em abril/2025.

Tabela 3.1: Relação dos trabalhos citados.

Autores	Bases de dados	Algoritmos	Paradigma de AF	# de clientes	Resultado (Acurácia)
(Liu et al., 2022)	Banco, e-commerce, UCI datasets	Florestas Aleatórias	Centralizado	8	99,50
(Souza et al., 2020)	CTU-13	Florestas Aleatórias	Descentralizado	N/A	98,00 (F1-score)
(Mothukuri et al., 2022)	Modbus network data set (logs de ataques)	LSTM, GRU, Florestas Aleatórias	Centralizado	N/A	90,25
(Wardana et al., 2024)	CIC-IoT2023	Florestas Aleatórias	Centralizado	5	99,68
(Rey et al., 2022)	N-BaIoT	MLP	Centralizado	8	99,96
(Campos et al., 2022)	ToN_IoT	Regressão Logística	Centralizado	10	91,00
(Neto et al., 2022)	CIC-IDS2017	Simulated Annealing	Centralizado	150	96,00
(Markovic et al., 2022)	KDD, NSL-KDD, UNSW-NB15, CIC-IDS2017	Florestas Aleatórias	Centralizado	N/A	93,51
(Friha et al., 2022)	CIC-IDS2018, MQTTset, InSDN	DNN, CNN	Centralizado	15	99,71
(Nakip et al., 2023)	Kitsune, Bot-IoT	RNA	Descentralizado	3	98,00
(Cholakoska et al., 2023)	IoTID20	RNA	Centralizado	50	98,30
(Alkhpor and Alserhani, 2023)	UNSW-NB15	CNN, Florestas Aleatórias	Centralizado	4	90,18
(Ribera, 2024)	IoTID20	RNA	Centralizado	5	98,30

3.3 Considerações finais

Neste capítulo foram analisadas diversas pesquisas que abordam a detecção de intrusões em redes de ambientes federados. A revisão também identificou a importância dos modelos de AM na classificação de eventos intrusivos, evidenciando seu papel na segurança computacional.

A revisão indicou que o AF é uma abordagem consolidada, pois permite o treinamento de modelos de AM sem a necessidade de compartilhamento de dados, garantindo maior privacidade e escalabilidade. Entretanto, parte das pesquisas que aplicam AF à detecção de intrusão utiliza redes neurais ou variações, que, apesar de sua alta capacidade de aprendizado, são computacionalmente custosas e de difícil interpretação.

Diante desse cenário, este trabalho explora o uso de florestas aleatórias no contexto do AF para detecção de intrusão, pois, diferente das abordagens baseadas em *deep learning*, as florestas aleatórias caracterizam modelos mais leves, interpretáveis e robustos, tornando-se uma alternativa viável para ambientes com limitações computacionais como os dispositivos IoT.

A partir desta revisão, algumas concepções de melhorias foram identificadas e podem ser adicionadas ao modelo proposto para melhorar seus resultados, como a utilização de *oversampling* para contornar o desbalanceamento de classes, técnicas de seleção de atributos para a redução da dimensionalidade do conjunto de dados e rodadas de execução para avaliar o desempenho geral do modelo.

Capítulo 4

Materiais e Métodos

4.1 Considerações iniciais

Para o desenvolvimento deste trabalho, optou-se pela utilização de um ambiente computacional local. Essa decisão foi motivada pela necessidade de assegurar maior controle sobre as condições de execução dos experimentos, possibilitando o isolamento das variáveis que impactam diretamente o desempenho dos algoritmos propostos. Essa abordagem contribui para a exatidão, a reprodutibilidade e a validade dos resultados obtidos. No entanto, reconhece-se que o ambiente simulado apresenta limitações inerentes, uma vez que não reproduz integralmente a complexidade e as variabilidades presentes em aplicações reais, especialmente em cenários distribuídos envolvendo dispositivos IoT e o Aprendizado Federado (AF).

4.2 Materiais

O desenvolvimento do método experimental foi realizado aplicando o ambiente de desenvolvimento *Spyder*[1] (*Scientific Python Development Environment*), uma *Integrated Development Environment* (IDE) de código aberto especialmente projetada para a programação em linguagem *Python*. Além desse, outros artefatos também foram utilizados:

- Linguagem de programação *Python*[2], versão 3.11;
- Bibliotecas auxiliares Numpy[3], Scikit-Learn[4], Pandas[5] e Matplotlib[6];
- Conjunto de dados IoTID20 (seção 4.2.1);
- Software estatístico JASP[7];
- Computador *desktop* com sistema operacional Fedora Linux 40, processador Intel Core i5-10500T @ 2.30GHz e memória RAM de 16 GB.

[1] https://www.spyder-ide.org/
[2] https://www.python.org/
[3] https://numpy.org/
[4] https://scikit-learn.org
[5] https://pandas.pydata.org/
[6] https://matplotlib.org/
[7] https://jasp-stats.org/

4.2.1 Conjunto de dados IoTID20

O IoTID20[8] é um conjunto de dados utilizado na área de segurança computacional, tendo sido criado a partir do tráfego de rede trocado entre dispositivos IoT, como câmeras, *laptops*, *tablets*, *smartphones* e outras estruturas, em um ambiente típico de casa inteligente.

Tal conjunto é comumente empregado para pesquisas e criação de algoritmos de detecção de intrusão e segurança e, de acordo com os seus autores, (Ullah and Mahmoud, 2020), uma das principais justificativas para o seu uso nessas tarefas é a capacidade de replicar a atual tendência de comunicação em redes IoT.

A base é composta por 625.783 eventos e 86 atributos, dos quais 83 são as variáveis independentes e que fornecem informações sobre os dados e são usadas como entradas para os modelos de AM. Os três atributos restantes são as variáveis dependentes ou alvos que os modelos tentam prever ou explicar com base nos atributos independentes.

Para a tarefa de classificação binária, o atributo *Label* é utilizado para categorizar os eventos como normais ou anormais. Já na classificação multiclasses, um segundo atributo, *Cat*, indica a categoria específica dos eventos, correspondendo a diferentes tipos de ataques, tais como *Denial of Service* (DoS), *Man-in-the-Middle* (MITM) ou *Scan*.

Adicionalmente, o conjunto inclui o atributo preditivo *Sub_Cat* que permite a subclassificação dos ataques. Esse atributo permite identificar a técnica associada ao ataque principal, por exemplo, a varredura de portas (*Scan Host Port*) ou da versão do sistema operacional (*Scan Port OS*) em uso, no caso de eventos do tipo *Scan*. O resumo da distribuição das classes pode ser visto na Tabela 4.1 e alguns eventos originais podem ser vistos no Apêndice A.1.

Tabela 4.1: Distribuição de classes do conjunto de dados IoTID20.

Binária (*Label*)	Categorias (*Cat*)	Subcategorias (*Sub_Cat*)
Normal (40.073), Anormal (585.710)	Normal (40.073), DoS (59.391), Mirai (415.677), MITM (35.377), Scan (75.265)	Normal (40.073), DoS (59.391), Mirai Ack Flooding (55.124), Mirai Brute Force (121.181), Mirai HTTP Flooding (55.818), Mirai UDP Flooding (183.554), MITM (35.377), Scan Host Port (22.192), Scan Port OS (53.073)

Fonte: (Ullah and Mahmoud, 2020).

Neste trabalho, o atributo *Label* é utilizado para identificar os eventos normais e os ataques, por meio de modelos de florestas aleatórias. Para isso, baseia-se no conjunto de atributos extraídos do IoTID20, adotando, portanto, uma abordagem de detecção de intrusão por anomalia, conforme seção 2.4.2.

[8] https://sites.google.com/view/iot-network-intrusion-dataset/home/. Acesso em abril/2025.

4.3 Método

Inicialmente foram realizadas pesquisas bibliográficas, abrangendo as áreas de segurança computacional e cibernética (seção 2.2), detecção de intrusão (seção 2.4) e métodos de inteligência artificial e Aprendizado de Máquina (AM) (seção 2.6). De modo complementar, foram levantados trabalhos da literatura considerando o estado da arte relacionado ao tema desta pesquisa (Capítulo 3).

O presente trabalho consiste no desenvolvimento e na avaliação do desempenho de um modelo computacional baseado em florestas aleatórias, aplicado em um ambiente simulado de Aprendizado Federado (AF) horizontal, visando a classificação binária de eventos intrusivos, usando o conjunto de eventos relacionados à segurança computacional IoTID20. A arquitetura computacional proposta é demonstrada por meio da Figura 4.1.

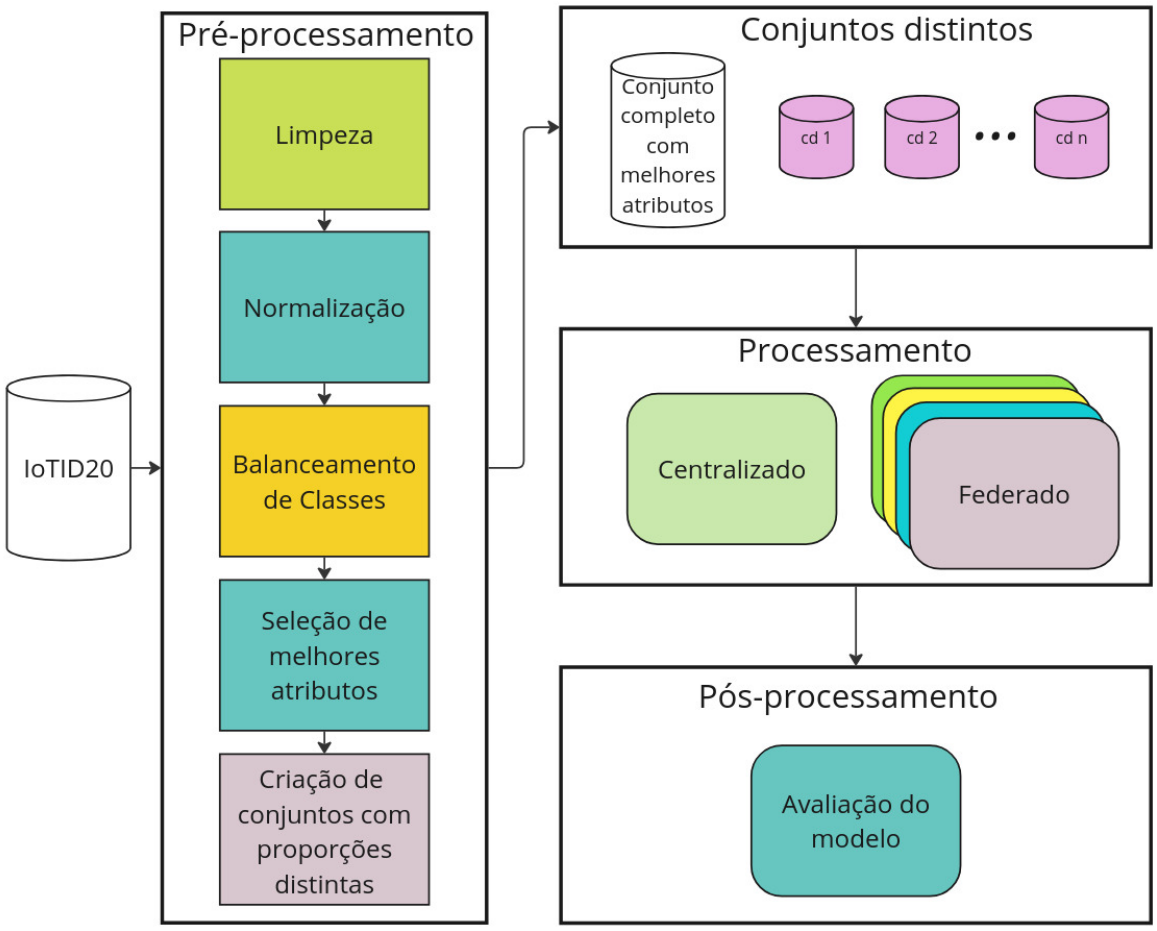

Figura 4.1: Visão macro do método proposto.
Fonte: O autor.

Inicialmente, o método contempla tarefas de pré-processamento no conjunto de dados, incluindo limpeza e normalização, balanceamento das classes e seleção dos melhores atributos. Tais tarefas são aplicadas com o intuito de preparar o conjunto de eventos para as etapas de processamento.

Um servidor central é responsável pelas etapas de processamento dos aprendizados cen-

tralizado (AC) e federado (AF). Na etapa centralizada, o servidor cria um modelo inicial de florestas aleatórias, utilizando os dados disponíveis localmente. Em seguida, na etapa federada, o servidor distribui o modelo global, e vários conjuntos de dados distintos, para os dispositivos clientes.

Seis clientes participam da etapa federada, realizando o treinamento e melhoria do modelo global recebido e posteriormente enviando ao servidor central as melhorias encontradas, as quais são avaliadas se melhoram o desempenho do modelo, antes de serem agregadas ao modelo global. Ao final de cada etapa, os resultados são coletados para avaliação.

A acurácia é utilizada como principal métrica de avaliação, por fornecer uma visão geral sobre a capacidade do modelo em distinguir entre tráfego normal ou anômalo, sendo a mais usada nos estudos relacionados. Adicionalmente, as medidas F1-score e o Recall são considerados em momentos específicos da análise, proporcionando uma avaliação mais aprofundada da performance do modelo, conferindo maior robustez e confiabilidade aos resultados obtidos.

Os algoritmos utilizados neste método foram executados em sua configuração *default*, e seus principais hiperparâmetros e valores podem ser consultados na Tabela 4.2.

Tabela 4.2: Hiperparâmetros utilizados nos algoritmos.

Florestas aleatórias Pré-proces. e AC	**CatBoost**	**XGB**	**LightGBM**
n_estimators=100 random_state=42 n_jobs=-1 max_depth=10 criterion=gini	random_state=42 iterations=10 depth=10	random_state=42 n_jobs=8 max_depth=10 objective=binary:logistic	learning_reate=0.1 max_depth=10
Florestas aleatórias AF	**Gain Ratio**	**Information Gain**	**Chi-Squared**
n_estimators=100∼1000 random_state=randômico n_jobs=-1 max_depth=10	Implementação manual em Python	Implementação manual em Python	Scikit-learn chi2()

4.3.1 Pré-processamento

O pré-processamento desempenha um papel vital em um sistema de detecção de intrusão (IDS) e é um passo essencial para melhorar o processo de treinamento em modelos de AM (Chimphlee and Chimphlee, 2023).

A etapa de pré-processamento é única e prepara os dados para serem utilizados nos processos de AC e AF. Na Figura 4.2 ilustra-se as atividades realizadas nesta etapa e que estão detalhadas na sequência.

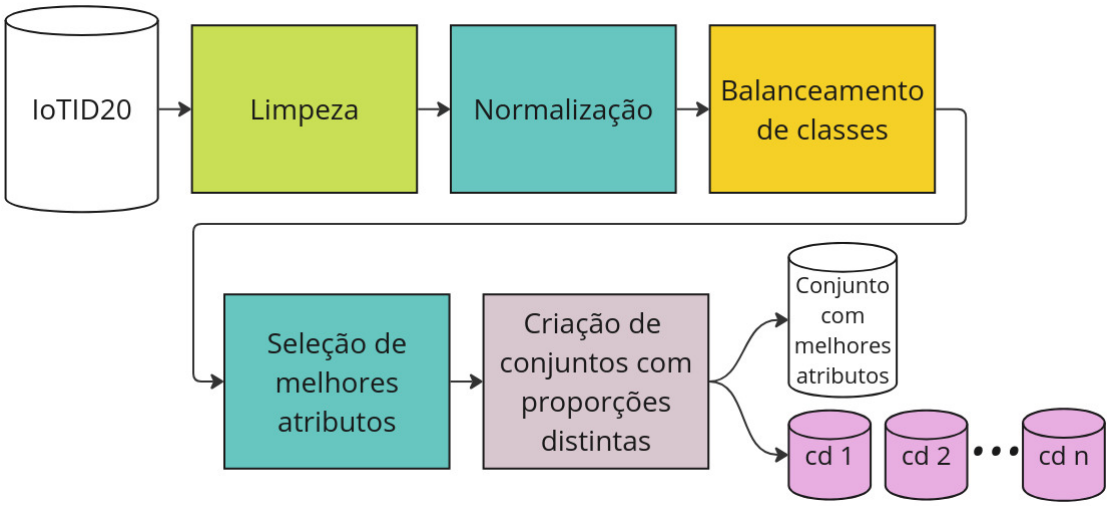

Figura 4.2: Resumo das atividades do pré-processamento.
Fonte: O autor.

4.3.1.1 Limpeza dos dados

Os atributos inicialmente removidos do conjunto de dados IoTID20 estão listados na Tabela 4.3. *Cat* e *Sub_Cat* fazem referência a possibilidade do conjunto de dados ser utilizado para a classificação multiclasses, entretanto, neste estudo, foi utilizado apenas o atributo *Label*, que indica se um evento é normal ou uma anomalia (classificação binária). Os demais atributos que constam na tabela foram removidos pelo fato de serem específicos da época e do ambiente de criação do conjunto de dados.

Tabela 4.3: Atributos irrelevantes removidos.

Atributo	Descrição
Flow_ID	Identificação do fluxo de comunicação entre IP de origem e destino
Cat e Sub_Cat	Categoria e subcategoria do evento (classificação multiclasses)
Timestamp	Data e hora do evento
Src_IP	Endereço IP de origem
Src_Port	Porta de conexão de origem
Dst_IP	Endereço IP de destino
Dst_Port	Porta de conexão de destino

Os atributos apresentados na Tabela 4.4 foram inteiramente removidos da base de eventos, pois todos os seus valores eram constantes ou com variância menor do que 0,01% ao longo do conjunto de dados. Deste modo, os mesmos não agregam valor para a construção do modelo de AM.

Tabela 4.4: Atributos removidos com valores constantes ou com baixa variância.

Atributos	
Bwd_Blk_Rate_Avg	Fwd_Pkts/b_Avg
Bwd_Byts/b_Avg	Fwd_PSH_Flags
Bwd_Pkts/b_Avg	Fwd_Seg_Size_Min
Bwd_URG_Flags	Fwd_URG_Flags
ECE_Flag_Cnt	Init_Fwd_Win_Byts
Fwd_Blk_Rate_Avg	URG_Flag_Cnt
Fwd_Byts/b_Avg	

Durante a limpeza dos dados, foram encontrados 368 registros com valor *Inf* nos atributos *Flow_Pkts/s* e *Flow_Byts/s*, representando valores extremamente altos (infinitos). Como esses registros representam aproximadamente 0,06% do total de registros, eles foram removidos do conjunto de dados, pois caracterizam *outliers*. O atributo *Init_Bwd_Win_Byts* possuía algumas ocorrências de valores negativos, os quais foram substituídos pela mediana da coluna. Ainda durante a análise, foram identificados e removidos 468.336 registros duplicados, dos quais 455.070 pertenciam à classe majoritária e 13.260 à classe minoritária.

4.3.1.2 Normalização

O próximo passo foi normalizar os dados, utilizando o método *Z-score*, com o objetivo de garantir que todos os atributos estejam na mesma escala, evitando que ordens de grandeza diferentes dominem a análise. Seguindo o padrão adotado por (Ribera, 2024), todos os atributos foram submetidos ao processo de normalização, com exceção do atributo preditivo *Label*.

4.3.1.3 Balanceamento de classes

Para aumentar a quantidade de eventos da classe minoritária e equalizar o número de eventos entre as classes, foi aplicada a *Synthetic Minority Oversampling Technique* (SMOTE). A SMOTE utiliza o algoritmo *K-Nearest Neighbors* (KNN) para selecionar os vizinhos mais próximos de cada instância da classe minoritária, e, a partir deles, criar amostras artificiais.

Na aplicação da SMOTE, a estratégia adotada para gerar os novos exemplos foi de que o número de eventos da classe minoritária fosse aproximadamente equivalente ao número de eventos da classe majoritária. Desta maneira, foi utilizado k=40 vizinhos no algoritmo KNN. A escolha desse valor foi fundamentada em avaliações empíricas preliminares, que indicaram que essa configuração proporciona um número reduzido de novos eventos duplicados. Com esse valor de k, foram criados aproximadamente 102.000 novos exemplos para a classe minoritária.

4.3.1.4 Seleção de atributos

Dando continuidade ao pré-processamento, foi realizada a redução da dimensionalidade, com a seleção dos melhores atributos que possam ser utilizados pelos modelos de AM. De

acordo com (Faceli et al., 2021), a redução do número de atributos pode melhorar o desempenho do modelo, reduzir seu o custo computacional e tornar os resultados mais compreensíveis.

Para a seleção dos atributos foi tomado como base a pesquisa de (Leevy et al., 2021). A escolha por este método se justifica pelo fato de que essa abordagem apresenta similaridade com os objetivos e características do presente trabalho, especialmente no que se refere à aplicação de técnicas de aprendizado supervisionado para detecção de ataques e a utilização de algoritmos baseados em árvores de decisão.

Tal método consiste em um *ensemble* de sete técnicas, sendo três delas baseadas em estatísticas e quatro em algoritmos de aprendizado supervisionado, sendo o resultado final alcançado por votação. Para cada uma das técnicas, os atributos selecionados foram classificados em ordem decrescente de acordo com as importâncias calculadas, mantendo os melhores e eliminando importâncias iguais a zero. Por fim, para a definição dos atributos selecionados para a composição do melhor grupo, foram escolhidos os que estiveram presentes pelo menos em quatro das sete técnicas aplicadas.

Para a utilização neste trabalho, foi implementado um *ensemble* para seleção de atributos, o qual pode ser visto na Figura 4.3. São utilizadas as métricas estatísticas *Information Gain* (IG), *Gain Ratio* (GR) e *Chi-Squared* (CS), e técnicas de aprendizado supervisionado, como Florestas Aleatórias (FA), *Categorial Boosting* (CatBoost), *eXtreme Gradient Boosting* (XGBoost) e *Light Gradient-Boosting Machine* (LightGBM). Essas técnicas foram previamente descritas na seção 2.6.

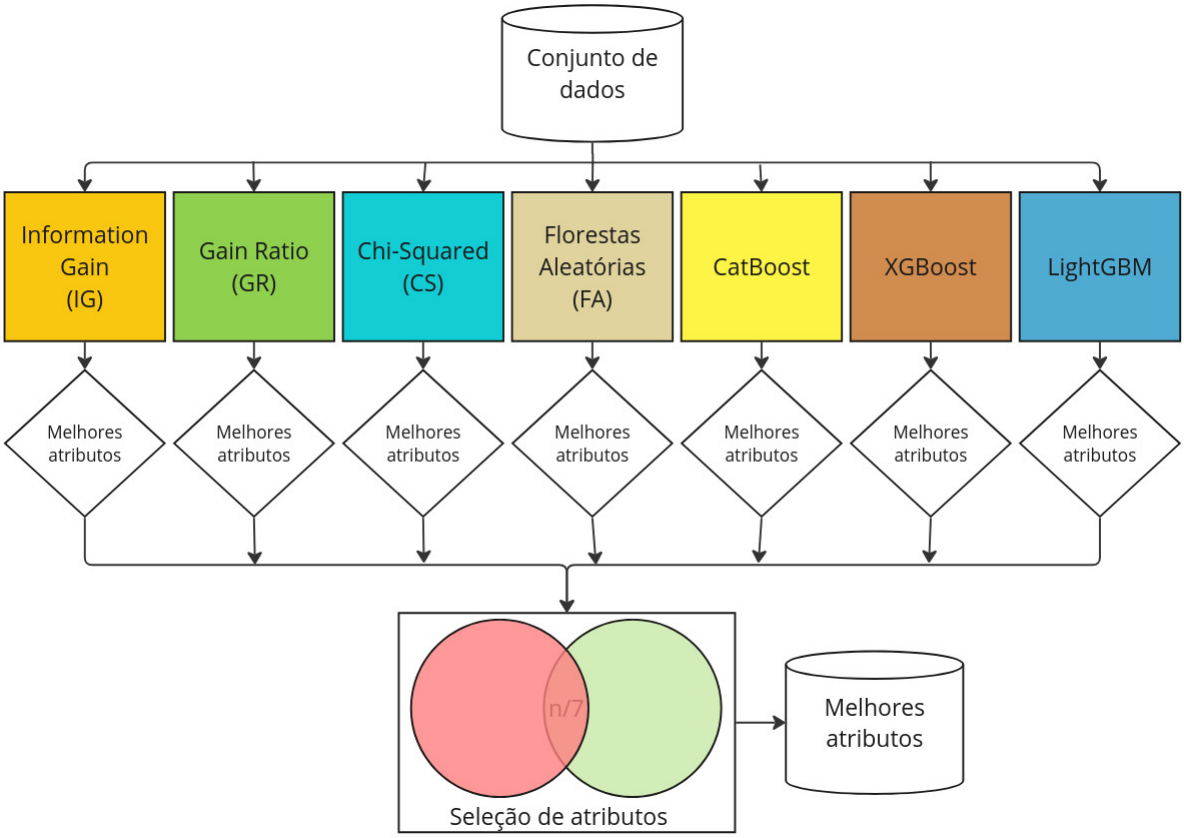

Figura 4.3: *Ensemble* de seleção dos melhores atributos.
Fonte: O autor.

Para o cálculo do IG, GR e CS, não foram fornecidos parâmetros específicos de configuração, sendo utilizadas as configurações padrão dos algoritmos. No caso dos modelos de aprendizado supervisionado, esses foram executados com seus hiperparâmetros padrão, como o número de árvores de decisão (*n_estimators*), fixado em 100 e a profundidade máxima de cada árvore (*max_depth*) limitada a 10. Adicionalmente, para garantir a reprodutibilidade dos experimentos foi utilizado o parâmetro *random_state* igual a 42. Detalhes específicos dos hiperparâmetros foram exibidos na Tabela 4.2.

O conjunto de dados original possuía 83 atributos, sendo que 19 foram eliminados durante as etapas iniciais de pré-processamento. Assim, restaram 64 atributos que foram submetidos aos sete algoritmos do *ensemble*, onde cada um gerou uma lista ordenada decrescente por ordem de importância com até 32 atributos.

Conforme destacado por (Faceli et al., 2021), "a ordenação de atributos pode ser vista como uma forma simples de seleção, em que os atributos são ordenados de acordo com sua relevância para um dado critério".

A escolha do limite de 32 atributos, equivalente a 50% do total, teve como objetivo otimizar a etapa de pré-processamento, reduzindo a complexidade do conjunto de dados. Além disso, o limite de até 32 atributos considera que alguns podem ser desconsiderados por determinados algoritmos de seleção, especialmente aqueles cujo ganho de informação é nulo, resultando em

listas com menos atributos.

Foram consideradas diferentes abordagens para a seleção final, partindo da identificação dos atributos comuns a pelo menos três dos sete algoritmos até os atributos compartilhados por todos os sete algoritmos. A seleção dos atributos foi realizada com base no critério de votação: os atributos que não atingiram o número mínimo de votos estabelecido naquela avaliação foram excluídos do conjunto de 64 atributos. Os detalhes dos grupos de atributos criados estão na Tabela 4.5 e os atributos selecionados por cada algoritmo do *ensemble* podem ser vistos no Apêndice A - Tabela A.2.

Tabela 4.5: Grupos de atributos criados: A, B, C, D e E.

Grupo	Abordagem (Nº de Votos)	Nº de Atributos	Nº de Eventos
A	3~7	43	121.235 Anormal (48,55%) 128.520 Normal (51,45%)
B	4~7	38	120.444 Anormal (48,38%) 128.519 Normal (51,62%)
C	5~7	28	114.532 Anormal (47,14%) 128.468 Normal (52,86%)
D	6~7	19	114.419 Anormal (47,11%) 128.463 Normal (52,89%)
E	7	6	81.913 Anormal (40,67%) 119.513 Normal (59,33%)

Cada grupo de atributos foi avaliado por meio de um processo de validação cruzada com dez *folds* (k=10), utilizando como classificador o algoritmo de florestas aleatórias. Em cada iteração, um novo modelo era treinado com 90% dos dados e testado nos 10% restantes, garantindo que todas as instâncias fossem utilizadas tanto para treinamento quanto para validação. O objetivo dessa avaliação foi identificar, entre os cinco grupos de atributos testados, aquele que proporcionasse o melhor desempenho preditivo.

A configuração usada nos hiperparâmetros do classificador para a validação cruzada foi definida da seguinte forma: 100 árvores estimadoras (n_estimators=100), semente de controle de aleatoriedade 42 (random_state=42), profundidade máxima das árvores igual a 10 (max_depth=10) e critério de divisão o índice de Gini.

A partir dos resultados da validação cruzada, foram calculadas as acurácias médias de cada grupo de atributos, as quais foram submetidas a análises estatísticas. Inicialmente, foi feito o teste de normalidade de Shapiro-Wilk, considerando um nível de significância de 0,05. Os resultados indicaram ausência de evidências suficientes para rejeitar a hipótese de normalidade nos dados. Com base nesse resultado, foi feita a análise de variância, ANOVA, a qual apontou diferenças estatisticamente significativas entre os grupos *B* e *E*, bem como entre os grupos *D* e *E*. Esses resultados da avaliação ANOVA estão registrados na Tabela 4.6.

Tabela 4.6: Resultados do teste ANOVA entre os pares de grupos de atributos.

Comparação entre pares	ANOVA (p-*value* < 0,05)	Diferença
Grupo A x Grupo B	0,9582	
Grupo A x Grupo C	1,0000	
Grupo A x Grupo D	0,5736	
Grupo A x Grupo E	0,1847	
Grupo B x Grupo C	0,9288	
Grupo B x Grupo D	0,9282	
Grupo B x Grupo E	0,0408	✓
Grupo C x Grupo D	0,5021	
Grupo C x Grupo E	0,2280	
Grupo D x Grupo E	0,0046	✓

Uma análise mostrou que o grupo *B* manteve o equilíbrio no número de eventos por classe, utilizando um conjunto maior de atributos (38). Em contrapartida, o grupo *D* também apresentou equilíbrio entre as classes, mas com uma configuração mais enxuta, empregando apenas 19 atributos. Por outro lado, o grupo *E* apresentou o maior desequilíbrio entre as classes, além de utilizar um número reduzido de atributos, apenas seis, em relação aos outros grupos.

Diante dessas análises, foi escolhido o grupo *D* para dar continuidade à construção dos conjuntos de dados com melhores atributos, estando estes relacionados na Tabela 4.7.

Tabela 4.7: Melhores atributos selecionados após *ensemble* (Grupo D).

Atributo	# Votos	Atributo	# Votos
ACK_Flag_Cnt	7	Flow_Pkts/s	6
Flow_Duration	7	Pkt_Len_Max	6
Pkt_Size_Avg	7	Bwd_Pkts/s	6
Bwd_Header_Len	7	Fwd_Pkts/s	6
Pkt_Len_Min	7	Idle_Min	6
Idle_Max	7	Bwd_Pkt_Len_Max	6
Init_Bwd_Win_Byts	6	TotLen_Bwd_Pkts	6
Flow_IAT_Min	6	Flow_IAT_Max	6
Bwd_Pkt_Len_Mean	6	Bwd_Pkt_Len_Min	6
Pkt_Len_Var	6		

4.3.1.5 Criação de grupos e conjuntos de dados

Para estruturar a realização dos testes e a avaliação dos resultados dos modelos de aprendizado de máquina (AM), foram definidos quatro grupos de dados. Cada grupo é composto por seis conjuntos de dados distintos, variando conforme critérios de distribuição dos eventos. Essa variação nos percentuais de ocorrência dos eventos tem como objetivo avaliar a capacidade de generalização dos modelos frente a diferentes níveis de desbalanceamento, simulando cenários próximos da realidade:

- Grupo 1 – completo: contém o conjunto de dados completo, com o mesmo número de

eventos em cada classe (normais e anormais), garantindo um cenário de equilíbrio (Tabela 4.8).

- Grupo 2 - balanceado: cada conjunto contém apenas eventos únicos, ou seja, um mesmo evento aparece apenas uma vez e em apenas um dos conjuntos, evitando qualquer sobreposição entre eles (Tabela 4.9).
- Grupo 3 - desbalanceamento moderado: os conjuntos são construídos com proporções de 25%, 50% e 75% dos eventos de uma das classes (eventos normais), mantendo-se 100% dos eventos da classe oposta (eventos anormais). Em seguida, essa lógica é invertida para gerar mais três conjuntos, totalizando os seis conjuntos do grupo (Tabela 4.10).
- Grupo 4 - desbalanceamento extremo: inclui conjuntos de dados com distribuições altamente desbalanceadas entre as classes: inicialmente são criados três conjuntos utilizando 1%, 3% e 5% dos eventos da classe normal, combinados com 100% dos eventos da classe anomalia. Em seguida, a abordagem é invertida: outros três conjuntos são formados com 7%, 9% e 11% dos eventos da classe anomalia, mantendo-se 100% dos eventos da classe normal (Tabela 4.11). Diferentemente do grupo anterior (grupo 3), aqui não foi seguido o padrão simétrico de inversão de percentuais, pois o objetivo é simular cenários mais extremos e assimétricos, alinhados com situações reais de detecção de intrusões.

Estes percentuais foram escolhidos para simular uma variedade representativa de condições de desbalanceamento de classes, que são frequentemente observadas em ambientes reais de detecção de intrusão, cobrindo cenários de desequilíbrio moderado a extremo, a fim de testar a robustez do modelo em diversas situações práticas.

O padrão de nomes utilizado para a criação dos conjuntos de dados foi:

- cd: indica "conjunto de dados".
- A: indica a classe anomalia, seguido do percentual de eventos.
- N: indica a classe normal, seguido do percentual de eventos.
- U: identificador único utilizado para diferenciar conjuntos que possuem o mesmo percentual de eventos em ambas as classes.

Tomando como exemplo "cd_A100-N25", é o conjunto de dados com 100% de eventos de anomalia (A100) e 25% de eventos normais (N25).

Tabela 4.8: Conjuntos de dados - grupo 1 - completo.*

Conjunto de dados	Eventos Anormais	Eventos Normais	Total de Eventos
cd_A100-N100	114.419 (100%)	128.463 (100%)	242.882

* Diferente dos outros grupos que possuem seis conjuntos de dados, para otimizar o processamento neste grupo, o conjunto de dados foi armazenado em uma única vez, evitando a duplicação. Quando necessário, essa mesma cópia é reutilizada.

Tabela 4.9: Conjuntos de dados - grupo 2 - balanceado.

Conjunto de dados	Eventos Anormais	Eventos Normais	Total de Eventos
cd_A16-N16-U1	19.070 (± 16,60%)	21.410 (± 16,60%)	40.480
cd_A16-N16-U2	19.070 (± 16,60%)	21.410 (± 16,60%)	40.480
cd_A16-N16-U3	19.070 (± 16,60%)	21.410 (± 16,60%)	40.480
cd_A16-N16-U4	19.069 (± 16,60%)	21.411 (± 16,60%)	40.480
cd_A16-N16-U5	19.070 (± 16,60%)	21.411 (± 16,60%)	40.481
cd_A16-N16-U6	19.070 (± 16,60%)	21.411 (± 16,60%)	40.481

Tabela 4.10: Conjuntos de dados - grupo 3 - desbalanceamento moderado.

Conjunto de dados	Eventos Anormais	Eventos Normais	Total de Eventos
cd_A100-N25	114.419 (100%)	32.116 (25%)	146.535
cd_A100-N50	114.419 (100%)	64.232 (50%)	178.651
cd_A100-N75	114.419 (100%)	96.347 (75%)	210.766
cd_A25-N100	28.605 (25%)	128.463 (100%)	157.068
cd_A50-N100	57.210 (50%)	128.463 (100%)	185.673
cd_A75-N100	85.814 (75%)	128.463 (100%)	214.277

Tabela 4.11: Conjuntos de dados - grupo 4 - desbalanceamento extremo.

Conjunto de dados	Eventos Anormais	Eventos Normais	Total de Eventos
cd_A100-N1	114.419 (100%)	1.285 (1%)	115.704
cd_A100-N3	114.419 (100%)	3.854 (3%)	118.273
cd_A100-N5	114.419 (100%)	6.423 (5%)	120.842
cd_A7-N100	8.009 (7%)	128.463 (100%)	136.472
cd_A9-N100	10.298 (9%)	128.463 (100%)	138.761
cd_A11-N100	12.586 (11%)	128.463 (100%)	141.049

Com a criação desses conjuntos de dados, é concluída a etapa de pré-processamento, marcando o início das etapas de processamento do AC e do AF.

4.3.2 Processamento centralizado

Nesta etapa do AC, foi utilizado o grupo 1 (conjunto completo), apresentado na Tabela 4.8, para a construção do modelo base de florestas aleatórias. Esse modelo tem como finalidade servir como referência para comparação entre as abordagens centralizada e federada, além de atuar como ponto de partida para o processamento federado.

O AC foi realizado com os seguintes hiperparâmetros: 100 árvores estimadoras na floresta (n_estimators=100), controle de aleatoriedade com valor fixo de 42 (random_state=42), profundidade máxima das árvores igual a 10 (max_depth=10) e critério de divisão baseado no índice de Gini.

A avaliação de desempenho do modelo foi realizada por meio de validação cruzada estratificada, com dez *folds* (k=10), aplicada ao conjunto de dados. Esse processo faz com que,

em cada uma das dez iterações, o conjunto de dados seja particionado em subconjuntos mutuamente exclusivos, mantendo a proporção das classes em cada divisão. Desse modo, são utilizados 90% dos dados para treinamento e 10% para testes em cada iteração, garantindo que todas as instâncias fossem utilizadas tanto para treinamento quanto para validação.

Durante a validação cruzada, um novo modelo de floresta aleatória é treinado do zero em cada iteração. Isso ocorre porque a função *cross_validate*[9] da biblioteca Scikit-learn utilizada neste processo, clona o classificador original e realiza o *fit* exclusivamente sobre os dados da partição de treino correspondente àquela rodada. Assim, a cada iteração, é construída uma nova floresta, composta por árvores treinadas apenas com os dados daquela subdivisão.

Após cada iteração, os valores de acurácia, F1-score e Recall são registrados com base no desempenho do modelo sobre os dados de teste daquele *fold*. Ao final de todas as iterações, os resultados são analisados e o modelo com melhor acurácia é selecionado como modelo de referência.

4.3.3 Processamento federado

Para o AF, foram utilizados seis clientes simulando dispositivos IoT. Esse número foi definido com base na quantidade de conjuntos de dados gerados nos grupos 2, 3 e 4, ao final da etapa de pré-processamento.

Ao realizar a simulação de dispositivos, são descartadas questões como gestão de bateria, conectividade à Internet e outros fatores específicos de cada modelo de equipamento, com o intuito de simplificar o processo e focar em aspectos do modelo de AM.

Para analisar o comportamento do modelo no AF, o processo foi dividido em quatro experimentos, sendo que nos dois primeiros os dados analisados são considerados IID, enquanto nos demais são non-IID:

- 1º Experimento (AF1): cada cliente possui o conjunto de dados do grupo 1 (Tabela 4.8).
- 2º Experimento (AF2): cada cliente possui um conjunto de dados do grupo 2 (Tabela 4.9).
- 3º Experimento (AF3): cada cliente possui um conjunto de dados do grupo 3 (Tabela 4.10).
- 4º Experimento (AF4): cada cliente possui um conjunto de dados do grupo 4 (Tabela 4.11).

Cada experimento foi repetido dez vezes, sendo registrada, ao final de cada repetição, a instância do modelo que obteve a melhor acurácia, juntamente com os seus valores de F1-score e Recall. No início de cada repetição, o conjunto de dados do grupo 1 (completo) é dividido em 70% para treinamentos e 30% para testes. No entanto, apenas a parcela destinada aos testes

[9]https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html. Acesso em abril/2025.

(30%) é efetivamente utilizada para avaliar o desempenho do modelo durante o processo de agregação, enquanto os 70% reservados para treinamento são desconsiderados.

Em cada execução de um experimento, são executadas 50 épocas. Define-se uma época como o ciclo completo de troca de informações entre o servidor e os clientes. A definição de 50 épocas foi baseada em avaliações empíricas, nas quais esse número demonstrou ser suficiente para alcançar resultados consistentes nas métricas de desempenho analisadas. A figura 4.4 ilustra a representação de uma época de AF.

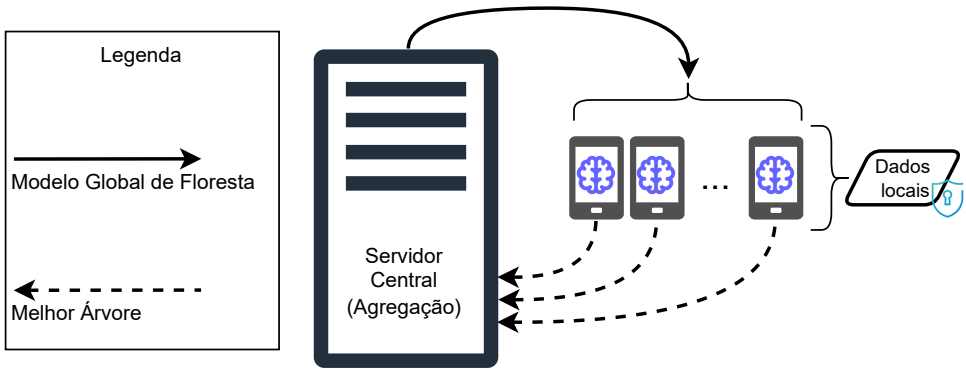

Figura 4.4: Representação de uma época do modelo federado.
Fonte: O autor.

Em cada época do AF, o servidor envia o modelo base de floresta aleatória para os clientes. Cada cliente, ao receber o modelo, divide o seu conjunto de dados local em 70% para treinamentos e 30% para testes, e adiciona mais árvores à floresta recebida.

A criação das novas árvores é feita utilizando os métodos próprios e otimizados das florestas aleatórias, aproveitando suas vantagens, como o paralelismo, o *bagging* e a capacidade de lidar de forma mais eficaz com grandes volumes de dados. Essa abordagem substitui a construção individual de árvores por métodos convencionais.

Após a fase de treinamentos, a identificação da melhor árvore da floresta no cliente é realizada por meio da avaliação da sua acurácia, utilizando a porção local dos dados de testes. São analisadas todas as árvores da floresta e, após a avaliação, a melhor árvore de decisão de cada cliente é enviada ao servidor para agregação.

A estratégia de agregação proposta para a construção do modelo global de floresta aleatória baseia-se no envio, por parte de cada cliente, da sua melhor árvore de decisão ao servidor. Essa abordagem foi inspirada no estudo de (Markovic et al., 2022), que avaliou diferentes formas de agregação das árvores individuais das florestas locais para compor o modelo global. No referido estudo, os melhores resultados foram obtidos ao ordenar as árvores de decisão de cada floresta com base na métrica de acurácia e selecionar, dentre elas, as mais eficientes.

O processo de agregação das árvores ao modelo global ocorre da seguinte forma: em cada época, a árvore enviada pelo cliente ao servidor é incorporada à floresta aleatória global. Em seguida, a acurácia da floresta expandida (com a nova árvore) é avaliada utilizando o con-

junto de dados de teste, previamente reservado para essa finalidade no início da execução do experimento.

Caso a inclusão da nova árvore resulte em um aumento na acurácia do modelo, ou se essa métrica permanecer inalterada, a árvore é mantida na floresta. Dessa forma, o número de classificadores do modelo global aumenta, potencialmente melhorando sua capacidade de generalização. Caso contrário, ela é descartada, evitando que um classificador de desempenho inferior comprometa o resultado do modelo.

Essa abordagem é simples e eficiente, pois só aceita melhorias claras e imediatas. Isso torna o algoritmo rápido e garante que apenas árvores que contribuem para a melhoria da floresta global sejam mantidas.

Este processo de troca de informações entre servidor e clientes é repetido ao longo das 50 épocas e a expectativa é que, a cada época, o número de árvores na floresta global aumente, tornando o modelo mais robusto e capaz de melhor generalizar os dados.

O padrão de processamento de 50 épocas repetidas dez vezes permanece o mesmo em todos os quatro experimentos e os resultados de acurácia, F1-score e Recall do melhor modelo de cada execução são registrados para análise comparativa do desempenho do modelo sob as diferentes distribuições de dados. O pseudocódigo do AF pode ser visto no Algoritmo 1.

Algoritmo 1 Pseudocódigo do AF com Florestas Aleatórias.

Entrada: Conjuntos de dados de 1 até 4; modelo base de RF; $N_{\text{épocas}} = 50$; $N_{\text{repetições}} = 10$; $N_{\text{clientes}} = 6$
Para cada experimento **FL1, FL2, FL3** e **FL4 faça**
    Para cada repetição de 1 até $N_{\text{repetições}}$ **faça**
        Servidor envia os conjuntos de dados para todos os clientes
        Para cada época de 1 até $N_{\text{épocas}}$ **faça**
            Servidor envia RF_{global} para todos os clientes
            Para cada cliente c_i **faça**
                Dividir o conjunto de dados local: 70% treinamento / 30% testes
                Incrementar e treinar o modelo local RF_i com novas árvores de decisão
                Avaliar a acurácia de cada árvore de decisão em RF_i
                Selecionar a árvore T_i^* com o melhor desempenho
                Enviar T_i^* ao servidor
            Fim Para
            Servidor: Recebe a melhor árvore de cada cliente
            Para cada árvore T_i^* recebida **faça**
                Temporariamente adicione T_i^* ao RF_{global}
                Avalie a acurácia atualizada usando dos dados de validação
                Se a acurácia melhora ou permanece a mesma **então**
                    Mantenha T_i^* na RF_{global}
                se não
                    Remova e descarte a T_i^* da RF_{global}
                Fim Se
            Fim Para
        Fim Para
        Armazene a melhor RF_{global} final da atual repetição
    Fim Para
Fim Para
Saída: Modelos finais dos experimentos FL1, FL2, FL3 e FL4

4.3.4 Pós-processamento

Na etapa de pós-processamento do modelo de AM, são realizadas tarefas com o objetivo principal de analisar os resultados, avaliar as métricas de desempenho e realizar a visualização dos dados. Essas atividades permitem compreender e interpretar o desempenho do modelo de forma mais clara e detalhada e serão tratadas no próximo capítulo.

Dentre as métricas utilizadas para avaliação do modelo, são usadas a acurácia, o F1-score e o Recall. Essas métricas foram previamente definidas na seção 2.6.9.4 e são amplamente empregadas na literatura, por oferecerem diferentes perspectivas sobre o desempenho de modelos preditivos.

4.4 Considerações finais

Neste capítulo, foram apresentados os métodos centralizado e federado aplicados à detecção de intrusão, bem como as bases de dados criadas e utilizadas nos experimentos. Também foram descritos o processo de seleção de atributos e os procedimentos de pré-processamento realizados sobre a base de dados original. No próximo capítulo, serão apresentados e discutidos os resultados obtidos neste trabalho.

Capítulo 5

Resultados e Discussão

5.1 Considerações iniciais

Neste capítulo são apresentados os resultados experimentais, acompanhados da respectiva discussão, das contribuições alcançadas e da comparação com estudos correlatos, considerando o estado da arte. Além disso, são incluídas informações obtidas a partir da aplicação dos métodos descritos nos capítulos anteriores e, em seguida, realiza-se uma avaliação dos resultados, com base nas hipóteses levantadas ao longo da pesquisa.

5.2 Pré-processamento

5.2.1 Limpeza dos dados

A avaliação inicial do conjunto de dados visou eliminar registros com atributos faltantes ou incompletos, valores atípicos ou irrelevantes, assim como fora de escala, entre outros. Importante destacar que informações incoerentes podem comprometer a performance dos resultados dos modelos de Aprendizado de Máquina (AM).

A limpeza de dados traz uma série de benefícios importantes. Além de contribuir para a redução do tamanho do conjunto de dados e, consequentemente, diminuir o uso de recursos computacionais durante o treinamento dos modelos, ela melhora a qualidade da informação disponível, reduz a complexidade do modelo, aumenta a exatidão das previsões e diminui o risco de *overfitting*. Portanto, a etapa de limpeza é essencial para garantir maior confiabilidade nas análises e nos resultados (Faceli et al., 2021).

5.2.2 Normalização

A normalização dos dados foi realizada com base no método Z-score, por meio da biblioteca Scikit-learn *StandardScaler*[1]. Essa abordagem é particularmente importante para modelos de AM, pois garante que os atributos sejam comparáveis em escala, reduzindo o viés em atributos com magnitudes maiores. Como consequência, a normalização favorece um aprendizado mais estável e equilibrado, contribuindo para o desempenho do modelo (Singh and Singh,

[1] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. Acesso em abril/2025.

2020).

5.2.3 Balanceamento de classes

Após a limpeza e a normalização, o conjunto de dados ainda apresentava desbalanceamento, composto por mais de 82% dos registros pertencentes à classe anomalia. Diante desse cenário, aplicou-se a SMOTE para aumentar a quantidade de registros da classe minoritária.

Para isso, foram realizados testes prévios com o objetivo de determinar o número de vizinhos (k) a ser utilizado pelo algoritmo KNN, de modo que o resultado da aplicação da técnica gerasse a menor quantidade de novos registros duplicados. Como resultado desses testes, foi utilizado o valor de k=40, com o qual aproximadamente menos de 2% dos registros gerados apresentaram duplicação e foram removidos. De forma definitiva, foram criados e adicionados ao conjunto cerca de 102.000 novos registros.

A escolha pela estratégia de aumentar o número de eventos da classe minoritária (*oversampling*) em contrapartida a diminuir o número de eventos da classe majoritária (*undersampling*) foi feita pelos seguintes motivos:

- Reduzir o número de exemplos da classe majoritária poderia levar a perda de exemplos importantes.
- Reduzir ainda mais a classe minoritária poderia levar a uma perda significativa de exemplos.
- Os novos exemplos são derivados a partir de dados reais da própria classe minoritária.

As métricas exibidas na Tabela 5.1 foram obtidas a partir do processo de validação cruzada com dez folds (k=10), usando o classificador florestas aleatórias. Os dados referem-se aos eventos existentes no conjunto de dados, antes e depois da aplicação da SMOTE, processados com a finalidade de verificar se houve melhoria com a utilização dessa técnica. Apesar de ter ocorrido uma pequena diminuição no valor da acurácia, observou-se uma melhoria nas métricas de F1-score e Recall, o que justifica o uso da SMOTE.

Tabela 5.1: Resultados da aplicação da SMOTE no conjunto de dados.

	Acurácia		F1-Score		Recall		Número de Eventos
	μ	σ	μ	σ	μ	σ	(A)normal (N)ormal
Antes SMOTE	98,3564	0,0906	95,0201	0,2869	90,9867	0,5299	128.706 (82,77%): A 26.805 (17,23%): N
Após SMOTE	97,2569	0,1462	97,2232	0,1497	96,0467	0,2174	128.706 (50,00%): A 128.704 (50,00%): N

A diminuição da acurácia pode ser atribuída ao fato de o modelo, ao se tornar mais sensível à classe positiva após a aplicação da SMOTE, passou a classificar mais amostras como positivas, o que aumentou o número de falsos positivos. Por outro lado, o aumento do F1-score

e do Recall ocorreu porque o modelo conseguiu identificar mais corretamente os verdadeiros positivos.

5.2.4 Seleção de atributos

Para a etapa de escolha dos melhores atributos, com o intuito de reduzir a dimensionalidade, foi implementado um *ensemble* baseado em (Leevy et al., 2021), utilizando as métricas estatísticas *Information Gain* (IG), *Gain Ratio* (GR) e *Chi-Squared* (CS) e técnicas de aprendizado supervisionado, como Florestas Aleatórias (FA), *Categorical Boosting* (CatBoost), *eXtreme Gradient Boosting* (XGBoost) e *Light Gradient-Boosting Machine* (LightGBM).

A escolha dessas técnicas deve-se ao fato de que as mesmas estão relacionadas a algoritmos ou classificadores baseados em árvores de decisão, ou que, de algum modo, mantêm uma conexão com elas, fato que está alinhado à linha de pesquisa deste trabalho.

Para a execução das métricas IG, GR e CS, não foram fornecidos parâmetros especiais de configuração. Já os algoritmos de aprendizado supervisionado foram executados com seus hiperparâmetros *default*, com o número de árvores de decisão (*n_estimators*=100) e a profundidade máxima de cada árvore (*max_depth*=10). Para possibilitar a reprodução dos experimentos, foi utilizado (*random_state*=42) para que as operações que envolvem aleatoriedade produzissem sempre os mesmos resultados.

Os atributos do conjunto de dados foram processados por cada uma das técnicas elencadas, sendo que cada uma classificou os atributos em ordem decrescente de importância, mantendo os 32 mais relevantes. Os atributos escolhidos por cada classificador podem ser vistos no Apêndice A - Tabela A.2.

Para selecionar a melhor combinação de atributos, os mesmos foram reunidos em grupos, identificados pelas letras *A* até *E*. Cada grupo identificado, é, respectivamente, o resultado da técnica de seleção de atributos, considerando a participação dos atributos em pelo menos três dos sete grupos, quatro dos sete e assim progressivamente até a participação em todos os sete grupos de resultados. Cada grupo foi submetido ao processo de validação cruzada utilizando florestas aleatórias, usando dez *folds* (k=10) e registradas as métricas de acurácia para comparação com os demais grupos.

O teste de normalidade de Shapiro-Wilk foi aplicado para a acurácia de cada grupo e pode ser visto na Tabela 5.2. O *p-value* em todos os casos foi superior à significância definida de 0,05, o que não permite rejeitar a hipótese nula (H_0) de que os dados seguem uma distribuição normal.

Tabela 5.2: Acurácias dos grupos de atributos A, B, C, D e E.

Iteração (*fold*)	Grupo A	Grupo B	Grupo C	Grupo D	Grupo E
1	97,2493	97,0116	97,3950	97,2868	97,2149
2	97,0411	97,2044	97,2098	97,4021	97,2744
3	97,1732	97,2888	97,1769	97,3855	97,2844
4	97,2053	97,3971	97,1193	97,2867	97,1354
5	97,3334	97,2686	97,2222	97,1467	97,0014
6	97,3253	97,2003	97,1687	97,1590	97,0262
7	97,3013	97,3931	97,3621	97,3402	97,1105
8	97,3533	97,1642	97,1646	97,4884	97,2594
9	97,3453	97,5136	97,3374	97,2414	96,8871
10	97,0770	97,3449	97,1851	97,4967	96,9665
μ	97,2405	97,2787	97,2341	97,3234	97,1160
σ	0,1133	0,1426	0,0952	0,1226	0,1415
(*p-value* > 0,05)	0,1240	0,9772	0,0895	0,6327	0,4211

Como o resultado do teste de normalidade indicou que os dados seguem uma distribuição normal, o próximo teste feito foi a ANOVA, técnica estatística usada para comparar as médias de três ou mais grupos diferentes e verificar se existem diferenças significativas entre elas. A hipótese nula (H_0) afirma que as médias dos grupos são iguais e a hipótese alternativa (H_1) que pelo menos uma das médias dos grupos é diferente.

Os dados da ANOVA, apresentados anteriormente na Tabela 4.6, indicaram diferenças significativas entre os grupos, evidenciadas pelo *p-value* inferior a 0,05. Assim, a H_0 é rejeitada, permitindo afirmar com um nível de significância de 95% que pelo menos um dos grupos é diferente dos demais. A análise apontou diferenças entre os grupos *B* e *E*, assim como entre os grupos *D* e *E*. Uma avaliação mais detalhada, realizada em conjunto com os dados da Tabela 4.5, evidenciou que:

- O grupo *B* possui 38 atributos, com uma distribuição de classes de 48,38% de eventos anormais e 51,62% de eventos normais.
- O grupo *D* apresenta 19 atributos, com 47,11% de eventos anormais e 52,89% de eventos normais.
- O grupo *E* conta com apenas seis atributos, com 40,67% de eventos anormais e 59,33% de eventos normais.

O uso do grupo *E* foi descartado devido ao maior desequilíbrio na distribuição das classes e ao número reduzido de atributos em comparação aos grupos *B* e *D*. Os grupos *B* e *D* mantiveram uma proporção próxima na distribuição dos eventos, mas o grupo *B* possui o dobro de atributos em comparação ao grupo *D*.

Dessa forma, o grupo *D* foi o escolhido para a construção do conjunto de dados com os melhores atributos. Essa escolha se justifica pelo fato de o grupo *D* manter um equilíbrio na

distribuição dos eventos entre as classes, ao mesmo tempo em que possui a metade do número de atributos do grupo *B*.

Como resultado da aplicação da técnica *ensemble* para seleção dos melhores atributos, o número de atributos foi reduzido de 64 para 19, considerando ainda que tarefas anteriores de pré-processamento também removeram atributos inválidos.

5.2.5 Criação de conjuntos de dados com proporções distintas

Após a seleção dos melhores atributos, foram criados quatro grupos de dados: completo, balanceado, desbalanceamento moderado e desbalanceamento extremo. Esses grupos foram identificados, respectivamente, como grupo 1, 2, 3 e 4 (subseção 4.3.1.5).

Em cada grupo, foram gerados seis conjuntos de dados, cada um contendo 19 atributos e diferentes proporções de eventos. Essa estratégia permitiu que os modelos fossem avaliados em condições distintas de distribuições dos dados e possibilitou a participação de seis clientes no AF, cada um utilizando um conjunto específico em cada experimento.

No caso do grupo 1, como forma de otimização do processamento, apenas uma cópia do conjunto de dados cd_A100-N100 foi criada, em vez de seis, uma vez que replicar os mesmos dados várias vezes não seria necessário.

5.2.6 Distribuição final dos eventos após pré-processamento

Ao término do pré-processamento do conjunto de dados, o número total de eventos foi reduzido de 625.783 para 242.882, uma diminuição de aproximadamente 61,19%, sendo que a maior parte desta redução foi obtida pela remoção de eventos originais duplicados. A classe anomalia, que inicialmente correspondia a 93,60% dos eventos, passou a representar 47,11% com 114.419 registros, enquanto a classe normal aumentou de 6,40% para 52,89% com 128.463 registros. Essas informações estão exibidas na Figura 5.1.

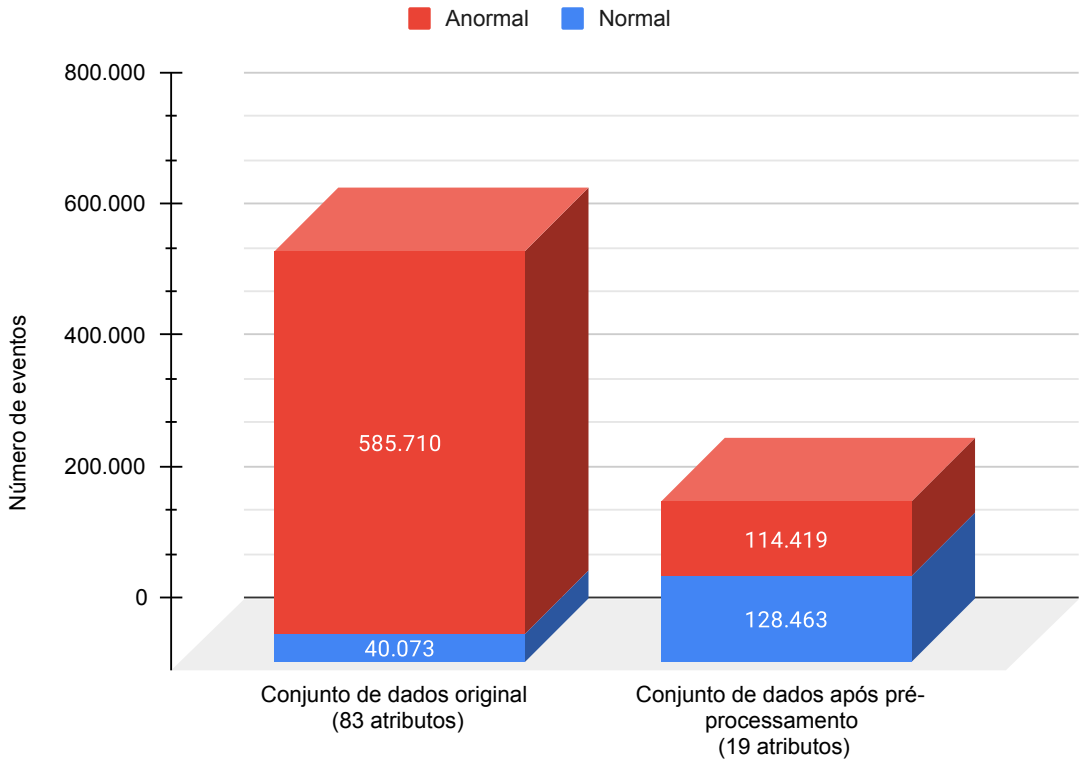

Figura 5.1: Distribuição final dos eventos no conjunto de dados após pré-processamento. Fonte: O autor.

5.3 Processamento centralizado

Na fase do Aprendizado Centralizado (AC), foi utilizado o grupo 1, representado pelo conjunto de dados cd_A100-N100, para gerar o modelo base de florestas aleatórias.

Os eventos selecionados passaram por um processo de validação cruzada, utilizando dez *folds* (k=10), aplicando florestas aleatórias. Após cada iteração da validação cruzada, o valor de acurácia foi registrado, o qual refletia o desempenho do modelo naquela divisão específica dos dados. Os resultados parciais das acurácias foram comparados, e a floresta com o melhor resultado foi reservada.

Esse processo permitiu identificar qual configuração de floresta obteve o melhor desempenho ao longo das iterações, sendo que o melhor resultado foi obtido na 10ª iteração. Os resultados do processamento centralizado podem ser vistos na Figura 5.2 e são apresentados na Tabela 5.3.

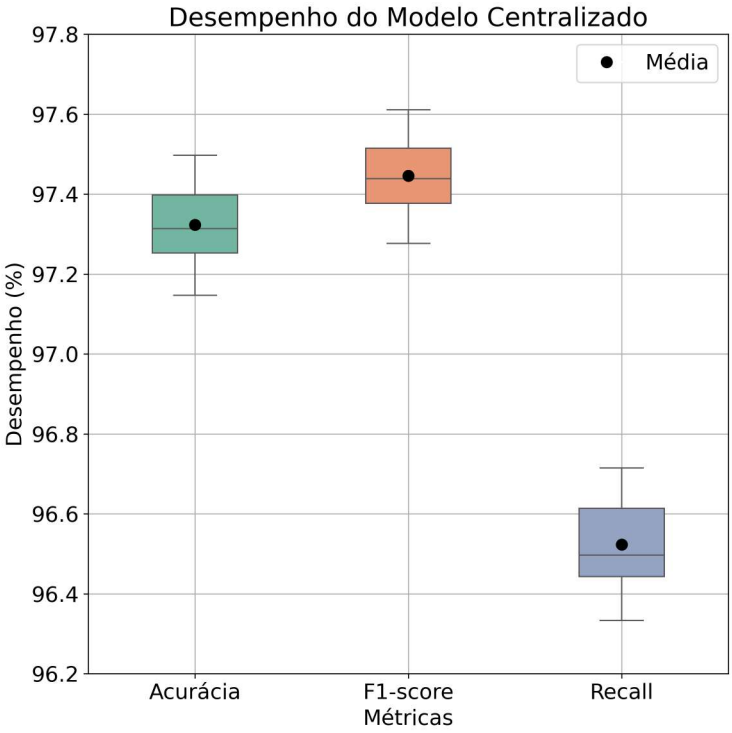

Figura 5.2: Desempenho do processamento centralizado para criação do modelo base.
Fonte: O autor.

Tabela 5.3: Resultados do processamento centralizado para criação do modelo base.

Iteração (*fold*)	Acurácia	F1-score	Recall
1	97,2868	97,4093	96,4350
2	97,4021	97,5188	96,5206
3	97,3885	97,5019	96,4658
4	97,2867	97,4138	96,6137
5	⇓ 97,1467	97,2763	96,3335
6	97,1591	97,2916	96,4736
7	97,3403	97,4635	96,6137
8	97,4885	97,6039	96,7149
9	97,2414	97,3657	96,3880
10	⇑ ***97,4967***	***97,6108***	***96,6763***
μ	97,3234	97,4456	96,5235
σ	0,1226	0,1166	0,1267

A floresta que obteve o melhor desempenho é selecionada como modelo base, servindo como ponto de partida para o processamento federado. A justificativa por escolher o melhor modelo centralizado é garantir que a comparação com os resultados dos modelos federados seja feita com valores realistas e atingíveis.

5.4 Processamento federado

O Aprendizado Federado (AF) foi dividido em quatro experimentos (AF1, AF2, AF3 e AF4), sendo que cada um foi repetido por dez vezes. Cada experimento refere-se ao processa-

mento exclusivo de um grupo de dados (seção 4.3.1.5). Em cada execução de um experimento, são executadas 50 épocas, que é a troca de informações entre servidor e clientes. Desta forma de execução, conclui-se que cada grupo de dados foi analisado por 500 vezes, correspondentes a dez execuções independentes de 50 épocas cada.

Cada experimento, repetido por dez vezes, resultou no registro de dez valores de acurácia, os quais são utilizados para a avaliação do modelo federado, e referem-se ao melhor resultado em cada repetição.

Na execução da primeira época de cada experimento, o servidor envia aos clientes uma cópia do modelo global de floresta aleatória com 100 árvores. Cada cliente, então, adiciona novas árvores de decisão à floresta, até o limite de 1.000 árvores. Este valor foi definido de forma arbitrária, mas deve ser superior ao número de árvores que estão no modelo global recebido, permitindo novas árvores de decisão possam ser adicionadas na floresta.

Quanto à escolha de 100 árvores para o modelo global, foram realizados testes preliminares com diferentes valores desse hiperparâmetro. Contudo, esse número apresentou os melhores resultados e, como a busca exaustiva pela otimização de hiperparâmetros não faz parte das atividades deste trabalho, optou-se por utilizá-lo.

Ao final dos treinamentos e testes pelos clientes, a melhor árvore identificada é enviada para o servidor para agregação. A agregação é projetada para minimizar a comunicação e otimizar o tempo de processamento, tornando o modelo escalável mesmo com vários participantes.

À medida que as épocas são executadas, novas árvores são adicionadas à floresta global pelo servidor, que a redistribui novamente aos clientes, agora, com um número maior de árvores.

A Figura 5.3 representa como ocorre a evolução do desempenho dos modelos federados ao longo das épocas de treinamento. Inicialmente, o modelo global registra uma acurácia de 97,4967% e é composto por 100 árvores de decisão, parâmetros que estabelecem a referência inicial do modelo. Conforme as épocas avançam e novas árvores são incorporadas ao modelo global, observa-se a melhoria gradual no desempenho.

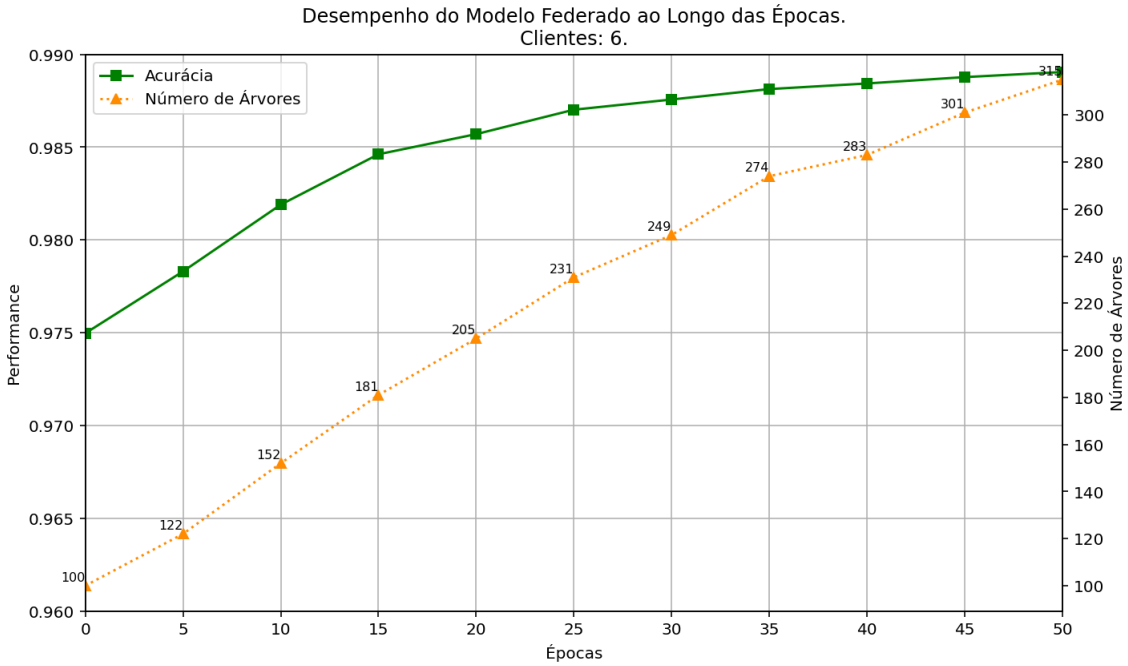

Figura 5.3: Exemplo da evolução do aprendizado federado ao longo das épocas.
Fonte: O autor.

Processamento federado - AF1

No 1º experimento do AF, cada cliente recebeu uma cópia do conjunto de dados do grupo 1, representado pelo conjunto de dados cd_A100-N100. Esse conjunto contém todos os eventos disponíveis, totalizando 114.419 (47,11%) da classe anomalia e 128.463 (52,89%) da classe normal.

Essa configuração foi escolhida com o objetivo de simular um cenário em que há homogeneidade entre os dados locais dos clientes, ou seja, todos possuem o mesmo volume e distribuição de amostras. Dessa forma, o experimento se aproxima de uma abordagem de aprendizado centralizado e permite avaliar os efeitos do processo de agregação federada sobre o desempenho do modelo global.

Durante a execução dos experimentos, a floresta global manteve uma média de 323 árvores e, o menor valor de acurácia foi de 99,1628%, superior ao do modelo centralizado que é de 97,4967%. Os resultados do AF1 (acurácia, F1-score e Recall) podem ser vistos na Figura 5.4 e na Tabela 5.4.

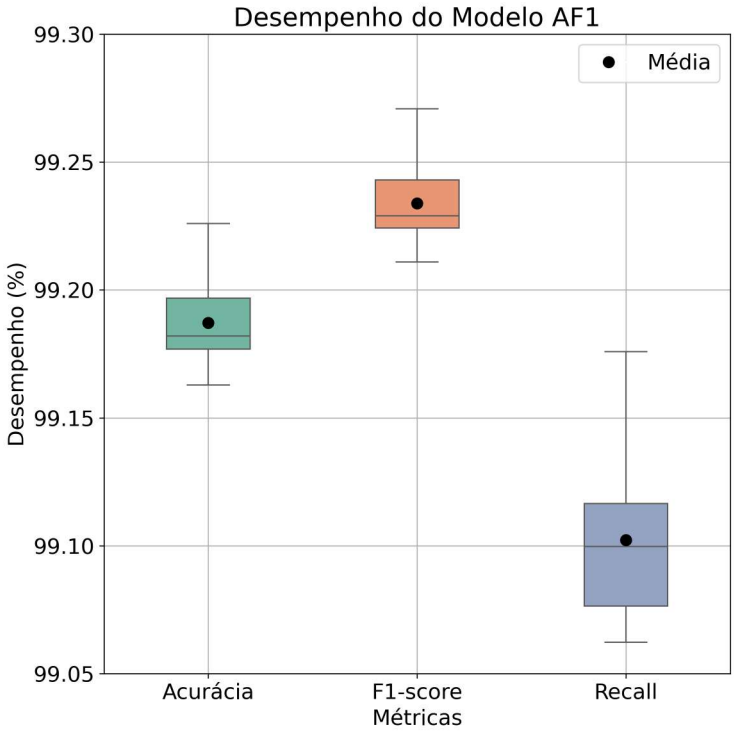

Figura 5.4: Desempenho do modelo AF1.
Fonte: O autor.

Tabela 5.4: Desempenho do modelo AF1.

Execução	Acurácia	F1-score	Recall	Número de Árvores na Floresta
1	99,1958	99,2421	99,1165	331
2	99,1985	99,2447	99,1191	329
3	99,1779	99,2253	99,1010	320
4	99,1724	99,2201	99,0881	314
5	⇓ 99,1628	99,2109	99,0623	310
6	99,1793	99,2264	99,0726	312
7	99,1971	99,2434	99,1165	331
8	99,1766	99,2238	99,0726	312
9	99,1848	99,2317	99,0984	321
10	⇑ 99,2260	99,2708	99,1759	348
μ	99,1871	99,2339	99,1023	323
σ	0,0179	0,0170	0,0328	12

Processamento federado - AF2

O 2º experimento do AF distribuiu para os clientes os conjuntos de dados do grupo 2, que são formados por registros únicos. Cada cliente recebeu aproximadamente 19.070 (16,60%) eventos da classe anomalia e 21.410 (16,60%) da classe normal.

Neste cenário, os clientes passaram a trabalhar com conjuntos distintos dos dados, embora ainda mantendo uma distribuição equilibrada entre as classes em cada conjunto local. Essa forma de particionamento apresenta uma situação realista de AF, em que os dados estão distri-

buídos entre diferentes fontes e não estão replicados.

O número médio de árvores na floresta global foi de 207 árvores, e a menor acurácia foi de 97,7973%, superior aos 97,4967% do modelo centralizado. Os resultados de acurácia, F1-score e Recall do AF2 podem ser vistos na Figura 5.5 e Tabela 5.5.

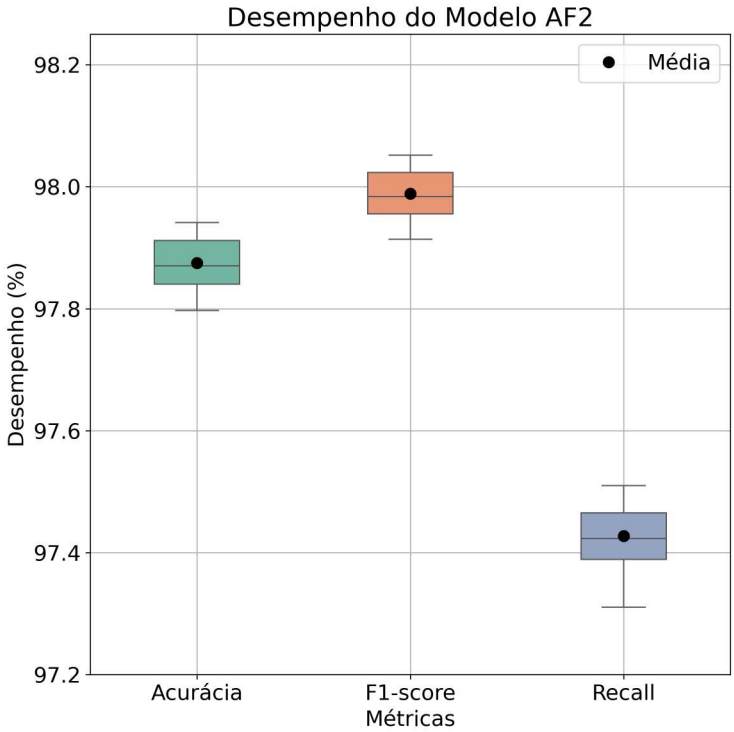

Figura 5.5: Desempenho do modelo AF2.
Fonte: O autor.

Tabela 5.5: Desempenho do modelo AF2.

Execução	Acurácia	F1-score	Recall	Número de Árvores na Floresta
1	97,8714	97,9849	97,4141	209
2	97,8330	97,9485	97,3779	192
3	97,8618	97,9763	97,4296	195
4	97,9098	98,0216	97,4658	238
5	97,9126	98,0241	97,4632	216
6	⇑ 97,9414	98,0517	97,5071	236
7	⇓ 97,7973	97,9140	97,3108	168
8	97,9222	98,0339	97,5097	226
9	97,8687	97,9824	97,4167	207
10	97,8330	97,9486	97,3805	184
μ	97,8751	97,9886	97,4275	207
σ	0,0460	0,0439	0,0618	23

Processamento federado - AF3

No 3º experimento do AF, cada cliente recebeu um dos conjuntos de dados do grupo 3. Nessa configuração, três conjuntos (cd_A100-N25, cd_A100-N50 e cd_A100-N75) apresentam

100% de eventos anormais, enquanto a proporção de eventos normais varia entre 25%, 50% e 75%, respectivamente. Nos três conjuntos restantes (cd_A25-N100, cd_A50-N100 e cd_A75-N100), ocorre o inverso: os eventos normais compõem 100% e os eventos anormais aparecem em proporções de 25%, 50% e 75%.

Essa configuração foi considerada para representar um cenário de distribuição não homogênea entre os clientes, com muitos eventos compartilhados entre os clientes, uma característica que pode ocorrer em aplicações de AF. A floresta global manteve, em média, 305 árvores, e a menor acurácia registrada foi de 98,8472%, valor superior ao obtido pelo modelo centralizado, cuja acurácia foi de 97,4967%. Os resultados da acurácia, do F1-score e do Recall do AF3 podem ser vistos na Figura 5.6 e na Tabela 5.6.

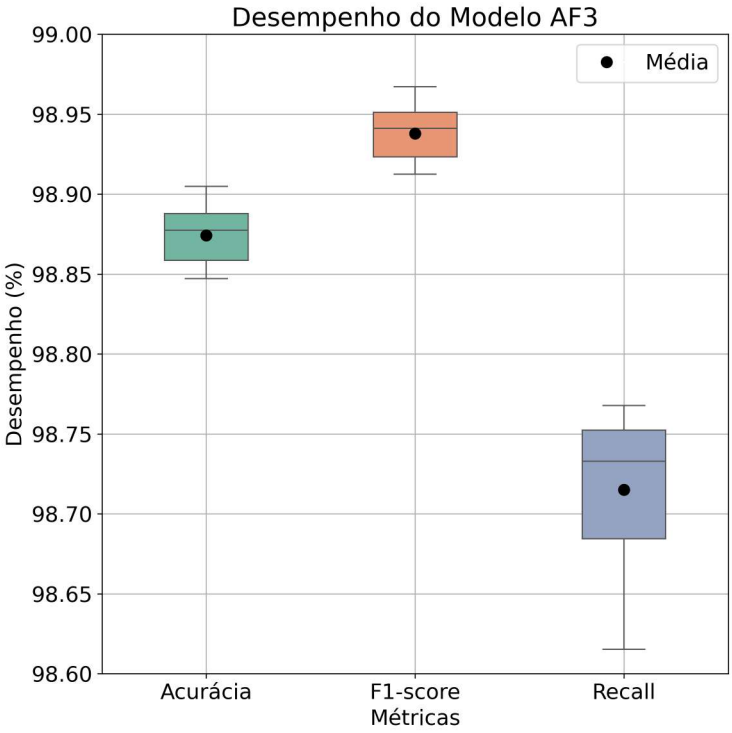

Figura 5.6: Desempenho do modelo AF3.
Fonte: O autor.

Tabela 5.6: Desempenho do modelo AF3.

Repetição	Acurácia	F1-score	Recall	Número de Árvores na Floresta
1	⇓ 98,8472	98,9125	98,6825	298
2	98,8733	98,9376	98,7523	303
3	⇑ 98,9048	98,9671	98,7600	320
4	98,8815	98,9446	98,6903	314
5	98,8856	98,9491	98,7523	314
6	98,8884	98,9518	98,7678	304
7	98,8938	98,9567	98,7497	315
8	98,8595	98,9238	98,6644	298
9	98,8582	98,9231	98,7161	288
10	98,8486	98,9130	98,6153	294
μ	98,8741	98,9379	98,7151	305
σ	0,0199	0,0190	0,0505	11

Processamento federado - AF4

Por fim, no 4º experimento do AF, cada cliente recebeu um dos conjuntos de dados do grupo 4, caracterizado por um desbalanceamento extremo entre as classes, com proporções variando entre 1% e 11% para uma classe, enquanto a outra estava presente em 100% dos registros.

Essa configuração representa um dos cenários mais desafiadores para o aprendizado federado, pois simula situações reais em que diferentes fontes de dados (clientes) enfrentam forte desequilíbrio local, contribuindo de forma para o modelo global, principalmente pela existência de poucos registros em uma das classes.

A floresta global manteve uma média de 212 árvores, e a menor acurácia foi de 97,7698%, acima dos 97,4967% do modelo centralizado. Os resultados de acurácia, F1-score e Recall do AF4 estão representados na Figura 5.7 e na Tabela 5.7.

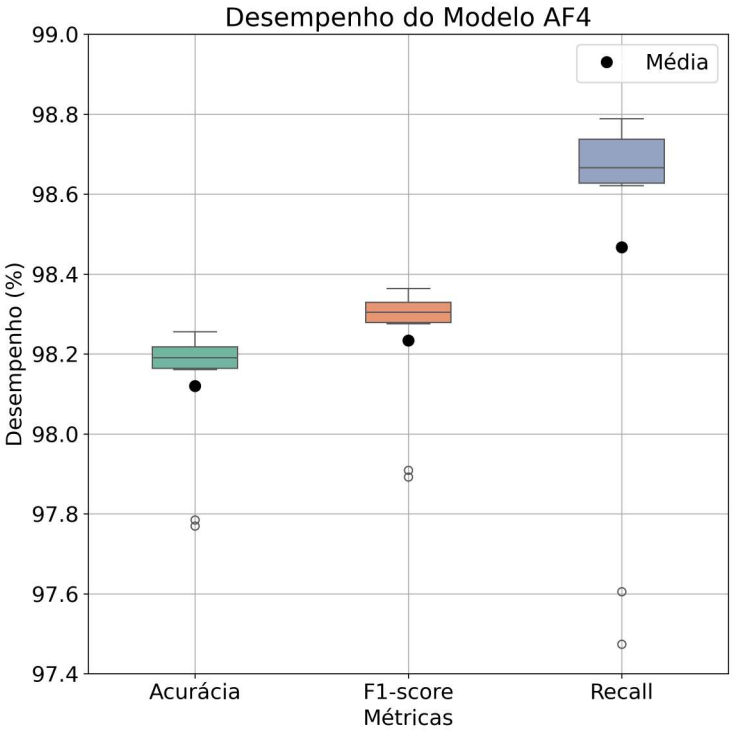

Figura 5.7: Desempenho do modelo AF4.
Fonte: O autor.

Tabela 5.7: Desempenho do modelo AF4.

Repetição	Acurácia	F1-score	Recall	Número de Árvores na Floresta
1	98,2118	98,3249	98,7884	223
2	98,2035	98,3166	98,7471	240
3	98,1610	98,2752	98,6205	204
4	98,2406	98,3495	98,6696	258
5	98,1733	98,2888	98,7497	205
6	⇓ 97,7698	97,8920	97,4735	140
7	98,2200	98,3301	98,6489	237
8	97,7849	97,9088	97,6053	132
9	98,1775	98,2919	98,7058	216
10	⇑ 98,2557	98,3633	98,6618	262
μ	98,1198	98,2341	98,4671	212
σ	0,1829	0,1780	0,4926	45

5.5 Avaliação dos resultados

Para avaliar os resultados dos experimentos, foi aplicado o teste de Shapiro-Wilk com o objetivo de verificar a normalidade dos dados. A Tabela 5.8 apresenta os resultados da acurácia obtidos para todos os experimentos, incluindo o AC. A hipótese nula (H_0) sugere que os dados seguem uma distribuição normal, enquanto a hipótese alternativa (H_1), os dados não seguem uma distribuição normal.

Os valores do *p-value* para os experimentos AC, AF1, AF2 e AF3 foram superiores à significância de 0,05, indicando que esses seguem uma distribuição normal. Entretanto, o AF4 obteve o valor de 0,0003 para o *p-value*, inferior ao limite de significância. Desta maneira, a H_0 deve ser rejeitada, sugerindo que os dados não seguem uma distribuição normal.

Tabela 5.8: Resultados do teste de normalidade de todos os experimentos.

Iteração	Acurácia				
	AC	AF1	AF2	AF3	AF4
1	97,2868	99,1958	97,8714	98,8472	98,2118
2	97,4021	99,1985	97,8330	98,8733	98,2035
3	97,3855	99,1779	97,8618	98,9048	98,1610
4	97,2867	99,1724	97,9098	98,8815	98,2406
5	97,1467	99,1628	97,9126	98,8856	98,1733
6	97,1591	99,1793	97,9414	98,8884	97,7698
7	97,3403	99,1971	97,7973	98,8938	98,2200
8	97,4885	99,1766	97,9222	98,8595	97,7849
9	97,2414	99,1848	97,8687	98,8582	98,1775
10	97,4967	99,2260	97,8330	98,8486	98,2557
μ	97,3234	99,1871	97,8751	98,8741	98,1198
σ	0,1226	0,0179	0,0460	0,0199	0,1829
(*p-value* > 0,05)	0,6328	0,3959	0,7588	0,5725	0,0003

Considerando que os resultados indicaram que os dados não seguem distribuição normal, o teste estatístico adequado para verificar a existência de diferenças entre os experimentos é o Kruskal-Wallis. A hipótese nula (H_0) sugere que todos os grupos têm a mesma distribuição ou a mesma mediana, e a hipótese alternativa (H_1) que pelo menos um grupo tem distribuição diferente das demais.

Após a aplicação do teste, os resultados levaram à rejeição da H_0, indicando diferenças estatisticamente significativas entre os experimentos, evidenciadas por um *p-value* inferior a 0,05. Para identificar especificamente quais pares de experimentos apresentaram essas diferenças, foi aplicado o teste *post-hoc* de Dunn, que aponta os pares com diferenças significativas e seus respectivos níveis de significância. Esses resultados estão detalhados na Tabela 5.9:

Tabela 5.9: Resultados do teste Kruskal-Wallis entre os pares de experimentos.

Comparação entre pares	Kruskal-Wallis (*p-value* < 0,05)	Diferença
AC x AF1	0,0000	✓
AC x AF2	0,6566	
AC x AF3	0,0000	✓
AC x AF4	0,0576	
AF1 x AF2	0,0002	✓
AF1 x AF3	1,0000	
AF1 x AF4	0,0074	✓
AF2 x AF3	0,0576	
AF2 x AF4	1,0000	
AF3 x AF4	0,6566	

Comparação entre AC x AF1 e AC x AF3

Foram identificadas diferenças significativas entre os pares de experimentos AC x AF1 e AC x AF3. Esses resultados indicam que o modelo AF1, treinado com o conjunto de dados completo e o modelo AF3, treinado com o conjunto de dados com desbalanceamento moderado, obtiveram um desempenho estatisticamente diferente, mas superior ao modelo AC. O AC foi treinado de forma centralizada com uma única floresta de 100 árvores e os modelos AF1 e AF3, por outro lado, foram construídos agregando as melhores árvores de decisão dos seis clientes ao longo de 50 épocas. No caso do AF1, onde todos os clientes tinham o mesmo conjunto de dados completo, a colaboração resultou em um modelo global mais robusto. Embora o AF3 utilizasse dados moderadamente desbalanceados entre clientes, o processo federado ainda conseguiu superar o modelo centralizado.

Comparação AF1 x AF2

O modelo AF1, onde cada cliente tinha o conjunto de dados completo, apresentou desempenho estatisticamente superior ao modelo AF2, onde cada cliente tinha um conjunto de dados balanceado, com eventos únicos e não compartilhados entre si. A floresta global no AF2 teve, em média, 207 árvores, menos que em AF1, que tinha 323 árvores. Embora ambos os cenários usem dados balanceados (IID), a diferença pode residir na distribuição dos dados: no AF1, todos clientes têm a mesma distribuição geral de eventos e sem diversidade, enquanto no AF2, cada cliente tem uma partição exclusiva do conjunto de dados.

O desempenho superior do AF1 sugere que ter clientes treinando com a mesma distribuição completa de dados, e agregando seus modelos, resultou em um modelo global mais forte do que quando cada cliente aprende apenas a partir de uma parte única (exclusiva) do conjunto de dados total, como no AF2. A menor quantidade de árvores na floresta do AF2 também pode indicar um processo de convergência ligeiramente diferente ou menos eficaz para essa distribuição de dados.

Comparação AF1 x AF4

O modelo AF1, com dados completos e balanceados distribuídos entre os clientes, apresentou desempenho estatisticamente superior ao modelo AF4, onde os clientes possuíam conjuntos de dados com desbalanceamento extremo entre as classes. O desempenho em cenários de dados non-IID, especialmente com desbalanceamento severo como no AF4, é um desafio conhecido no AF. Clientes com distribuições de classes muito distintas aprendem padrões diferentes, o que pode dificultar a convergência do modelo global e impactar negativamente o desempenho. A diferença entre AF1 e AF4 confirma o impacto negativo do desbalanceamento extremo de dados non-IID no desempenho do modelo federado, em comparação com um cenário IID onde todos os clientes compartilham a mesma distribuição completa e balanceada.

Avaliação geral

Foi observada uma relação direta entre o número de árvores geradas em cada experimento federado e seu desempenho final: quanto maior o número de árvores na floresta, melhores foram os resultados, o que é esperado para esse tipo de modelo de AM.

Os experimentos AF1 e AF3 foram os que apresentaram o maior número de árvores na floresta e obtiveram as melhores acurácias. Já para os experimentos AF2 e AF4, eles apresentaram um número médio de árvores menor do que experimentos anteriores, mas seus resultados de acurácia ainda foram superiores aos do AC. Esses resultados podem ser vistos na Figura 5.8.

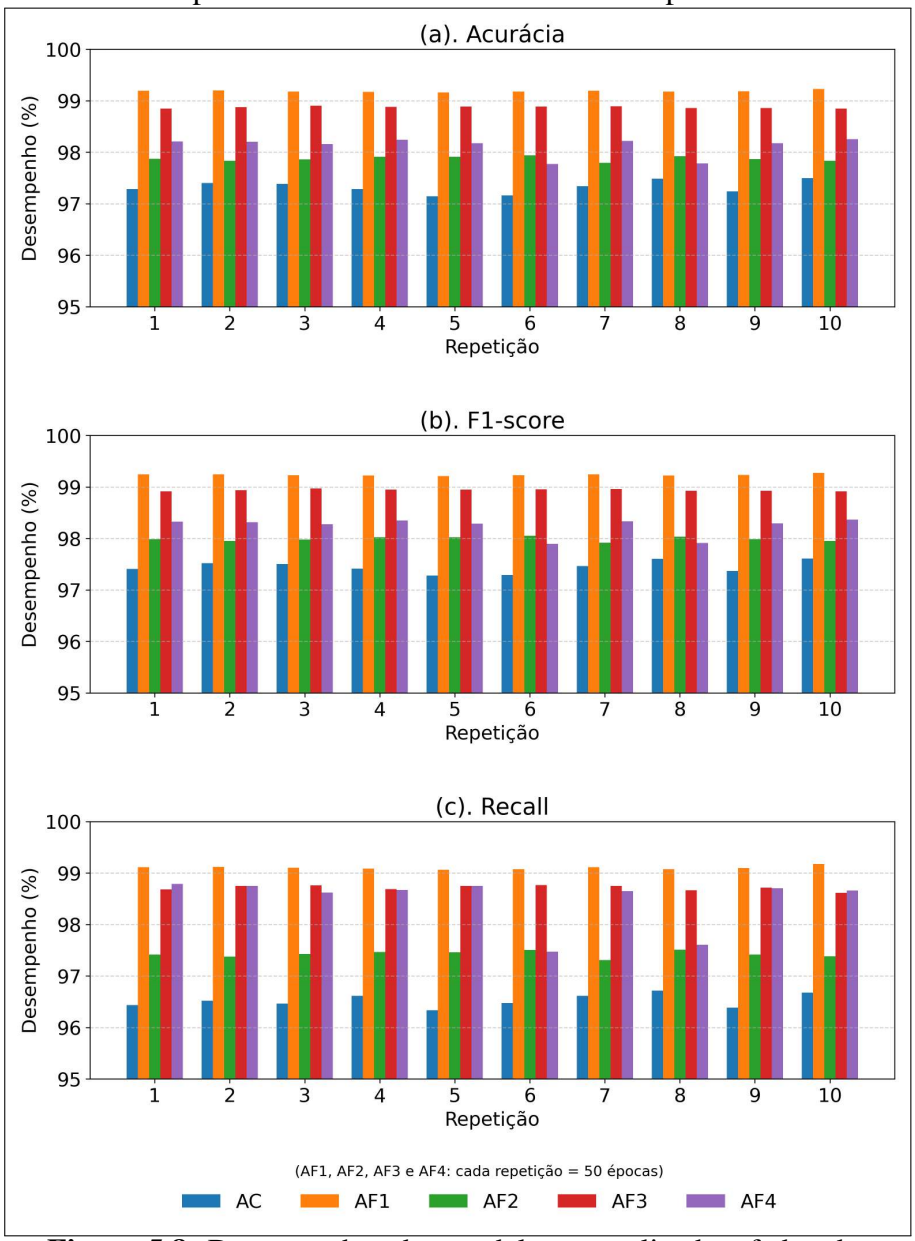

Figura 5.8: Desempenhos dos modelos centralizado e federado.
Fonte: O autor.

Em termos quantitativos, os modelos federados mantiveram entre 207 e 323 árvores na floresta global, enquanto o modelo centralizado utilizou 100 árvores. Como o modelo federado

alcançou um desempenho próximo ao do modelo centralizado que tem menos árvores, isso pode sugerir que a complexidade do problema analisado não é grande, ou que a eficácia do modelo federado pode estar mais relacionada à seleção das melhores árvores do que à quantidade delas.

Outro fator que pode ter contribuído para o modelo federado não alcançar resultados expressivos é o fato de não ter sido realizada nenhuma otimização dos hiperparâmetros dos algoritmos, os quais foram utilizados com suas configurações padrão. Ao ajustar os hiperparâmetros, a capacidade do modelo de se adaptar a diferentes padrões nos dados pode ser aprimorada, resultando em um desempenho significativamente melhor. Entretanto, a partir deste estudo, a linha base de resultados está criada, podendo servir como parâmetro inicial para qualquer estudo com vistas a melhorar a sua performance.

Contudo, com base nos dados apresentados nas seções anteriores, ainda assim comprova-se que o método proposto para o AF supera o desempenho do AC. No entanto, essa superioridade não se reflete necessariamente em um ganho expressivo de acurácia, mas sim em aspectos como estabilidade e capacidade de generalização dos dados.

A literatura aponta condições em que o AF é mais vantajoso do que o AC, especialmente em cenários onde a privacidade e a descentralização dos dados são fundamentais. No entanto, também se sabe que o desempenho do AF pode ser inferior ao do AC devido às características non-IID dos dados distribuídos entre os clientes. Isso significa que clientes com diferentes quantidades de eventos e distribuições de classes podem impactar negativamente a convergência e o desempenho geral do modelo federado (McMahan et al., 2016).

Para o cálculo do desempenho geral do modelo federado, foram utilizados os valores médios de acurácia, F1-score e recall obtidos nos experimentos e, a partir desses valores, foi calculada a média aritmética simples. Os dados utilizados no cálculo e os resultados estão apresentados na Tabela 5.10:

Tabela 5.10: Cálculo do desempenho final do modelo federado.

	Acurácia	F1-score	Recall	Nº de Árvores
AF1	99,1871	99,2339	99,1023	323
AF2	97,8751	97,9886	97,4275	207
AF3	98,8741	98,9379	98,7151	305
AF4	98,1198	98,2341	98,4671	212
μ	98,51	98,60	98,43	262
σ	0,62	0,58	0,72	61

5.6 Considerações em relação a trabalhos relacionados

Após analisar as contribuições alcançadas pelo método proposto, retoma-se uma das questões levantadas na proposta:

- Comparar, baseado na literatura associada, o desempenho do modelo de florestas aleatórias com outros modelos de aprendizado de máquina em termos de métricas de classifica-

ção, como acurácia, no ambiente de aprendizado federado.

Com base nas métricas de desempenho de todos os experimentos do AF, foi alcançado 98,51% para a acurácia, 98,60% para o F1-score e 98,43% para o Recall, resultados que colocam esta pesquisa em posição de destaque em comparação com os estudos apresentados na revisão da literatura. A classificação correspondente pode ser consultada na Tabela 5.11.

Basicamente, os trabalhos relacionados exibem resultados de acurácia e poucos apresentam outras métricas, como F1-score ou Recall. Em cenários onde uma classe é muito mais frequente do que outra, um modelo pode atingir alta acurácia simplesmente prevendo sempre a classe majoritária, sem realmente ser eficaz na identificação da classe minoritária. Sendo assim, para confirmar os resultados do modelo, são exibidas as métricas de F1-score e Recall deste trabalho.

Tabela 5.11: Comparação de resultados com os trabalhos relacionados.

Autores	Classificação	Seleção de atributos?	Acurácia	F1-score	Recall
(Rey et al., 2022)	Binária e multiclasses	Sim	99,96		99,98
(Friha et al., 2022)	Multiclasses	Sim	99,71		
(Wardana et al., 2024)	Binária	Sim	99,68	99,68	99,68
(Liu et al., 2022)	Binária e multiclasses	Sim	99,50		
este trabalho	***Binária***	Sim	***98,51***	***98,60***	***98,43***
(Ribera, 2024)	Binária	Sim	98,30		
(Cholakoska et al., 2023)	Multiclasses	Sim	98,30		
(Nakip et al., 2023)	Multiclasses	Sim	98,00		100,00
(Souza et al., 2020)	Binária	Sim	98,00		
(Neto et al., 2022)	Binária	Sim	96,00		
(Markovic et al., 2022)	Binária	Sim	93,51		
(Campos et al., 2022)	Multiclasses	Sim	91,00		
(Mothukuri et al., 2022)	Multiclasses	Sim	90,25		
(Alkhpor and Alserhani, 2023)	Multiclasses	Sim	90,18	90,09	90,18

5.7 Considerações finais

Neste capítulo foram apresentados e analisados os resultados dos experimentos conduzidos para avaliar a eficácia do método proposto, que combina florestas aleatórias e aprendizado federado, utilizando o conjunto de dados IoTID20. As etapas de pré-processamento foram detalhadas, com destaque para o balanceamento de classes e a seleção de atributos, o que resultou em conjuntos de dados otimizados para o treinamento dos modelos.

Foram realizados experimentos centralizados e federados, comparando o desempenho de modelos de florestas aleatórias em diferentes configurações de distribuição de dados entre os clientes. A avaliação dos resultados utilizando métricas de desempenho revelou que os modelos

federados alcançaram desempenho superior ao modelo base centralizado. No capítulo seguinte, são apresentadas as conclusões deste trabalho, bem como sugestões para o desenvolvimento de futuras pesquisas.

Capítulo 6

Conclusão

Com base nas buscas bibliográficas realizadas, tudo indica que este é o primeiro trabalho a utilizar florestas aleatórias em um ambiente de Aprendizado Federado (AF) com o conjunto de dados IoTID20. Essa abordagem combina a robustez das florestas aleatórias na classificação de intrusões aos benefícios do AF, que viabiliza a colaboração entre múltiplos dispositivos, sem a necessidade de centralizar os dados. O uso do IoTID20, amplamente reconhecido na análise de segurança em ambientes de Internet das Coisas (IoT), posiciona este estudo como uma contribuição relevante para o avanço da pesquisa em segurança cibernética e aprendizado de máquina (AM).

Durante a análise comparativa com os trabalhos relacionados, observou-se que poucos estudos da literatura aplicam validação estatística sobre os resultados de seus modelos. Muitos limitam-se à apresentação de métricas pontuais, como acurácia, sem fornecer informações adicionais que permitam avaliar esses resultados. A ausência dessa validação compromete a interpretação precisa dos modelos, sobretudo em cenários com dados desbalanceados. Neste trabalho, a utilização de métricas complementares como F1-score e Recall, aliada à repetição sistemática dos experimentos, permitiu superar essa limitação, agregando rigor e confiabilidade à análise. Essa escolha metodológica posiciona esta pesquisa em um patamar diferenciado entre os estudos sobre aprendizado federado, evidenciando um compromisso com a qualidade estatística dos resultados apresentados.

As tarefas de pré-processamento aplicadas ao conjunto de dados contribuíram significativamente para otimizar o desempenho do modelo. A aplicação da técnica de *oversampling* (SMOTE) mitigou o desbalanceamento entre as classes, aumentando a capacidade do modelo em identificar padrões menos frequentes. Além disso, o uso de uma abordagem baseada em *ensemble* para a seleção de atributos permitiu, de forma controlada, a criação de múltiplos conjuntos de dados. Isso possibilitou uma análise mais aprofundada do comportamento do modelo sob diferentes configurações, resultando em melhorias tanto no desempenho quanto na interpretabilidade dos resultados.

A repetição dos experimentos ao longo dos processos de treinamentos e de testes no ambiente de AF contribuiu para a robustez dos resultados, minimizando a influência de variações pontuais e reforçando a confiabilidade dos achados. Essa abordagem experimental sistemática permitiu a aplicação de análises estatísticas com maior confiabilidade, viabilizando compa-

rações mais precisas entre os diferentes cenários avaliados e assegurando que as conclusões fossem fundamentadas em evidências sólidas e reprodutíveis.

Os resultados experimentais demonstraram que a aplicação de florestas aleatórias em um contexto federado superou a acurácia do modelo centralizado. Essa abordagem apresentou desempenho superior ao uso de modelos locais, mesmo considerando diferentes distribuições de dados entre os clientes — tanto em cenários com dados independentes e identicamente distribuídos (IID) quanto com dados não independentes e não identicamente distribuídos (non-IID). A combinação de florestas aleatórias com AF, aplicada ao conjunto IoTID20, demonstrou não apenas viabilidade técnica, mas também excelente desempenho em condições realistas. Esse diferencial evidencia o potencial da proposta frente a outras soluções da literatura, especialmente ao considerar aspectos como privacidade dos dados, escalabilidade e robustez em ambientes heterogêneos.

Adicionalmente, enfatiza-se que a otimização dos hiperparâmetros do modelo representa uma etapa promissora para alcançar resultados ainda mais expressivos. Ajustes mais precisos nesses parâmetros podem melhorar a adaptação do modelo às variações presentes nos dados distribuídos, contribuindo para maior capacidade de predição e estabilidade em cenários federados e centralizados.

Com base nas contribuições alcançadas, identificam-se diversas possibilidades para trabalhos futuros, entre as quais se destacam:

- Avaliar o método sem a redução da dimensionalidade, verificando se a preservação de todas as variáveis melhora o desempenho do modelo ou se a redução é realmente benéfica para evitar ruído e redundância nos dados;
- Considerar a utilização de outros métodos de seleção de atributos, de forma a avaliar se diferentes abordagens podem influenciar o desempenho do modelo;
- Aplicar um mecanismo de parada antecipada (*early stop*) para interromper o treinamento quando não houver evolução significativa no desempenho, economizando tempo e recursos computacionais;
- Realizar a otimização dos hiperparâmetros com busca aleatória (Bergstra and Bengio, 2012) ou busca em grade (Hsu, Chang and Lin, 2003), com a finalidade de encontrar configurações mais eficientes ao problema;
- Investigar a classificação em múltiplas classes utilizando o próprio IoTID20 ou outros conjuntos de dados relevantes ao contexto IoT, para avaliar a capacidade de generalização do modelo;
- Explorar cenários em que diferentes clientes recebem subconjuntos distintos de atributos, de forma a testar a convergência do modelo e sua capacidade de aprendizado com informações parciais.

Referências Bibliográficas

Aggarwal, C. C. (2023). *Neural Networks and Deep Learning: A Textbook*, 2 edn, Springer International Publishing.
URL: *https://doi.org/10.1007/978-3-031-29642-0* Citado na página 19.

Alkhpor, H. K. and Alserhani, F. M. (2023). Collaborative federated learning-based model for alert correlation and attack scenario recognition, *Electronics (Switzerland)* **12**.
URL: *https://doi.org/10.3390/electronics12214509* Citado 3 vezes nas páginas 73, 74 e 111.

Alpaydin, E. (2021). *Machine Learning, revised and updated edition*, The MIT Press. Citado na página 19.

Alves, I. N. L. (2021). *Detecção de intrusão em nós sensores de redes de sensores sem fio*, Master's thesis, Universidade Estadual do Oeste do Paraná.
URL: *https://tede.unioeste.br/handle/tede/5905* Citado na página 21.

Antonakakis, M., April, T., Bailey, M., Bernhard, M., Arbor, A., Bursztein, E., Cochran, J., Durumeric, Z., Halderman, J. A., Arbor, A., Invernizzi, L., Kallitsis, M., Kumar, D., Lever, C., Ma, Z., Mason, J., Menscher, D., Seaman, C., Sullivan, N., Thomas, K. and Zhou, Y. (2017). Understand the mirai botnet, *Proceedings of the 26th USENIX Security Symposium* . Citado na página 26.

Armstrong, R. A. (2014). When to use the bonferroni correction.
URL: *https://doi.org/10.1111/opo.12131* Citado na página 65.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization, *Journal of Machine Learning Research* **13**: 281–305. Citado na página 114.

Breiman, L. (1996). Bagging predictors, *Machine Learning* **24**: 123–140.
URL: *https://doi.org/10.1007/BF00058655* Citado na página 50.

Breiman, L. (2001). Random forests, *Machine Learning* **45**: 5–32.
URL: *https://doi.org/10.1023/A:1010933404324* Citado 2 vezes nas páginas 19 e 50.

Breiman, L., Friedman, J. H., Olshen, R. A. and Stone, C. J. (1984). *Classification and Regression Trees*, 1 edn, Chapman and Hall/CRC.
URL: *https://doi.org/10.1201/9781315139470* Citado na página 46.

Campos, E. M., Saura, P. F., González-Vidal, A., Hernández-Ramos, J. L., Bernabé, J. B., Baldini, G. and Skarmeta, A. (2022). Evaluating federated learning for intrusion detection in internet of things: Review and challenges, *Computer Networks* **203**.
URL: *https://doi.org/10.1016/j.comnet.2021.108661* Citado 3 vezes nas páginas 69, 74 e 111.

Chawla, N. V., Bowyer, K. W., Hall, L. O. and Kegelmeyer, W. P. (2002). Smote: Synthetic minority over-sampling technique, *Journal of Artificial Intelligence Research* **16**: 321–357.
URL: *https://doi.org/10.48550/arXiv.1106.1813* Citado na página 38.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system, *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Vol. 13-17-August-2016, Association for Computing Machinery, pp. 785–794.
URL: *https://doi.org/10.1145/2939672.2939785* Citado na página 53.

Chimphlee, S. and Chimphlee, W. (2023). Machine learning to improve the performance of anomaly-based network intrusion detection in big data, *Indonesian Journal of Electrical Engineering and Computer Science* **30**: 1106–1119.
URL: *http://doi.org/10.11591/ijeecs.v30.i2.pp1106-1119* Citado na página 79.

Cholakoska, A., Gjoreski, H., Rakovic, V., Denkovski, D., Kalendar, M., Pfitzner, B. and Arnrich, B. (2023). Federated learning for network intrusion detection in ambient assisted living environments, *IEEE Internet Computing* **27**: 15–22.
URL: *https://doi.org/10.1109/MIC.2023.3264700* Citado 3 vezes nas páginas 72, 74 e 111.

de Oliveira, E. M. (2020). *Combinação autoajustada de modelos de aprendizagem de máquina com otimização de hiperparâmetros para detecção de intrusos em redes de computadores*, Master's thesis, Universidade Estadual do Oeste do Paraná.
URL: *https://tede.unioeste.br/handle/tede/5331* Citado na página 21.

de Souza, C. A. (2018). *Método híbrido de detecção de intrusão aplicando inteligência artificial*, Master's thesis, Universidade Estadual do Oeste do Paraná.
URL: *https://tede.unioeste.br/handle/tede/3534* Citado na página 21.

Dhanya, K. A., Vajipayajula, S., Srinivasan, K., Tibrewal, A., Kumar, T. S. and Kumar, T. G. (2023). Detection of network attacks using machine learning and deep learning models, *Procedia Computer Science* **218**: 57–66.
URL: *https://doi.org/10.1016/j.procs.2022.12.401* Citado na página 22.

Ertel, W. (2017). *Undergraduate Topics in Computer Science Introduction to Artificial Intelligence*, 2 edn, Springer.
URL: *http://www.springer.com/series/7592* Citado 3 vezes nas páginas 39, 41 e 60.

Everitt, B. S. and Dunn, G. (2001). *Applied Multivariate Data Analysis*, John Wiley & Sons, Ltd.
URL: *https://doi.org/10.1002/9781118887486* Citado na página 65.

Faceli, K., Lorena, A. C., Gama, J., de Almeida, T. A. and Carvalho, A. C. P. L. F. (2021). *Inteligência Artificial - Uma Abordagem de Aprendizado de Máquina*, 2 edn, LTC. Citado 4 vezes nas páginas 36, 82, 83 e 93.

Farah, A. (2020). *Cross dataset evaluation for iot network intrusion detection*, Master's thesis, University of Wisconsin-Milwaukee.
URL: *http://digital.library.wisc.edu/1793/92438* Citado na página 41.

Farnaaz, N. and Jabbar, M. A. (2016). Random forest modeling for network intrusion detection system, *Procedia Computer Science* **89**: 213–217.
URL: *https://doi.org/10.1016/j.procs.2016.06.047* Citado 2 vezes nas páginas 22 e 50.

Fayyad, U., Piatetsky-Shapiro, G. and Smyth, P. (1996). From data mining to knowledge discovery in databases, *AI Magazine* **17**: 37–53.
URL: *https://doi.org/10.1609/aimag.v17i3.1230* Citado 2 vezes nas páginas 35 e 38.

Ferrag, M. A., Friha, O., Maglaras, L., Janicke, H. and Shu, L. (2021). Federated deep lear-

ning for cyber security in the internet of things: Concepts, applications, and experimental analysis, *IEEE Access* **9**: 138509–138542.
URL: *https://doi.org/10.1109/ACCESS.2021.3118642* Citado 2 vezes nas páginas 19 e 55.

Fisher, R. A. (1992). *Statistical Methods for Research Workers*, Springer New York, pp. 66–70.
URL: *https://doi.org/10.1007/978-1-4612-4380-9_6* Citado na página 64.

Friha, O., Ferrag, M. A., Shu, L., Maglaras, L., Choo, K. K. R. and Nafaa, M. (2022). Felids: Federated learning-based intrusion detection system for agricultural internet of things, *Journal of Parallel and Distributed Computing* **165**: 17–31.
URL: *https://doi.org/10.1016/j.jpdc.2022.03.003* Citado 3 vezes nas páginas 71, 74 e 111.

Goodfellow, I., Bengio, Y. and Courville, A. (2016). *Deep Learning*, MIT Press. Citado na página 19.

Guo, Y. (2023). A review of machine learning-based zero-day attack detection: Challenges and future directions, *Computer Communications* **198**: 175–185.
URL: *https://doi.org/10.1016/j.comcom.2022.11.001* Citado na página 25.

Gupta, B. B. and Badve, O. P. (2017). Taxonomy of dos and ddos attacks and desirable defense mechanism in a cloud computing environment, *Neural Computing and Applications* **28**: 3655–3682.
URL: *https://doi.org/10.1007/s00521-016-2317-5* Citado na página 26.

Haibo, H., Yang, B., Garcia, E. A. and Shutao, L. (2008). Adasyn: Adaptive synthetic sampling approach for imbalanced learning, *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence*, pp. 1322–1328.
URL: *https://doi.org/10.1109/IJCNN.2008.4633969* Citado na página 38.

Hastie, T., Tibshirani, R. and Friedman, J. (2017). *The Elements of Statistical Learning Data Mining, Inference and Prediction*, 2 edn, Springer.
URL: *https://doi.org/10.1007/978-0-387-84858-7* Citado 4 vezes nas páginas 38, 42, 49 e 50.

Hsu, C.-W., Chang, C.-C. and Lin, C.-J. (2003). A practical guide to support vector classification. Citado na página 114.

Hussein, A. H. (2019). Internet of things (iot): Research challenges and future applications, *IJACSA) International Journal of Advanced Computer Science and Applications* **10**: 77–82.
URL: *https://doi.org/10.14569/IJACSA.2019.0100611* Citado na página 17.

IBM (2024). Custos de uma violação de dados 2018 - 2024.
URL: *https://www.ibm.com/downloads/cas/50ALZL8W* Citado na página 18.

Iot-Analytics (2024). Número de dispositivos iot conectados.
URL: *https://iot-analytics.com/number-connected-iot-devices* Citado na página 17.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., D'Oliveira, R. G. L., Eichner, H., Rouayheb, S. E., Evans, D., Gardner, J., Garrett, Z., Gascón, A., Ghazi, B., Gibbons, P. B., Gruteser, M., Harchaoui, Z., He, C., He, L., Huo, Z., Hutchinson, B., Hsu, J., Jaggi, M., Javidi, T., Joshi, G., Khodak, M., Konečný, J., Korolova, A., Koushanfar, F., Koyejo, S., Lepoint,

T., Liu, Y., Mittal, P., Mohri, M., Nock, R., Özgür, A., Pagh, R., Raykova, M., Qi, H., Ramage, D., Raskar, R., Song, D., Song, W., Stich, S. U., Sun, Z., Suresh, A. T., Tramèr, F., Vepakomma, P., Wang, J., Xiong, L., Xu, Z., Yang, Q., Yu, F. X., Yu, H. and Zhao, S. (2021). Advances and open problems in federated learning, *Foundations and Trends in Machine Learning* **14**: 1–121.
URL: *https://doi.org/10.48550/arXiv.1912.04977* Citado 2 vezes nas páginas 19 e 57.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q. and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree, *NIPS'17: Proceedings of the 31st International Conference on Neural Information Processing Systems*, Curran Associates Inc., pp. 3149–3157.
URL: *https://dl.acm.org/doi/10.5555/3294996.3295074* Citado na página 52.

Kelleher, J. D., Namee, B. M. and D'Arcy, A. (2015). *Fundamentals of Machine Learning for Predictive Data Analytics.*, 1st edn, The MIT Press. Citado na página 62.

Khan, A. A., Chaudhari, O. and Chandra, R. (2023). A review of ensemble learning and data augmentation models for class imbalanced problems: combination, implementation and evaluation.
URL: *http://arxiv.org/abs/2304.02858* Citado na página 38.

Kruskal, W. H. and Wallis, W. A. (1952). Use of ranks in one-criterion variance analysis, *Journal of the American Statistical Association* **47**: 583–621.
URL: *https://doi.org/10.2307/2280779* Citado na página 64.

Kumar, S. and Sudarsan, S. D. (2014). An innovative udp port scanning technique, *International Journal of Future Computer and Communication* **3**: 381–384.
URL: *https://doi.org/10.7763/IJFCC.2014.V3.332* Citado na página 26.

Leevy, J. L., Hancock, J., Zuech, R. and Khoshgoftaar, T. M. (2021). Detecting cybersecurity attacks across different network features and learners, *Journal of Big Data* **8**.
URL: *https://doi.org/10.1186/s40537-021-00426-w* Citado 3 vezes nas páginas 37, 82 e 95.

Leon, M., Markovic, T. and Punnekkat, S. (2022). Comparative evaluation of machine learning algorithms for network intrusion detection and attack classification, *2022 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8.
URL: *https://doi.org/10.1109/IJCNN55064.2022.9892293* Citado na página 22.

Li, Y. and Liu, Q. (2021). A comprehensive review study of cyber-attacks and cyber security; emerging trends and recent developments, *Energy Reports* **7**: 8176–8186.
URL: *https://doi.org/10.1016/j.egyr.2021.08.126* Citado na página 25.

Liu, Y., Liang, Y., Liu, Y., Liu, Z., Meng, C., Zhang, J. and Zheng, Y. (2022). Federated forest, *IEEE Transactions on Big Data* **8**: 843–854.
URL: *https://doi.org/10.1109/TBDATA.2020.2992755* Citado 3 vezes nas páginas 67, 74 e 111.

Markovic, T., Leon, M., Buffoni, D. and Punnekkat, S. (2022). Random forest based on federated learning for intrusion detection, *Artificial Intelligence Applications and Innovations*, Springer International Publishing.
URL: *https://doi.org/10.1007/978-3-031-08333-4_11* Citado 4 vezes nas páginas 70, 74, 89 e 111.

McMahan, H. B., Moore, E., Ramage, D., Hampson, S. and y Arcas, B. A. (2016).

Communication-efficient learning of deep networks from decentralized data, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)* **54**. **URL:** *http://arxiv.org/abs/1602.05629* Citado 7 vezes nas páginas 19, 55, 56, 57, 69, 70 e 110.

Mohammed, R., Jordan, M. A. A., Rawashdeh, J. and Abdullah, M. (2020). Machine learning with oversampling and undersampling techniques: Overview study and experimental results, *11th International Conference on Information and Communication Systems (ICICS)*, IEEE, pp. 243–248. **URL:** *https://doi.org./10.1109/ICICS49469.2020.239556* Citado na página 37.

Morsy, S. M. and Nashat, D. (2022). D-arp: An efficient scheme to detect and prevent arp spoofing, *IEEE Access* **10**: 49142–49153. **URL:** *https://doi.org/10.1109/ACCESS.2022.3172329* Citado na página 26.

Mothukuri, V., Khare, P., Parizi, R. M., Pouriyeh, S., Dehghantanha, A. and Srivastava, G. (2022). Federated-learning-based anomaly detection for iot security attacks, *IEEE Internet of Things Journal* **9**: 2545–2554. **URL:** *https://doi.org/10.1109/JIOT.2021.3077803* Citado 3 vezes nas páginas 68, 74 e 111.

Nakip, M., Gül, B. C. and Gelenbe, E. (2023). Decentralized online federated g-network learning for lightweight intrusion detection. **URL:** *https://doi.org/10.48550/arXiv.2306.13029* Citado 3 vezes nas páginas 71, 74 e 111.

Neto, H. N. C., Dusparic, I., Mattos, D. M. F. and Fernandes, N. C. (2022). Fedsa: Accelerating intrusion detection in collaborative environments with federated simulated annealing, *Proceedings of the 2022 IEEE International Conference on Network Softwarization: Network Softwarization Coming of Age: New Challenges and Opportunities, NetSoft 2022* pp. 420–428. **URL:** *https://doi.org/10.1109/NetSoft54395.2022.9844024* Citado 4 vezes nas páginas 59, 70, 74 e 111.

Nápoles, G. and Grau, I. (2023). Presumably correct undersampling, *Lecture Notes in Computer Science*, Springer, pp. 420–433. **URL:** *https://doi.org/10.1007/978-3-031-49018-7_30* Citado 2 vezes nas páginas 38 e 39.

Orlandi, F. C., Anjos, J. C. S. D., Leithardt, V. R. Q., Santana, J. F. D. P. and Geyer, C. F. R. (2023). Entropy to mitigate non-iid data problem on federated learning for the edge intelligence environment, *IEEE Access* **11**: 78845–78857. **URL:** *https://doi.org/10.1109/ACCESS.2023.3298704* Citado na página 60.

Panda, R. M. and Sagar, B. S. D. (2022). *Decision Tree*, Springer, Cham, pp. 1–7. Citado na página 45.

Patel, M. B. R. and Rana, M. K. K. (2014). A survey on decision tree algorithm for classification. **URL:** *www.ijedr.org* Citado na página 48.

Pol, P. (2017). Modeling fancy bear cyber attacks designing a unified kill chain for analyzing, comparing and defending against cyber attacks. **URL:** *https://studenttheses.universiteitleiden.nl/handle/1887/64569* Citado 2 vezes nas

páginas 27 e 28.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V. and Gulin, A. (2017). Catboost: unbiased boosting with categorical features.
URL: *http://arxiv.org/abs/1706.09516* Citado na página 53.

Quinlan, J. R. (1986). Induction of decision trees, *Machine Learning* **1**: 81–106.
URL: *https://doi.org/10.1007/BF00116251* Citado 2 vezes nas páginas 37 e 47.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*, Vol. 16, Morgan Kaufmann Publishers, Inc., pp. 235–240.
URL: *https://dl.acm.org/doi/10.5555/152181* Citado na página 48.

Revathi, S. and Malathi, A. (2013). A detailed analysis on nsl-kdd dataset using various machine learning techniques for intrusion detection, *International Journal of Engineering Research and Technology (IJERT)* **2**: 1848–1853. Citado na página 22.

Rey, V., Sánchez, P. M. S., Celdrán, A. H. and Bovet, G. (2022). Federated learning for malware detection in iot devices, *Computer Networks* **204**.
URL: *https://doi.org/10.1016/j.comnet.2021.108693* Citado 3 vezes nas páginas 69, 74 e 111.

Ribera, C. D. R. (2024). Detecção de intrusão em dispositivos de internet das coisas com uma abordagem de aprendizado federado.
URL: *https://tede.unioeste.br/handle/tede/7452* Citado 6 vezes nas páginas 21, 55, 73, 74, 81 e 111.

Rokach, L. and Maimon, O. (2006). *Decision Trees*, Springer-Verlag, pp. 165–192.
URL: *https://doi.org/10.1007/0-387-25465-x_9* Citado 2 vezes nas páginas 44 e 49.

Russell, S. J. and Norvig, P. (2010). *Artificial Intelligence A Modern Approach*, 3 edn, Pearson Education, Inc. Citado 2 vezes nas páginas 18 e 19.

Sagiroglu, S. and Sinanc, D. (2013). Big data: A review, *Proceedings of the 2013 International Conference on Collaboration Technologies and Systems, CTS 2013*, pp. 42–47.
URL: *https://doi.org/10.1109/CTS.2013.6567202* Citado na página 53.

Samet, R., Aslan, O., Gupta, D. and Ozkan-Okay, M. (2021). A comprehensive systematic literature review on intrusion detection systems, *IEEE Access* **9**: 157727–157760.
URL: *https://doi.org/10.1109/ACCESS.2021.3129336* Citado 2 vezes nas páginas 32 e 34.

Saranya, T., Sridevi, S., Deisy, C., Chung, T. D. and Khan, M. (2020). Performance analysis of machine learning algorithms in intrusion detection system: A review, *Procedia Computer Science* **171**: 1251–1260.
URL: *https://doi.org/10.1016/j.procs.2020.04.133* Citado na página 40.

Scarfone, K. and Mell, P. (2007). Special publication 800-94 guide to intrusion detection and prevention systems (idps). recommendations of the national institute of standards and technology, *Technical report*, National Institute of Standards and Technology.
URL: *https://doi.org/10.6028/NIST.SP.800-94* Citado 4 vezes nas páginas 18, 30, 32 e 33.

Schiliro, F. (2023). Towards a contemporary definition of cybersecurity.
URL: *https://doi.org/10.48550/arXiv.2302.02274* Citado na página 24.

Schonlau, M. and Zou, R. Y. (2020). The random forest algorithm for statistical learning, *Stata*

Journal **20**: 3–29.
URL: *https://doi.org/10.1177/1536867X20909688* Citado na página 44.

Shah, K., Patel, H., Sanghvi, D. and Shah, M. (2020). A comparative analysis of logistic regression, random forest and knn models for the text classification, *Augmented Human Research* **5**.
URL: *https://doi.org/10.1007/s41133-020-00032-0* Citado na página 51.

Shannon, C. E. (1948). A mathematical theory of communication, *The Bell System Technical Journal* **27**: 623–656.
URL: *https://doi.org/10.1002/j.1538-7305.1948.tb01338.x* Citado na página 54.

Shapiro, S. S. and Wilk, M. B. (1965). An analysis of variance test for normality (complete samples), *Biometrika* **52**: 591–611.
URL: *https://doi.org/10.1093/biomet/52.3-4.591* Citado na página 64.

Shubhendu, S. and Vijay, J. (2013). Applicability of artificial intelligence in different fields of life, *International Journal of Scientific Engineering and Research (IJSER)* **1**: 2347–3878.
URL: *https://doi.org/10.70729/1130915* Citado na página 40.

Silva, C. L. S. (2024). *Abordagens para o problema do desbalanceamento em detecção de intrusão - um estudo de caso aplicando cic-ids2018*, Master's thesis, Universidade Estadual do Oeste do Paraná.
URL: *https://tede.unioeste.br/handle/tede/7258* Citado 3 vezes nas páginas 21, 42 e 52.

Singh, D. and Singh, B. (2020). Investigating the impact of data normalization on classification performance, *Applied Soft Computing Journal* **97**.
URL: *https://doi.org/10.1016/j.asoc.2019.105524* Citado 2 vezes nas páginas 36 e 93.

Souza, L. A. C. D., Rebello, G. A., Camilo, G. F., Guimaraes, L. C. B. and Duarte, O. C. M. B. (2020). Dfedforest: Decentralized federated forest, *Proceedings - 2020 IEEE International Conference on Blockchain, Blockchain 2020* pp. 90–97.
URL: *https://doi.org/10.1109/Blockchain50366.2020.00019* Citado 3 vezes nas páginas 67, 74 e 111.

Speiser, J. L., Miller, M. E., Tooze, J. and Ip, E. (2019). A comparison of random forest variable selection methods for classification prediction modeling, *Expert Systems with Applications* **134**: 93–101.
URL: *https://doi.org/10.1016/j.eswa.2019.05.028* Citado na página 50.

Stallings, W. (2017). *Network security essentials : applications and standards*, 6 edn, Pearson Education Limited. Citado 2 vezes nas páginas 23 e 25.

Tantithamthavorn, C., McIntosh, S., Hassan, A. E. and Matsumoto, K. (2017). An empirical comparison of model validation techniques for defect prediction models, *IEEE Transactions on Software Engineering* **43**: 1–18.
URL: *https://doi.org/10.1109/TSE.2016.2584050* Citado na página 61.

Thakkar, A. and Lohiya, R. (2022). A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions, *Artificial Intelligence Review* **55**: 453–563.
URL: *https://doi.org/10.1007/s10462-021-10037-9* Citado na página 63.

Tiwari, T., Tiwari, S. and Tiwari, T. (2018). How artificial intelligence, machine learning and deep learning are radically different?, *International Journal of Advanced Research in Com-*

puter Science and Software Engineering **8**: 1.
URL: *https://doi.org/10.23956/ijarcsse.v8i2.569* Citado na página 40.

Turhan, N. S. (2020). Karl pearsons chi-square tests, *Educational Research and Reviews* **15**: 575–580.
URL: *https://doi.org/10.5897/ERR2019.3817* Citado 2 vezes nas páginas 37 e 54.

Ullah, I. and Mahmoud, Q. H. (2020). A scheme for generating a dataset for anomalous activity detection in iot networks, *Advances in Artificial Intelligence* **12109 LNAI**: 508–520.
URL: *https://doi.org/10.1007/978-3-030-47358-7_52* Citado na página 77.

Valencio, J. D. G. (2021). *Abordagem de detecção de intrusão em ambientes fog computing e internet of things*, Master's thesis, Universidade Estadual do Oeste do Paraná.
URL: *https://tede.unioeste.br/handle/tede/5958* Citado na página 21.

Vimarlund, V., Borycki, E. M., Kushniruk, A. W. and Avenberg, K. (2021). Ambient assisted living: Identifying new challenges and needs for digital technologies and service innovation, *Yearbook of medical informatics* **30**: 141–149.
URL: *https://doi.org/10.1055/s-0041-1726492* Citado na página 72.

Wardana, A. A., Sukarno, P., Basuki, S. and Utomo, S. B. (2024). Federated random forest with feature selection for collaborative intrusion detection in internet of things, *Procedia Computer Science* **246**: 20–29.
URL: *https://doi.org/10.1016/j.procs.2024.09.193* Citado 3 vezes nas páginas 68, 74 e 111.

Wei, Y. and Shangguan, M. (2023). A review of deep learning based intrusion detection systems, *Highlights in Science, Engineering and Technology AICT* **2023**.
URL: *https://doi.org/10.54097/hset.v56i.10104* Citado na página 41.

Williams, P., Dutta, I. K., Daoud, H. and Bayoumi, M. (2022). A survey on security in internet of things with a focus on the impact of emerging technologies, *Internet of Things (Netherlands)* **19**.
URL: *https://doi.org/10.1016/j.iot.2022.100564* Citado na página 17.

Witten, I. H., Frank, E., Hall, M. A. and Pal, C. J. (2017). *Data Mining*, 4 edn, Elsevier. Citado 5 vezes nas páginas 37, 42, 43, 54 e 61.

Yang, T., Andrew, G., Eichner, H., Sun, H., Li, W., Kong, N., Ramage, D. and Beaufays, F. (2018). Applied federated learning: Improving google keyboard query suggestions.
URL: *http://arxiv.org/abs/1812.02903* Citado na página 56.

Zhu, H., Xu, J., Liu, S. and Jin, Y. (2021). Federated learning on non-iid data: A survey.
URL: *https://doi.org/10.48550/arXiv.2106.06843* Citado 3 vezes nas páginas 55, 57 e 58.

ZorarpacI, E. and Ozel, S. A. (2016). A hybrid approach of differential evolution and artificial bee colony for feature selection, *Expert Systems with Applications* **62**: 91–103. Citado na página 37.

Zuech, R., Hancock, J. and Khoshgoftaar, T. M. (2021). Detecting web attacks using random undersampling and ensemble learners, *Journal of Big Data* **8**.
URL: *https://doi.org/10.1186/s40537-021-00460-8* Citado na página 38.

Apêndice A

Exemplos originais do conjunto de dados IoTID20

Tabela A.1: Exemplos originais do conjunto de dados IoTID20.

Flow_ID	Src_IP	Src_Port	Dst_IP	Dst_Port	Protocol	Timestamp
192.168.0.13-192.168.0.16-10000-10101-17	192.168.0.13	10000	192.168.0.16	10101	17	25/07/2019 03:25:53 AM
192.168.0.13-222.160.179.132-554-2179-6	222160179132	2179	192.168.0.13	554	6	26/05/2019 10:11:06 PM
192.168.0.13-192.168.0.16-9020-52727-6	192.168.0.16	52727	192.168.0.13	9020	6	11/07/2019 01:24:48 AM
192.168.0.13-192.168.0.16-9020-52964-6	192.168.0.16	52964	192.168.0.13	9020	6	04/09/2019 03:58:17 AM
192.168.0.1-239.255.255.250-36763-1900-17	192.168.0.1	36763	239255255250	1900	17	10/09/2019 01:41:18 AM
192.168.0.24-101.79.244.148-41980-443-6	192.168.0.24	41980	101.79.244.148	443	6	10/09/2019 01:39:13 AM
192.168.0.24-210.89.164.90-60175-8899-17	192.168.0.24	60175	210.89.164.90	8899	17	25/07/2019 03:21:01 AM
192.168.0.24-58.225.75.83-41467-443-6	192.168.0.24	41467	58.225.75.83	443	6	11/07/2019 01:52:37 AM
192.168.0.13-210.89.164.90-60132-8899-17	192.168.0.13	60132	210.89.164.90	8899	17	25/07/2019 03:21:13 AM
192.168.0.13-111.149.163.151-554-7953-6	111149163151	7953	192.168.0.13	554	6	26/05/2019 10:20:36 PM
192.168.0.24-40.100.49.34-10102-443-6	192.168.0.24	10102	40.100.49.34	443	6	04/09/2019 03:58:32 AM
192.168.0.24-104.118.134.215-43238-443-6	104118134215	443	192.168.0.24	43238	6	25/07/2019 03:26:18 AM
192.168.0.13-52.219.36.20-53604-443-6	192.168.0.13	53604	52.219.36.20	443	6	25/07/2019 03:24:59 AM
192.168.0.13-192.168.0.16-10000-10101-17	192.168.0.13	10000	192.168.0.16	10101	17	25/07/2019 03:25:48 AM
192.168.0.24-210.89.164.90-60079-8899-17	192.168.0.24	60079	210.89.164.90	8899	17	25/07/2019 03:20:43 AM
192.168.0.13-192.168.0.23-9020-44144-6	192.168.0.13	9020	192.168.0.23	44144	6	26/05/2019 10:06:28 PM
192.168.0.13-192.168.0.16-56361-10101-17	192.168.0.13	56361	192.168.0.16	10101	17	25/07/2019 03:24:09 AM
192.168.0.13-192.168.0.16-9020-49784-6	192.168.0.13	9020	192.168.0.16	49784	6	20/05/2019 04:56:32 AM
192.168.0.13-192.168.0.16-56361-10101-17	192.168.0.13	56361	192.168.0.16	10101	17	25/07/2019 03:24:23 AM
192.168.0.24-210.89.164.90-60183-8899-17	192.168.0.24	60183	210.89.164.90	8899	17	25/07/2019 03:21:07 AM

Tabela A.1: Exemplos originais do conjunto de dados IoTID20 (continuação).

Flow_Duration	Tot_Fwd_Pkts	Tot_Bwd_Pkts	TotLen_Fwd_Pkts	TotLen_Bwd_Pkts	Fwd_Pkt_Len_Max	Fwd_Pkt_Len_Min
75	1	1	982.0	1430.0	982.0	982.0
5310	1	2	0.0	0.0	0.0	0.0
141	0	3	0.0	2806.0	0.0	0.0
151	0	2	0.0	2776.0	0.0	0.0
153	2	1	886.0	420.0	452.0	434.0
157	2	1	0.0	0.0	0.0	0.0
139	20	1	640.0	32.0	32.0	32.0
112	0	2	0.0	2896.0	0.0	0.0
86	1	1	32.0	32.0	32.0	32.0
6799	0	2	0.0	0.0	0.0	0.0
54	2	3	0.0	4076.0	0.0	0.0
165	1	1	1441.0	1441.0	1441.0	1441.0
199	2	1	2800.0	1400.0	1400.0	1400.0
212	3	1	4290.0	1430.0	1430.0	1430.0
40	10	1	320.0	32.0	32.0	32.0
60431	5	7	5680.0	1097.0	1388.0	1041.0
293	0	5	0.0	168.0	0.0	0.0
120	1	1	30.0	1388.0	30.0	30.0
154	0	2	0.0	72.0	0.0	0.0
267	2	1	64.0	32.0	32.0	32.0

Tabela A.1: Exemplos originais do conjunto de dados IoTID20 (continuação).

Fwd_Pkt_Len_Mean	Fwd_Pkt_Len_Std	Bwd_Pkt_Len_Max	Bwd_Pkt_Len_Min	Bwd_Pkt_Len_Mean	Bwd_Pkt_Len_Std
982.0	0.0	1430.0	1430.0	1430.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1388.0	30.0	935.3333333333336	784.0416655595117
0.0	0.0	1388.0	1388.0	1388.0	0.0
443.0	12.727922061357855	420.0	420.0	420.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0
32.0	0.0	32.0	32.0	32.0	0.0
0.0	0.0	1448.0	1448.0	1448.0	0.0
32.0	0.0	32.0	32.0	32.0	0.0
0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	1460.0	1156.0	1358.6666666666667	175.51448183364622
1441.0	0.0	1441.0	1441.0	1441.0	0.0
1400.0	0.0	1400.0	1400.0	1400.0	0.0
1430.0	0.0	1430.0	1430.0	1430.0	0.0
32.0	0.0	32.0	32.0	32.0	0.0
1136.0	144.46106741956459	1097.0	0.0	156.71428571428572	414.6270268911223
0.0	0.0	40.0	32.0	33.6	3.577708763999663
30.0	0.0	1388.0	1388.0	1388.0	0.0
0.0	0.0	40.0	32.0	36.0	5.656854249492381
32.0	0.0	32.0	32.0	32.0	0.0

Tabela A.1: Exemplos originais do conjunto de dados IoTID20 (continuação).

Flow_Byts/s	Flow_Pkts/s	Flow_IAT_Mean	Flow_IAT_Std	Flow_IAT_Max	Flow_IAT_Min
32160000.000000004	26666.66666666667	75.0	0.0	75.0	75.0
0.0	564.9717514124294	2655.0	2261.327486234579	4254.0	1056.0
19900709.219858155	21276.59574468085	70.5	0.7071067811865476	71.0	70.0
18384105.9602649	13245.033112582783	151.0	0.0	151.0	151.0
8535947.712418301	19607.843137254906	76.5	0.7071067811865476	77.0	76.0
0.0	19108.280254777063	78.5	6.363961030678928	83.0	74.0
4834532.37410072	151079.1366906475	6.9499999999999975	1.6693837501494853	10.0	4.0
25857142.85714286	17857.14285714286	112.0	0.0	112.0	112.0
744186.046511628	23255.81395348837	86.0	0.0	86.0	86.0
0.0	294.1609060155905	6799.0	0.0	6799.0	6799.0
75481481.48148148	92592.5925925926	13.5	4.795831523312719	20.0	9.0
17466666.666666668	12121.212121212122	165.0	0.0	165.0	165.0
21105527.638190955	15075.376884422109	99.5	33.23401871576773	123.0	76.0
26981132.0754717	18867.924528301886	70.66666666666667	0.5773502691896257	71.0	70.0
8800000.0	275000.0	4.0	1.563471919941143	7.0	1.0
112144.42918369712	198.57357978520957	5493.727272727273	12649.922451073833	43394.0	121.0
573378.8395904436	17064.84641638225	73.25	0.4999999999999992	74.0	73.0
11816666.666666664	16666.666666666668	120.0	0.0	120.0	120.0
467532.4675324675	12987.012987012988	154.0	0.0	154.0	154.0
359550.5617977528	11235.955056179775	133.5	185.969083452062	265.0	2.0

Tabela A.1: Exemplos originais do conjunto de dados IoTID20 (continuação).

Fwd_IAT_Tot	Fwd_IAT_Mean	Fwd_IAT_Std	Fwd_IAT_Max	Fwd_IAT_Min	Bwd_IAT_Tot	Bwd_IAT_Mean
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	5310.0	5310.0
0.0	0.0	0.0	0.0	0.0	141.0	70.5
0.0	0.0	0.0	0.0	0.0	151.0	151.0
76.0	76.0	0.0	76.0	76.0	0.0	0.0
74.0	74.0	0.0	74.0	74.0	0.0	0.0
132.0	6.947368421052631	1.7150861400630166	10.0	4.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	112.0	112.0
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	6799.0	6799.0
40.0	40.0	0.0	40.0	40.0	43.0	21.5
0.0	0.0	0.0	0.0	0.0	0.0	0.0
76.0	76.0	0.0	76.0	76.0	0.0	0.0
141.0	70.5	0.7071067811865476	71.0	70.0	0.0	0.0
37.0	4.111111111111111	1.6158932858054431	7.0	1.0	0.0	0.0
56804.0	14201.000000000002	21738.29375395717	46693.0	898.0	60431.0	10071.833333333332
0.0	0.0	0.0	0.0	0.0	293.0	73.25
0.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	154.0	154.0
265.0	265.0	0.0	265.0	265.0	0.0	0.0

Tabela A.1: Exemplos originais do conjunto de dados IoTID20 (continuação).

Bwd_IAT_Std	Bwd_IAT_Max	Bwd_IAT_Min	Fwd_PSH_Flags	Bwd_PSH_Flags	Fwd_URG_Flags	Bwd_URG_Flags
0.0	0.0	0.0	0	0	0	0
0.0	5310.0	5310.0	0	0	0	0
0.7071067811865476	71.0	70.0	0	0	0	0
0.0	151.0	151.0	0	0	0	0
0.0	0.0	0.0	0	0	0	0
0.0	0.0	0.0	0	0	0	0
0.0	0.0	0.0	0	0	0	0
0.0	112.0	112.0	0	0	0	0
0.0	0.0	0.0	0	0	0	0
0.0	6799.0	6799.0	0	0	0	0
2.1213203435596424	23.0	20.0	0	0	0	0
0.0	0.0	0.0	0	0	0	0
0.0	0.0	0.0	0	0	0	0
0.0	0.0	0.0	0	0	0	0
0.0	0.0	0.0	0	0	0	0
18541.28181563148	47389.0	371.0	0	1	0	0
0.4999999999999992	74.0	73.0	0	0	0	0
0.0	0.0	0.0	0	0	0	0
0.0	154.0	154.0	0	0	0	0
0.0	0.0	0.0	0	0	0	0

Tabela A.1: Exemplos originais do conjunto de dados IoTID20 (continuação).

Fwd_Header_Len	Bwd_Header_Len	Fwd_Pkts/s	Bwd_Pkts/s	Pkt_Len_Min	Pkt_Len_Max
8	8	13333.333333333336	13333.333333333336	982.0	1430.0
20	44	188.32391713747646	376.64783427495286	0.0	0.0
0	96	0.0	21276.59574468085	30.0	1388.0
0	64	0.0	13245.033112582783	1388.0	1388.0
16	8	13071.895424836599	6535.9477124182995	420.0	452.0
64	32	12738.853503184711	6369.426751592358	0.0	0.0
160	8	143884.89208633095	7194.2446043165455	32.0	32.0
0	64	0.0	17857.14285714286	1448.0	1448.0
8	8	11627.906976744185	11627.906976744185	32.0	32.0
0	44	0.0	294.1609060155905	0.0	0.0
40	60	37037.03703703704	55555.555555555555	0.0	1460.0
32	32	6060.606060606061	6060.606060606061	1441.0	1441.0
40	20	10050.251256281406	5025.125628140703	1400.0	1400.0
24	8	14150.943396226416	4716.981132075472	1430.0	1430.0
80	8	250000.0	25000.0	32.0	32.0
160	224	82.73899157717067	115.83458820803892	0.0	1388.0
0	40	0.0	17064.84641638225	32.0	40.0
32	32	8333.333333333334	8333.333333333334	30.0	1388.0
0	16	0.0	12987.012987012988	32.0	40.0
16	8	7490.636704119851	3745.3183520599255	32.0	32.0

Tabela A.1: Exemplos originais do conjunto de dados IoTID20 (continuação).

Pkt_Len_Mean	Pkt_Len_Std	Pkt_Len_Var	FIN_Flag_Cnt	SYN_Flag_Cnt	RST_Flag_Cnt	PSH_Flag_Cnt
1280.6666666666667	258.6529205969524	66901.33333333334	0	0	0	0
0.0	0.0	0.0	0	1	0	0
1048.5	679.0	461041.0	0	0	0	0
1388.0	0.0	0.0	0	0	0	0
431.5	15.176736583776279	230.33333333333331	0	0	0	0
0.0	0.0	0.0	0	0	0	0
32.0	0.0	0.0	0	0	0	0
1448.0	0.0	0.0	0	0	0	0
32.0	0.0	0.0	0	0	0	0
0.0	0.0	0.0	0	1	0	0
922.6666666666667	724.327734293439	524650.6666666667	0	0	0	0
1441.0	0.0	0.0	0	0	0	0
1400.0	0.0	0.0	0	0	0	0
1430.0	0.0	0.0	0	0	0	0
32.0	0.0	0.0	0	0	0	0
605.6923076923076	589.742370957492	347796.0641025641	0	0	0	1
34.66666666666667	4.1311822359545785	17.06666666666667	0	0	0	0
935.3333333333333	784.0416655595117	614721.3333333335	0	0	0	0
37.333333333333336	4.618802153517006	21.33333333333333	0	0	0	0
32.0	0.0	0.0	0	0	0	0

Tabela A.1: Exemplos originais do conjunto de dados IoTID20 (continuação).

ACK_Flag_Cnt	URG_Flag_Cnt	CWE_Flag_Count	ECE_Flag_Cnt	Down/Up_Ratio	Pkt_Size_Avg	Fwd_Seg_Size_Avg
0	0	0	0	1.0	1921.0	982.0
0	0	0	0	2.0	0.0	0.0
1	0	0	0	0.0	1398.0	0.0
1	0	0	0	0.0	2082.0	0.0
0	0	0	0	0.0	575.3333333333334	443.0
1	0	0	0	0.0	0.0	0.0
0	0	0	0	0.0	33.52380952380953	32.0
1	0	0	0	0.0	2172.0	0.0
0	0	0	0	1.0	48.0	32.0
0	0	0	0	0.0	0.0	0.0
1	0	0	0	1.0	1107.2	0.0
1	0	0	0	1.0	2161.5	1441.0
1	0	0	0	0.0	1866.666666666667	1400.0
0	0	0	0	0.0	1787.5	1430.0
0	0	0	0	0.0	34.90909090909091	32.0
1	0	0	0	1.0	656.1666666666666	1136.0
0	0	0	0	0.0	41.6	0.0
1	0	0	0	1.0	1403.0	30.0
0	0	0	0	0.0	56.0	0.0
0	0	0	0	0.0	42.66666666666666	32.0

Tabela A.1: Exemplos originais do conjunto de dados IoTID20 (continuação).

Bwd_Seg_Size_Avg	Fwd_Byts/b_Avg	Fwd_Pkts/b_Avg	Fwd_Blk_Rate_Avg	Bwd_Byts/b_Avg	Bwd_Pkts/b_Avg
1430.0	0	0	0	0	0
0.0	0	0	0	0	0
935.3333333333336	0	0	0	0	0
1388.0	0	0	0	0	0
420.0	0	0	0	0	0
0.0	0	0	0	0	0
32.0	0	0	0	0	0
1448.0	0	0	0	0	0
32.0	0	0	0	0	0
0.0	0	0	0	0	0
1358.6666666666667	0	0	0	0	0
1441.0	0	0	0	0	0
1400.0	0	0	0	0	0
1430.0	0	0	0	0	0
32.0	0	0	0	0	0
156.71428571428572	0	0	0	0	0
33.6	0	0	0	0	0
1388.0	0	0	0	0	0
36.0	0	0	0	0	0
32.0	0	0	0	0	0

Tabela A.1: Exemplos originais do conjunto de dados IoTID20 (continuação).

Bwd_Blk_Rate_Avg	Subflow_Fwd_Pkts	Subflow_Fwd_Byts	Subflow_Bwd_Pkts	Subflow_Bwd_Byts	Init_Fwd_Win_Byts
0	1	982	1	1430	-1
0	1	0	2	0	-1
0	0	0	3	2806	-1
0	0	0	2	2776	-1
0	2	886	1	420	-1
0	2	0	1	0	-1
0	20	640	1	32	-1
0	0	0	2	2896	-1
0	1	32	1	32	-1
0	0	0	2	0	-1
0	2	0	3	4076	-1
0	1	1441	1	1441	-1
0	2	2800	1	1400	-1
0	3	4290	1	1430	-1
0	10	320	1	32	-1
0	5	5680	7	1097	-1
0	0	0	5	168	-1
0	1	30	1	1388	-1
0	0	0	2	72	-1
0	2	64	1	32	-1

Tabela A.1: Exemplos originais do conjunto de dados IoTID20 (continuação).

Init_Bwd_Win_Byts	Fwd_Act_Data_Pkts	Fwd_Seg_Size_Min	Active_Mean	Active_Std	Active_Max	Active_Min
-1	1	0	0.0	0.0	0.0	0.0
14600	0	0	0.0	0.0	0.0	0.0
1869	0	0	0.0	0.0	0.0	0.0
1869	0	0	0.0	0.0	0.0	0.0
-1	2	0	0.0	0.0	0.0	0.0
5840	0	0	0.0	0.0	0.0	0.0
-1	20	0	0.0	0.0	0.0	0.0
155	0	0	0.0	0.0	0.0	0.0
-1	1	0	0.0	0.0	0.0	0.0
14600	0	0	0.0	0.0	0.0	0.0
2048	0	0	0.0	0.0	0.0	0.0
252	1	0	0.0	0.0	0.0	0.0
2800	2	0	0.0	0.0	0.0	0.0
-1	3	0	0.0	0.0	0.0	0.0
-1	10	0	1.0	0.0	1.0	1.0
32678	5	0	443.0	326.401593133367	898.0	121.0
-1	0	0	0.0	0.0	0.0	0.0
1869	1	0	0.0	0.0	0.0	0.0
-1	0	0	0.0	0.0	0.0	0.0
-1	2	0	0.0	0.0	0.0	0.0

Tabela A.1: Exemplos originais do conjunto de dados IoTID20 (continuação).

Idle_Mean	Idle_Std	Idle_Max	Idle_Min	Label	Cat	Sub_Cat
75.0	0.0	75.0	75.0	Anomaly	Mirai	Mirai-Ackflooding
2655.0	2261.327486234579	4254.0	1056.0	Anomaly	DoS	DoS-Synflooding
70.5	0.7071067811865476	71.0	70.0	Anomaly	Scan	Scan Port OS
151.0	0.0	151.0	151.0	Anomaly	Mirai	Mirai-Hostbruteforceg
76.5	0.7071067811865476	77.0	76.0	Anomaly	Mirai	Mirai-Hostbruteforceg
78.5	6.363961030678928	83.0	74.0	Anomaly	Mirai	Mirai-Hostbruteforceg
6.9499999999999975	1.6693837501494853	10.0	4.0	Anomaly	Mirai	Mirai-UDP Flooding
112.0	0.0	112.0	112.0	Anomaly	Scan	Scan Port OS
86.0	0.0	86.0	86.0	Anomaly	Mirai	Mirai-UDP Flooding
6799.0	0.0	6799.0	6799.0	Anomaly	DoS	DoS-Synflooding
13.5	4.795831523312719	20.0	9.0	Anomaly	Mirai	Mirai-Hostbruteforceg
165.0	0.0	165.0	165.0	Anomaly	Mirai	Mirai-UDP Flooding
99.5	33.23401871576773	123.0	76.0	Anomaly	Mirai	Mirai-UDP Flooding
70.66666666666667	0.5773502691896257	71.0	70.0	Anomaly	Mirai	Mirai-HTTP Flooding
4.333333333333332	1.224744871391589	7.0	3.0	Anomaly	Mirai	Mirai-UDP Flooding
9708.333333333334	16525.22015183661	43394.0	1394.0	Anomaly	DoS	DoS-Synflooding
73.25	0.4999999999999992	74.0	73.0	Anomaly	Mirai	Mirai-HTTP Flooding
120.0	0.0	120.0	120.0	Normal	Normal	Normal
154.0	0.0	154.0	154.0	Anomaly	Mirai	Mirai-Ackflooding
133.5	185.969083452062	265.0	2.0	Anomaly	Mirai	Mirai-UDP Flooding

Tabela A.2: Atributos selecionados por cada algoritmo do *ensemble*.

#	Information Gain	Gain Ratio	Chi-Squared
1	Flow_Duration	RST_Flag_Cnt	Fwd_Seg_Size_Avg
2	Flow_Pkts/s	ACK_Flag_Cnt	Fwd_Pkt_Len_Mean
3	Flow_IAT_Mean	Protocol	Fwd_Pkt_Len_Std
4	Idle_Mean	SYN_Flag_Cnt	ACK_Flag_Cnt
5	Flow_IAT_Max	Bwd_Header_Len	Pkt_Len_Var
6	Idle_Max	Fwd_Pkt_Len_Max	Subflow_Bwd_Pkts
7	Bwd_Pkts/s	Active_Min	Tot_Bwd_Pkts
8	Idle_Min	Fwd_Pkt_Len_Min	TotLen_Fwd_Pkts
9	Flow_IAT_Min	Bwd_Pkt_Len_Min	Subflow_Fwd_Byts
10	Fwd_Pkts/s	Pkt_Len_Max	Pkt_Len_Mean
11	Pkt_Size_Avg	Bwd_Pkt_Len_Max	Pkt_Size_Avg
12	TotLen_Bwd_Pkts	Fwd_Pkt_Len_Std	Bwd_Seg_Size_Avg
13	Subflow_Bwd_Byts	Pkt_Len_Min	Bwd_Pkt_Len_Mean
14	Pkt_Len_Mean	Fwd_Pkt_Len_Mean	Fwd_Pkt_Len_Max
15	Bwd_Pkt_Len_Mean	Fwd_Seg_Size_Avg	SYN_Flag_Cnt
16	Bwd_Seg_Size_Avg	Bwd_Pkt_Len_Mean	Pkt_Len_Max
17	Bwd_Header_Len	Bwd_Seg_Size_Avg	Bwd_Header_Len
18	Bwd_Pkt_Len_Min	Active_Max	Bwd_IAT_Tot
19	Pkt_Len_Min	TotLen_Bwd_Pkts	Bwd_IAT_Max
20	Pkt_Len_Max	Subflow_Bwd_Byts	Flow_IAT_Mean
21	Bwd_Pkt_Len_Max	Active_Mean	Flow_Duration
22	Subflow_Fwd_Byts	Subflow_Fwd_Byts	Protocol
23	TotLen_Fwd_Pkts	TotLen_Fwd_Pkts	Bwd_IAT_Mean
24	Fwd_Pkt_Len_Min	Fwd_Header_Len	Flow_IAT_Max
25	Fwd_Pkt_Len_Max	Pkt_Len_Mean	Idle_Max
26	Fwd_Pkt_Len_Mean	Active_Std	Bwd_Pkt_Len_Max
27	Fwd_Seg_Size_Avg	Pkt_Size_Avg	Bwd_IAT_Min
28	ACK_Flag_Cnt	Pkt_Len_Var	Flow_IAT_Min
29	Init_Bwd_Win_Byts	Pkt_Len_Std	Bwd_Pkt_Len_Min
30	Fwd_IAT_Mean	Flow_Duration	Pkt_Len_Std
31	Flow_Byts/s	Idle_Max	Idle_Mean
32	Fwd_IAT_Tot	Flow_IAT_Max	Idle_Min

Tabela A.2: Atributos selecionados por cada algoritmo do *ensemble* (continuação).

#	Florestas aleatórias	CatBoost	XGB	LightGBM
1	Bwd_Header_Len	ACK_Flag_Cnt	ACK_Flag_Cnt	Init_Bwd_Win_Byts
2	ACK_Flag_Cnt	Init_Bwd_Win_Byts	Flow_Duration	Flow_Duration
3	Bwd_IAT_Tot	Flow_Duration	Fwd_Pkt_Len_Mean	Flow_IAT_Min
4	Flow_Pkts/s	Pkt_Size_Avg	Fwd_Pkt_Len_Min	Bwd_Header_Len
5	Idle_Max	Flow_IAT_Min	Bwd_Header_Len	Flow_Pkts/s
6	Flow_IAT_Min	Bwd_Header_Len	Init_Bwd_Win_Byts	Bwd_Pkts/s
7	Bwd_Pkts/s	Subflow_Fwd_Pkts	Bwd_Pkts/s	Flow_Byts/s
8	Idle_Min	Bwd_Pkt_Len_Mean	Pkt_Len_Max	Pkt_Len_Max
9	Init_Bwd_Win_Byts	Fwd_Pkt_Len_Mean	TotLen_Bwd_Pkts	TotLen_Bwd_Pkts
10	Flow_Duration	Flow_IAT_Mean	Pkt_Size_Avg	Bwd_Pkt_Len_Min
11	Flow_IAT_Max	Fwd_Header_Len	Fwd_Act_Data_Pkts	ACK_Flag_Cnt
12	Flow_IAT_Std	Pkt_Len_Var	Pkt_Len_Std	Pkt_Size_Avg
13	Protocol	Tot_Bwd_Pkts	Idle_Min	Flow_IAT_Mean
14	Fwd_IAT_Max	Flow_Pkts/s	Bwd_Pkt_Len_Max	Pkt_Len_Var
15	Bwd_IAT_Min	Bwd_IAT_Min	Bwd_IAT_Mean	Idle_Mean
16	Pkt_Size_Avg	Pkt_Len_Max	Flow_IAT_Min	Fwd_Pkt_Len_Std
17	Idle_Std	Bwd_Pkts/s	Idle_Max	Flow_IAT_Max
18	Tot_Bwd_Pkts	Fwd_Pkts/s	Bwd_IAT_Min	Fwd_Header_Len
19	Bwd_Seg_Size_Avg	Pkt_Len_Min	Bwd_Pkt_Len_Min	Pkt_Len_Mean
20	Subflow_Bwd_Byts	Idle_Std	TotLen_Fwd_Pkts	Flow_IAT_Std
21	Fwd_Header_Len	Idle_Max	Flow_IAT_Mean	Idle_Max
22	Fwd_Pkt_Len_Std	Idle_Min	Fwd_Pkts/s	Fwd_Pkts/s
23	Bwd_Pkt_Len_Mean	Bwd_Seg_Size_Avg	Bwd_IAT_Tot	Idle_Min
24	Bwd_Pkt_Len_Min	Bwd_Pkt_Len_Max	Flow_Pkts/s	Protocol
25	Fwd_Pkt_Len_Max	TotLen_Bwd_Pkts	Idle_Mean	Bwd_Pkt_Len_Std
26	Fwd_Act_Data_Pkts	Flow_IAT_Max	Protocol	Bwd_Pkt_Len_Mean
27	Fwd_IAT_Std	Bwd_IAT_Std	Fwd_Pkt_Len_Std	Pkt_Len_Min
28	Pkt_Len_Var	Subflow_Bwd_Pkts	Pkt_Len_Var	TotLen_Fwd_Pkts
29	Bwd_Pkt_Len_Max	Flow_IAT_Std	Pkt_Len_Min	Tot_Bwd_Pkts
30	Fwd_Pkts/s	Subflow_Bwd_Byts	Fwd_IAT_Min	Pkt_Len_Std
31	Pkt_Len_Min	Subflow_Fwd_Byts	Tot_Fwd_Pkts	Fwd_Act_Data_Pkts
32	Tot_Fwd_Pkts	Active_Std	Fwd_Pkt_Len_Max	Bwd_IAT_Tot