DISSERTAÇÃO DE MESTRADO PROFISSIONAL

F-NIDS: Sistema de Detecção de Intrusão baseado em Aprendizado Federado

Jonathas Alves de Oliveira

Programa de Pós-Graduação Profissional em Engenharia Elétrica
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

F-NIDS — SISTEMA DE DETECÇÃO DE INTRUSÃO BASEADO EM APRENDIZADO FEDERADO

JONATHAS ALVES DE OLIVEIRA

ORIENTADOR: GERALDO PEREIRA ROCHA FILHO, Ph.D

DISSERTAÇÃO DE MESTRADO PROFISSIONAL EM ENGENHARIA ELÉTRICA

PUBLICAÇÃO: PPEE.MP.020
BRASÍLIA/DF, JANEIRO–2024

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DISSERTAÇÃO DE MESTRADO PROFISSIONAL

F-NIDS: Sistema de Detecção de Intrusão baseado em Aprendizado Federado

Jonathas Alves de Oliveira

Dissertação de Mestrado Profissional submetida ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica

Banca Examinadora

Prof. Dr. Geraldo P. Rocha Filho _____
Orientador

Prof Dr. João Gondim _____
Examinador Interno

Prof. Dr. Miguel Elias M. Campista _____
Examinador Externo

Prof. Dr. Vinícius P. Gonçalves _____
Examinador Suplente

FICHA CATALOGRÁFICA

OLIVEIRA, JONATHAS ALVES
F-NIDS: Sistema de Detecção de Intrusãobaseado em Aprendizado Federado [Distrito Federal] 2024.
xvi, 89 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2024).
Dissertação de Mestrado Profissional - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica

1. Sistemas de Detecção de Intrusão	2. Aprendizado Federado
3. Privacidade Diferencial	4. Cloud Computing
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

OLIVEIRA, J. A. (2024). *F-NIDS: Sistema de Detecção de Intrusãobaseado em Aprendizado Federado*. Dissertação de Mestrado Profissional, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 89 p.

Jonathas Alves de Oliveira
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasília (UnB)
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

DEDICATÓRIA

Dedico esse trabalho a todos que acreditam na ciência e no conhecimento como meio definitivo de se obter o bem-estar e o progresso da sociedade. Dedico, em especial, a todos as nações e governos aos quais fazem investimentos visando o progresso cientifico, de modo a alcançar o bem comum da humanidade.

AGRADECIMENTOS

Toda conquista merece uma profunda reflexão acerca daqueles que apoiaram seu alcance. Primeiramente, agradeço aos meus pais por terem me ensinado o quão valioso é o conhecimento. Agradeço ao meu orientador, professor Geraldo Pereira Rocha Filho, e ao meu coorientador, professor Vinícius Pereira Gonçalves, que sempre me apoiaram a manter o foco e a suscitar a capacidade de pesquisa. Agradeço também à minha família, que me apoiou a enfrentar esse desafio. Não poderia de deixar de agradecer ainda aos meus colegas de trabalho e ao Instituto Eldorado, que certamente contribuíram para a minha dedicação a esta pesquisa. Por fim, agradeço a Deus por despertar em mim uma imensa curiosidade e inquietude na busca pelo conhecimento ao país Brasil por apoiar a ciência e investir na construção da ciencia e do conhescimento.

RESUMO

O aumento das redes de IoT apresentou novos desafios em termos de escalabilidade e segurança para os sistemas de detecção de intrusão de rede (NIDS) distribuídos devido a preocupações com a privacidade. Embora tenha havido algum progresso na abordagem desses desafios, ainda há perguntas não respondidas sobre como alcançar um equilíbrio entre desempenho e robustez para garantir a privacidade de forma distribuída. Além disso, é necessário desenvolver uma arquitetura confiável e dimensionável para NIDS distribuídos que possam ser implantados com eficiência em vários cenários de IoT. Essas questões sobre robustez se basearam principalmente na escolha de técnicas de aprendizado de máquina distribuídas e com proteção da privacidade. Neste trabalho, propomos o F-NIDS, um detector de intrusão que utiliza inteligência artificial federada e técnicas de comunicação assíncrona entre as entidades do sistema para proporcionar escalabilidade horizontal, juntamente com técnicas de privacidade diferencial para tratar de questões de confidencialidade de dados. A arquitetura do F-NIDS foi projetada para ser adaptável ao uso em redes de IoT, adequada para ser usada em ambientes baseados em nuvem ou neblina. Os resultados de nossos experimentos mostraram que o modelo de detecção confidencial empregado no F-NIDS – considerando as métricas de acurácia multi-classe, acurácia binária, precisão e *recall* – conseguiu prever e determinar a natureza dos ataques quando eles ocorrem. Para determinar os parâmetros ideais que atingem um equilíbrio entre a privacidade dos dados e o desempenho da classificação, foram empregadas três estratégias, cada uma avaliada pelo desempenho de robustez correspondente. Em primeiro lugar, os modelos foram treinados com valores variáveis de ruído gaussiano e submetidos a ataques de inferência de membros baseados em regras de caixa preta. Em segundo lugar, foram realizados ataques regulares de inferência de membros, utilizando diferentes amostras com tamanhos variados para determinar a quantidade máxima de dados que poderiam ser armazenados com segurança nos agentes de detecção, para tarefas de treinamento. Por fim, a robustez dos modelos treinados foi avaliada contra ataque de inversão de modelo, e os resultados foram comparados por meio de comparações gráficas. Com base nessas avaliações, o valor do nível de ruído gaussiano para o treino dos modelos locais foi determinado em 21, com tamanhos de amostra para cada agente de treino variando entre 10K a 25K.

ABSTRACT

The rise of IoT networks has presented fresh challenges in terms of scalability and security for distributed Network Intrusion Detection Systems (NIDS) due to privacy concerns. While some progress has been made in addressing these challenges, there are still unanswered questions regarding how to achieve a balance between performance and robustness to ensure privacy in a distributed manner. Additionally, there is a need to develop a reliable and scalable architecture for distributed NIDS that can be effectively deployed in various IoT scenarios. These questions about robustness relied mainly on choosing privacy-

secured and distributed Machine Learning techniques. In this work, we propose the F-NIDS, an intrusion detector that utilizes federated artificial intelligence and asynchronous communication techniques between system entities to provide horizontal scalability, along with differential privacy techniques to address data confidentiality concerns. The architecture of F-NIDS is designed to be adaptable for usage in IoT networks, suited to be used in cloud or fog-based environments. Results from our experiments have shown that the confidential detection model employed in F-NIDS – considering multi-class accuracy, binary accuracy, precision, and recall metrics – was capable of predicting and determining the nature of attacks when they occur. In order to determine optimal parameters that strike a balance between data privacy and classification performance, three strategies were employed, each evaluated for its corresponding robustness performance. Firstly, models were trained with varying Gaussian noise values, and subjected to membership inference black box rule-based attacks. Secondly, regular membership inference black box attacks were performed, utilizing different stolen samples with varying sizes to determine the maximum amount of data that could be securely stored on the detection agents for training tasks. Lastly, the robustness of the trained models was evaluated against a model inversion attack, and the results were compared through graphical comparisons. Based on these evaluations, Gaussian noise level and sample size values of 21 were obtained for each detection agent in the system, with sample sizes ranging from 10K to 25K.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Siglas

ML	*Machine Learning (Aprendizado de Máquina)*
DNN	*Deep Neural Network* (Rede Neural Profunda)
CNN	*Convolutional Neural Networks* (Redes Neurais Convolucionais)
FN	*False Negative* (Falso Negativo)
FP	*False Positive* (Falso Positivo)
GPU	*Graphics Processing Unit* (Unidade de Processamento Gráfico)
MLP	*Multilayer Perceptron* (Perceptron Multicamadas)
RNA	Rede Neural Artificial
RNN	*Recurrent Neural Network* (Rede Neural Recorrente)
PCA	*Principal Component Analysis (Análise de Componentes Principais)*
SVM	*Support Vector Machine*
TN	*True Negative* (Verdadeiro Negativo)
TP	*True Positive* (Verdadeiro Positivo)

1 INTRODUÇÃO

Na última década, observamos um aumento considerável na interconexão entre humanos, máquinas e serviços. Isso resultou no paradigma de comunicação da IoT (*Internet of Things* — Internet das Coisas) [13, 14]. Preocupações, como escalabilidade, latência e privacidade das informações, foram levantadas no contexto da IoT [15, 16]. As arquiteturas descentralizadas podem trazer algumas soluções para essas questões, oferecendo maior disponibilidade e escalabilidade superior [17, 18]. Na frente de segurança, as estratégias mais populares usadas em redes de IoT compreendem o uso de NIDS (*Network Intrusion Detection Systems* — Sistemas de Detecção de Intrusão em Rede). Esses sistemas são responsáveis por determinar e alertar se uma determinada atividade é normal ou mal-intencionada na rede [19]. Por outro lado, as técnicas convencionais de NIDS podem ser menos eficazes no contexto da IoT, devido ao seu dinamismo [20].

Um NIDS é criado para fornecer monitoramento e detecção contínuos durante o ciclo de vida das redes de computadores [21]. No entanto, devido à natureza dinâmica de um ambiente de IoT, cujos recursos são frequentemente limitados e podem conter dispositivos heterogêneos e com alto gral de interconectividade, nesse cenário as técnicas mais comuns de NIDS podem ser menos eficazes para sistemas de detecção de intrusão [20]. Portanto, o NIDS funciona em circunstâncias mais desafiadoras e restritivas quando usado nesse ambiente. Devido aos excelentes resultados, os pesquisadores adotaram a abordagem de ML (*Machine Learning — Aprendizado de Máquina*) para o desenvolvimento de NIDS a fim de melhorar a detecção de ataques cibernéticos [19]. Por outro lado, os autores discutiram em [22] que a questão da privacidade nos modelos de ML ainda é um desafio, especialmente em arquiteturas descentralizadas. Esse ainda é um campo promissor para o desenvolvimento de alternativas que aumentem a segurança e a confidencialidade dos dados e modelos de treinamento e, ao mesmo tempo, mantenham a escalabilidade e a resiliência que as arquiteturas descentralizadas podem oferecer.

Embora haja uma abundância de trabalhos sobre o tema da privacidade no contexto do ML, ainda há poucas aplicações em um cenário real de uso [23, 24]. No aspecto da descentralização, o FL (*Federated Learning* — Aprendizado Federado) é um paradigma proposto recentemente para permitir a distribuição de tarefas de ML com maior preservação da privacidade dos dados de treinamento, e essa técnica demonstrou uma ampla gama de aplicações, especialmente quando a confidencialidade é um aspecto importante [25, 26]. Apesar de o FL ser usado para distribuir tarefas de treinamento de ML, outra questão importante é a distribuição dos serviços de detecção entre vários agentes na rede. A abordagem de comunicação assíncrona oferece tal habilidade aos sistemas de segurança em atender com maior escalabilidade a demanda por requisições de detecção de intrusão. A abordagem de comunicação assíncrona baseia-se em técnicas de enfileiramento de mensagens, retornos de chamada, arquitetura orientada por eventos e modelo de *publish/subscribe* (publicação/assinatura).

Trabalhos como [27], apresentam uma visão geral relevante dos padrões e arquiteturas de mensagens, incluindo comunicação assíncrona, enfileiramento de mensagens e sistemas orientados por eventos. Além disso, [27] oferece uma coleção de padrões para projetar sistemas de mensagens assíncronas, abrangendo conceitos como filas de mensagens, padrões para o mecanismo de *publish/subscribe* e arquiteturas orienta-

das a eventos. Os benefícios da comunicação assíncrona em sistemas distribuídos são discutidos em [28], destacando as vantagens de escalabilidade e resistência a falhas proporcionadas por arquiteturas com tais características. Entretanto, essa abordagem tem algumas limitações e desafios, tais como a complexidade da implementação e no tratamento e correção de erros. Os tempos de latência também podem ser maiores que arquiteturas de comunicação síncronas [29].

1.1 JUSTIFICATIVA

A implementação do NIDS desempenha um papel fundamental na segurança cibernética, especialmente diante do cenário em constante evolução das ameaças digitais. A importância desses sistemas de detecção de ameaças, fornecem uma camada crítica de segurança para prevenir a exploração de vulnerabilidades em sistemas de informação [30]. No entanto, como destacado em [31], existe a necessidade contínua de inovações nesse campo. A implementação de IDS torna-se imperativa para garantir a integridade, confidencialidade e disponibilidade dos dados em ambientes digitais cada vez mais interconectados e suscetíveis a ameaças sofisticadas [30, 32, 33].

A preservação da privacidade surge como aspecto crítico na implementação NIDS, dado o caráter sensível das informações trafegadas em ambientes digitais [34]. A coleta e análise de dados para detecção de intrusões podem inadvertidamente violar a privacidade dos usuários, impactando negativamente a confiança e a aceitação desses sistemas [35]. Portanto, a integração de mecanismos que garantam a anonimização e a proteção das informações pessoais é crucial para mitigar potenciais riscos à privacidade [36]. Em [37] destacam a necessidade de considerações éticas e jurídicas na implementação de NIDS, descartando a importância de abordagens que conciliem eficácia na detecção de ameaças com a preservação dos direitos individuais. Em um cenário onde as questões de privacidade são cada vez mais urgentes, a integração de mecanismos de proteção de dados sensíveis torna-se indispensável para promover a aceitação e a eficácia contínua dos NIDS na detecção de intrusões.

O desafio de equilibrar a preservação da privacidade com a eficiência de classificação em modelos de ML para os NIDS é amplamente reconhecido na literatura acadêmica. Autores como [38] destacam a complexidade inerente à proteção da privacidade em ambientes de análise de dados, especialmente quando se visa manter a confidencialidade dos indivíduos durante a identificação de padrões de intrusão. Essa preocupação é ressaltada por trabalhos em [39] que discutem as dificuldades práticas de alcançar a privacidade dos dados de treino de um modelo de aprendizagem de máquina. Já em [40] a importância da eficiência e do desempenho dos modelos de NIDS para enfrentar ameaças em tempo real é destacada. Desse modo, o desafio consiste em encontrar estratégias inovadoras que conciliem a robustez da detecção de intrusões com a preservação rigorosa da privacidade, considerando a natureza sensível dos dados envolvidos. Esse equilíbrio delicado destaca a necessidade contínua de pesquisa e desenvolvimento para superar os obstáculos técnicos associados à implementação bem-sucedida de modelos de classificação confidenciais para um NIDS.

Tento em vista a quantidade de dados coletados para treinar um modelo de ML, surgem preocupações sobre a segurança, extração não autorizada de dados e o uso indevido dessas informações. As ameaças à

privacidade de modelos de ML geralmente vem na forma de alguns tipos de ataques que visam obter dados de treino dos modelos. Entre esses ataques, destaca-se a inversão de modelo, a inferência de membros em caixa preta e as baseadas em regras. No caso da inversão de modelo, os invasores buscam reconstruir dados de treinamento ou informações confidenciais a partir das saídas do modelo original, comprometendo a privacidade dos dados originais [41]. Já a inferência de membros em caixa preta envolve a dedução de informações sobre os dados de entrada com base nas respostas do modelo, mesmo sem acesso direto aos dados de treinamento. Por fim, os ataques baseados em regras exploram padrões de comportamento do modelo para inferir informações sensíveis sobre os dados de entrada, contornando medidas de segurança [42]. Esses ataques representam desafios significativos para a proteção da privacidade dos dados e requerem estratégias eficazes de defesa para mitigar seus impactos negativos. Estabelecer técnicas de ML colaborativa que possam lidar com essas ameaças, sem afetar o desempenho, constitui um importante desafio [43].

1.2 OBJETIVO

O objetivo deste trabalho é propor o F-NIDS, um sistema de detecção de intrusão distribuído para superar as limitações de privacidade mencionadas em tais cenários distribuídos, balanceando o desempenho e a robustez da classificação em termos de métricas de acurácia, precisão e *recall*. Para fornecer serviços de detecção de intrusão distribuída, com escalabilidade horizontal em cenários de IoT, uma arquitetura distribuída de divisão das tarefas, entre os agentes de detecção, é proposta. Já a arquitetura de treinamento distribuída irá focar na comunicação confidencial dos dados, sem a necessidade de trocar dados de treino, ao dispensar a transmissão de informações individuais dos clientes.

Outro objetivo importante é obter modelos locais seguros para serem armazenados nos clientes e que sejam robustos em relação aos ataques de reconstrução e inferência de membros. Para tal, usa-se a técnicas de DP, permitindo uma transmissão, dos pesos sinápticos dos modelos entre os clientes, de maneira confidencial.

Em resumo, o F-NIDS visa prover os seguintes recursos principais: (i) apresentar uma alta acurácia na classificação do tráfego de dados em um cenário de IoT distribuído; (ii) possuir uma arquitetura descentralizada capaz de permitir resistir as variações de demanda por meio de um redimensionamento dinâmico do sistema; e (iii) garantir maior confidencialidade dos dados de treinamento e dos modelos treinados. Para confirmar que esses objetivos foram atingidos, o F-NIDS foi avaliado e comparado com três abordagens diferentes utilizando métricas de acurácia, precisão e *recall*.

1.3 CONTRIBUIÇÕES

Os resultados obtidos durante a elaboração do F-NIDS produziu duas publicações relevantes. A primeira consta nos Anais do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos — SBRC 2023 [44]. A segunda publicação consta no Volume 236 da revista *Computer Networks* [45]:

1. OLIVEIRA, J.; MENEGUETTE, R.; GONÇALVES, V.; JR., R. S.; GUIDONI, D.; OLIVEIRA,

J.; FILHO, G. R. F-NDIS – Sistema de Detecção de Intrusão Descentralizado baseando em Aprendizado Federado. Anais do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos. Porto Alegre, RS, Brasil: SBC, 2023. p. 29–42. ISSN 2177-9384. Disponível em: <https://sol.sbc.org.br/index.php/sbrc/article/view/24527>;

2. OLIVEIRA, J. A.; GONÇALVES, V. P.; MENEGUETTE, R. I.; de Sousa, R. T.; GUIDONI; D. L.; OLIVEIRA, J. C.; Rocha Filho, G. P. F-NIDS — *A Network Intrusion Detection System based on Federated Learning. Computer Networks*, v. 236, p. 110010, 2023. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128623004553>.

1.4 ESTRUTURA DA DISSERTAÇÃO

O restante desse trabalho está organizado da seguinte forma. O Capítulo 2 apresenta os conceitos fundamentais pelos quais esse trabalho se estabelece. No Capítulo 3 os demais trabalhos, que recentemente foram publicados sobre o tema, são discutidos. O Capítulo 4 descreve como o sistema funciona, aspectos técnicos e de arquitetura, além de como as técnicas utilizadas para distribuição de tarefas de treinamento e de privacidade diferencial são combinadas e orquestradas. O Capítulo 5 apresenta os resultados por meio da avaliação de desempenho de classificação e robustez do sistema. O Capítulo 6 conclui a dissertação com as discussões, conclusões e proposições de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Compreender um NIDS distribuído diz respeito não somente a saber sobre sua utilidade prática e suas vantagens, mas também todos os fundamentos que dão origem a esse tipo de solução de segurança e técnicas pelas quais se confere um diferencial à solução. Dessa maneira, dominar os motivos pelos quais um sistema dessa natureza se torna necessário, o que o qualifica com um detector de intrusão, como é implementado e como mensurar sua eficiência são todos aspectos que devem ser compreendidos. Com esse trabalho visa propor um NIDS com características únicas, os conceitos que sustentam tais características precisam ser adequadamente introduzidos.

No decorrer desse capítulo, os conceitos utilizados na construção do F-NIDS serão apresentados. As questões como cada conceito é baseado, como é implementado, as vantagens e possíveis desvantagens serão esclarecidas. Tais conceitos dizem respeitos ao Aprendizado de Máquina, Aprendizado Federado, Sistemas Distribuídos, Aprendizado Federado, Privacidade Diferencial e as Métricas de desempenho para um sistema de detecção de intrusão.

2.1 APRENDIZADO DE MÁQUINA

O aprendizado de máquina é um campo da inteligência artificial que revolucionou a maneira como as máquinas podem aprender a partir de dados [46]. Conforme o ilustrado na Figura 2.1, o aprendizado de máquina é um subcampo da inteligência artificial que se concentra no desenvolvimento de algoritmos e modelos que permitem que os sistemas computacionais melhorem seu desempenho em tarefas específicas à medida que são expostos a mais dados. Em vez de serem programados explicitamente para realizar uma tarefa, os sistemas de aprendizado de máquina aprendem com exemplos e experiência [47, 46].

Ainda segundo [47, 46], o aprendizado de máquina possui uma ampla gama de aplicações em diversas áreas, incluindo:

- **Classificação:** Classificar e rotular automaticamente dados, como detecção de spam em e-mails ou diagnóstico médico.
- **Regressão:** Prever valores contínuos com base em dados, como previsão de preços de ações.
- **Agrupamento:** Agrupar dados em categorias semelhantes, útil em análise de mercado e segmentação de clientes.
- **Recomendação:** Sugerir produtos, conteúdo ou serviços com base no comportamento do usuário, como o algoritmo de recomendação da Netflix.
- **PLN (Processamento de Linguagem Natural):** Compreender e gerar texto natural, o que é usado em chatbots, tradução automática e análise de sentimentos.

- **Visão Computacional:** Interpretar imagens e vídeos, usados em reconhecimento facial, veículos autônomos e controle de qualidade industrial.

De acordo com [47, 46], o processo de aprendizado de máquina pode abranger cinco etapas distintas. A primeira delas é a coletada de dados, preparação, onde ocorre a limpeza e formatação, e a separação entre conjunto de teste e treinamento. A segunda etapa envolve a seleção dos algoritmos e é a etapa onde os algoritmos de treino e as técnicas de treinamento serão selecionadas, levando-se em consideração os aspectos de performance de predição e outros fatores tais como segurança e privacidade. Na terceira etapa o processo de treino do modelo em si é trabalhado, onde são construídas relações entre as entradas e as saídas desejadas. A quarta etapa diz respeito a avaliação do modelo obtido, em que vários índices de desempenho são analisados além da avaliação dos demais aspectos definidos anteriormente, na segunda etapa. Na última etapa é feita a implantação do modelo, onde a plataforma de execução recebe o modelo e os agentes de predição, e as operações de aprendizado de máquina são monitoradas. Outros autores como [48] preferem fazer uma divisão em seis etapas, em que a coleta é colocada em uma etapa exclusiva e a etapa de implantação é substituída por uma etapa de otimização e ajustes dos modelos. Sendo que, no aprendizado de máquina, há duas abordagens básicas, essas abordagens são chamadas de aprendizado supervisionado e o não supervisionado [48].

O aprendizado supervisionado é uma abordagem na qual um algoritmo é treinado com um conjunto que compreende pares consistindo em uma entrada e na saída correspondente. O objetivo é que o algoritmo aprenda a mapear as entradas para as saídas desejadas, de modo que possa fazer previsões ou classificações precisas em novos dados. Já o aprendizado supervisionado é comumente aplicado em tarefas de classificação e regressão, onde os rótulos são fornecidos para orientar o processo de aprendizado [48].

O aprendizado não supervisionado é uma abordagem em que um algoritmo é treinado em um conjunto de dados que não contém rótulos explícitos. O objetivo é que o algoritmo encontre estruturas intrínsecas nos dados, como clusters, realizar associações ou representações. Essa abordagem é usada em tarefas de agrupamento, redução de dimensionalidade e análise de componentes principais [48].

Existem alguns fatores que ajudaram a massificar a utilização da aprendizagem de máquina. Dentre os quais pode-se elencar primeiramente a capacidade dos modelos de aprendizado de máquina de generalizar a partir de dados de treinamento para fazer previsões em novos dados. Há também a capacidade de automatizar tarefas que seriam difíceis ou impossíveis de serem programadas manualmente, sendo esse um fator preponderante. Por fim, a possibilidade de melhoria contínua dos modelos para se aprimorar com o tempo à medida que mais dados se tornam disponíveis foi fundamental para a larga adesão dessa técnica [48, 46].

Apesar dos fatores que colaboraram para sua popularização, ainda há alguns pontos de evolução, pois os modelos podem não funcionar adequadamente em situações com dados diferentes dos quais foram usados para treinamento do modelo, fenômeno esse denominado *overfitting*, ou sobreajuste [46]. Há ainda outro desafio com relação a interpretabilidade dos modelos, pois alguns modelos de aprendizado de máquina, como redes neurais profundas, são difíceis de interpretar, o que pode ser problemático em aplicações críticas [49]. Por fim, alguns algoritmos de aprendizado de máquina podem exigir um volume significativo de dados para produzir um modelo de predição eficaz.

2.1.1 Redes Neurais Artificiais

As RNAs (Redes Neurais Artificias) são modelos que compreendem uma das técnicas de aprendizado de máquina e que se estabeleceram como uma das mais transformadoras revoluções no campo da inteligência artificial. Essa abordagem permite que haja avanços notáveis em áreas como visão computacional e processamento de linguagem natural, além de robótica e na área de medicina. Essa técnica consiste em um subcampo do aprendizado de máquina. Nessa subseção será discutida essa técnica, como funciona e as aplicações que a tornam tão impactante. Já o DL (*Deep Learning* — Aprendizado Profundo), também consiste em uma subárea do aprendizado de máquina que trata das RNAs que possuem em sua arquitetura várias camadas ocultas compostas de diferentes algoritmos [46].

Figura 2.1: Diagrama de cebola representando os conjunto de técnicas da IA. Adaptado de [1].

Essas redes são inspiradas no funcionamento do cérebro humano, sendo compostas por camadas interconectadas de neurônios artificiais. A presença de múltiplas camadas permite eliminar algumas etapas previstas no processo de aprendizado de máquina tradicional, isso dá ao modelo capacidade de aprendizado usando representações cada vez mais abstratas dos dados de entrada [50, 51].

As RNAs são geralmente treinadas através do processo de aprendizado supervisionado, embora permita também o uso da abordagem não supervisionada. Durante o treinamento, as redes ajustam automaticamente os pesos sinápticos das conexões entre os neurônios visando minimizar a diferença entre as saídas previstas e as saídas reais, com base em um conjunto de dados de treinamento. Mediante sucessivas iterações, que envolve a retropropagação do erro e atualização dos pesos, a rede se torna capaz de representar características complexas e realizar tarefas como classificação, regressão, geração de texto e detecção de objetos [52, 46].

Para a compreensão do funcionamento da aprendizagem de máquina primeiramente é necessário compreender o perceptron. O perceptron é uma estrutura que consegue realizar uma classificação binária, separando os dados de entrada em duas categorias com base em um limite de decisão determinado pelos pesos e pela função de ativação. A Figura 2.3 ilustra a arquitetura básica dessa estrutura.

No perceptron as entradas e saídas poderão ser números não binários e cada conexão entre camadas é associada com um peso sináptico. É feita uma soma ponderada dos valores dos pesos sinápticos $z =$

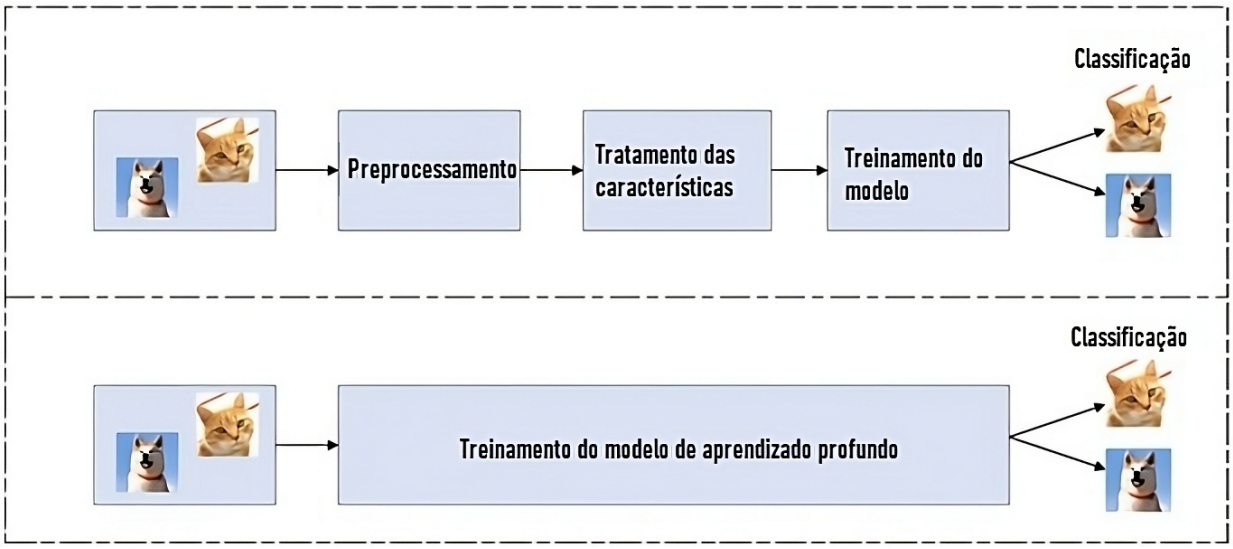


Figura 2.2: Diferença entre o aprendizado profundo e o aprendizado de máquina tradicional. Adaptado de [2].

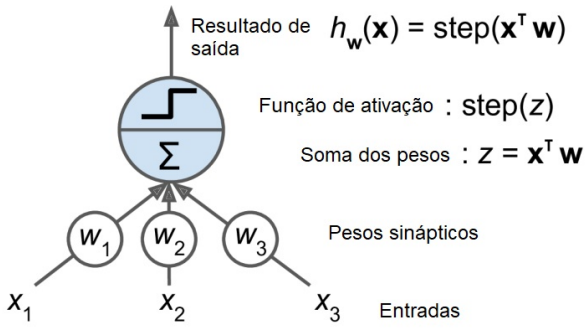


Figura 2.3: Perceptron com três neurônios de entrada, três pesos sinápticos e um neurônio de saída. Adaptado de [3].

$w_1x_1 + w_2x_2 + ... + w_nx_n = x^T w$, por fim é aplicada a função degrau *(step function)* de ativação $h_w(x) = step(z)$, em que $z = X^T W$. Os pesos são inicialmente inicializados com valores aleatórios e o treinamento é feito pela função de otimização seguida do ajuste dos pesos. É importante destacar o papel que a função de ativação tem de distinção entre os dois tipos de resultados de classificação. Um perceptron básico também pode vir dotado de um neurônio de viés inserido na primeira posição de cada camada com um valor constante $x_0 = 1$. A função de otimização, que calcula o valor do próximo passo de ajuste $w_{i,j}^{(i+1)}$ do perceptron, é descrita na Equação 2.1, com $w_{i,j}$ representando a conexão sináptica entre o i-ésimo neurônio de entrada x_i com o j-ésimo valor de saída $\hat{y}_j$, η a taxa de aprendizagem e y_j o valor original esperado como resultado:

$$w_{i,j}^{(i+1)} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i \tag{2.1}$$

Embora o perceptron consiga realizar algumas tarefas de separação entre duas regiões linearmente distinguíveis, ela se mostra insuficiente para resolver problemas mais complexos, onde não há uma divisão linear entre classes, por exemplo [52, 46]. Uma vasta gama de problemas existentes são em sua maioria não linearmente separáveis, então um mecanismo que permita tratar desses problemas significa um grande salto em relação ao cenário anterior. A Figura 2.4 ilustra como se dá esses problemas de separação.

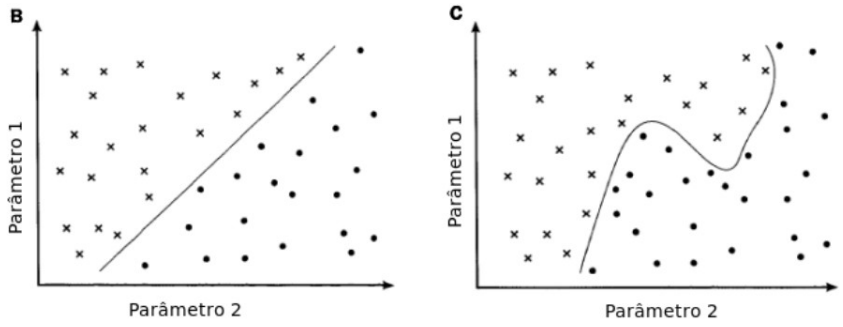


Figura 2.4: À esquerda o exemplo de um problema linearmente separável que pode ser resolvido por um perceptron sem camadas ocultas e à direita um não linearmente separável cuja resolução demanda um perceptron com uma ou mais camadas ocultas. Retirado de [4].

Uma estrutura de MLP (*Multilayer Perceptron* — Perceptron Multicamadas) é como se chama o perceptron que possui uma camada oculta entre a camada de entrada e a de saída. Essa camada oculta dá ao perceptron a habilidade de resolução de problemas não linearmente separáveis. A Figura 2.5 (adaptada de [3]) ilustra um MLP com duas entradas, uma camada oculta com 4 neurônios na camada oculta e três neurônios na camada de saída, totalizando três camadas.

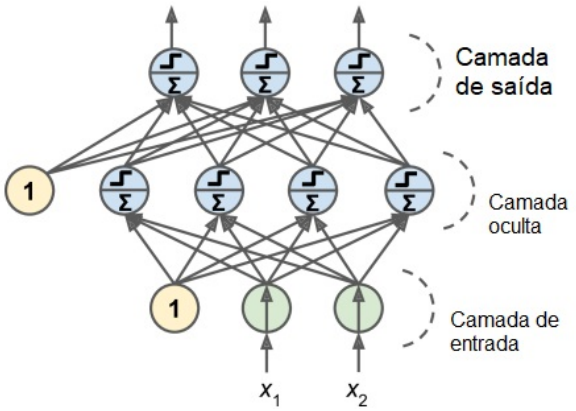


Figura 2.5: Exemplo básico da arquitetura MLP. Adataptado [3].

Quando ocorre o empilhamento de várias camadas ocultas intermediárias, o perceptron dá origem a uma RNA, onde cada camada terá um papel de interpretar determinado nível de abstração. Tal abordagem resulta no aprendizado profundo. Embora alguns autores consideram que o MLP seja um caso particular de RNA [3].

Após a introdução dos MLPs, entendeu-se que a função de ativação usada anteriormente, na Equação 2.1, se mostrou inadequada para viabilizar o treinamento dessas redes com mais de duas camadas. Isso se deve ao fato do melhor ajuste dos pesos sinápticos envolve a resolução de problemas de otimização, onde o uso de derivadas é imprescindível. Sendo que a função *step*, até então utilizada, não possui derivada no ponto zero. Portanto, foram introduzidas novas funções de ativação juntamente com o algoritmo de *backpropagation* — retropropagação dos pesos sinápticos —. A Figura 2.6 ilustra o processo de retropropagação em uma rede neural artificial.

O algoritmo de *backpropagation* desempenha um papel fundamental no sucesso das redes neurais

artificiais, tornando-as capazes de aprender a partir de dados e melhorar seu desempenho ao longo do tempo. Ele é usado para ajustar os pesos das conexões entre os neurônios da rede visando minimizar o erro entre as previsões do modelo e as saídas desejadas. O processo funciona retroativamente, calculando o gradiente do erro em relação aos pesos, camada por camada, a partir da camada de saída até a camada de entrada. Esse gradiente é usado para atualizar os pesos das conexões, permitindo que a rede aprenda a representação dos dados e a realizar tarefas complexas, como classificação, por exemplo. Entretanto, não é possível ser feito sem as funções de ativação. É importante destacar que o algoritmo pode ser aplicado não apenas no cálculo do gradiente, podendo ser empregado em outras tarefas de como, por exemplo, na análise do modelo aprendido [46].

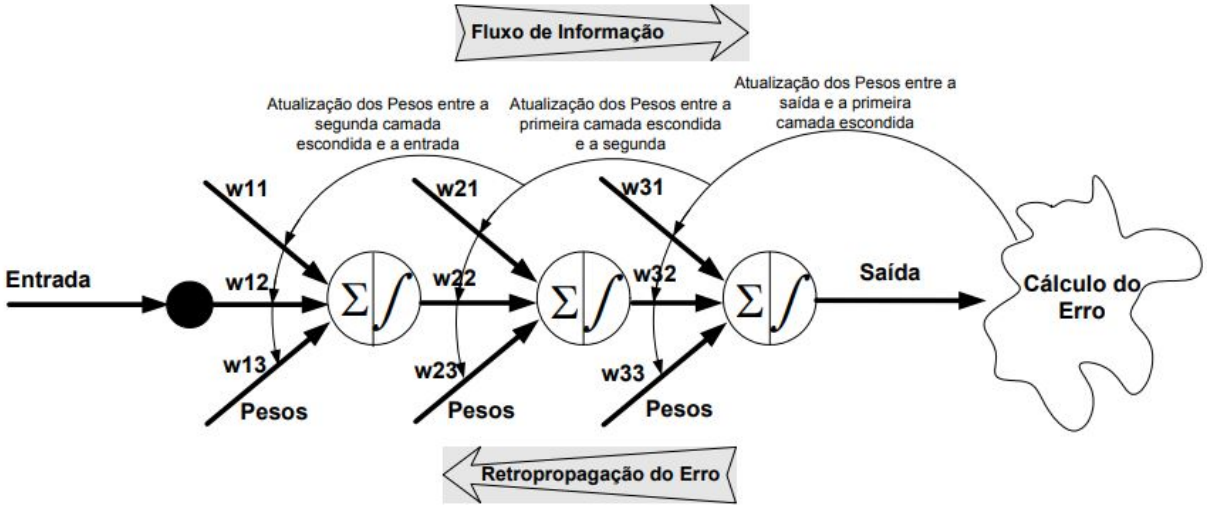


Figura 2.6: Sentido do fluxo de informação e de retropropagação do erro. Adaptado de [5].

Dentre os métodos de Descida de Gradiente, o método SGD (*Stochastic Gradient Descent* — Descida do Gradiente Estocástico) é amplamente utilizado no treinamento de redes neurais e no aprendizado de máquina em geral [46, 47]. Esse algoritmo é um aperfeiçoamento do método tradicional existente na regressão linear, ao consistir numa técnica para gradualmente encontrar os valores de θ, com θ representando os parâmetros ótimos a serem encontrados. Encontrar tais valores ótimos envolve minimizar a função de custo $J(\theta)$ e atualizar gradualmente os valores. Sendo η o tamanho do passo de atualização do treino (ou taxa de aprendizagem). A atualização dos valores segue a seguinte fórmula:

$$\theta' = \theta - \eta \bigtriangledown_\theta J(\theta) \tag{2.2}$$

No algoritmo de Descida do Gradiente envolve o cálculo do gradiente para a função $J(\theta)$, exemplo a exemplo. Tal operação eleva exageradamente o custo computacional associado. O algoritmo de Descida do Gradiente Estocástico reduz esse esforço usando apenas uma amostra dos exemplos disponíveis. Essa amostra é chamada de *minibatch* $\mathbb{B} = \{x^{(1)}, ..., x^{(m')}\}$ escolhidos aleatoriamente sobre o conjunto de treino. Sendo que o tamanho de $\mathbb{B}$ fica tipicamente entre um e algumas centenas de exemplos, dependendo do tamanho do conjunto de treino. A Equação 2.3 descreve a função L de custo usada no cálculo do gradiente para o método de SGD:

$$J(\theta) = \frac{1}{m'} \bigtriangledown_\theta \sum_{i=1}^{m'} L(x^{(i)}, y^{(i)}, \theta) \tag{2.3}$$

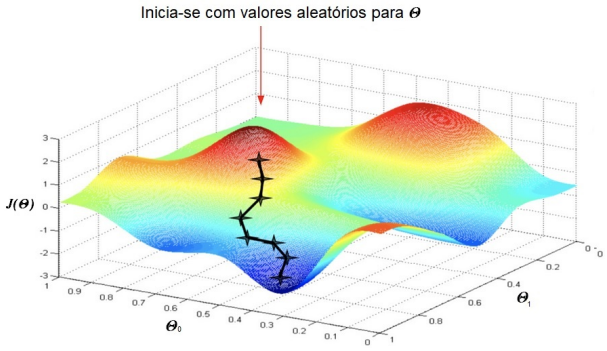


Figura 2.7: Atuação do algoritmo SGD sobre um plano tridimensional. Adaptado de [6].

Dessa maneira, o algoritmo de SGD produz uma alteração gradual, porém bastante efetiva dos parâmetros de treino, conforme ilustrado na Figura 2.7. Nessa figura, é possível observar que houveram sete pequenas correções graduais até que se encontrasse os valores ótimos para θ que minimizam o valor da função de custo $J(\theta)$. O resultado mínimo dessa função, aliado ao algortimo de *backpropagation*, irá produzir ajustes dos pesos sinápticos mais precisos e, portanto, a rede neural terá aprendido.

2.2 *FEDERATED LEARNING*

A FL é uma técnica de aprendizagem de máquina que permite a aprendizagem colaborativa em dados distribuídos e, ao mesmo tempo, mantém alguns níveis de privacidade dos dados. Introduzida em 2016, tal técnica permite que vários dispositivos treinem coletivamente um modelo compartilhado, realizando treinamento local em seus dados sem compartilhá-los centralmente [53]. Isso é possível por meio da troca de pesos sinápticos do modelo entre o dispositivo de treinamento e o agregador de modelos, que transforma vários modelos em um modelo global. Esse paradigma de treinamento descentralizado atenua as preocupações com a privacidade e a sobrecarga dos canais de comunicação [54]. A Figura 2.8 ilustra quais as entidades presentes no FL e como eles interagem entre si para o treinamento federado.

O processo envolve a inicialização de um modelo global em um servidor central, a seleção de dispositivos para participar, a execução de treinamento local nos dados de cada dispositivo, a agregação dos modelos atualizados no servidor central e a iteração dessas etapas até que a convergência ou o desempenho desejado do modelo seja alcançado [53, 54, 55, 56]. Ao manter os dados locais e evitar o compartilhamento de dados, o aprendizado federado aborda as preocupações com a privacidade e reduz os custos de comunicação [54], tornando-o adequado para cenários com fontes de dados distribuídas e muitos dispositivos [57].

O FL oferece uma abordagem escalável para distribuir tarefas de treinamento de modelos. Nesse paradigma de aprendizado descentralizado, o processo de treinamento ocorre em dispositivos ou agentes locais, permitindo a paralelização e o treinamento eficiente. Esse processo de treinamento de modelo distribuído reduz a sobrecarga de comunicação e aumenta a escalabilidade. A capacidade de prover escalabilidade

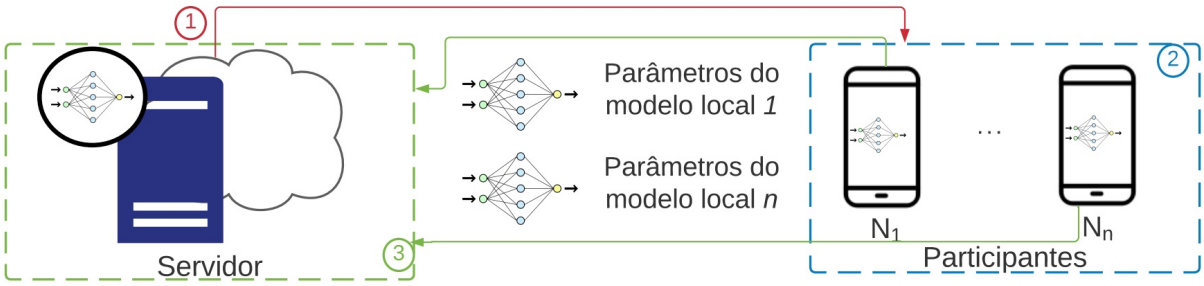


Figura 2.8: Arquitetura de funcionamento Aprendizado Federado. Adaptado de [7].

da aprendizagem federada foi destacado em vários estudos, incluindo [58], demonstrando sua capacidade de lidar com tarefas de aprendizagem distribuída em grande escala. No entanto, [58] afirma que um dos principais desafios é encontrar o equilíbrio certo entre a preservação da privacidade e a precisão do modelo, pois a introdução de perturbação gaussiana para proteção da privacidade pode afetar a utilidade do modelo treinado. Além de gerenciar efetivamente o custo de privacidade, quantificar a proteção de privacidade fornecida é outro desafio no aprendizado federado com privacidade diferencial. O gerenciamento dos canais de comunicação, sincronização e recursos para acomodar o treinamento com inúmeros membros federados é outro desafio importante a ser abordado [58, 54].

Tal técnica também permite a integração com mecanismos de DP (*Differential Privacy* — Privacidade Diferencial), atenuando algumas das lacunas de segurança remanescentes do FL. Tais lacunas referem-se a possibilidade de reconstrução do modelo a partir dos pesos. Gerando a possibilidade de inferência de informações a partir desses modelos reconstruídos [59]. A DP introduz maior garantia de que qualquer versão de um conjunto de dados estatísticos permaneça igualmente confiável, independentemente de conter ou não um item específico [38], minimizando a possibilidade de inferência de informações individuais confidenciais, mas mantendo as propriedades estatísticas do conjunto de dados.

2.3 PRIVACIDADE DIFERENCIAL

A privacidade diferencial é um framework para compartilhar publicamente informações sobre um conjunto de dados, mantendo as propriedades estatísticas no conjunto de dados sem que informações sobre indivíduos no conjunto de dados sejam passíveis de extração [60, 61]. A ideia por trás da privacidade diferencial é a de se minimizar o efeito de uma substituição única no conjunto, de modo que o efeito no conjunto de dados seja indistinguível de antes da operação ter sido realizada. Caso essa minimização for significativa, o resultado de uma única consulta não poderá ser usado para inferir muito sobre um indivíduo em particular e, portanto, os indivíduos do conjunto terão sua privacidade assegurada. Resumidamente, algoritmos de privacidade diferencial são todos aqueles onde o observador, vendo sua saída, não puder dizer se a informação de um indivíduo em particular é fidedigna. A Figura 2.9 ilustra o efeito esperado após o algoritmo de privacidade diferencial atuar sobre um conjunto de dados.

Para se alcançar esse resultado, algoritmos de privacidade diferencial introduzem ruídos estatísticos nos dados no durante o armamento ou produção. Esses ruídos conseguem tornar confidencial dados de

cada membro do conjunto ao passo que ainda possibilita que o público, destinado para acessar os dados, extraiam as informações de interesse contidas nesses dados sem exibir informações de identificação individual dos membros [62, 60]. Essa abordagem é possível por meio de parâmetros que regulam a quantidade de ruído, ou ainda pode haver parâmetros que regulam um trecho do conjunto que será considerado. Por exemplo, ao lançar uma moeda para cada indivíduo e adicionar aleatoriedade aos dados ou descartar arbitrariamente alguns lançamentos, ainda sim é possível garantir que a probabilidade de uma consulta estatística produzir um determinado resultado seja a mesma, independentemente da circunstância onde a consulta será realizada.

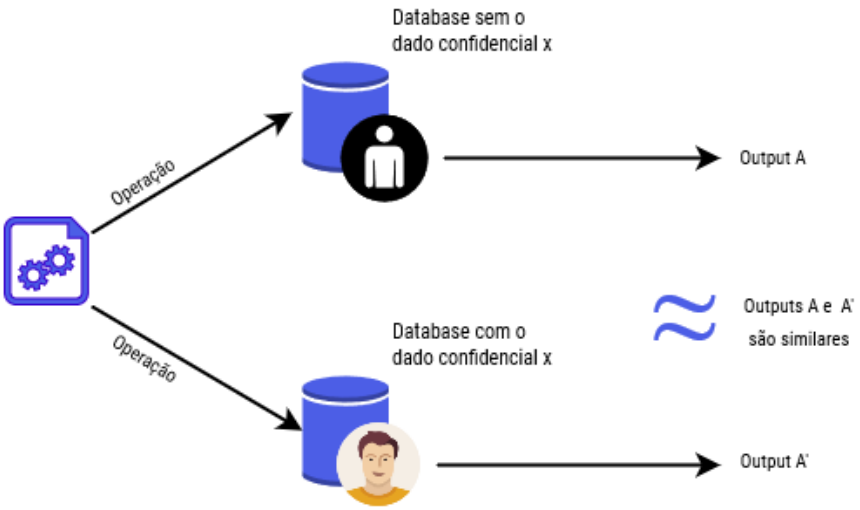


Figura 2.9: Princípio básico de funcionamento da Privacidade Diferencial. Adaptado de [8].

A privacidade diferencial tem aplicações importantes nas publicações de dados oficiais conduzidas pelo setor público ou ainda o aprimoramento de dados em organizações do setor privado. Ela tem se revelado uma ferramenta que permite com que organizações compartilhem informações agregadas sem comprometer a confidencialidade das respostas individuais [61]. Além disso, algoritmos diferencialmente privados possuem resistência contra os ataques de inferência de membros e inversão do modelo. Portanto, essa abordagem desempenha um papel crucial na proteção da privacidade em um mundo cada vez mais orientado por dados.

2.3.1 Definições e fundamentos matemáticos

Sendo dois bancos de dados $\mathcal{D}$ e $\mathcal{D}'$ que diferem em apenas um registro. Considerando um mecanismo de randomização $\mathbf{M}[\bullet]$ que recebe um conjunto de dados e retorna um novo conjunto. Esse mecanismo é diferencialmente privado se o resultado de $\mathbf{M}[\mathcal{D}]$ e $\mathbf{M}[\mathcal{D}']$ são indistinguíveis para quaisquer $\mathcal{D}$ e $\mathcal{D}'$, ou seja:

Um mecanismo $\mathbf{M}[\bullet]$ é ϵ-diferencialmente privado se, para todos os subconjuntos $\mathcal{S} \subset \mathbf{M}$ e conjuntos $\mathcal{D}$ e $\mathcal{D}'$, a seguinte expressão for válida:

$$\mathbf{Pr}(\mathbf{M}[\mathcal{D}] \in \mathcal{S}) \leq \mathbf{exp}(\varepsilon)\mathbf{Pr}(\mathbf{M}[\mathcal{D}'] \in \mathcal{S}) \tag{2.4}$$

O termo ε controla quanto o resultado produzido pelo mecanismo pode divergir após o processamento dos dois conjuntos de dados adjacentes, além de registrar quanto da privacidade é perdida quando o mecanismo é aplicado nos dados. Portanto, quanto menor o valor de ε mais eficaz é o mecanismo de privacidade, pois a ausência do registro removido se torna imperceptível. Tal definição foi proposta inicialmente em [38] e serviu de base para trabalhos subsequentes. Esses trabalhos concluíram não ser flexível o suficiente para produzir mecanismos de DP capazes de lidar com conjuntos com outras disparidades, além da ausência de apenas um registro entre si. Sendo assim, uma definição mais flexível foi introduzida em [63].

O conceito de (ε, δ)-privacidade diferencial foi introduzida como uma solução aos problemas de flexibilidade encontrados na definição anterior. Além de ser considerada uma forma particular da definição de privacidade diferencial expressa na Equação 2.4, é descrita formalmente da seguinte maneira:

$$\mathbf{Pr}(\mathbf{M}[\mathcal{D}] \in \mathcal{S}) \leq \mathbf{exp}(\varepsilon)\mathbf{Pr}(\mathbf{M}[\mathcal{D}'] \in \mathcal{S}) + \delta \tag{2.5}$$

O termo δ é o responsável pelo relaxamento da noção anterior de DP. A definição 2.5 embora seja mais fraca que a definição anterior, descrita na Equação 2.4, ela permite que se flexibilize a quantidade de perturbação a ser inserida pelo mecanismo nos conjuntos até um determinado valor de δ. Tal flexibilização se traduz nas possibilidades de ajuste fino dos limiares de privacidade, dependendo das necessidades de privacidade requeridas pela aplicação e também em função das propriedades específicas aos conjuntos de dados que se deseja trabalhar.

2.3.2 Mecanismo gaussiano de privacidade diferencial

Dentre os diversos mecanismos existentes, o mecanismo Gaussiano trabalha com a sensividade S_f adicionando ao conjunto de dados inicial uma perturbação que segue uma distribuição normal de média 0 e desvio padrão $S_f\sigma$, da forma $\mathcal{N}(0, S_f^2 \cdot \sigma^2)$ [38].

A sensividade S_f de uma função $f : \mathbb{N}^{|\mathbf{X}|} \rightarrow \mathbb{R}^k$, com x, y entradas adjacentes, é:

$$S_f \stackrel{\text{def}}{=} \max_{\|x-y\|_1=1} \big\|f(x) - f(y)\big\|_2 \tag{2.6}$$

Portanto, o mecanismo Gaussiano de privacidade diferencial é definido por:

$$\mathbf{M}(d) \stackrel{\text{def}}{=} f(d) + \mathcal{N}(0, S_f^2 \cdot \sigma^2) \tag{2.7}$$

Sendo $\varepsilon \in (0, 1)$ arbitrário, o mecanismo Gaussiano satisfaz a (ε, δ)-privacidade diferencial se $\sigma \geq cS_f/\varepsilon$, com $c^2 > 2ln(1.25/\sigma)$.

Dentre as vantagens do mecanismo Gaussiano pode-se elencar o tipo de ruído adicionado, sendo do mesmo tipo de ruído que eventualmente já exista no conjunto. Além disso, a soma de dois ruídos Gaussianos continua sendo um ruído que continua sendo modelado pelo próprio mecanismo, tornando o efeito estatístico fácil de entender e ajustar. A Seção 4.3 apresenta o algoritmo de privacidade diferencial através do mecanismo Gaussiano.

2.4 ESTRATÉGIAS COMUNS DE VIOLAÇÃO DA PRIVACIDADE EM ML

Embora os modelos de ML possibilitarem grande eficiência para realização tarefas complexas de classificação, existem ameaças relacionadas a confidencialidade dos dados usados no treino. Compreender como ocorrem como essas ameaças se configuram, consiste em um prerrequisito fundamental para a construção e treino seguro desses modelos. Em ataques que ameaçam a confidencialidade dos dados, existem três tipos estratégicas comuns: a inferência de membros baseada em regras, a inferência baseada em caixa preta e a inversão do modelo.

2.4.1 Inferência de membros baseado em regra

O ataque utilizando o ataque funciona explorando o modelo de ML treinado. Basicamente, o atacante utiliza um conjunto de regras para avaliar se determinados pontos de dados foram utilizados no treinamento do modelo. Tais regras podem ser obtidas a partir do próprio modelo original. Isso é feito observando as respostas do modelo alvo para diferentes consultas e comparando essas respostas com um conjunto os critérios estabelecidos. Se o comportamento do modelo indicar de modo consistente se ele foi treinado com determinados exemplos, o atacante pode inferir que tal amostra de dados faz parte do conjunto de treinamento original. Com acesso a essas informações, o atacante pode comprometer a privacidade dos dados e até mesmo a segurança do modelo, explorando vulnerabilidades relacionadas à privacidade e à generalização do modelo.

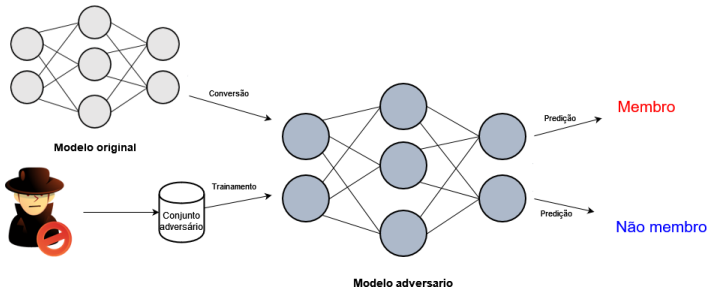


Figura 2.10: Inferência através da técnica baseada em regra. Adaptado de [9].

A Figura 2.10 ilustra o processo típico de ataque de inferência de membros baseado em regra. Nesse ataque, o agente malicioso consegue obter o modelo original e em seguida o converte em um modelo adversário que é capaz da realização de classificação binária, entre membros e não membros. Para tornar o modelo adversário mais eficaz, um retreino do modelo adversário pode ser realizado por meio de um conjunto de dados adversário composto ou não de exemplos membros. Privacidade diferencial é uma das técnicas utilizadas para proteção contra esse cenário de ataque.

2.4.2 Ataque de Inferência de membros em caixa preta

O ataque de inferência de membros em caixa preta, assim como a baseada em regras, é estratégia em que um adversário tenta determinar se uma amostra particular de dados foi utilizada no treinamento de um modelo, mesmo que sem o acesso direto ao modelo alvo. No entanto, na modalidade de caixa preta

ocorre normalmente por meio de um conjunto de dados denominado *Shadow Dataset*. Um *Shadow Dataset* consiste em um conjunto de dados adversário com dois rótulos, membros e não membros, gerados a partir de uma fração de exemplos de membros reais, extraídos de maneira não autorizada do conjunto de dados original. Já a outra parte é composta de exemplos de não membros. Por fim, esse conjunto adversário é utilizado pelo atacante para treinar um modelo de classificação binária adversário, conhecido como *Shadow Model*. Uma vez que esse modelo adversário é treinado, ele pode ser usado para inferir se determinados exemplos foram ou não utilizados no treinamento do modelo de ML alvo. Portanto, tal ataque pode ser muito eficiente para a violação de privacidade dos dados de treino.

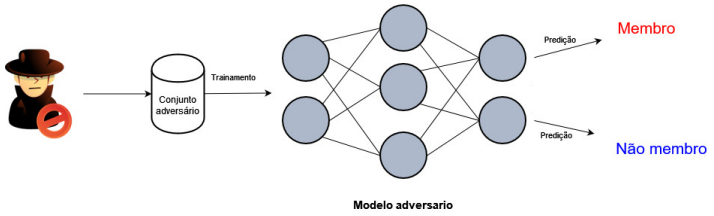


Figura 2.11: Técnica de ataque de inferência baseada em caixa preta. Adaptado de [9].

A Figura 2.11 ilustra o processo de ataque de inferência baseado em caixa preta. Nesse ataque, o agente malicioso obtém os dados de treino e em seguida constrói um conjunto adversário com dois rótulos (membros e não membros). Então, sabendo a arquitetura do modelo alvo, um *shadow model* é montado e treinado a partir do *shadow dataset*. Tal modelo então será capaz de predizer quais exemplos pertencem ou não ao conjunto de dados completo. É possível proteger contra esse cenário de ataque ao se fracionar o conjunto de dados em amostras menores que inviabilizem o treinamento do modelo adversário que possua utilidade mínima, o FL pode ser um importante aliado para tal.

2.4.3 Ataque de Inversão do modelo

A técnica de inversão de modelo é um ataque que visa reconstruir dados sensíveis ou de treinamento a partir das saídas do modelo, comprometendo a confidencialidade das informações originais. Ao explorar as respostas do modelo, os invasores podem inferir detalhes sobre os dados de entrada, como características individuais dos membros do conjunto e suas informações confidenciais. Esse tipo de ataque é especialmente preocupante em cenários nos quais os modelos de ML são utilizados para lidar com dados sensíveis, como informações médicas ou financeiras.

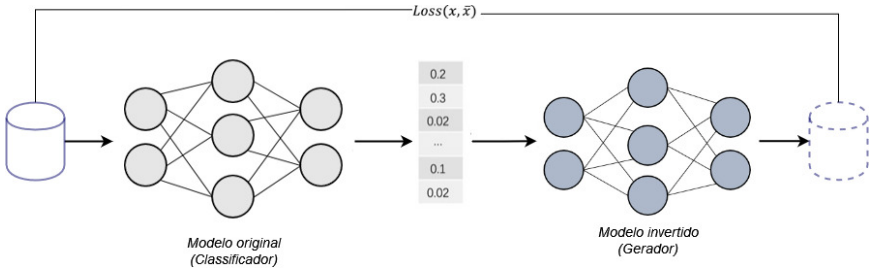


Figura 2.12: Ataque utilizando a técnica de inversão do modelo. Adaptado de [10].

A Figura 2.12 ilustra como esse ataque é possível. Assim como o ataque de inferência de membros

baseado em regra, nessa modalidade de ataque é utilizada apenas o modelo original, porém invertendo as entradas e saídas. Esse ataque, no entanto, ao invés de produzir resultado de classificação binária, o ataque gera exemplos próximos ao *dataset* original utilizado para treinamento desse modelo. Tal ataque é possível de ser mitigado através da DP.

2.5 MÉTRICAS DE DESEMPENHO UTILIZADAS NA DETECÇÃO DE INTRUSÃO

A tarefa de mensuração da performance de um detector de intrusão baseado em anomalia é importante para confirmar sua efetividade. Efetividade para um NIDS significa uma baixa taxa de falsos positivos e de falsos negativos, por outro lado, uma alta taxa de verdadeiros positivos e negativos. Os NIDS com essa característica podem apresentar disparidades para detecção de cada classe de tráfego [64]. Essas disparidades são mais comuns quando o modelo de classificação é treinado utilizando conjuntos de dados desbalanceados [65, 66]. Portanto, esses sistemas precisam obedecer certos critérios de avaliação para assegurar que seu funcionamento está calibrado e adequado, conseguindo detectar ameaças adequadamente ou não criar erroneamente o alerta.

2.5.1 Matriz de Confusão

Dentre os métodos de avaliação gráfica do desempenho do classificador usado no sistema de detecção de intrusão, a Matriz de Confusão — *Confusion Matrix* — é um método que consiste em posicionar os resultados reais em linhas horizontais e as predições posicionadas em colunas. Dessa maneira, os resultados desejáveis ficam posicionados na diagonal principal e os indesejáveis ficam posicionados no entorno diagonal. Isso porque, na diagonal principal, TPs (*True Positives* — Verdadeiros Positivos) e TNs (*True Negatives* — Verdadeiros Negativos) representam exatamente a intersecção entre as predições obtidas e os valores reais. Quanto maior a proporção de exemplos contidos na diagonal principal, mais bem-sucedido é o modelo de classificação. É importante ressaltar que os exemplos usados para a plotagem da matriz precisam ser de exemplos do conjunto de teste, para um resultado mais correto do ponto de vista metodológico. A Figura 5.3 ilustra uma matriz de confusão 2×2, destinada a avaliar um cenários de duas classes.

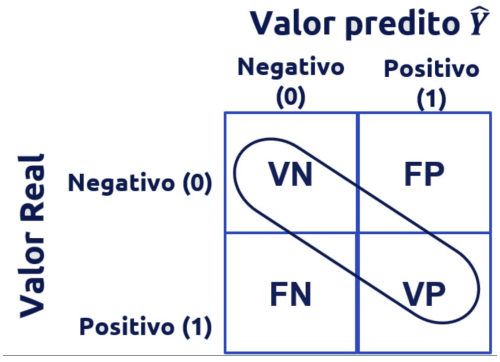


Figura 2.13: Exemplo de uma matriz de confusão para duas classes V e F.

.

2.5.2 *Precision*, *Recall* e *F1 Score*

A métrica de *precision* — precisão — mede a proporção de ocorrências classificadas como positivas que são realmente pertence ao conjunto de TP. Ou seja, ela consegue responder se todas as instâncias classificadas como positivas, quantas delas são realmente positivas. Uma alta precisão indica que o modelo faz poucos erros ao classificar instâncias como positivas, o que é importante quando se deseja minimizar falsos positivos. Em sistemas de detecção de intrusão, a alta precisão é crucial para evitar falsas mensagens de alerta que podem sobrecarregar os administradores de segurança com informações irrelevantes. Essa métrica é definida matematicamente na Equação 2.8.

$$Precision = \frac{TP}{TP + FP} \tag{2.8}$$

O *recall*, por sua vez, avalia a proporção de todas as instâncias verdadeiramente positivas que foram corretamente identificadas pelo modelo. Isso significa que consegue informar se todas as instâncias positivas existentes, quantas o modelo conseguiu encontrar. Um alto *recall* é essencial quando se deseja identificar a maioria das instâncias positivas e minimizar falsos negativos. Em sistemas de detecção de intrusão, um *recall* eficaz é fundamental para garantir que as ameaças reais não sejam ignoradas. Essa métrica também pode ser chamada de sensividade, sendo definida matematicamente na Equação 2.9.

$$Recall = \frac{TP}{TP + FN} \tag{2.9}$$

Para obter um valor que engloba tanto a precisão quanto o *recall* o *F1 score* pode ser utilizado, resultando em valores equilibrados entre as demais métricas em análise. Essa métrica é uma média harmônica entre a precisão e o *recall*, onde os valores menores ganham maior preponderância no resultado. A Equação 2.10 define matematicamente tal conceito.

$$F1\ score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{2.10}$$

É importante notar que a precisão e o *recall* geralmente podem se encontrar uma condição de oposição. O que significa que aumentar a precisão, em casos relativamente comuns, pode produzir uma diminuição do *recall* e vice-versa. Portanto, encontrar o equilíbrio certo entre essas duas métricas é importante, dependendo dos requisitos específicos do sistema. Em algumas situações, como na detecção de ataques, pode ser necessário priorizar uma métrica sobre a outra com base nos requisitos de segurança da organização.

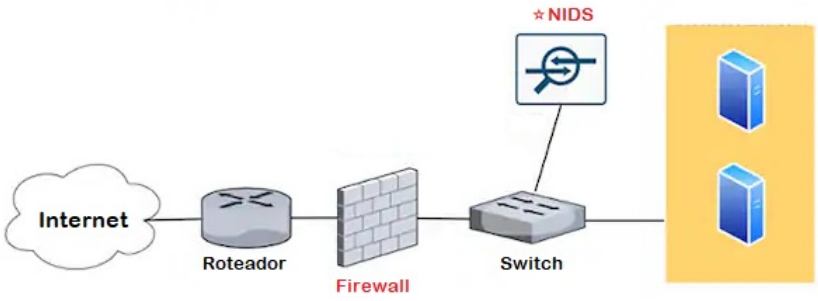
2.6 DETECÇÃO DE INTRUSÃO EM REDES DE COMPUTADORES

A detecção de intrusão é um componente de grande importância para a segurança cibernética. Ela envolve a monitorização contínua de uma rede de computadores em busca de atividades suspeitas ou não autorizadas. Seu objetivo é identificar qualquer tentativa de acesso malicioso ou fora das finalidades previstas. Tal monitoramento visa mitigar o acesso ao uso indevido de recursos ou ainda ameaças à integridade

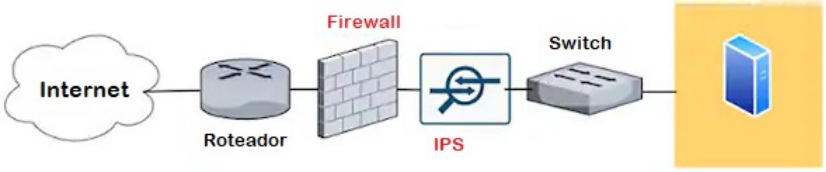
da rede e dos dados. Isso é alcançado por meio da análise de padrões de tráfego, eventos anômalos e comportamentos que podem indicar uma violação de segurança [64, 67]. A detecção de intrusão em redes de computadores é feita pelos NIDS cujas tarefas incluem:

- **Prevenção de ameaças**. Os sistemas de detecção auxiliam na identificação de ameaças antes que elas causem danos significativos. Isso permite uma ação proativa para mitigar riscos.
- **Proteção de dados sensíveis**. Em ambientes que armazenam informações sensíveis, como dados financeiros ou médicos, tais sistemas auxiliam na manutenção da confidencialidade e a integridade dos dados.
- **Conformidade regulatória**. As regulamentações as quais as organizações estão submetidas, podem exigir a existência desses sistemas.

A Figura 2.14a mostra uma configuração típica de um NIDS. Ele é um dispositivo posicionado lado a lado aos demais dispositivos conectados à rede local. Dessa maneira, é possível considerar o NIDS um sistema que detecta ameças que transitam em rede local, proveniente de máquinas que estão sobre controle de agentes maliciosos. Tal sistema age emitindo mensagens de alerta e notificações para agentes interessados. Já o IPS ilustrado na Figura 2.14b é um dispositivo posicionado imediatamente atrás de um firewall e se destina a realizar uma função de atuar na execução de políticas e contramedidas contra possíveis ameaças. Tais medidas podem incluir o bloqueio de pacotes suspeitos que transitam de ou para a Internet [64].



(a) Arquitetura de funcionamento de um NIDS.



(b) Arquitetura de funcionamento de um IPS.

Figura 2.14: Sistemas de detecção e prevenção de intrusão. Adaptado de [11].

Há dois tipos principais de NIDS, os baseados em assinatura e os baseados em anomalias. Os NIDS baseados em assinatura usam padrões conhecidos para identificar ameaças, essas ameaças são armazenadas num banco de dados internos, os pacotes de redes são comparados com esses registros. Caso haja uma compatibilidade entre o pacote e a assinatura, então o NIDS age para evitar um possível ataque. Essa

técnica é eficaz contra ameaças conhecidas, no entanto, pode ser inadequado para lidar com outros tipos de ameaças desconhecidos. Já os NIDS baseados em anomalias fazem o monitoramento do tráfego visando encontrar padrões de comportamento incomuns. A detecção por anomalias é normalmente feita usando técnicas de aprendizado de máquina e outros meios estatísticos. Esses NIDS são eficazes para detecção de ameaças desconhecidas ou que visam explorar brechas na segurança abertas recentemente, porém a desvantagem reside no fato de que podem ser suscetíveis a ocorrências de detecções com falsos positivos.

Outro aspecto importante sobre o funcionamento dos NIDS são as técnicas usadas na captura do tráfego. As técnicas de captura mais comuns é o *Probe Mode* — Modo Promíscuo —, onde o mecanismo de captura é conectado como um dispositivo qualquer na rede, porém configurado para capturar todos os pacotes via configurações na camada de rede da pilha TCP/IP. Há também a técnica de captura que envolve o uso de Agentes de *Software*, instalados nos dispositivos que captura os pacotes que transitam na placa de rede desses nós apenas. Há também a técnica de espelhamento portas, que ocorre nas camadas física e de enlace por meio de uma configuração no *switch* da rede. Nessa configuração, o tráfego que passa pelas portas espelhadas também é enviado ao NIDS para detecção, como descrito na Figura 2.15.

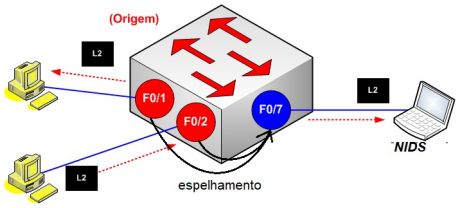


Figura 2.15: Exemplo de espelhamento de portas.

2.7 CLOUD E FOG COMPUTING

A computação em nuvem é uma extensão da computação distribuída em que os recursos de servidores remotos são usados para armazenar dados, executar aplicativos e disponibilizar serviços pela Internet. Ao contrário do paradigma local, todos os serviços são movidos para *datacenter* para além das dependências físicas onde os serviços são consumidos e acessados através da Internet [68]. Nesse paradigma, os custos de se manter um *datacenter* escalável, e altamente resistente a falhas, reduzem significativamente, entretanto existem alguns desafios e desvantagens associados. Tais desvantagens dizem respeito às questões de interoperabilidade, portabilidade, integração e governança [69], além de segurança, localização dos dados, dependência da Internet e processo de migração [70].

A *Fog computing* — computação em névoa — estende os conceitos de computação distribuída, trazendo o processamento para borda da rede. Isso reduz a latência e aprimora a eficiência, tornando-a crucial para aplicações de IoT [71]. Dessa maneira é possível manter os benefícios da computação distribuída local onde se preserva aspectos de segurança e latência, e quando é conveniente também utilizam-se os sistemas distribuídos no ambiente de nuvem em que é possível e desejável que alguns dados e sistemas estejam acessíveis através da Internet.

A Figura 2.16 ilustra como esses dois conceitos se relacionam para atender as necessidades dos clientes. Aplicações e informações, que demandam menor latência e maior desempenho e segurança, ficam armazenadas nas proximidades dos dispositivos que as acessam, ou seja, na camada *Fog*, os demais serviços e dados que demandam maior disponibilidade geográfica e resistência a falhas ficam armazenados na camada *Cloud*. Dispositivos clientes se comunicam com os dispositivos da camada neblina normalmente através da rede local e com os serviços da nuvem geralmente por meio da Internet.

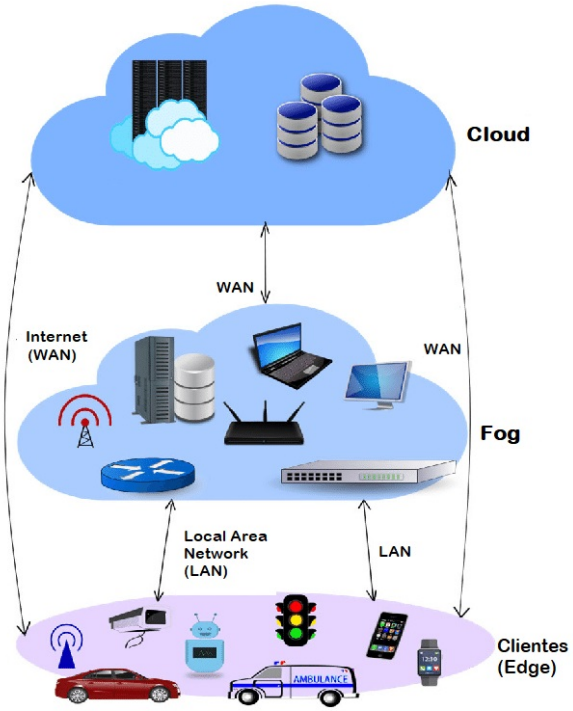


Figura 2.16: Arquitetura de comunicação em rede utilizando Cloud e Fog. Adaptado de [12].

2.8 CONSIDERAÇÕES FINAIS

Durante esse capítulo foi apresentado os principais conceitos, fundamentos e princípios necessários para construção de um sistema de detecção de intrusão com propriedades de segurança de dados e de escalabilidade. No decorrer do trabalho, esses conceitos serão aprofundados e será feito a conexão entre cada uma desses conhecimentos para fundamentar decisões de projeto, aspectos e demais detalhes técnicos. O Capítulo 3 trará luz sobre alguns trabalhos recentes que abordam soluções de NIDS sobre diversos aspectos e serão discutidos de um ponto de vista comparativo.

3 TRABALHOS CORRELATOS

Neste capítulo, serão apresentados os trabalhos que abordam soluções de detecção de intrusão, bem como serviram de inspiração para a criação do F-NIDS. Serão discutidas as questões sobre os aspectos técnicos sobre a qual cada solução se apoia. A performance de cada solução, além de possíveis vantagens e desvantagens de cada uma, em relação ao F-NIDS, também serão apresentadas e discutidas.

Soluções de NIDS centralizadas, embora estejam fora do escopo deste trabalho, foram desenvolvidas recentemente visando ampliar o nível de precisão e o desempenho das detecções de intrusão [72, 73, 74]. Especificamente, em [75], eles propõem o uso de ConvNets (*Convolutional Neural Networks* — Redes Neurais Convolucionais) em sistemas de detecção de intrusão. Em [76], uma rede neural profunda é definida com o empilhamento de *autoencoders* assimétricos, combinados com uma camada de saída em SVM (*Support Vector Machines* — Máquinas de Vetores de Suporte) para obter melhores níveis de precisão nas classificações de ataques, uma técnica chamada SNDAE (*Stacked Nonsymmetric Deep Autoencoder* — Autoencoder profundo não simétrico empilhado). Entretanto, além da falta de mecanismos para garantir a confidencialidade dos modelos resultantes, os trabalhos citados apresentam limitações quanto à sua escalabilidade para atender a redes altamente distribuídas, conforme pontuado em [19].

3.1 DETECTORES DE INTRUSÃO DISTRIBUÍDOS

No contexto de NIDS distribuídos, diferentes soluções foram propostas [77, 78]. Essas soluções abordam o uso de vários agentes com aprendizado de máquina e se concentram principalmente em problemas de desempenho e dimensionamento. O uso de FL também foi avaliado para a construção de detectores de intrusão [55], porém, eles se limitam a analisar dados provenientes de redes convencionais, sem o foco nas características específicas das redes de IoT e na confidencialidade dos modelos. No entanto, além da falta de mecanismos para garantir a confidencialidade dos modelos resultantes, os trabalhos citados têm limitações principalmente para atender a redes altamente distribuídas, conforme explorado em [19].

Outras soluções descentralizadas de NIDS, que exploram a questão da privacidade e da confiabilidade dos dados com mais ênfase, foram investigadas em [79, 80, 79] propõe o SP-CIDS, uma solução de ML para NIDS distribuído destinado a atender redes de veículos autônomos, aplicando técnicas de DP aos dados de treinamento. [80] desenvolve um NIDS distribuído para sistemas de saúde. Essa proposta usa redes neurais generativas e codificadores automáticos para proteger a confidencialidade dos modelos. Para agregação e transmissão de modelos [79] adota DMS (*Distributed Machine Learning* — Aprendizado de Máquina Distribuído) na agregação de modelos. Embora sejam arquiteturas destinadas a proteger os dados de treinamento, ambas as soluções admitem um ponto central de vulnerabilidade, permitindo a transferência de dados brutos entre um agente central e, portanto, são ineficazes para a proteção da privacidade [79, 81]. Além disso, ambos os trabalhos usam a suposição de que as redes neurais generativas produzem inerentemente modelos mais confidenciais para armazenamento e transmissão; no entanto, conforme demonstrado em [82], essas técnicas de ML também podem ser vulneráveis a violações de confidencialidade, especifi-

camente nos ataques de inferência nos dados de treinamento.

3.2 DETECTORES DE INTRUSÃO FEDERADOS

Em [83], é proposto um NIDS federado que privilegia a confidencialidade, usando técnicas de DP. Essa é a solução que mais se assemelha ao F-NIDS. Os autores adotaram o algoritmo de agregação de modelos `Fed+`. Os autores afirmam que esse algoritmo proporciona maior acurácia do modelo global, reduzindo os efeitos do ruído causado pelo DP e atenuando as perdas causadas pela própria agregação. Embora proponha um NIDS descentralizado, o cenário aborda apenas um subconjunto de aplicativos de IoT (ou seja, aplicações industriais) e não avalia sua generalização para outras aplicações de IoT, como, por exemplo, carros autônomos, cidades inteligentes e em *datacenters* em geral. Além disso, algumas limitações do algoritmo de agregação são levantadas, com relação à questão da capacidade de personalização dos agentes e também na robustez dos modelos gerados [84]. Vale ressaltar que [83] utilizou, na avaliação dos resultados, uma quantidade muito pequena de clientes, o que pode ter impactado os números obtidos, sendo uma estratégia que pode ser inadequada para testar um cenário real de aprendizagem federada, no qual são previstos algumas dezenas ou até milhões de clientes. Um estudo mais robusto com uma quantidade maior de clientes pode ser realizado para validar um NIDS usando aprendizado federado [58].

A proposta do F-NIDS se distingue do que foi proposto em [83] em dois fatores. O primeiro diz respeito ao uso do Algoritmo de agregação, em que no F-NIDS foi usado `FedAvg` e em [83] foi usado `Fed+`. Isso é relevante, pois o algoritmo `FedAvg` já demonstrou em trabalhos anteriores que possui grande potencial de escalabilidade, enquanto o `Fed+` ainda demanda avaliações adicionais a respeito de sua capacidade de se adaptar em configurações com centenas de clientes de treino. O segundo e mais importante fator de diferenciação se dá na arquitetura proposta. A arquitetura do F-NIDS se propõe a ser mais abrangente, a medida que aborda a escalabilidade não apenas as tarefas de treino federado do modelo, mas também do mecanismo de detecção em si. Ainda sobre o proposto em [83], é importante frisar que, embora a ausência de avaliação de robustez não implica em um sistema menos robusto, a ausência de tal avaliação aponta no sentido de estar incompleta para adoção imediata, visto que tal proposta necessita de apontar indicadores de robustez.

3.3 MECANISMO DE COMUNICAÇÃO EM AMBIENTE DISTRIBUÍDO.

Alguns trabalhos já foram realizados propondo sistemas que usam mecanismo *publish/subcribe* para fornecer comunicação entre as entidades do sistema. Assim, validando a escalabilidade e a disponibilidade proporcionadas por essa técnica. Em [85] foi proposta uma estrutura, usando essa técnica, que conseguiu melhorar a velocidade de processamento de dados de grande capacidade, garantindo mais estabilidade e resiliência em condições adversas, como largura de banda restrita em plataformas de nuvem de IoT. Por outro lado, [86] propôs uma solução de comunicação que aplica a técnica pub/sub em conjunto com NDN (*Named Data Networking* — Rede de dados nomeados) e aproveita a disponibilidade e a escalabilidade da troca de dados entre entidades. As avaliações realizadas em [87] chegaram à conclusão de que existem

ferramentas de código aberto para *publish/subcribe*, tanto em ambientes de IoT quanto de nuvem, que são tolerantes a erros e oferecem suporte a operações de processamento de fluxo ao vivo, podendo operar bem em um cluster. Assim, a capacidade de tal mecanismo na manutenção da tolerância a falhas e o suporte a cargas de trabalho pesadas o colocam como uma alternativa viável para distribuição de tarefas de detecção de intrusão.

3.4 COMPARAÇÃO DOS TRABALHOS

A Tabela 3.1 fornece uma comparação entre alguns dos outros trabalhos recentes de NIDS em termos de vantagens e limitações. Nessa tabela, as diferenças entre os trabalhos recentes de NIDS são destacadas, como o tipo, em termos de arquitetura e técnicas usadas; o número de agentes usados no treinamento do sistema e na avaliação do desempenho; o desempenho do sistema em termos de precisão e também as vantagens e limitações de cada NIDS estudado. Um aspecto importante nessa relação é a capacidade de prover detecções distribuídas, em que nenhum dos trabalhos listados proveu uma solução para abordar esse aspecto, como, por exemplo, através de algum mecanismo de comunicação assíncrona, como *publish/subscribe*.

Tabela 3.1: Estudo comparativo

Sistema	Tipo	Agentes	Detecção distribuida	Vantagens	Limitações
FELIDS - Friha et al. 2022 [88]	FL distribuído sem DP	15	não	Avaliação de muitos classificadores	Falta de mecanismo de preservação da privacidade; baixa quantidade de clientes
Fed+ IDS - Ruzafa-Alcáza et al. 2021 [83]	FL com DP	5	não	Avaliação de muitas técnicas de DP	Baixa quantidade de clientes; nenhuma avaliação de robustez
MS CNN - Jing et al. 2022 [75]	CNN centralizada	1	não	Métricas de alto desempenho	Ponto único de falha
Mbasuva & Zodi - 2022 [72]	Centralizado sem DP	1	não	Alto desempenho de classificação.	Sem mecanismo de preservação da privacidade.
NMAIFS MOP-AQAI - Ling & Hao - 2022 [73]	Centralizado sem DP	1	não	Altíssimo desempenho de classificação.	Avaliação ausente para ambientes de IoT.
Yu et. al. - 2022 [75]	Centralizado sem DP	1	não	Detecção de múltiplas classes de ataques.	Baixa acurácia, levando-se em consideração a complexidade do modelo.
Qazi et. al. 2022 [76]	Centralizado sem DP	1	não	Altíssimo desempenho de classificação e responsividade.	Ponto único de falha e sem mecanismo de preservação da privacidade.
Al-Yaseen et. al. 2017 [77]	Distribuído sem DP	Não informado	não	Alto desempenho num cenário distribuído.	Sem mecanismo de preservação da privacidade dos dados de treino.
Riyad et. al. 2019 [78]	Distribuído sem DP	Não informado	não	Alta tolerância à falhas.	Ausência de mecanismo de preservação da privacidade dos dados de treino.
Raja et. al. 2021 [79]	Distribuído com DP	Não informado	não	Alto desempenho de classificação com mecanismo de privacidade diferencial.	Ausência do número de agentes distribuídos.

3.5 CONSIDERAÇÕES FINAIS

A avaliação desses trabalhos permite concluir que ainda há problemas em aberto nesse campo. Entre esses problemas está a ausência de uma proposta de arquitetura distribuída capaz de prover detecções

distribuídas de modo escalável e resiliente a falhas, atendendo a várias cargas de trabalho. O outro problema, não menos importante, diz respeito à robustez contra ameaças a privacidade. A confidencialidade das informações é um quesito importante e que ou não é levada em consideração em alguns casos, ou não é feita de modo aprofundado. De modo que há espaço para uma proposta que aborde com maior ênfase a questão da privacidade dos dados, para modelos treinados colaborativamente. Nesse sentido, a relação de trabalhos demonstra que a robustez contra vazamentos de informações não foi adequadamente endereçada nos sistemas trabalhos relacionados.

No capítulo seguinte o F-NIDS é apresentado e os recursos que fazem do propriamente dito serão elencadas, incluindo os algoritmos, *frameworks* estratégias que embasaram a estrutura do sistema de um IDS Federado e Distribuído serão expostos.

4 F-NIDS - *FEDERATED NETWORK INTRUSION DETECTION SYSTEM*

Este capítulo apresenta o F-NIDS, um sistema de detecção de intrusão dotado de Privacidade Diferencial e Aprendizado Federado para permitir a escalabilidade horizontal das tarefas de treinamento. Na Seção 4.1 é apresentada a arquitetura do sistema, com uma visão geral e os principais mecanismos de funcionamento e como cada um desses elementos interage e resolve os problemas abordados até aqui. Já na Seção 4.2, o mecanismo de treinamento distribuído é introduzido, através do aprendizado federado. Na Seção 4.3, o mecanismo de privacidade diferencial do sistema, que confere robustez na confidencialidade, é descrito. Já na Seção 4.4, os aspectos de detecção distribuída no sistema serão melhor abordados. Por fim, o capítulo encerra com a Seção 4.5, onde são feitas as considerações finais sobre a arquitetura.

4.1 ARQUITETURA

O F-NIDS usa o FL para distribuir as tarefas de treinamento, orquestrar a troca de dados entre entidades por meio de rodadas federadas e agregar os modelos locais por meio do algoritmo FedAvg (*Federated Averaging* — Média Federada). A escolha pelo FL, e com FedAvg como método distribuído de treino e agregação, foi devido aos seus recursos de privacidade, como isolamento dos conjuntos de dados nos agentes e o compartilhamento apenas dos parâmetros do treino. Além disso, a técnica DP é usada pelo F-NIDS para fornecer uma camada de segurança adicional, protegendo a confidencialidade dos dados do modelo. Essa técnica aplica o algoritmo (*DP-SGD Differentially Private — Stochastic Gradient Descent* — Descida do Gradiente Estocástico diferencialmente privado) para treinar modelos locais, aplicando ruído nos pesos locais do modelo durante a execução do algoritmo de gradiente descendente no lado do agente. Em seguida, os modelos locais podem ser armazenados e trocados entre os agentes de forma mais segura. A escolha pelo DP-SGD se deu pelas propriedades de ser menos exigente em termos de capacidade computacional, tendo em vista que o F-NIDS é voltado para IoT, onde os dispositivos geralmente não possuem grande capacidade de processamento, a DP-SGD se mostra mais adequada. O serviço de detecção distribuída é implementado por meio do mecanismo de *publish/subscribe* para detecção distribuída de intrusão. Esse mecanismo foi escolhido, pois já é amplamente adotado por redes IoT, portanto os dispositivos voltados para esse paradigma já suportam normalmente os protocolos comunicação assíncrona via *pub/sub*, além das bibliotecas são nativamente instaladas, facilitando a implementação e adoção.

A arquitetura do F-NIDS é ilustrada na Figura 4.1. Na figura, o agente central (rótulo AC) gera os pesos globais iniciais (rótulo G), propagando-os para os outros membros do sistema. O CA tem a função de agregar pesos e produzir um modelo global usando uma técnica de agregação de peso local e também executa a orquestração, propagando os pesos obtidos nessa agregação. As tarefas de detecção, o recebimento dos pesos globais e o treinamento de um modelo local são realizados pelos agentes de detecção (rótulo DA - *Detection Agent*). Tais agentes poder ficar localizados na nuvem ou em *fog* visando proteção dos modelos, evitando assim vazamento do mesmo e a exploração maliciosa dos mesmos. Caso seja necessário, novos

DAs podem ser adicionados a esse sistema, escalonando as tarefas de treinamento. Na tarefa de detecção, o cliente enviará pacotes para serem analisados pelos DAs, de maneira assíncrona.

Além de treinar um modelo local, usando um conjunto de dados individual, essa estratégia permite subdividir em tarefas em ciclos menores de treinamento entre DAs heterogêneos. Tal estratégia é possível por meio do fracionamento do conjunto de treino em tamanhos mais adequados para as capacidades de *hardware* de cada DA. Como os DAs transmitirão apenas pesos locais ao final do treino, essa estratégia também se traduz num menor consumo de largura de banda de rede, o que o torna adequado para cenários distribuídos de IoT onde excessos de comunicação podem sobrecarregar a rede.

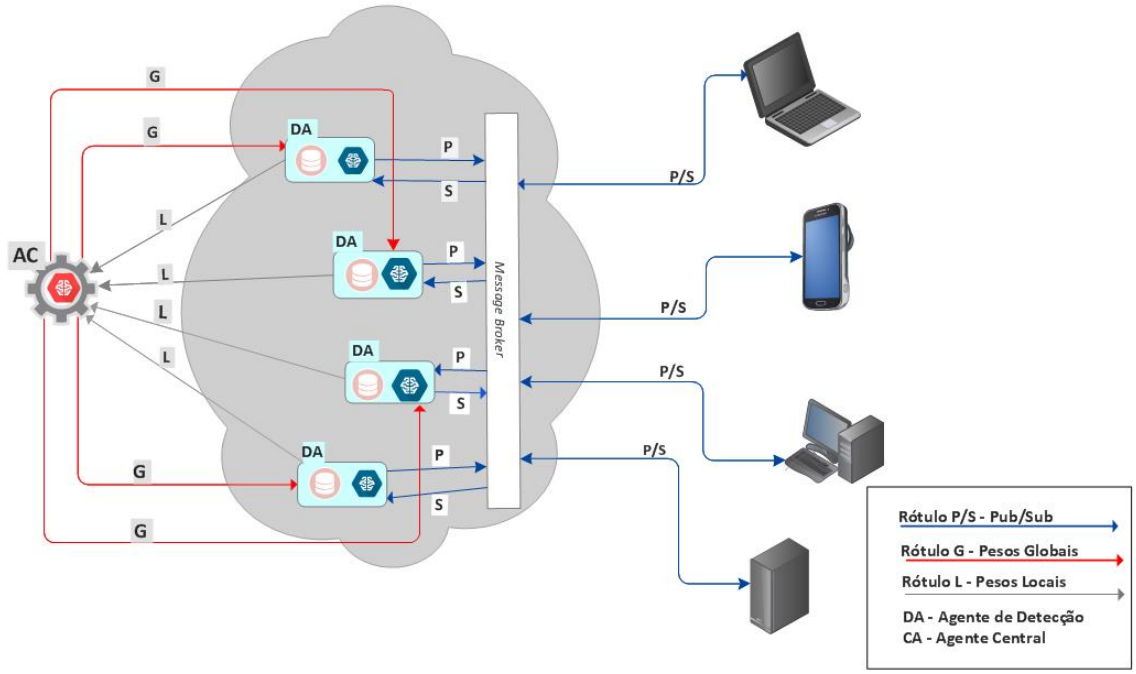


Figura 4.1: Arquitetura geral do F-NIDS.

O MB (*Message Broker* — Roteador de Mensagens) lida com tarefas assíncronas usando o modelo *publish/subscribe* e gerenciando filas. Ele coloca em fila as solicitações de mensagens de detecção dos clientes e as respostas do DA nas filas apropriadas. Dessa maneira os DA's estarão responsáveis pela classificação. Dessa maneira, os modelos estarão isolados dos clientes e, portanto, protegidos contra acessos visando a exploração maliciosa das propriedades estatísticas desse modelo visando acessar dados confidenciais. Embora seja real a possibilidade de que tal comunicação esteja sujeita a variações na latência, entende-se que tal efeito possa ser associado custos associados a uma maior segurança. Além disso, essa latência pode ser diluída a medida que existe uma quantidade elevada de DAs para realizar a detecção no menor espaço de tempo possível.

Outra característica importante do sistema é que tanto os clientes quanto os agentes de detecção têm

uma camada de *software* específica para ouvir eventos, publicar mensagens e subscrever filas no MB. A comunicação entre esses agentes ocorre por meio do MQTT (*Message Queuing Telemetry Transport* — Transporte de Filas de Mensagem de Telemetria), um protocolo leve que permite tratar eventos assíncronos e o desacoplamento de agentes. São utilizados três tipos de filas: a fila de solicitação de detecção, a fila de notificação (uma para cada cliente) e a fila de alerta. A fila de solicitação de detecção recebe mensagens de clientes assinadas por todos os agentes de detecção, enquanto as filas de notificação garantem que clientes específicos recebam notificações individuais, publicadas pelos agentes de detecção. A fila de alerta, subscrita por todos os clientes, permite a publicação de alerta pelos DAs que serão propagadas para todos os dispositivos do sistema. Ela permitirá que o sistema seja dimensionado horizontalmente, seja tolerante a falhas e esteja mais disponível.

Os clientes enviam solicitações de detecção publicando-as em filas internas de MB, permitindo assim ganhos de escalabilidade e confiabilidade. Para isso, a tarefa de captura de pacotes é feita pelos próprios clientes individualmente. Para oferecer maior robustez contra ataques de inferência nos modelos, cada DA vem equipado com uma camada de segurança adicional usando DP. Assim, o AC produz um modelo global que herda algumas propriedades de DP dos outros modelos locais. Essa arquitetura se baseia em três mecanismos: (i) mecanismo de treinamento descentralizado usando FL; (ii) mecanismo de treinamento com DP e (iii) mecanismo de detecção descentralizado e distribuído por meio de troca de mensagens assíncronas. Outro aspecto importante a ser enfatizado é que o F-NIDS, por ser baseado em FL, só transacionará os pesos sinápticos dos modelos. Visando minimizar a transferência de dados que podem ser vulneráveis a violações de confidencialidade e comprometer a largura de banda disponível. Para proporcionar maior robustez contra ataques de inferência nos modelos, durante o processo de obtenção dos pesos locais pelos DAs, foi incluída uma camada de segurança adicional usando DP, para proteger a privacidade dos modelos obtidos.

O F-NIDS foi projetado para ser implantado na nuvem, com os clientes realizando solicitações de detecção de intrusão pela Internet. Em uma implantação em nuvem, todos os DAs e o MB estarão localizados geograficamente distantes dos clientes. No entanto, ele também pode ser implantado em uma camada de neblina. Nesse caso, a arquitetura do sistema permite que os agentes de detecção sejam implantados localmente, mas perto dos clientes. Apenas o agente central é mantido na nuvem, apenas para fins de agregação do modelo. Como o modelo tem medidas de segurança para evitar a exploração maliciosa, devido ao mecanismo de DP, essa abordagem permanece segura para ser usada. As características distribuídas permitem o dimensionamento horizontal rápido, o que o torna adequado para o cenário de IoT.

4.2 MECANISMO DE APRENDIZADO FEDERADO COM PRIVACIDADE

Embora o F-NIDS seja distribuído, o agente central gera o modelo inicial com parâmetros aleatórios e realiza a orquestração do treinamento do modelo e a transmissão desses parâmetros. Essa orquestração é realizada por meio de variáveis chamadas hiperparâmetros federados. Esses hiperparâmetros (dispostos na Tabela 4.1) definem as configurações usadas nos classificadores treinados. Quando todas as condições definidas nos hiperparâmetros são atendidas, é iniciada uma rodada federada na qual a agregação de peso é feita por meio do algoritmo `FedAvg`. A rodada federada termina quando a propagação dos parâmetros

do modelo federado para os agentes de detecção é concluída.

Tabela 4.1: Lista de hiperparâmetros usados no treinamento do modelo

Hiperparâmetro	**Valor padrão**
Neurônios na camada oculta	160
Taxa de aprendizado	0,02
Épocas	10
Rodadas	10
Tamanho do minilote	1000
Conjunto de validação	20%
DP-SGD — Norma L_2	1,5
DP-SGD — Ruído σ	0,5
FL — Fração mínima de DAs de treinamento	0,1
FL — Fração mínima de DAs de avaliação	0,1
FL — Mínimo de DAs de treinamento	10
FL — Mínimo de DAs disponíveis	75

No `FedAvg` [53], o agente de treino e detecção $k_t \in K$ (com $n_k = |\mathcal{P}_k|$, em que $\mathcal{P}_k$ são os índices do conjunto de dados contidos no agente k_t, considerando $C = 1$ como o conjunto de dados completo, e η é a taxa de aprendizado) calcula o vetor de gradiente $g_k = \nabla F_k(w_t)$ correspondente ao treinamento local do modelo w_t. Em seguida, os próprios agentes recebem a tarefa de atualizar os pesos localmente por $k_t \leftarrow k_t - \eta \nabla F_k(k_t)$ várias vezes antes da etapa de agregação, que ainda é executada pelo CA. Nessa estratégia, o custo computacional é controlado por três hiperparâmetros: C, a fração de agentes que executam os cálculos em cada rodada; E, o número de épocas de cada agente em seu conjunto de dados local; e B, o tamanho do minilote usado por cada cliente. Assim, o `FedAvg` tem o valor de $B = \infty$ (tamanho do minilote igual ao tamanho do conjunto de dados local) e $E = 1$. O CA agrega esses gradientes e aplica o método $w_{t+1} \leftarrow w_t - \eta \nabla f(w_t)$, em que $\nabla f(w_t) = \sum_{k=1}^{K} \frac{n_k}{n} g_{t+1}^k$. No caso do F-NIDS, cada agente $k_t \in K$ representa um DA do sistema.

O Algoritmo 1 descreve esse processo de treinamento dos modelos de classificação F-NIDS. Ele consiste em duas etapas, a etapa global (das linhas 1 a 9) e a etapa local (das linhas 10 a 16). Na linha 1, um vetor de pesos globais é inicializado. Na próxima linha, ele itera sobre cada rodada e calcula internamente o número de clientes que participarão do treinamento em uma determinada rodada. A quinta linha itera sobre os agentes de treino e detecção registrados no F-NIDS, passando como argumento os pesos do modelo global e atualizando o vetor de pesos locais com os novos valores de peso retornados do DA (linha 6). Na linha 8, é feita a agregação real de todos os pesos dos modelos treinados localmente, para cada DA. Em seguida, os pesos globais são atualizados para serem usados na próxima rodada. A etapa local começa na linha 10 em diante, com o treinamento dos modelos locais nos agentes. A etapa começa com os dados de treinamento local divididos em um conjunto de B de minilotes. Nas linhas 11 e 12, ela itera sobre cada uma das épocas i dos agentes. Para cada uma dessas épocas, todos os minilotes de $b \in B$ são usados como argumentos para calcular os pesos locais do DA atual (linha 13), usando os pesos globais e da taxa de aprendizado do agente em questão como argumento. A etapa local termina quando um conjunto de pesos locais w do modelo é obtido e repassado ao agente central.

Adotando o FL na estratégia de projeto do F-NIDS, o sistema torna-se apto para a realização de tarefas de treinamento descentralizado do modelo, sem a necessidade de troca de dados de treino. Dessa maneira

Algorithm 1 `FederatedAveraging`. Os agentes *K* são indexados por *k*; *B* é o tamanho do minilote local, *E* é o número de épocas locais e η é a taxa de aprendizado.

AgenteCentralExecuta() ▷ //Executa no Agente Central

inicialize w_0
para a cada rodada t = 1, 2, ... **faça**
  $m \leftarrow \max(C \cdot K, 1)$
  $S_t \leftarrow$(conjunto aleatório de m clientes)
  para cada agente $k \in S_t$ **em paralelo faça**
    $w_{t+1}^k \leftarrow$AgenteDetecçaoAtualiza(k, w_t)
  fim para
  $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
fim para

AgenteDetecçaoAtualiza(k, w)**:** ▷ //Executar no agente de detecção k

$\mathcal{B} \leftarrow$ (divida $\mathcal{P}_k$ em lotes de tamanho B)
para cada época local i de 1 a E **faça**
  para lote $b \in \mathcal{B}$ **faça**
    $w \leftarrow w - \eta \nabla \ell(w; b)$
  fim para
fim para
retorna w ao agente central

o sistema supera as limitações mencionadas nos trabalhos [72, 73, 75, 76] da Tabela 3.1.

4.3 MECANISMO DE PRIVACIDADE DIFERENCIAL DO CLASSIFICADOR

Para garantir a mais alta confidencialidade dos modelos treinados, o F-NIDS implementa o algoritmo DP-SGD. Essa técnica é implementada localmente em cada DA durante a execução do algoritmo de descida de gradiente. Antes de os pesos serem atualizados, um trecho do gradiente é selecionado e o ruído gaussiano é adicionado a ele, produzindo um modelo seguro a ser armazenado ou seus pesos trocados na rede pelo algoritmo FL. Quando a CA recebe esse modelo seguro, o algoritmo `FedAvg` pode executar as tarefas de agregação normalmente, mas o modelo global herdará as propriedades de privacidade dos modelos locais.

Tal abordagem é a versão clássica do algoritmo de otimização do modelo SGD, mas inclui o DP. Esse algoritmo limita a sensibilidade de cada gradiente [89]. Seja $clip_c : g_t(x_i) \in \mathbb{R}^p \rightarrow g_t(x_i)/max(1, \frac{\|g_t(x_i)\|_2}{C}) \in \mathbb{R}^p$ a função de seleção aplicada sobre os valores de entrada de modo que o resultado tenha a norma máxima de ℓ_2 de C. Assim, a etapa de atualização do algoritmo DP-SGD é dada por: $w^{(t+1)} = w^{(t)} - \eta_t\{\frac{1}{B} \sum_{i \in \mathbb{B}_t} clip_c(\nabla_{w_t} \mathcal{L}(w_t, x_i)) + \delta\}$ e $\delta \sim \mathcal{N}(0, \sigma^2 C^2 I)$ é a variável aleatória correspondente ao DP gaussiano e σ^2 o desvio padrão do ruído [90].

O pseudocódigo completo de `DP-SGD` é apresentado no Algoritmo 2. A linha 1 contém a lista de hiperparâmetros usados no treinamento, a qual são os exemplos de treinamento (ou seja, taxa de aprendizado, escala de ruído σ, tamanho L e o limite de norma C). Na linha 2, um conjunto de pesos iniciais é

inicializado aleatoriamente. A partir da linha 3, o algoritmo faz uma iteração em cada época. Na linha 4, a cada época, uma amostra L_t é selecionada e, para cada subconjunto i de L_t, ele calcula o vetor de gradiente $g_t(x_i)$. Na linha 8, é feito o *clipping*, uma seleção de um trecho do vetor de gradiente inicial e, em seguida, é adicionado o ruído gaussiano na linha 9 e, por fim, são ajustados os pesos sinápticos do modelo como uma função do gradiente obtido e da taxa de aprendizado.

Algorithm 2 `DP-SGD`. Algoritmo SGD diferencialmente privado

1: **Input:** Amostras $\{x_1, ..., x_N\}$ e função de custo: $\mathcal{L}(w) = \frac{1}{N}\sum_i \mathcal{L}(w, x_i)$. Parâmetros: taxa de aprendizagem η_t, perturbação σ, tamanho do minilote L, norma do gradiente C e T é o número de épocas.
2: **Inicialização aleatória** w_0
3: **para** $t \in [T]$ **faça**
4: Obtenha uma amostra L_t com distribuição de probabilidade $\frac{L}{N}$
5: **para** $i \in L_t$ **faça** ▷ `//Obter gradiente`
6: $g_t(x_i) \leftarrow \nabla_{w_t}\mathcal{L}(w_t, x_i)$
7: **fim para**
8: $\bar{g}_t(x_i) \leftarrow clip_c(g_t(x_i))$ ▷ `//Seleção de um trecho do gradiente`
9: $\tilde{g}_t \leftarrow \frac{1}{L}\sum_i(\bar{g}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 I))$ ▷ `//Passo de pertubação`
10: $w_{t+1} \leftarrow w_t - \eta_t \tilde{g}_t$ ▷ `//Ajuste dos pesos`
11: **fim para**
12: **Retorna** w_T.

A escolha desse algoritmo para prover privacidade deve-se principalmente por ser um método de proteção da privacidade dos dados que exige menor custo computacional. Tornando o F-NIDS adequado para os dispositivos com limitações de desempenho, em que são muito comuns de serem encontrados em redes IoT. Ao adotar essa técnica, o F-NIDS supera as limitações de robustez na confidencialidade dos dados, existentes nos trabalhos [72, 73, 75, 76, 77, 78] da Tabela 3.1.

4.4 MECANISMO DE DETECÇÃO DISTRIBUÍDA

O F-NIDS divide o processo de detecção de intrusão em três etapas: captura, classificação e contramedidas. A captura é feita diretamente pelos clientes, usando um mecanismo de captura interno. Para atender aos casos em que pode ser impraticável a captura pelo cliente e o envio de pacotes a serem detectados, um tipo de agente especial pode ser implantado no sistema para capturar pacotes, por meio de alguma das técnicas de captura discutidas em 2.6, e enviá-los aos agentes de detecção. A detecção é realizada pelos DAs, publicando e assinando as filas de MB. As contramedidas podem ser executadas em conjunto pelos agentes e dispositivos com base em suas próprias regras, permitindo que cada um dos membros do sistema implemente sua política de repúdio individual contra agentes mal-intencionados.

A interação proposta do F-NIDS entre um cliente, interessado em detectar um pacote específico, e o DA é ilustrada na Figura 4.2. O cliente inicia o processo capturando um pacote e verificando se o remetente está na lista de repúdio. Se o remetente não estiver bloqueado anteriormente, o cliente envia um pacote com a solicitação de classificação para o DA. É importante ressaltar que o cliente não sabe qual DA será responsável pela detecção da intrusão, nem sua localização, pois o MB desacopla as partes. Entretanto, o MB garante que a comunicação terá uma resposta assíncrona, por meio do mecanismo *publish/subscribe*.

O classificador, ao prever que o pacote é benigno, notifica o cliente interessado. Se o pacote for classificado como mal-intencionado, o F-NIDS notificará o cliente inicial e emitirá um alerta para todos os outros clientes que se inscreveram na fila de alerta do MB. Essa notificação contém a probabilidade da classificação feita, qual classe de ataque foi detectada e a origem que emitiu o pacote. O cliente, ao receber notificações de ataque ou um alerta, inclui a origem na lista de repúdio e encerra todas as conexões com o remetente do pacote malicioso.

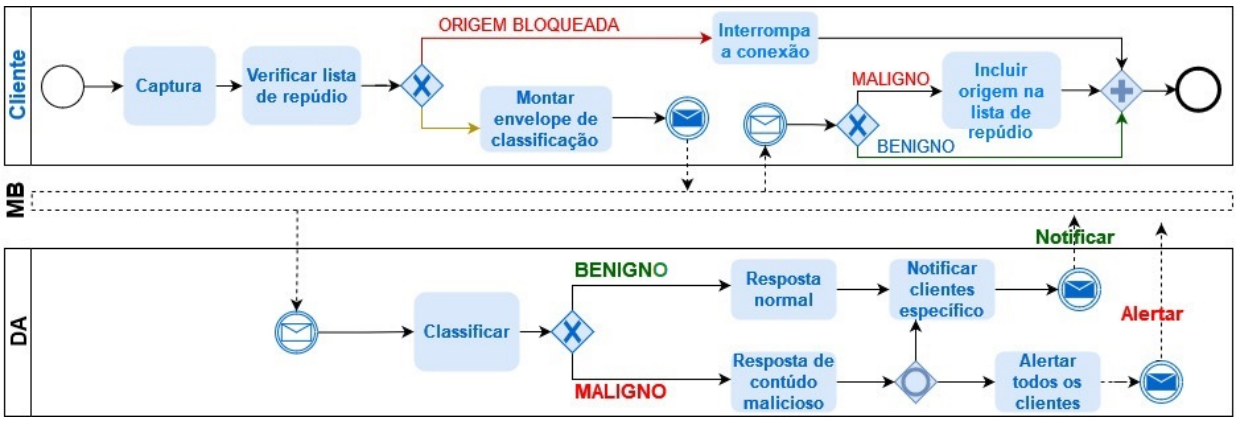


Figura 4.2: Operação básica entre um determinado DA e um cliente.

A figura 4.3 ilustra o modelo de comunicação entre todos os DAs disponíveis e os clientes interessados na detecção de intrusão. Para a detecção de uma intrusão, um cliente c publica uma mensagem m, formatada a partir de um pacote capturado, invocando a operação $pub(m)$. Ao realizar a classificação benigna de uma mensagem, o cliente interessado é notificado com uma mensagem de resposta r, por meio de uma operação $notify(c, r)$. Além de detectar uma intrusão, o agente publicará uma mensagem de alerta endereçada ao cliente interessado e também a todos os outros clientes, por meio da operação $alert(r, t)$, em que t é o resultado da classificação e o tipo de alerta. Um cliente subscreve aos resultados de alerta por meio da operação $sub(t)$; t é opcional; portanto, se t for inserido, o cliente assinará somente o tipo de alerta inserido; caso contrário, subscreverá os alertas de todos os tipos. No meio, um cluster de MBs trata as mensagens, garantindo que apenas um dos agentes de detecção as obtenha e processe. No caso de uma instância de um agente ficar indisponível, a comunicação continua a fluir normalmente, pois o cluster consegue lidar com esse evento adequadamente. É importante observar que a lista de repúdio é mantida individualmente por cada cliente e essa configuração foi pensada para permitir que cada um desses agentes implemente sua política individual contra agentes mal-intencionados.

Cada mensagem *m*, de notificação ou alerta, é emitida pelos dispositivos clientes seguindo o padrão de formato de mensagem IDEA (*Intrusion Detection Extensible Alert* — Alerta de Detecção de Intrusão Extensível) formulado por [91], um formato de comunicação que usa a notação JSON (*Javascript Object Notation* — Notação de objetos para *Javascript*) e que se baseia no IDMEF (*Intrusion Detection Message Exchange Format* — Formato de Troca de Mensagens de Detecção de Intrusão), proposto por [92]. Nesse padrão, todas as mensagens enviadas por agentes de detecção precisam ter apenas uma classificação e ser fornecidas com um conjunto de atributos com seus respectivos tipos que identificam a origem, o destinatário e o rastreamento de tempo de um pacote. Os atributos fundamentais são: *AnalyzerID*, sendo o identificador do agente que fez o alerta; *CreateTime* é a data em que a mensagem foi gerada pelo remetente do pacote; *DetectTime* e *AnalyzerTime* são os horários relativos à data em que o pacote foi enviado para

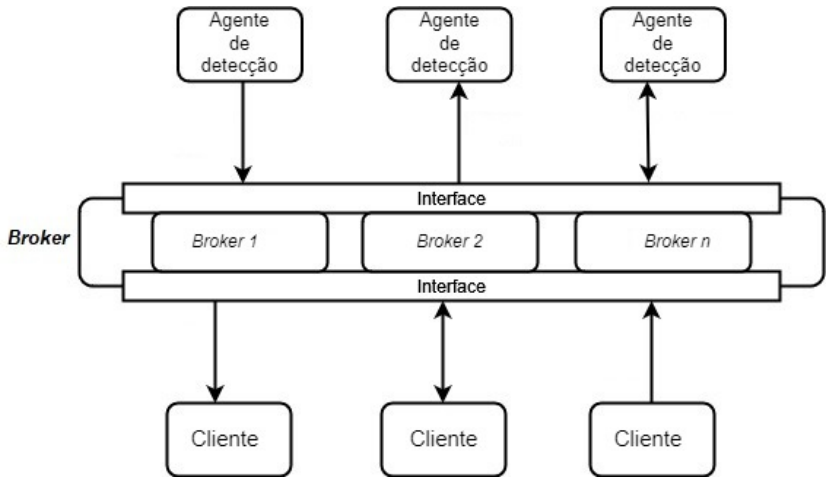


Figura 4.3: Modelo de comunicação F-NIDS.

análise e o horário em que foi analisado por um dos agentes de detecção, respectivamente. Os endereços IP do remetente e o destino do pacote são representados pelos atributos *Source* e *Target*.

Através dessa abordagem, o F-NIDS de torna capaz de prover o mecanismo de detecção descentralizada. O que o credencia para atender as diversas cargas de trabalho em redes IoT, onde a demanda pode sofrer flutuações abruptas e inesperadas. Ao adotar essa estratégia de projeto, o F-NIDS se torna capaz de fornecer, além de um treinamento descentralizado, também uma interface de detecção descentralizada, superando assim a limitação existente no trabalho [83] da Tabela 3.1.

4.5 CONSIDERAÇÕES FINAIS

Esse capítulo focou nos detalhes de desenvolvimento e funcionamento do sistema F-NIDS. Dois métodos de computação distribuída foram apresentadas, o primeiro foi para o treinamento distribuição das tarefas de treinamento de um modelo global de classificação de ameaças. O segundo método tratou de como disponibilizar os serviços de detecção em um ambiente de rede IoT com camadas de *Fog* e/ou *Cloud*, para o maior número de clientes possível.

Embora que, através da DP, o F-NIDS seja capaz de prover maior robustez para proteção da privacidade e assim evitar que agentes possam acessar dados de treino uns dos outros, é importante destacar que para haver privacidade ampla é necessário pensar no contexto em que os agentes e clientes estão inseridos. Dessa maneira, a implementação do F-NIDS precisa agrupar clientes e os agentes do sistema num contexto de segurança semelhante, por exemplo, numa mesma organização ou propósito de utilização. Outro ponto a se destacar é que se os DAs contiverem dados em excesso, essa arquitetura poderá oferecer vulnerabilidades com relação aos ataques de inferência de membros em caixa preta. No capítulo seguinte, os resultados das avaliações de desempenho de classificação foram efetuados no sistema. Para essas avaliações, uma metodologia foi estabelecida para a correta mensuração entre as principais abordagens de IDS baseado em anomalia.

5 METODOLOGIA E RESULTADOS

Nesse capítulo, é apresentada a avaliação de desempenho do F-NIDS. O sistema é avaliado comparando as métricas de desempenho de predição binária e multiclasse em termos de acurácia, precisão e *recall*. Cada um desses indicadores de desempenho do F-NIDS foi comparado com os outros três métodos comumente usados em outros trabalhos de NIDS. A métrica de acurácia visa verificar em qual rodada ocorre a convergência do F-NIDS em comparação com outros métodos.

O desempenho dos métodos foi avaliado usando, além da acurácia, a precisão e a *recall*. A precisão (P_c) é descrita na Equação 5.1 e é a medida da proporção de previsões positivas verdadeiras (TP_c) de todas as previsões feitas para uma determinada classe c, em que (TP_c) são os exemplos classificados corretamente como sendo de c e (FP_c) são os exemplos classificados incorretamente como sendo da classe c. O *recall* R_c é a proporção de previsões positivas verdadeiras de todas as instâncias positivas reais para uma determinada classe, descrita na Equação 5.2, em que FN_c são os exemplos classificados incorretamente como sendo de classe diferente da classe c. As métricas de precisão e *recall* são computadas entre uma classe em relação a todas outras.

$$P_c = \frac{TP_c}{TP_c + FP_c} \tag{5.1}$$

$$R_c = \frac{TP_c}{TP_c + FN_c} \tag{5.2}$$

O conjunto de dados usado é baseado no `NF-ToN-IoT-v2` [1]. Esses dados foram produzidos a partir dos arquivos `.pcap` do conjunto de dados `ToN-IoT` e, em seguida, processados para gerar dados na ferramenta `NetFlow` [2] [93], onde o banco de dados produzido tem 43 atributos relevantes. Para este trabalho, `IPV4_SRC_ADDR e IPV4_DST_ADDR` – que representam as informações dos endereços IP do remetente e do destinatário, foram removidos. A pesquisa usou apenas 41 dos atributos restantes. O conjunto de dados resultante é não identicamente distribuído e tem 2,5 milhões de registros extraídos aleatoriamente dos dados originais, representando aproximadamente 14,75% do volume de dados contidos no `NF-ToN-IoT-v2`. Considerando esses dados, 80% foram separados para o conjunto de treinamento e 20% foram alocados para o conjunto de teste. Para o treinamento federado, a fração aos 80% do *dataset*, destinados ao treino, foram divididos igualitariamente entre os 100 agentes de treinamento. A descrição completa do *dataset* pode ser acessado na Tabela de atributos 7.1 e na Tabela de classes 7.2.

Para comparar o F-NIDS com outros métodos de ML existentes, já estudados em outros trabalhos, os métodos de treinamento de classificadores avaliados são o método centralizado denominado ANN; o centralizado com DP denominado ANN-DP; o método de treinamento que aplica somente o algoritmo federado, denominado FED e o método federado com DP, do qual consiste o F-NIDS. Dez modelos individuais foram treinados para cada método. Na avaliação de desempenho, cada um desses modelos é avaliado

[1] <https://staff.itee.uq.edu.au/marius/NIDS_datasets/>

[2] <https://www.cisco.com/c/pt_br/tech/quality-of-service-qos/netflow/index.html>

com uma fração de 10% do conjunto de dados de teste. Para obter o desempenho geral binário e multiclasse nos quatro métodos, a acurácia média será analisada nas rodadas de treinamento na Seção 5.1.1. Essa estratégia visa observar em que rodada cada método atinge níveis relativos de convergência. A Seção 5.1.2 trata da avaliação da precisão obtida e dos resultados de *recall*, visando mensurar o desempenho da classificação em cada uma das classes individualmente. Nas avaliações realizadas nas subseções 5.2.1, 5.2.2 e 5.2.3, foram treinadas nove instâncias do F-NIDS, e cada uma dessas instâncias foi submetida a dez ataques adversários diferentes. As médias de acurácia, precisão e o *recall* foram usadas para comparar cada um dos resultados para cada modelo.

O experimento se dá através da implementação do detector federado (classificador) e da avaliação do mesmo usando as métricas mencionadas. O mecanismo de captura de pacotes e de comunicação distribuída e assíncrona entre clientes e DAs será feita fora do escopo desse trabalho. Na implementação do detector foram usadas as bibliotecas `Tensorflow` [3] para treinamento de rede neural, `TF Privacy` [4] para privacidade diferencial e Flower [5] para aprendizado federado. A tabela 4.1 apresenta a lista de hiperparâmetros usados no modelo de treinamento. Os hiperparâmetros do modelo foram obtidos usando o método *hyperparameter tuning*, e o tamanho do minilote foi o maior suportado pela GPU usada (NVIDIA V100 com 16 GB GDDR5). Os hiperparâmetros relacionados ao aprendizado federado foram escolhidos levando-se em conta a CPU e a memória RAM disponível (8 núcleos e 24 GB de RAM). Outros valores de hiperparâmetros foram testados, mas com um desempenho inferior do classificador em termos de acurácia e convergência.

5.1 AVALIAÇÃO COMPARATIVA DE DESEMPENHO DO F-NIDS

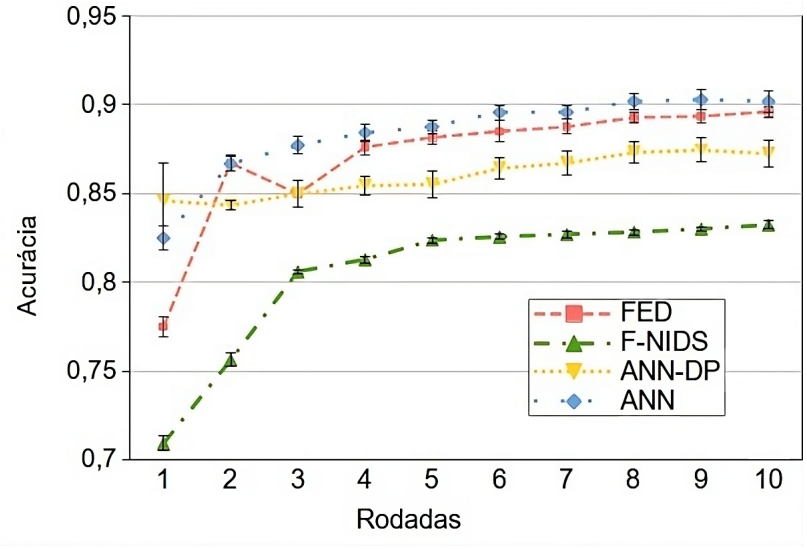
Nessa Seção serão apresentados e discutidos os resultados de desempenho do F-NIDS em relação aos demais métodos tradicionais de NIDS, que também utilizam ML por meio de um classificador MLP. A Seção 5.1.1 aborda os resultados de acurácia gerais multiclasse e a binária, respectivamente. Já nas Seções 5.1.2 e 5.1.3 o desempenho de cada classe são discutidos.

5.1.1 Resultados de acurácia do F-NIDS

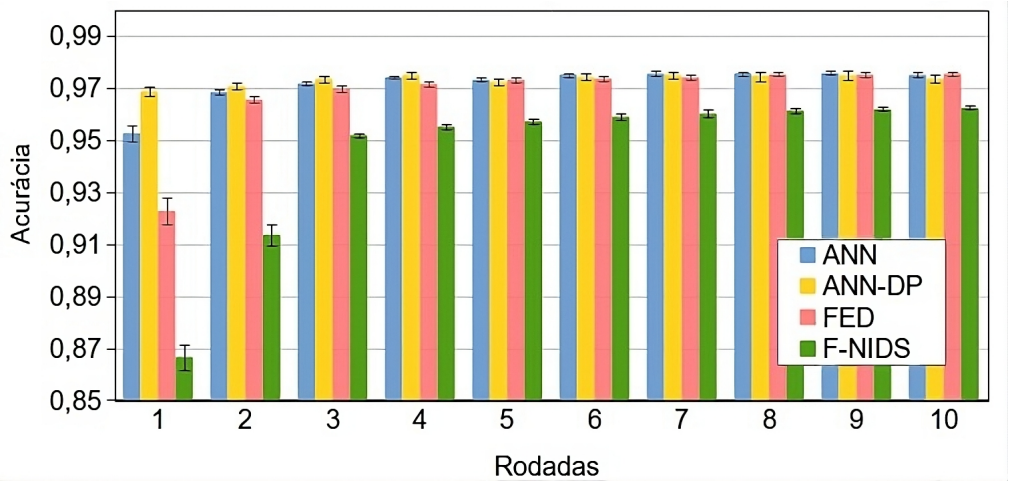
A Figura 5.1 apresenta os resultados de acurácia obtidos nos quatro métodos em estudo (ANN, ANN-DP, FED e F-NIDS), em função das rodadas de treinamento. Comparando os resultados na Figura 5.1a, o método ANN tem, em média, 7,7% mais acurácia do que o método F-NIDS. Ao comparar a acurácia do método ANN com os outros, as diferenças foram de 0,06% em relação ao FED e 3,0% em relação ao ANN-DP, respectivamente. No entanto, esses resultados estão dentro do esperado, pois, de acordo com [58], o algoritmo `FedAvg` exige um custo adicional em termos de acurácia do classificador, na medida que precisa agregar os pesos obtidos por modelos locais treinados com um número significativamente reduzido de exemplos. Portanto, sendo improvável que o modelo global seja exposto a todos os exemplos

[3] <https://www.tensorflow.org>
[4] <https://www.tensorflow.org/responsible_ai/privacy/guide>
[5] <https://flower.dev/>

de treino, em contraposição ao que ocorre no método centralizado de treinamento. É importante considerar também o efeito dos hiperparâmetros do algoritmo `DP-SGD`, que afetam significativamente a acurácia do classificador ao incluir ruídos no modelo. Apesar de todas essas considerações, e com base nos resultados observados e nas figuras apresentadas, é possível notar que as diferenças de desempenho multiclasse são relativamente pequenas entre o método F-NIDS e os demais métodos avaliados.



(a) Acurácia multiclasse.



(b) Acurácia binária.

Figura 5.1: Acurácia binária e multiclasse por rodada.

A Figura 5.1b apresenta os resultados de acurácia binária do classificador durante as dez rodadas de treinamento. Esses resultados são obtidos pelo agrupamento das classes de ataque em uma única classe. Nesse caso, é possível observar que a acurácia da detecção entre o tráfego normal e o mal-intencionado apresenta diferenças ainda menores. O impacto sobre a acurácia do F-NIDS, em relação a outros métodos, pode ser medido pela diferença entre as acurácias dos outros métodos e a acurácia do F-NIDS. Nesse caso, verificou-se que o impacto na acurácia binária média dos métodos ANN, ANN-DP e FED, em relação ao método F-NIDS, é de apenas 1,2%, 1,1% e 1,3%, respectivamente. Portanto, não foram observadas alterações relevantes no desempenho entre os quatro métodos nesse contexto. Esses resultados nos permitem

concluir que as perdas na acurácia binária, decorrentes da aplicação dos algoritmos `FedAvg` e `DP-SGD` juntos, resultam em impactos menores na acurácia binária do modelo resultante.

5.1.2 Desempenho do F-NIDS por avaliação de classe

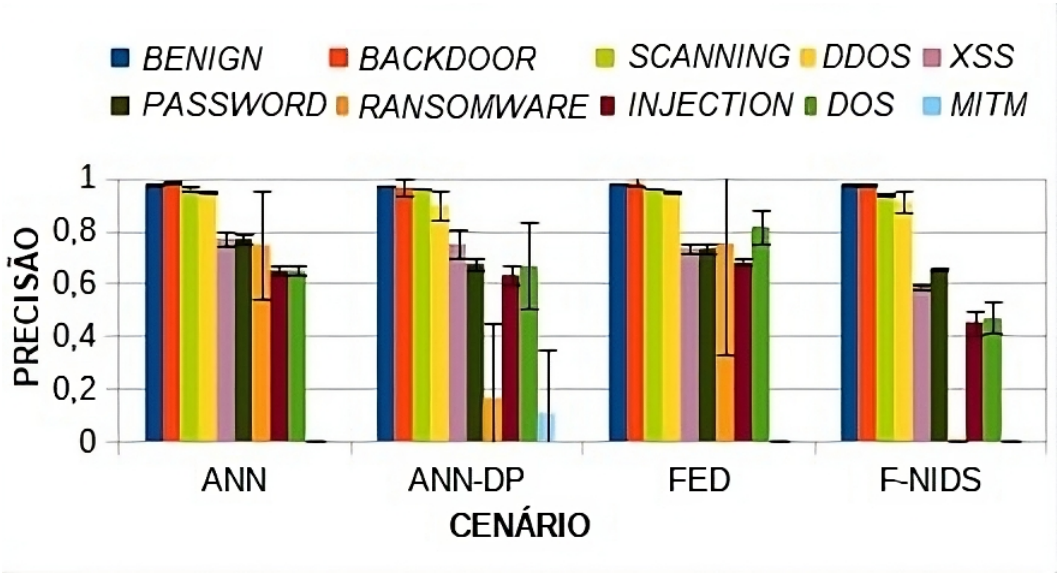
Na Figura 5.2, a precisão e o *recall* obtidos em cada classe usando o método de classificação F-NIDS são apresentados e comparados com os métodos ANN, ANN-DP e FED. A Figura 5.2a apresenta a precisão, ou proporção de acertos entre as previsões feitas, demonstrando a capacidade do modelo de classificar corretamente uma determinada classe, dentre as dez possíveis. A Figura 5.2b apresenta a *recall* de cada classe, separada pelos quatro métodos em estudo. Essa métrica é a proporção de acertos entre as classes reais. Para o F-NIDS em relação aos demais métodos, a precisão das previsões de *Benign, Backdoor, Scanning e DDOS* não mostra diferenças significativas do F-NIDS em relação aos outros métodos. Já especificamente nas classes *(XSS, Password, Ransomware, Injection, DDOS e MITM)*, podem ser observadas diferenças notáveis do F-NIDS para os demais.

É possível notar que houve uma diminuição acentuada na precisão de classificação nas classes *Ransomware e MITMT* no método F-NIDS. Aparentemente, esse comportamento está associado primeiramente ao número de exemplos disponíveis para essas duas classes, que é menor do que o tamanho do minilote escolhido. Em segundo lugar, a perturbação e o *clipping* do algoritmo *DP-SGD*, aliado ao aprendizado federado, ressalta ainda mais o efeito de baixa precisão para classes com poucos exemplos de treino. No método F-NIDS, as classes *DDOS* e *Injection* tiveram desempenho significativamente inferior ao dos outros métodos, embora tivessem um número de exemplos de treinamento comparativamente próximo ao das classes com melhor precisão. Considerando todas as classes de uma maneira ampla, os resultados nos permitem concluir que a aplicação do algoritmo `FedAvg`, em conjunto com o `DP-SGD`, apresentou um pequeno impacto na precisão geral e na *recall* de várias classes. Sendo que esse impacto se deu principalmente pela baixa quantidade de exemplos em algumas classes, consequência do *dataset* original não ser identicamente distribuído.

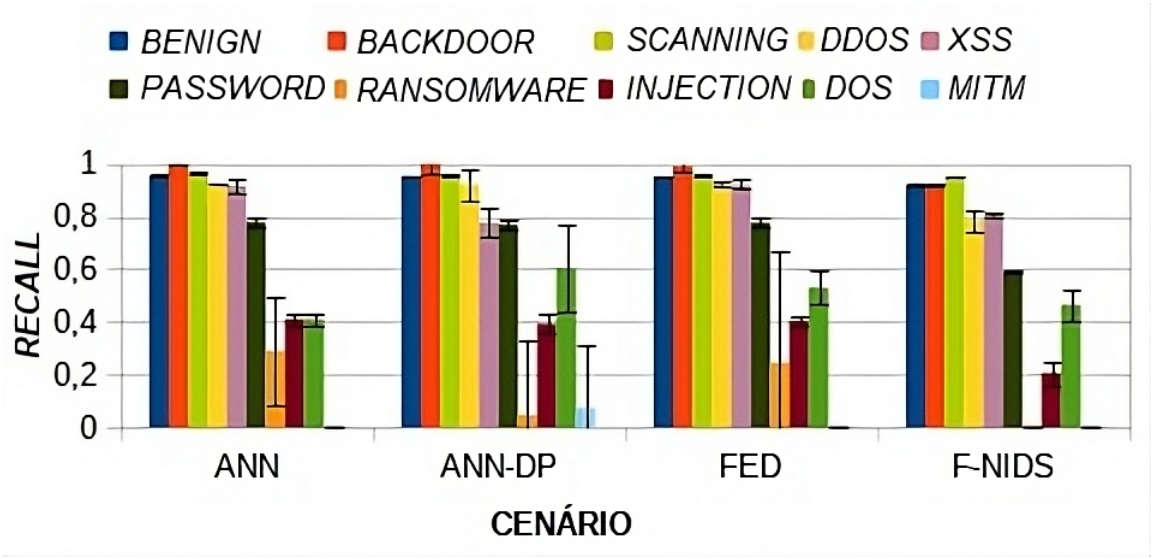
5.1.3 Eficiência binária do F-NIDS

A avaliação dos resultados por classe nos permite registrar a eficiência dos classificadores na detecção de cada classe individualmente, mas prever se um determinado pacote será provavelmente classificado corretamente como benigno ou malicioso é o objetivo primário do F-NIDS. Assim, as classes de ataque foram agrupadas em apenas uma classe, denominada *Attack*, resultando em duas classificações possíveis (ou seja, benigno ou ataque).

A Figura 5.3 apresenta as matrizes de confusão para cada um dos métodos, mostrando a interseção entre os exemplos classificados como benignos ou de ataque e seus valores reais correspondentes. O método F-NIDS (Figura 5.3d) obteve um resultado semelhante aos demais na detecção correta do tráfego benigno, o que também contribuiu para manter uma quantidade reduzida de falsos positivos. Na detecção de ataques, o F-NIDS apresentou uma quantidade de detecção apenas ligeiramente inferior à dos outros métodos, representada pelas Figuras 5.3a, 5.3b e 5.3c. Além disso, os ataques verdadeiros classificados incorretamente como benignos pelo método F-NIDS tiveram uma pequena diferença em relação aos outros



(a) Precisão

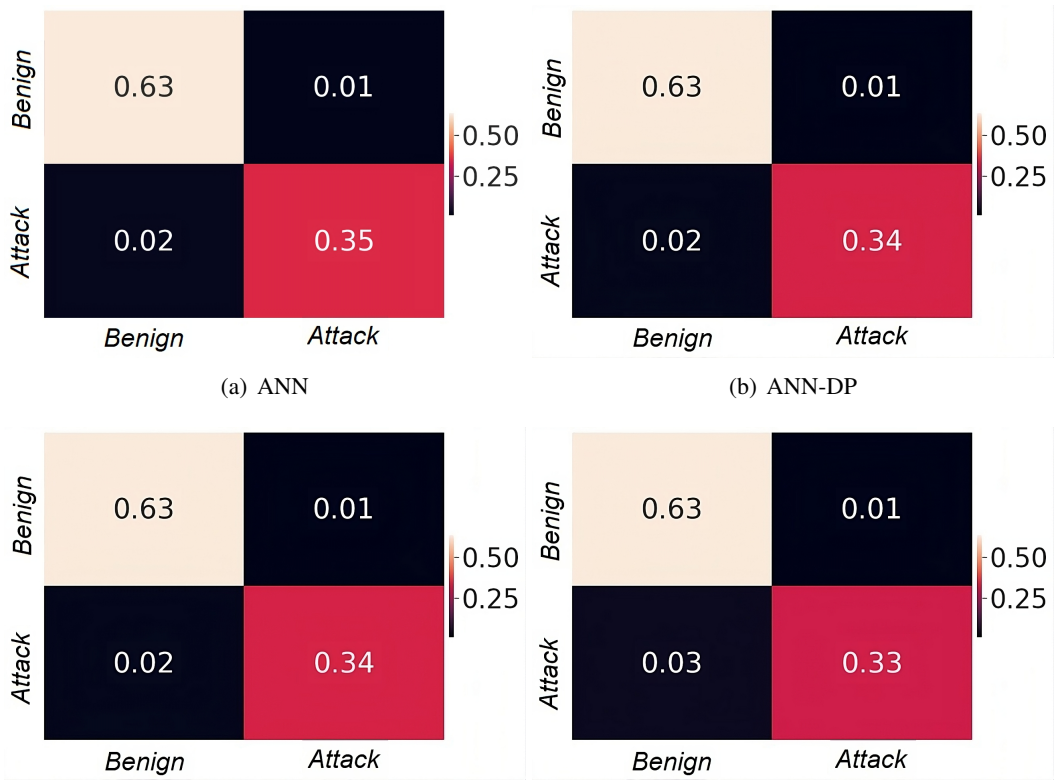


(b) *Recall*

Figura 5.2: Resultados de precisão e *recall* multiclasse.

métodos. Esses resultados demonstram que o método F-NIDS é quase tão eficiente quanto os métodos ANN (Figura 5.3a), ANN-DP (Figura 5.3b) e FED (Figura 5.3c) para detectar pacotes benignos e quase tão eficiente para classificar corretamente os ataques quanto as demais abordagens mencionadas. Tal resultado está associado a quantidade de classes de ataque, que somadas possuem exemplos em número suficiente para treinar classificadores precisos para detecção de ataques, tanto para os métodos centralizados quanto os federados. A Tabela 5.1 apresenta numericamente os resultados das métricas obtidas. Ela contém as médias das dez observações e os erros representados por seu desvio padrão.

Ao se analisar os efeitos do algoritmo `DP-SGD` no desempenho da classificação de ataques, comparando o método ANN (Figura 5.3a) com ANN-DP (Figura 5.3b), pode-se observar que esse algoritmo influencia ligeiramente a capacidade de detectar ataques verdadeiros. Considerando todas as possibilida-



(a) ANN (b) ANN-DP

(c) FED (d) F-NIDS

Figura 5.3: Matriz de confusão binária dos métodos de detecção.

Tabela 5.1: Avaliação das métricas de classificação binária dos métodos. Cada resultado é seguido por seu desvio padrão $\pm$.

Método	**Precisão**	***Recall***	**Acurácia**
ANN	$0{,}975 \pm 0{,}005$	$0{,}955 \pm 0{,}005$	$0{,}974 \pm 0$
ANN-DP	$0{,}971 \pm 0{,}002$	$0{,}954 \pm 0{,}003$	$0{,}973 \pm 0{,}001$
FED	$0{,}981 \pm 0$	$0{,}980 \pm 0$	$0{,}974 \pm 0$
F-NIDS	$0{,}885 \pm 0{,}28$	$0{,}970 \pm 0{,}264$	$0{,}962 \pm 0$

des, pode-se concluir que o algoritmo DP-SGD, com os hiperparâmetros de treinamento resulta, resulta em penalidades insignificantes sobre a eficácia binária do classificador. Comportamento semelhante foi verificado ao avaliar o efeito do algoritmo `FedAvg` no classificador, comparando os métodos ANN (Figura 5.3a) e FED (Figura 5.3c). No entanto, verificou-se que a utilização de FL, com o algoritmo de agregação `FedAvg`, juntamente com `DP-SGD` levou a uma diminuição na capacidade de detectar ataques reais, resultando em uma taxa maior de falsos negativos no caso do método F-NIDS.

Os resultados experimentais demonstraram ser possível construir um sistema de detecção de intrusão federado com privacidade diferencial, indicando que métricas de desempenho mais altas poderiam ser obtidas reduzindo o nível de ruído gaussiano ou diminuindo o número de DAs usados. No entanto, é importante observar que a implementação dessas reduções pode comprometer a robustez do sistema contra a inferência de membros ou ataques de inversão de modelo. Níveis mais baixos de ruído podem enfraquecer a robustez do sistema a cenários hipotéticos de roubo de modelos dos agentes, enquanto a redução do número de agentes de detecção pode torná-lo vulnerável ao roubo de amostras, comprometendo a privacidade

do sistema ao inferir o conjunto de dados de treinamento.

Além disso, é essencial manter um modelo de classificação útil para previsões precisas. Portanto, é fundamental encontrar um equilíbrio entre desempenho e robustez, sendo que as necessidades da organização é o principal fator a ser considerado ao usar o F-NIDS como uma ferramenta de detecção de intrusão. Por exemplo, em algumas determinadas aplicações médicas, na qual a privacidade é muito importante e as previsões devem ser revisadas por profissionais treinados, níveis mais baixos de precisão e acurácia podem ser aceitáveis. Por outro lado, em aplicativos não críticos para a privacidade, no qual a alta taxa de acurácia talvez seja crucial, como o reconhecimento facial, por exemplo, a privacidade dos dados pode não ser a principal preocupação. Nesse caso, níveis mais baixos de ruído podem ser aplicáveis para obter maior acurácia. Assim, o projetista do sistema deve avaliar cuidadosamente as necessidades e prioridades específicas da organização e do sistema pretendido ao determinar o balanceamento adequado entre desempenho e robustez para o F-NIDS. Nas seções a seguir, foi realizada uma avaliação de vulnerabilidade com ataques simulados para verificar qual configuração do F-NIDS produz detecções mais úteis, ao passo que permanece robusta contra ameaças à confidencialidade.

5.2 AVALIAÇÃO DE ROBUSTEZ DO F-NIDS

Essa Seção se dedicará a apresentar e discutir a robustez do F-NIDS usando diferentes configurações. Tais avaliações visam demonstrar quais os hiperparametros de privacidade que permitem ao F-NIDS manter um alto desempenho de detecção e, ao mesmo tempo, robusto contra os ataques que visam comprometer a privacidade dos dados de treino num contexto de ML. Nas Seções 5.2.1 e 5.2.2 a robustez do F-NIDS é testada com relação aos ataques de inferência de membro baseado em regra e em caixa preta, respectivamente. Já na Seção 5.2.3 a robustez do sistema é testada e avaliada mediante a um ataque de inversão de modelo. Os critérios para seleção desses ataques seguiu o descrito em [94], onde se coloca tais ataques à privacidade, no contexto de ML, como tendo a mais vasta documentação disponível, além de possuírem reprodução relativamente simples, segundo o exposto em [94].

5.2.1 Robustez contra ataques de inferência de membros baseados em regra

Ataques de inferência de membros baseados em regras envolvem a inferência dos dados de um indivíduo em um conjunto de dados confidenciais usando conhecimento externo prévio de algumas propriedades do modelo e dos dados de treino. Os invasores usam informações de segundo plano para criar regras permitem inferir dados [95]. Ao aplicar essas inferências aos atributos de um indivíduo, os invasores determinam se esse indivíduo estava ou não no conjunto de dados de treino, mesmo sem acesso direto ao conjunto de dados [96]. Esses ataques representam riscos à privacidade na medida que conseguem detectar a existência de um determinado exemplo em especial no conjunto original [59]. Como a técnica FL envolve a distribuição de pesos globais em vários locais ou dispositivos, cada modelo específico pode ser uma fonte de conhecimento a ser usada por um agente mal-intencionado hipotético.

Treinar o modelo usando o algoritmo `DP-SGD` com um hiperparâmetro de ruído gaussiano pode tornar o sistema robusto contra esses ataques. No entanto, como visto nas seções anteriores desse capítulo,

dependendo do ruído aplicado, isso acarreta custos de desempenho de classificação. Portanto, o equilíbrio entre robustez e desempenho é necessário para o treinamento efetivo do modelo. Tendo isso em mente, nove grupos de instâncias do F-NIDS contendo valores diferentes para o ruído σ foram treinados, visando selecionar o que apresenta melhor desempenho no ataque, produzindo o desempenho de classificação mais útil possível. Cada um desses detectores foi então submetido a ataques de inferência de membros baseados em regra. O grupo que apresentar melhor robustez contra o ataque, com o menor custo de desempenho de classificação, terá o nível de ruído relacionado selecionado. No entanto, é necessário estabelecer um critério baseado no desempenho e na robustez.

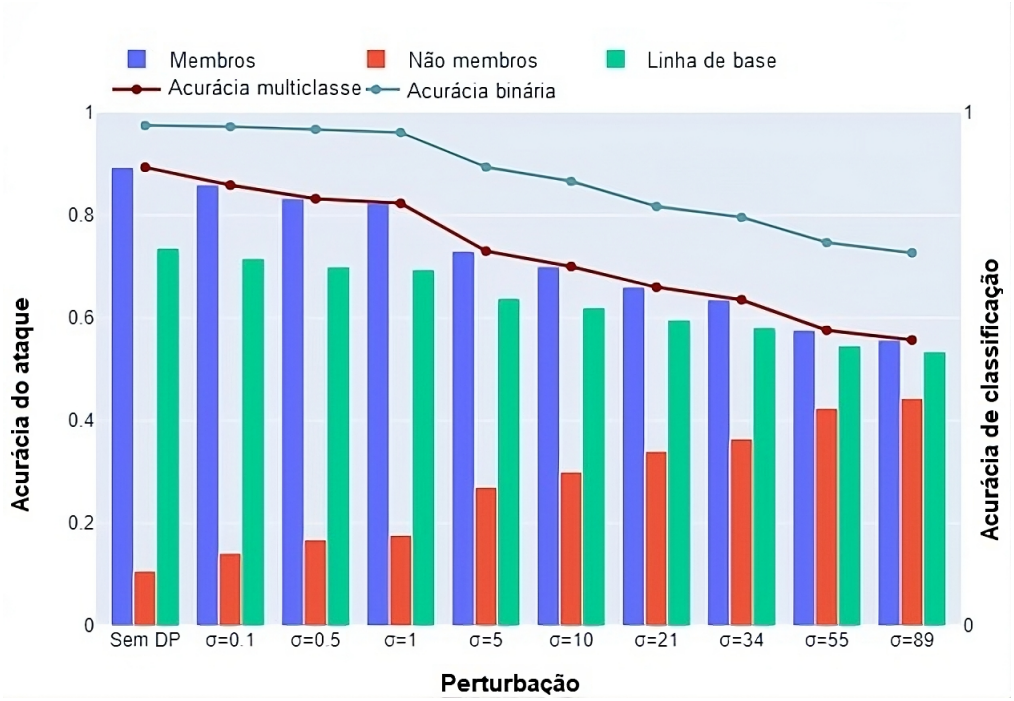
Com base nessas considerações, a acurácia é a medida de desempenho principal usada para avaliar a classificação e a capacidade de resistência do modelo. Além disso, a precisão e o *recall* do ataque foram avaliados, pois são medidas cruciais para determinar a capacidade do ataque em distinguir entre membros e não membros. O objetivo é manter a acurácia alta de classificação, enquanto se assegura a segurança do modelo, então é necessário definir valores para cada métrica que equilibrem o desempenho e a segurança da melhor maneira possível, do ponto de vista observacional. Para este trabalho, foi estabelecido que o melhor hiperparâmetro gaussiano será aquele em que a instância correspondente tenha uma acurácia binária superior a 80% e uma acurácia multiclasse superior a 60%. Além disso, a acurácia de inferência para membros reais deve ser inferior a 65%, e para não membros, inferior a 40%, resultando em uma linha de base média para o desempenho dos ataques de inferência inferior a 60%.

Nesse teste, foi usado o módulo `MembershipInferenceRuleBase` da biblioteca ART (*Adversarial Robustness Toolbox* — Caixa de Ferramentas de Robustez) [6]. Para produzir os resultados, foram realizados vinte ciclos independentes de avaliação de robustez. Para esse experimento, é necessário definir quais serão os dados de membros e os de não membros e então rotular os dados entre membros e não membros. Nesse caso, serão rotulados como membros os próprios dados usados para o treinamento do modelo original. Já o conjunto de dados de teste será rotulado como não membro, pois não foram usados para treinamento do modelo original. Após definidos os rótulos, esses conjuntos serão divididos pelo número de ciclos que será executado. Como serão vinte ciclos independentes, então os nove modelos foram testados com um conjunto contendo uma fração de 5% dos exemplos de dados de treinamento e teste.

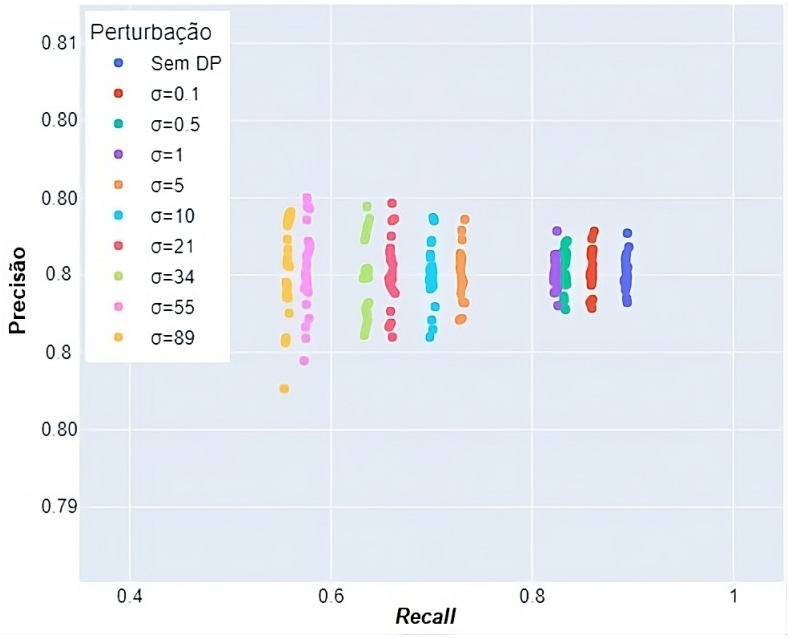
A Figura 5.4a apresenta os resultados de acurácia de um ataque de inferência de membros, juntamente com a acurácia de classificação do modelo original. Nessa figura, é possível observar três comportamentos principais. O primeiro é que, quanto maior for o valor do ruído gaussiano, menor será a linha de base do ataque obtido e a probabilidade de sucesso na inferência de não membros. Em segundo lugar, há um aumento na capacidade de detectar não membros à medida que o valor de σ aumenta, mas sem ultrapassar 50% de acurácia. O último comportamento notável diz respeito à acurácia da detecção do modelo. No entanto, essa acurácia permanece significativamente maior do que os valores de acurácia da linha de base do ataque, especialmente para a utilidade da classificação binária, mantendo ainda o equilíbrio entre utilidade de classificação e robustez. Baseando-se nesses resultados, o valor de ruído escolhido para o F-NIDS é $\sigma = 21$, pois ele pode atingir os critérios estabelecidos de robustez contra a inferência de membros e ainda assim manter a acurácia de detecção de intrusões acima de 80%. A tabela 5.2 apresenta alguns resultados relevantes do teste realizado, como a linha de base do ataque, a acurácia do ataque de inferência, a precisão da classificação binária e *F1 score* obtida pelo ataque.

[6] <https://adversarial-robustness-toolbox.org/>

Outro indicador importante é apresentado na Figura 5.4b, que contém a precisão em função do *recall* obtidos dos resultados do ataque. Observa-se que a precisão do resultado do ataque não é significativamente afetada pelo aumento do nível de ruído. No entanto, o *recall* apresentou uma significativa diminuição à medida que o nível de ruído aumentou. Os resultados mostram que, embora a probabilidade de um não membro ser considerado membro não seja significativamente afetada por σ, a probabilidade de um membro real ser considerado não membro tem um aumento considerável à medida que o nível de ruído aumenta.



(a) Performances de classificação e robustez



(b) Precisão e *recall* do ataque

Figura 5.4: Análise de acurácia e robustez para uma inferência de membros baseada em regra.

Tabela 5.2: Desempenho do ataque adversário baseado em regras.

σ	**L. Base**	**Acr. adv membros**	**Acr. binária**	**F1 Score adv**
0,1	0,72 ± 0,0010	0,86 ± 0,0010	0,97 ± 0,0010	0,83 ± 0,0007
1	0,69 ± 0,0008	0,82 ± 0,0011	0,96 ± 0,0014	0,81 ± 0,0006
21	0,60 ± 0,0013	0,66 ± 0,0015	0,82 ± 0,0029	0,72 ± 0,0012
55	0,55 ± 0,0016	0,58 ± 0,0016	0,75 ± 0,0030	0,71 ± 0,0012
89	0,53 ± 0,0016	0,56 ± 0,0015	0,73 ± 0,0030	0,66 ± 0,0013

5.2.2 Robustez contra ataques usando um modelo adversário

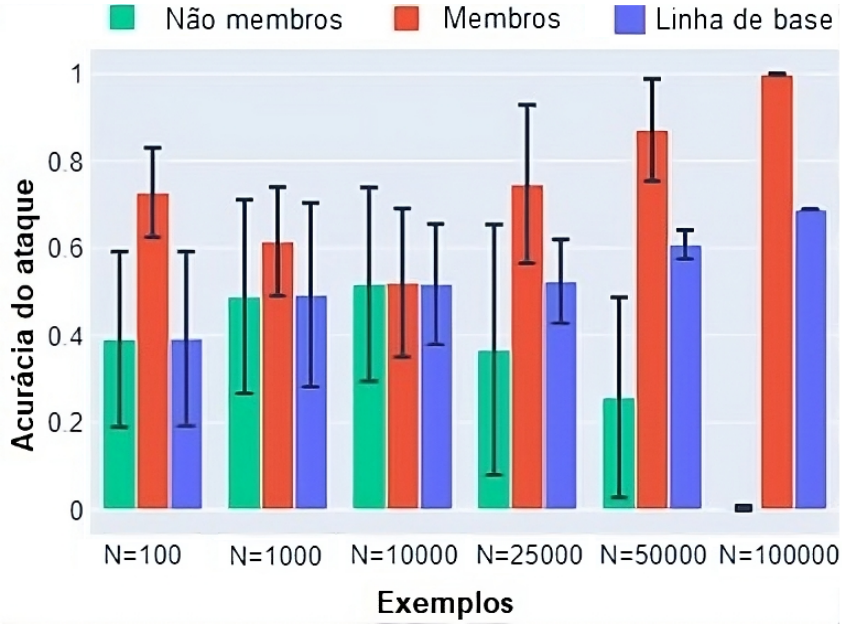
Esse cenário de ataque é um tipo de inferência de membros que pode ser feito usando apenas algumas amostras de conjunto de dados e conhecimento prévio sobre a arquitetura do modelo. Um novo tipo de modelo pode ser treinado e, em seguida, transformado em um modelo adversário, chamado *ShadowModel*, capaz de funcionar como um preditor dos membros pertencentes ao conjunto original, comprometendo a privacidade de todo o conjunto de dados.

Embora o classificador usado pelo F-NIDS seja protegido com privacidade diferencial, se uma amostra de tamanho excessivo for armazenada em alguns dos DAs, isso ainda pode representar um risco a ser considerado. Esse problema ocorre porque, dependendo do seu tamanho, uma amostra pode conter propriedades estatísticas semelhantes às do conjunto de dados original. O aprendizado federado, usado no F-NIDS, desempenha um papel essencial ao permitir a divisão de grandes volumes de dados em amostras menores distribuídas em um número maior de agentes, fazendo com que frações menores de dados confidenciais sejam armazenadas com maior confidencialidade. No entanto, ainda é necessário investigar o número máximo de exemplos de treinamento que podem ser armazenados nos DAs que melhor minimiza o risco de a amostra armazenada ser usada para treinar um *ShadowModel* preciso e acurado.

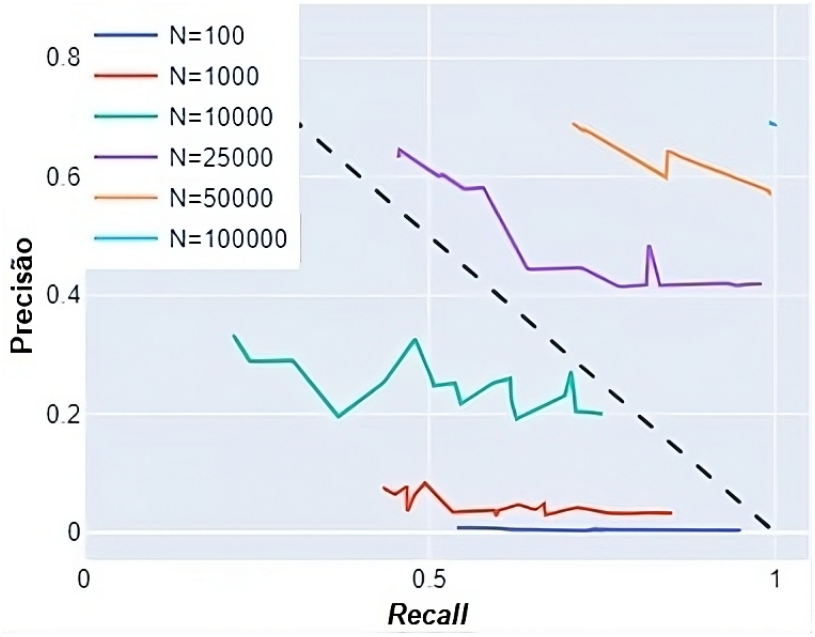
Para essa investigação, seis amostras do conjunto de dados de treinamento, de tamanhos distintos, foram selecionadas e usadas em um ataque adversário do *Membership Inference Black Box* — Inferência de Membros em Caixa Preta —, também contido na biblioteca ART. Nesse ataque, seis classificadores foram treinados, cada um com sua respectiva amostra, e o desempenho das métricas de acurácia de inferência de membro e não membro foi avaliado, bem como a precisão e a *recall* obtidas nos ataques. Cada uma das amostras mencionadas tem um tamanho N de exemplos de membros usados para treinar o *ShadowModel*. Esse processo foi repetido em 20 ciclos independentes e os resultados são mostrados na Tabela 5.3. Cada modelo adversário foi testado com $100K$ de exemplos de membros e $25K$ de exemplos de não membros.

A Figura 5.5a apresenta a eficácia dos ataques obtidos por classificadores treinados com diferentes valores de N. O eixo vertical mostra a porcentagem de sucesso do ataque e o horizontal, os resultados obtidos. É possível observar que a eficácia na detecção de membros e não membros diminui à medida que o valor de N diminui. Quando esse valor de $N = 100$ é atingido, ele coincide com o valor mais baixo registrado para a linha de base da precisão do ataque, pois os valores de precisão da detecção para membros e não membros permaneceram abaixo de 40%. Entretanto, um classificador treinado com um tamanho de exemplo tão baixo provavelmente não seria muito útil para a detecção de intrusões. Portanto, um número de exemplos de treinamento $10K \leq N \leq 25K$ atinge o nível razoavelmente aceitável para os três indicadores presentes na figura, em que ambos os indicadores têm valores próximos a 50% de eficiência de ataque. A

Tabela 5.3 apresenta os indicadores obtidos, incluindo a linha de base, a precisão da inferência de membros e o *F1 score* do ataque.



(a) Acurácia do ataque para cada *ShadowModel*



(b) Precisão e *Recall* curves for each *ShadowModel*

Figura 5.5: Análise de acurácia e robustez num cenário de ataque de inferência de membros.

Na figura 5.5b, a precisão e o *recall* obtidos nos resultados do ataque, foram comparados. Por meio do gráfico, é possível concluir que a precisão tem um aumento significativo à medida que o número de exemplos de treinamento é aumentado. Isso indica que quanto maior o número de exemplos capturados por um agente malicioso, maior a chance de construir um modelo capaz de inferir membros. No entanto, o *recall* não mostrou grande sensibilidade ao número de exemplos usados no treinamento do *ShadowModel*, influenciando pouco a capacidade desse modelo de categorizar corretamente os não membros.

Tabela 5.3: Indicadores de desempenho do ataque usando *ShadowModel*s.

N	L. Base	Acr. adv. membro	F1 Scr. adv.
100000	0,68 ± 0,00	0,99 ± 0,002	0,81 ± 0,001
50000	0,60 ± 0,03	0,87 ± 0,12	0,72 ± 0,06
25000	0,52 ± 0,1	0,74 ± 0,2	0,58 ± 0,11
10000	0,51 ± 0,13	0,52 ± 0,16	0,33 ± 0,07
1000	0,49 ± 0,21	0,61 ± 0,1	0,08 ± 0,03
100	0,39 ± 0,19	0,72 ± 0,1	0,010 ± 0,003

5.2.3 Robustez contra ataques generativos usando inversão de modelos

O último teste de robustez realizado no classificador F-NIDS é o MI (*Model Inversion* — Inversão de Modelo). Os ataques MI são ataques à privacidade em que um adversário tenta reconstruir informações confidenciais sobre indivíduos explorando as saídas de um modelo de aprendizado de máquina. Nesses ataques, o adversário usa consultas ao modelo, juntamente com o conhecimento obtidos à priori, para inferir atributos privados ou pontos de dados usados para gerar as previsões do modelo [97]. Ao consultar estrategicamente o modelo e analisar suas respostas, o invasor pode fazer engenharia reversa de informações confidenciais sobre indivíduos, possivelmente violando sua privacidade [98]. Os ataques de inversão de modelo destacam a importância de proteger informações confidenciais e considerar técnicas de preservação da privacidade ao desenvolver e implantar modelos de aprendizado de máquina [99].

Em um cenário prático, um DA precisa ser invadido e seu modelo extraído sem autorização. O agente mal-intencionado então usará as propriedades estatísticas existentes nos gradientes para gerar amostras que podem conter exemplos originais de outros DAs. Por fim, essas amostras geradas serão submetidas a avaliação de inferência de membros para obter os exemplos prováveis de serem pertencentes ao conjunto original.

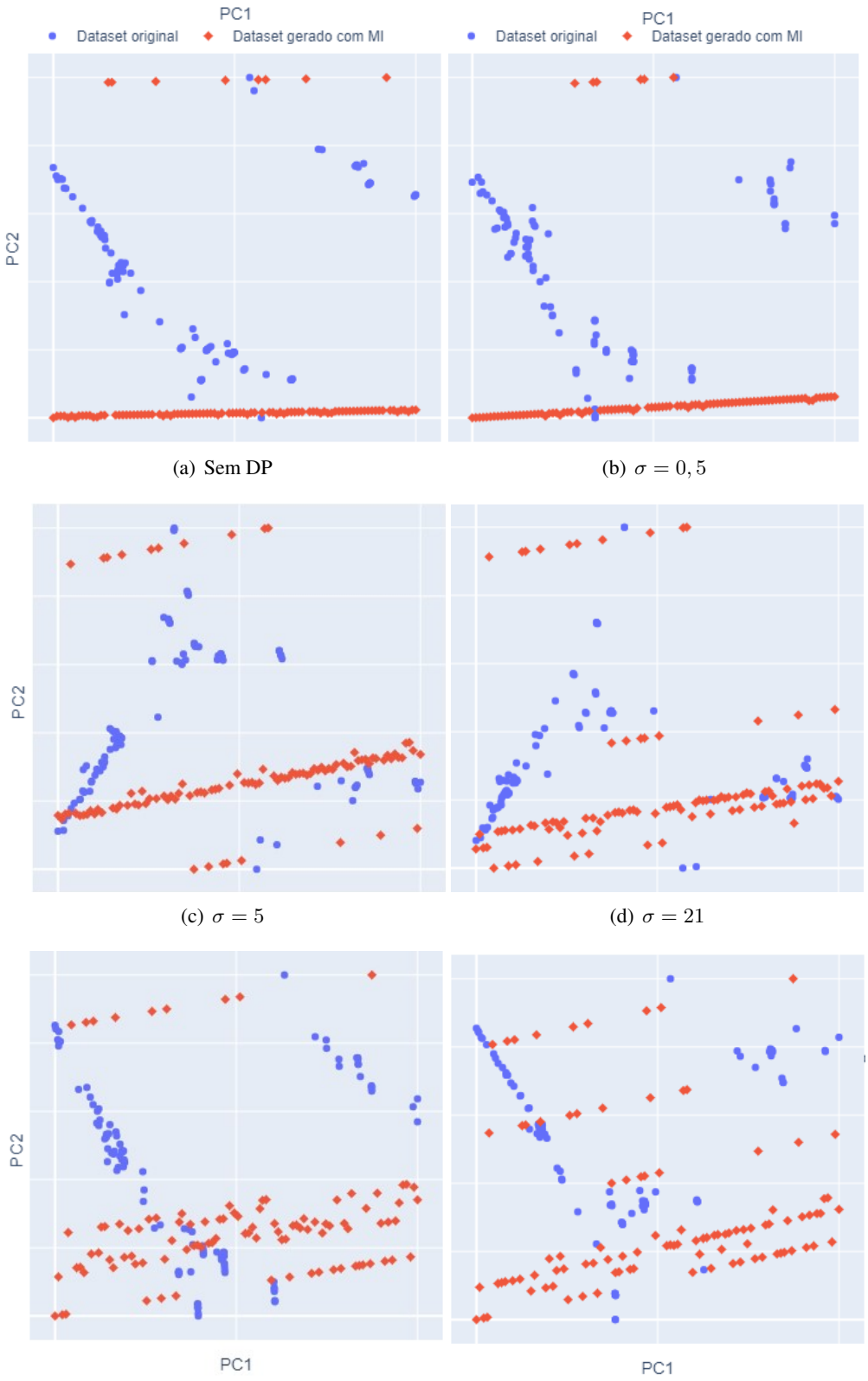
Por meio do teste, foi possível obter uma grande diferença nos dados gerados à medida que o nível de ruído no modelo de classificação original é aumentado. O algoritmo MI utilizado foi o `MIFace` [97], disponível na biblioteca ART. Para validar o teste, cem exemplos foram extraídos dos modelos, treinados com diferentes níveis de ruído. Os modelos usados nesse teste são os mesmos usados no teste de inferência de membros, já discutido na seção 5.2.1. Para cada um dos conjuntos de dados gerados, um conjunto original contendo a mesma quantidade de exemplos e uma distribuição de classe semelhante foi selecionado aleatoriamente. Ambos os conjuntos de dados tiveram sua dimensionalidade reduzida a dois componentes principais, usando a técnica de redução de dimensionalidade PCA (*Principal Component Analysis* — Análise de Componentes Principais), visando obter uma representação gráfica do efeito do ruído gaussiano nos conjuntos de dados.

A Figura 5.6 apresenta graficamente os resultados dos conjuntos de dados originais e adversários, gerados pelos seis modelos treinados usando diferentes níveis de ruído, com sua dimensionalidade reduzida a dois componentes principais. Como pode ser visto na Figura 5.6a, sem o uso do algoritmo DP no modelo de destino, o conjunto de dados gerado pelo MI mostra um padrão relativamente semelhante ao original, em que os pontos estão localizados em dois conjuntos próximos e com distribuição semelhante. Outro aspecto importante é que é possível traçar o mesmo limite de decisão para distinguir duas regiões em ambos os conjuntos de dados. À medida que o nível de ruído aumenta, é possível observar que os exemplos gerados

mostram um padrão cada vez mais distinto do conjunto de dados original. Assim, os exemplos recriados começam a mostrar um padrão de dispersão muito distinto do original, especialmente quando $\sigma = 89$, conforme a Figura 5.6f. Isso leva à conclusão de que, em um conjunto de dados produzido a partir de um *ShadowModel*, com um nível mais alto de ruído, torna-se, por exemplo, mais complexo para algum algoritmo de classificação traçar um limite de decisão que se ajuste a ambos os conjuntos de dados. Outra questão a ser observada é que, com $\sigma \geq 5$ presente na Figura 5.6c, não é mais possível desenhar a mesma região de decisão que pode separar linearmente os conjuntos de dados de treinamento e os adversários em dois limites de decisão com distribuição semelhante. Isso leva à conclusão de que os conjuntos de dados de treinamento e adversários tornam-se muito diferentes um do outro quando o ruído gaussiano é aumentado para cinco ou mais no modelo-alvo. Portanto, os dados adversários tornam-se inadequados para o treinamento de um modelo adversário capaz de adequadamente produzir amostras confidenciais do conjunto de dados de treino original.

5.3 CONSIDERAÇÕES FINAIS

No presente capítulo os resultados de avaliação de desempenho foram apresentados e discutidos, bem como a metodologia que sustentou os testes. O capítulo também foi usado para se estabelecer os hiperparâmetros ótimos de funcionamento para privilegiar a confidencialidade dos dados, sem sacrificar excessivamente o desempenho. É importante destacar que o foco foi na avaliação do desempenho de classificação e de robustez. Demais avaliações de desempenho com relação a métricas de desempenho em rede podem ser realizadas futuramente em outros trabalhos. No capítulo seguinte será feita a conclusão, discussões e o escopo de possíveis trabalhos futuros que esse trabalho possa fundamentar.



(a) Sem DP (b) $\sigma = 0,5$

(c) $\sigma = 5$ (d) $\sigma = 21$

(e) $\sigma = 55$ (f) $\sigma = 89$

Figura 5.6: Dimensões reduzidas para dois componentes dos conjuntos de dados originais e adversários.

6 CONCLUSÃO E TRABALHOS FUTUROS

Essa pesquisa mostrou que um dos principais desafios da IoT é a detecção de intrusão descentralizada e dimensionável nessas redes. Apesar das tentativas de obter detecções precisas em um ambiente de IoT, ainda havia possibilidade de melhoria em termos de escalabilidade, confidencialidade e confiabilidade. Os resultados mostraram que, com valores mais baixos de ruído gaussiano, o F-NIDS tem métricas de acurácia, precisão e *recall* semelhantes às estratégias tradicionais que usam apenas aprendizado centralizado. O F-NIDS foi projetado levando em consideração as necessidades e características de IoT, tendo esse paradigma balizado as decisões de projeto do sistema. Além disso, é projetado para ser executado como uma solução SaaS (*Software as a Service — Software* disponibilizado como um serviço através da rede). Além disso, a solução deve ser independente do setor, podendo funcionar em várias redes de IoT. No entanto, recomenda-se enfaticamente o uso do sistema em ambientes críticos para a privacidade, como os domínios médico e financeiro. Pois esses são setores em que pode ser extremamente vantajoso renunciar a uma fração de acurácia em favor da privacidade dos dados e ainda podendo usufruir de um desempenho de classificação aceitável.

Além disso, os resultados nos permitem definir boas configurações para dois hiperparâmetros que são muito importantes para garantir a confidencialidade e, ao mesmo tempo, manter o desempenho da classificação. Esses hiperparâmetros são o ruído gaussiano σ e o tamanho da amostra de treinamento do agente de detecção N. Os testes mostraram que $\sigma = 21$ protege contra ataques baseados em regras de caixa-preta de inferência de membros e ataques de inversão de modelo. Manter o tamanho da amostra, armazenado nos agentes de detecção, entre $N = 10K$ e $N = 25K$ ajuda a proteger a confidencialidade dos agentes de detecção, no caso de ataques inferência de membros em caixa-preta convencionais, por exemplo, se algum agente tiver sua confidencialidade comprometida, outros agentes de detecção podem manter a privacidade de seus próprios dados de treinamento garantida.

6.1 TRABALHOS FUTUROS

Embora as métricas usadas nos resultados do F-NIDS sejam bastante semelhantes às de outros trabalhos relacionados para verificar o desempenho da classificação, em trabalhos futuros, outros tipos de métricas serão investigados, como entropia e efeito avalanche. A escalabilidade e a disponibilidade do sistema também serão investigadas com métricas apropriadas, como indicadores de carga e estresse, tempos de resposta e taxa de erro. Embora a confidencialidade seja um pilar muito importante da segurança das informações, a integridade e a disponibilidade dos dados também são fatores igualmente importantes a serem considerados.

Nesse sentido, o F-NIDS precisa ter métricas que abordem também esses outros pilares. Tendo isso em mente, esse tipo de cenário será abordado em trabalhos futuros. Em adição, outros testes de aprendizado de máquina adversária podem ser realizados em modelos de classificação para medir sua robustez contra outros tipos de ataque de ML adversária. As outras ameaças de adversários que podem comprometer o

modelo de classificação do sistema são evasão e *poisoning*. Esses ataques estão relacionados aos outros pilares de segurança da informação. Portanto, a realização de medições e robustez podem ajudar a melhorar a segurança não apenas em termos de confidencialidade, mas também de integridade e disponibilidade dos dados.

REFERÊNCIAS BIBLIOGRÁFICAS

1 KODERA, S.; AKAZAWA, H.; MORITA, H.; KOMURO, I. Prospects for cardiovascular medicine using artificial intelligence. *Journal of Cardiology*, v. 79, n. 3, p. 319–325, 2022. ISSN 0914-5087. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0914508721002835>.

2 PANDIYAN, V.; SHEVCHIK, S.; WASMER, K.; CASTAGNE, S.; TJAHJOWIDODO, T. Modelling and monitoring of abrasive finishing processes using artificial intelligence techniques: A review. *Journal of Manufacturing Processes*, v. 57, p. 114–135, 2020. ISSN 1526-6125. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1526612520303923>.

3 GERON, A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd. ed. [S.l.]: O'Reilly Media, Inc., 2019. ISBN 1492032646.

4 GONçALVES, A. R. *Máquina de Vetores Suporte (Unicamp)*. Unicamp, 2009. Disponível em: <https://andreric.github.io/files/pdfs/svm.pdf>.

5 WILTGEN, F. *DESENVOLVIMENTO DE UM SISTEMA INTELIGENTE EM TEMPO REAL PARA CONTROLAR O DESLOCAMENTO DO PLASMA NO TOKAMAK - ETE (EXPERIMENTO TOKAMAK ESFÉRICO) DEVELOPMENT OF AN INTELLIGENT REAL-TIME SYSTEM TO CONTROL PLASMA DISPLACEMENT IN THE TOKAMAK - ETE (SPHERAL TOKAMAK EXPERIMENT)*. Tese (Doutorado), 10 2003.

6 NG, A. *Lecture 2 | Machine Learning (Stanford)*. Stanford, 2008. Disponível em: <https://www.youtube.com/watch?v=5u4G23_OohI>.

7 NETO, H. C.; MENEZES, D.; FERNANDES, N. Privacidade do usuário em aprendizado colaborativo: Federated learning, da teoria à prática. In: ____. [S.l.: s.n.], 2020. ISBN 9786587003856.

8 WATKINS, W. M.; CHEN, S. Y.-C.; YOO, S. Quantum machine learning with differential privacy. *Scientific Reports*, v. 13, n. 1, p. 2453, Feb 2023. ISSN 2045-2322. Disponível em: <https://doi.org/10.1038/s41598-022-24082-z>.

9 TRUEX, S.; LIU, L.; GURSOY, M.; YU, L.; WEI, W. Demystifying membership inference attacks in machine learning as a service. *IEEE Transactions on Services Computing*, IEEE Computer Society, Los Alamitos, CA, USA, v. 14, n. 06, p. 2073–2089, nov 2021. ISSN 1939-1374.

10 MO, K.; HUANG, T.; XIANG, X. Querying little is enough: Model inversion attack via latent information. In: CHEN, X.; YAN, H.; YAN, Q.; ZHANG, X. (Ed.). *Machine Learning for Cyber Security*. Cham: Springer International Publishing, 2020. p. 583–591. ISBN 978-3-030-62460-6.

11 WHAT is an Intrusion Detection System? Https://www.paloaltonetworks.com/cyberpedia/what-is-an-intrusion-detection-system-ids.

12 ANGEL, N. A.; RAVINDRAN, D.; VINCENT, P. M. D. R.; SRINIVASAN, K.; HU, Y.-C. Recent advances in evolving computing paradigms: Cloud, edge, and fog technologies. *Sensors*, v. 22, n. 1, 2022. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/22/1/196>.

13 RAHMAN, M. A.; ASYHARI, A. T. The emergence of internet of things (iot): Connecting anything, anywhere. *Computers*, v. 8, n. 2, 2019. ISSN 2073-431X. Disponível em: <https://www.mdpi.com/2073-431X/8/2/40>.

14 FILHO, P. G.; VILLAS, L. A.; GONÇALVES, V. P.; PESSIN, G.; LOUREIRO, A. A.; UEYAMA, J. Energy-efficient smart home systems: Infrastructure and decision-making process. *Internet of Things*, Elsevier, v. 5, p. 153–167, 2019.

15 HABIBZADEH, H.; NUSSBAUM, B. H.; ANJOMSHOA, F.; KANTARCI, B.; SOYATA, T. A survey on cybersecurity, data privacy, and policy issues in cyber-physical system deployments in smart cities. *Sustainable Cities and Society*, v. 50, p. 101660, 2019. ISSN 2210-6707. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2210670718316883>.

16 NAYAK, J.; NAIK, B.; DASH, P. B.; VIMAL, S.; KADRY, S. Hybrid bayesian optimization hypertuned catboost approach for malicious access and anomaly detection in iot nomalyframework. *Sustainable Computing: Informatics and Systems*, v. 36, p. 100805, 2022. ISSN 2210-5379. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2210537922001366>.

17 ROMAN, R.; ZHOU, J.; LOPEZ, J. On the features and challenges of security and privacy in distributed internet of things. *Computer Networks*, v. 57, n. 10, p. 2266–2279, 2013. ISSN 1389-1286. Towards a Science of Cyber Security Security and Identity Architecture for the Future Internet. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128613000054>.

18 CAVALCANTE, I. C.; MENEGUETTE, R. I.; TORRES, R. H.; MANO, L. Y.; GONÇALVES, V. P.; UEYAMA, J.; PESSIN, G.; NZE, G. D. A.; FILHO, G. P. R. Federated system for transport mode detection. *Energies*, MDPI, v. 15, n. 23, p. 9256, 2022.

19 CHAABOUNI, N.; MOSBAH, M.; ZEMMARI, A.; SAUVIGNAC, C.; FARUKI, P. Network intrusion detection for iot security based on learning techniques. *IEEE Communications Surveys Tutorials*, v. 21, n. 3, p. 2671–2701, 2019.

20 BERTINO, E.; ISLAM, N. Botnets and internet of things security. *Computer*, v. 50, n. 2, p. 76–79, 2017.

21 RAHMAN, M. A.; ASYHARI, A. T.; LEONG, L.; SATRYA, G.; Hai Tao, M.; ZOLKIPLI, M. Scalable machine learning-based intrusion detection system for iot-enabled smart cities. *Sustainable Cities and Society*, v. 61, p. 102324, 2020. ISSN 2210-6707. Disponível em: <https://www.sciencedirect.com/science/article/pii/S221067072030545X>.

22 CABRERO-HOLGUERAS, J.; PASTRANA, S. Sok: Privacy-preserving computation techniques for deep learning. *Proceedings on Privacy Enhancing Technologies*, v. 2021, n. 4, p. 139–162, 2021. Disponível em: <https://doi.org/10.2478/popets-2021-0064>.

23 KUMAR, R. S. S.; NYSTRöM, M.; LAMBERT, J.; MARSHALL, A.; GOERTZEL, M.; COMISSONERU, A.; SWANN, M.; XIA, S. Adversarial machine learning-industry perspectives. In: *2020 IEEE Security and Privacy Workshops (SPW)*. [S.l.: s.n.], 2020. p. 69–75.

24 CHEN, H.; HUSSAIN, S. U.; BOEMER, F.; STAPF, E.; SADEGHI, A. R.; KOUSHANFAR, F.; CAMMAROTA, R. Developing privacy-preserving ai systems: The lessons learned. In: *2020 57th ACM/IEEE Design Automation Conference (DAC)*. [S.l.: s.n.], 2020. p. 1–4.

25 ZHU, H.; ZHANG, H.; JIN, Y. From federated learning to federated neural architecture search: a survey. *Complex & Intelligent Systems*, v. 7, 01 2021.

26 STERGIOU, C. L.; PSANNIS, K. E.; GUPTA, B. B. Infemo: Flexible big data management through a federated cloud system. *ACM Trans. Internet Technol.*, Association for Computing Machinery, New York, NY, USA, v. 22, n. 2, oct 2021. ISSN 1533-5399. Disponível em: <https://doi.org/10.1145/3426972>.

27 HOHPE, G.; WOOLF, B. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 0321200683.

28 PAUTASSO, C.; ZIMMERMANN, O.; LEYMANN, F. Restful web services vs. "big"' web services: Making the right architectural decision. In: *Proceedings of the 17th International Conference on World Wide Web*. New York, NY, USA: Association for Computing Machinery, 2008. (WWW '08), p. 805–814. ISBN 9781605580852. Disponível em: <https://doi.org/10.1145/1367497.1367606>.

29 FOWLER, M. *Patterns of Enterprise Application Architecture*. USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0321127420.

30 STALLINGS, W. *Network Security Essentials: Applications and Standards*. Pearson, 2017. ISBN 9780134527338. Disponível em: <https://books.google.com.br/books?id=HEWwDAEACAAJ>.

31 ROESCH, M. Snort - lightweight intrusion detection for networks. In: *Proceedings of the 13th USENIX Conference on System Administration*. USA: USENIX Association, 1999. (LISA '99), p. 229–238.

32 BACE, R. *Intrusion Detection*. Macmillan Technical Publishing, 2000. (Macmillan technology series). ISBN 9781578701858. Disponível em: <https://books.google.com.br/books?id=VLgVMlV476IC>.

33 AXELSSON, S. Intrusion detection systems: A survey and taxonomy. 04 2000.

34 SHI, J.; GE, B.; LIU, Y.; YAN, Y.; LI, S. Data privacy security guaranteed network intrusion detection system based on federated learning. In: *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. [S.l.: s.n.], 2021. p. 1–6.

35 SCHNEIER, B. *Data and Goliath: The Hidden Battles to Capture Your Data and Control Your World*. 1st. ed. [S.l.]: W. W. Norton & Company, 2015. ISBN 0393244814.

36 CARVALHO, A. P.; CARVALHO, F. P.; CANEDO, E. D.; CARVALHO, P. H. P. Big data, anonymisation and governance to personal data protection. In: *The 21st Annual International Conference on Digital Government Research*. New York, NY, USA: Association for Computing Machinery, 2020. (dg.o '20), p. 185–195. ISBN 9781450387910. Disponível em: <https://doi.org/10.1145/3396956.3398253>.

37 MYERS, J.; FRIEDEN, T.; BHERWANI, K.; HENNING, K. Ethics in public health research: Privacy and public health at risk: Public health confidentiality in the digital age. *American journal of public health*, v. 98, p. 793–801, 06 2008.

38 DWORK, C. Differential privacy. In: BUGLIESI, M.; PRENEEL, B.; SASSONE, V.; WEGENER, I. (Ed.). *Automata, Languages and Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 1–12. ISBN 978-3-540-35908-1.

39 MCSHERRY, F.; TALWAR, K. Mechanism design via differential privacy. In: . [S.l.: s.n.], 2007. p. 94–103. ISBN 978-0-7695-3010-9.

40 VISHWAKARMA, M.; KESSWANI, N. Dids: A deep neural network based real-time intrusion detection system for iot. *Decision Analytics Journal*, v. 5, p. 100142, 2022. ISSN 2772-6622. Disponível em: <https://www.sciencedirect.com/science/article/pii/S277266222200073X>.

41 FREDRIKSON, M.; JHA, S.; RISTENPART, T. Model inversion attacks that exploit confidence information and basic countermeasures. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2015. (CCS '15), p. 1322–1333. ISBN 9781450338325. Disponível em: <https://doi.org/10.1145/2810103.2813677>.

42 BALUTA, T.; SHEN, S.; HITARTH, S.; TOPLE, S.; SAXENA, P. Membership inference attacks and generalization: A causal perspective. In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2022. (CCS '22), p. 249–262. ISBN 9781450394505. Disponível em: <https://doi.org/10.1145/3548606.3560694>.

43 MELIS, L.; SONG, C.; CRISTOFARO, E. D.; SHMATIKOV, V. Exploiting unintended feature leakage in collaborative learning. *2019 IEEE Symposium on Security and Privacy (SP)*, p. 691–706, 2018. Disponível em: <https://api.semanticscholar.org/CorpusID:53099247>.

44 OLIVEIRA, J.; MENEGUETTE, R.; GONçALVES, V.; JR., R. S.; GUIDONI, D.; OLIVEIRA, J.; FILHO, G. R. F-nids – sistema de detecção de intrusão descentralizado com base em aprendizado federado. In: *Anais do XLI Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*. Porto Alegre, RS, Brasil: SBC, 2023. p. 29–42. ISSN 2177-9384. Disponível em: <https://sol.sbc.org.br/index.php/sbrc/article/view/24527>.

45 de Oliveira, J. A.; GONçALVES, V. P.; MENEGUETTE, R. I.; de Sousa, R. T.; GUIDONI, D. L.; OLIVEIRA, J. C.; Rocha Filho, G. P. F-nids — a network intrusion detection system based on federated learning. *Computer Networks*, v. 236, p. 110010, 2023. ISSN 1389-1286. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1389128623004553>.

46 GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: The MIT Press, 2016. ISBN 0262035618.

47 BISHOP, C. M. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN 0387310738.

48 ALPAYDIN, E. *Introduction to Machine Learning*. 2nd. ed. [S.l.]: The MIT Press, 2010. ISBN 026201243X.

49 MUNN, M.; PITMAN, D. *Explainable AI for Practitioners*. O'Reilly Media, 2022. ISBN 9781098119096. Disponível em: <https://books.google.com.br/books?id=-dmYEAAAQBAJ>.

50 LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *nature*, Nature Publishing Group, v. 521, n. 7553, p. 436, 2015.

51 REVIEW of Deep Learning Algorithms and Architectures. *IEEE Access*, IEEE, v. 7, p. 53040–53065, 2019.

52 AGGARWAL, C. C. *Neural Networks and Deep Learning: A Textbook*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2018. ISBN 3319944622.

53 MCMAHAN, B.; MOORE, E.; RAMAGE, D.; HAMPSON, S.; ARCAS, B. A. y. Communication-Efficient Learning of Deep Networks from Decentralized Data. In: SINGH, A.; ZHU, J. (Ed.). *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. PMLR, 2017. (Proceedings of Machine Learning Research, v. 54), p. 1273–1282. Disponível em: <https://proceedings.mlr.press/v54/mcmahan17a.html>.

54 BONAWITZ, K. A.; EICHNER, H.; GRIESKAMP, W.; HUBA, D.; INGERMAN, A.; IVANOV, V.; KIDDON, C. M.; KONEčNý, J.; MAZZOCCHI, S.; MCMAHAN, B.; OVERVELDT, T. V.; PETROU, D.; RAMAGE, D.; ROSELANDER, J. Towards federated learning at scale: System design. In: *SysML 2019*. [s.n.], 2019. To appear. Disponível em: <https://arxiv.org/abs/1902.01046>.

55 ZHAO, R.; YIN, Y.; SHI, Y.; XUE, Z. Intelligent intrusion detection based on federated learning aided long short-term memory. *Physical Communication*, v. 42, p. 101157, 2020. ISSN 1874-4907. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1874490720302342>.

56 KONEčNý, J.; MCMAHAN, H. B.; YU, F. X.; RICHTARIK, P.; SURESH, A. T.; BACON, D. Federated learning: Strategies for improving communication efficiency. 2016. Disponível em: <https://arxiv.org/abs/1610.05492>.

57 YANG, Q.; LIU, Y.; CHEN, T.; TONG, Y. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, Association for Computing Machinery, New York, NY, USA, v. 10, n. 2, jan 2019. ISSN 2157-6904. Disponível em: <https://doi.org/10.1145/3298981>.

58 KAIROUZ, P.; MCMAHAN, H. B.; AVENT, B.; BELLET, A.; BENNIS, M.; BHAGOJI, A. N.; BONAWIT, 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.; KONECNý, J.; KOROLOVA, A.; KOUSHANFAR, F.; KOYEJO, S.; LEPOINT, T.; LIU, Y.; MITTAL, P.; MOHRI, M.; NOCK, R.; ÖZGüR, A.; PAGH, R.; QI, H.; RAMAGE, D.; RASKAR, R.; RAYKOVA, M.; 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.; ZHAO, S. *Advances and Open Problems in Federated Learning*. [S.l.]: Now Foundations and Trends, 2021.

59 SHOKRI, R.; STRONATI, M.; SONG, C.; SHMATIKOV, V. Membership inference attacks against machine learning models. In: *2017 IEEE Symposium on Security and Privacy (SP)*. [S.l.: s.n.], 2017. p. 3–18.

60 DWORK, C.; ROTH, A. [S.l.: s.n.], 2014.

61 LI, N.; LYU, M.; SU, D.; YANG, W. *Differential privacy : from theory to practice*. San Rafael, California: Morgan Claypool, 2017. (Synthesis lectures on information security, privacy, and trust, 18). ISBN 1-62705-297-6.

62 ZHU, T.; LI, G.; ZHOU, W.; YU, P. S. *Differential Privacy and Applications*. 1st. ed. [S.l.]: Springer Publishing Company, Incorporated, 2017. ISBN 3319620029.

63 DWORK, C.; MCSHERRY, F.; NISSIM, K.; SMITH, A. Calibrating noise to sensitivity in private data analysis. In: HALEVI, S.; RABIN, T. (Ed.). *Theory of Cryptography*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 265–284. ISBN 978-3-540-32732-5.

64 GHORBANI, A. A.; LU, W.; TAVALLAEE, M. Network intrusion detection and prevention - concepts and techniques. In: *Advances in Information Security*. [s.n.], 2009. Disponível em: <https://api.semanticscholar.org/CorpusID:41539305>.

65 YILMAZ, I.; MASUM, R.; SIRAJ, A. Addressing imbalanced data problem with generative adversarial network for intrusion detection. In: *2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI)*. [S.l.: s.n.], 2020. p. 25–30.

66 MBOW, M.; KOIDE, H.; SAKURAI, K. An intrusion detection system for imbalanced dataset based on deep learning. In: *2021 Ninth International Symposium on Computing and Networking (CANDAR)*. [S.l.: s.n.], 2021. p. 38–47.

67 NORTHCUTT, S.; NOVAK, J. *Network Intrusion Detection*. New Riders, 2002. (Voices (New Riders)). ISBN 9780735712652. Disponível em: <https://books.google.com.br/books?id=4ZdZJZhOT-YC>.

68 BUYYA, R.; BROBERG, J.; GOSCINSKI, A. M. *Cloud Computing Principles and Paradigms*. [S.l.]: Wiley Publishing, 2011. ISBN 9780470887998.

69 RAFAELS, R. J. *Cloud Computing: From Beginning to End*. North Charleston, SC, USA: CreateSpace Independent Publishing Platform, 2015. ISBN 1511404582.

70 CHEE, B. J. S.; FRANKLIN JR., C. *Cloud Computing: Technologies and Strategies of the Ubiquitous Data Center*. 1st. ed. USA: CRC Press, Inc., 2010. ISBN 1439806128.

71 BUYYA, R.; SRIRAMA, S. *Fog and Edge Computing: Principles and Paradigms*. Wiley, 2019. (Wiley Series on Parallel and Distributed Computing). ISBN 9781119524984. Disponível em: <https://books.google.com.br/books?id=v4B_DwAAQBAJ>.

72 MBASUVA, U.; ZODI, G.-A. L. Designing ensemble deep learning intrusion detection system for ddos attacks in software defined networks. In: *2022 16th International Conference on Ubiquitous Information Management and Communication (IMCOM)*. [S.l.: s.n.], 2022. p. 1–8.

73 LING, Z.; HAO, Z. J. An intrusion detection system based on normalized mutual information antibodies feature selection and adaptive quantum artificial immune system. *Int. J. Semant. Web Inf. Syst.*, IGI Global, USA, v. 18, n. 1, p. 1–25, jul 2022. ISSN 1552-6283. Disponível em: <https://doi.org/10.4018/IJSWIS.308469>.

74 LING, Z.; HAO, Z. J. Intrusion detection using normalized mutual information feature selection and parallel quantum genetic algorithm. *Int. J. Semant. Web Inf. Syst.*, IGI Global, USA, v. 18, n. 1, p. 1–24, jul 2022. ISSN 1552-6283. Disponível em: <https://doi.org/10.4018/IJSWIS.307324>.

75 YU, J.; YE, X.; LI, H. A high precision intrusion detection system for network security communication based on multi-scale convolutional neural network. *Future Generation Computer Systems*, v. 129, p. 399–406, 2022. ISSN 0167-739X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167739X21004143>.

76 QAZI, E. ul H.; IMRAN, M.; HAIDER, N.; SHOAIB, M.; RAZZAK, I. An intelligent and efficient network intrusion detection system using deep learning. *Computers and Electrical Engineering*, v. 99, p. 107764, 2022. ISSN 0045-7906. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045790622000684>.

77 AL-YASEEN, W. L.; OTHMAN, Z. A.; NAZRI, M. Z. A. Real-time multi-agent system for an adaptive intrusion detection system. *Pattern Recognition Letters*, v. 85, p. 56–64, 2017. ISSN 0167-8655. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167865516303415>.

78 M., R.; AHMED, M.; KHAN, R. An adaptive distributed intrusion detection system architecture using multi agents. *International Journal of Electrical and Computer Engineering (IJECE)*, v. 9, p. 4951, 12 2019.

79 RAJA, G.; ANBALAGAN, S.; VIJAYARAGHAVAN, G.; THEERTHAGIRI, S.; SURYANARAYAN, S. V.; WU, X.-W. Sp-cids: Secure and private collaborative ids for vanets. *IEEE Transactions on Intelligent Transportation Systems*, v. 22, n. 7, p. 4385–4393, 2021.

80 TABASSUM, A.; ERBAD, A.; MOHAMED, A.; GUIZANI, M. Privacy-preserving distributed ids using incremental learning for iot health systems. *IEEE Access*, v. 9, p. 14271–14283, 2021.

81 QIAN, B.; SU, J.; WEN, Z.; JHA, D. N.; LI, Y.; GUAN, Y.; PUTHAL, D.; JAMES, P.; YANG, R.; ZOMAYA, A. Y.; RANA, O.; WANG, L.; KOUTNY, M.; RANJAN, R. Orchestrating the development lifecycle of machine learning-based iot applications: A taxonomy and survey. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 53, n. 4, ago. 2020. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3398020>.

82 SALEM, A.; SAUTTER, Y.; BACKES, M.; HUMBERT, M.; ZHANG, Y. Baaan: Backdoor attacks against autoencoder and gan-based machine learning models. 10 2020.

83 RUZAFA-ALCAZAR, P.; FERNANDEZ-SAURA, P.; MARMOL-CAMPOS, E.; GONZALEZ-VIDAL, A.; RAMOS, J. L. H.; BERNAL, J.; SKARMETA, A. F. Intrusion detection based on privacy-preserving federated learning for the industrial iot. *IEEE Transactions on Industrial Informatics*, p. 1–1, 2021.

84 KUNDU, A. Fed+: A unified approach to robust personalized federated learning. 06 2021.

85 Sai Lohitha, N.; POUNAMBAL, M. Integrated publish/subscribe and push-pull method for cloud based iot framework for real time data processing. *Measurement: Sensors*, v. 27, p. 100699, 2023. ISSN 2665-9174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2665917423000351>.

86 HAN, S.; WOO, H. Ndn-based pub/sub system for scalable iot cloud. In: *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. [S.l.: s.n.], 2016. p. 488–491.

87 LAZIDIS, A.; TSAKOS, K.; PETRAKIS, E. G. Publish–subscribe approaches for the iot and the cloud: Functional and performance evaluation of open-source systems. *Internet of Things*, v. 19, p. 100538, 2022. ISSN 2542-6605. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2542660522000403>.

88 FRIHA, O.; FERRAG, M. A.; SHU, L.; MAGLARAS, L.; CHOO, K.-K. R.; NAFAA, M. Felids: Federated learning-based intrusion detection system for agricultural internet of things. *Journal of Parallel and Distributed Computing*, v. 165, p. 17–31, 2022. ISSN 0743-7315. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0743731522000570>.

89 BU, Z.; DONG, J.; LONG, Q.; SU, W. Deep Learning With Gaussian Differential Privacy. *Harvard Data Science Review*, v. 2, n. 3, sep 30 2020. Https://hdsr.mitpress.mit.edu/pub/u24wj42y.

90 ABADI, M.; CHU, A.; GOODFELLOW, I.; MCMAHAN, H. B.; MIRONOV, I.; TALWAR, K.; ZHANG, L. Deep learning with differential privacy. Association for Computing Machinery, New York, NY, USA, p. 308–318, 2016. Disponível em: <https://doi.org/10.1145/2976749.2978318>.

91 KÁCHA, P. Idea: security event taxonomy mapping. In: *18th International Conference on Circuits, Systems, Communications and Computers*. [S.l.: s.n.], 2014.

92 DEBAR, H.; VIINIKKA, J. Intrusion detection: Introduction to intrusion detection and security information management. In: . [S.l.: s.n.], 2005. v. 3655, p. 207–236. ISBN 978-3-540-28955-5.

93 SARHAN, M.; LAYEGHY, S.; PORTMANN, M. Towards a standard feature set for network intrusion detection system datasets. *Mobile Networks and Applications*, v. 27, n. 1, p. 357–370, Feb 2022. ISSN 1572-8153. Disponível em: <https://doi.org/10.1007/s11036-021-01843-0>.

94 ZHANG, X.; CHEN, C.; XIE, Y.; CHEN, X.; ZHANG, J.; XIANG, Y. A survey on privacy inference attacks and defenses in cloud-based deep neural network. *Computer Standards Interfaces*, v. 83, p. 103672, 2023. ISSN 0920-5489. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0920548922000435>.

95 NARAYANAN, A.; SHMATIKOV, V. Robust de-anonymization of large sparse datasets. In: *2008 IEEE Symposium on Security and Privacy (sp 2008)*. [S.l.: s.n.], 2008. p. 111–125.

96 GANTA, S. R.; KASIVISWANATHAN, S. P.; SMITH, A. *Composition Attacks and Auxiliary Information in Data Privacy*. 2008.

97 FREDRIKSON, M.; JHA, S.; RISTENPART, T. Model inversion attacks that exploit confidence information and basic countermeasures. In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. New York, NY, USA: Association for Computing Machinery, 2015. (CCS '15), p. 1322–1333. ISBN 9781450338325. Disponível em: <https://doi.org/10.1145/2810103.2813677>.

98 HITAJ, B.; ATENIESE, G.; PEREZ-CRUZ, F. *Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning*. 2017.

99 TRAMÈR, F.; ZHANG, F.; JUELS, A.; REITER, M. K.; RISTENPART, T. Stealing machine learning models via prediction APIs. In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, 2016. p. 601–618. ISBN 978-1-931971-32-4. Disponível em: <https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tramer>.

7 ANEXOS

Tabela 7.1: Descrição dos atributos do *dataset* NF-ToN-IoT-V2

Característica	**Descrição**
IPV4_SRC_ADDR	Endereço de origem IPv4
IPV4_DST_ADDR	Endereço de destino IPv4
L4_SRC_PORT	Número da porta de origem IPv4
L4_DST_PORT	Número da porta de destino IPv4
PROTOCOL	Byte identificador do protocolo IP
L7_PROTO	Protocolo da Camada 7 (numérico)
IN_BYTES	Número de bytes de entrada
OUT_BYTES	Número de bytes de saída
IN_PKTS	Número de pacotes de entrada
OUT_PKTS	Número de pacotes de saída
FLOW_DURATION_MILLISECONDS	Duração do fluxo em milissegundos
TCP_FLAGS	Soma cumulativa de todas as flags TCP
CLIENT_TCP_FLAGS	Soma cumulativa de todas as flags TCP do cliente
SERVER_TCP_FLAGS	Soma cumulativa de todas as flags TCP do servidor
DURATION_IN Cliente	para duração do fluxo do servidor (msec)
DURATION_OUT	Duração do fluxo do cliente para o servidor (msec)
MIN_TTL	TTL mínimo do fluxo
MAX_TTL	TTL máximo do fluxo
LONGEST_FLOW_PKT	Maior pacote (bytes) do fluxo
SHORTEST_FLOW_PKT	Menor pacote (bytes) do fluxo
MIN_IP_PKT_LEN	Comprimento do menor pacote IP do fluxo observado
MAX_IP_PKT_LEN	Comprimento do maior pacote IP do fluxo observado
SRC_TO_DST_SECOND_BYTES	Bytes/sec de origem para destino
DST_TO_SRC_SECOND_BYTES	Bytes/sec de destino para origem
RETRANSMITTED_IN_BYTES	Número de bytes de fluxo TCP retransmitidos (origem->destino)
RETRANSMITTED_IN_PKTS	Número de pacotes de fluxo TCP retransmitidos (origem->destino)
RETRANSMITTED_OUT_BYTES	Número de bytes de fluxo TCP retransmitidos (destino->origem)
RETRANSMITTED_OUT_PKTS	Número de pacotes de fluxo TCP retransmitidos (destino->origem)
SRC_TO_DST_AVG_THROUGHPUT	Taxa média de transferência de origem para destino (bps)
DST_TO_SRC_AVG_THROUGHPUT	Taxa média de transferência de destino para origem (bps)
NUM_PKTS_UP_TO_128_BYTES	Pacotes cujo tamanho de IP $\leq$ 128
NUM_PKTS_128_TO_256_BYTES	Pacotes cujo tamanho de IP > 128 e $\leq$ 256
NUM_PKTS_256_TO_512_BYTES	Pacotes cujo tamanho de IP > 256 e $\leq$ 512
NUM_PKTS_512_TO_1024_BYTES	Pacotes cujo tamanho de IP > 512 e $\leq$ 1024
NUM_PKTS_1024_TO_1514_BYTES	Pacotes cujo tamanho de IP > 1024 e $\leq$ 1514
TCP_WIN_MAX_IN	Janela TCP máxima (origem->destino)
TCP_WIN_MAX_OUT	Janela TCP máxima (destino->origem)
ICMP_TYPE	Tipo ICMP × 256 + código ICMP
ICMP_IPV4_TYPE	Tipo ICMP IPv4
DNS_QUERY_ID	Id de transação de consulta DNS
DNS_QUERY_TYPE	Tipo de consulta DNS (por exemplo, 1 = A, 2 = NS.)
DNS_TTL_ANSWER	TTL do primeiro registro A (se houver)
FTP_COMMAND_RET_CODE	Código de retorno do comando FTP do cliente

Tabela 7.2: Descrição das classes do *dataset* NF-ToN-IoT-V2

Classe	**Contagem**	**Descrição**
Benigno	6099469	Fluxos normais e não maliciosos
Backdoor	16809	Uma técnica que visa atacar computadores de acesso remoto respondendo a aplicativos clientes específicos construídos.
DoS	712609	Uma tentativa de sobrecarregar os recursos de um sistema de computador para impedir o acesso ou a disponibilidade de seus dados.
DDoS	2026234	Uma tentativa semelhante ao DoS, mas que possui múltiplas fontes distribuídas diferentes.
Injeção	684465	Uma variedade de ataques que fornecem entradas não confiáveis para alterar o curso da execução, sendo as injeções de SQL e código duas das principais.
MITM	7723	*Man In The Middle (Homem no Meio)* é um método que coloca um atacante entre uma vítima e o host com o qual a vítima está tentando se comunicar, para interceptar tráfego e comunicações.
Senha	1153323	Abrange uma variedade de ataques destinados a recuperar senhas por força bruta ou *sniffing*.
Ransomware	3425	Um ataque que criptografa os arquivos armazenados em um host e solicita compensação em troca da técnica/chave de descriptografia.
Varredura	3781419	Um grupo que consiste em uma variedade de técnicas que visam descobrir informações sobre redes e hosts, também conhecido como sondagem.
XSS	2455020	*Cross-site Scripting* (*Script entre Sites*) é um tipo de injeção no qual um atacante usa aplicativos da web para enviar *scripts* maliciosos para usuários finais.

8 CÓDIGO FONTE PARA O PREPROCESSAMENTO DO DATASET

8.1 CÓDIGOS PARA GERAR OS DOIS CONJUNTOS DE DADOS (TESTE E TREINO)

```
1
2  #Importacao das bibliotecas
3  import os
4  import codecs
5  import pandas as pd
6  from sklearn import preprocessing
7  import numpy as np
8  from sklearn.metrics import confusion_matrix
9  import seaborn as sns
10 import tensorflow as tf
11 import matplotlib.pyplot as plt
12 import matplotlib.pyplot as plt
13 from sklearn import preprocessing
14 from sklearn.preprocessing import LabelEncoder, OneHotEncoder,
       ↪ StandardScaler,MinMaxScaler
15 from sklearn.model_selection import train_test_split
16 from tensorflow.keras.utils import plot_model
17
18
19 #Constantes para os diretorios dos datasets
20 DIR_PATH = "/content/drive/MyDrive/datasets/"
21 PROCESSED_DIR_PATH = "/content/drive/MyDrive/datasets/NF-ToN-IoT-v2
       ↪ /"
22 FILE_PATH = os.path.join(DIR_PATH, "NF-ToN-IoT-v2.csv")
23
24 #Carregamento em memoria
25 url = os.path.join(FILE_PATH)
26 df = pd.read_csv(url)
27
28 #Randomizacao
29 df = df.sample(frac=1)
30
```

```
#Obter 2500000 de exemplos
df = df.head(2500000)

#Salvamento das classes
classes = np.load(os.path.join(DIR_PATH, 'label_encoder.npy'),
    ↪ allow_pickle=True)
classes = {y:x for x,y in enumerate(list_classes)}


# Limpezas das features desnecessarias

if "IPV4_SRC_ADDR" in df.columns:
  remove_columns = ['IPV4_SRC_ADDR','IPV4_DST_ADDR']
  df.drop(remove_columns, axis=1, inplace=True)

categorical_features = [ 'L4_SRC_PORT', 'L4_DST_PORT', 'PROTOCOL', '
    ↪ L7_PROTO']

numerical_features = df.columns.difference(categorical_features)
numerical_features = numerical_features.difference(['Label','Attack'
    ↪ ])

#Tratamento das feature numericas

scaler = MinMaxScaler()

df[numerical_features] = scaler.fit_transform(df[numerical_features
    ↪ ])

#Atualizacao do dataframe apos o tratamento

le = LabelEncoder()

for c in categorical_features:
  df[c] = le.fit_transform(df[c])


# Funcao de split do dataset para dividir entre treino e teste

def _split_train_test(df: pd.DataFrame) -> (pd.DataFrame,
    ↪ pd.DataFrame):
    # Sampling the dataset
```

```
68      x = df.iloc[:, df.columns != 'Attack']
69      y = df[['Attack']]
70
71      x_train, x_test, y_train, y_test = train_test_split(x, y,
            ↪ stratify=y, test_size=0.20,
72                                                  random_state=
                                                        ↪ np.random.randint
                                                        ↪ (10))
73
74      del x, y
75
76      train = pd.concat([x_train, y_train], axis=1, sort=False)
77      test = pd.concat([x_test, y_test], axis=1, sort=False)
78
79      return train, test
80
81  def _to_csv(df: pd.DataFrame, saving_path: str):
82      # if file does not exist write header
83      print(os.path.isfile(saving_path))
84      if not os.path.isfile(saving_path):
85          print(saving_path)
86          df.to_csv(saving_path, index=False)
87      # else it exists so append without writing the header
88      else:
89          df.to_csv(saving_path, index=False, mode='a', header=False)
90
91  train, test = _split_train_test(df)
92
93  #Armazenamento do resultado do preprocessamnto
94
95  _to_csv(train, os.path.join(PROCESSED_DIR_PATH, "train_NF-TON-IoT-
        ↪ v2.csv.csv"))
96
97  _to_csv(test, os.path.join(PROCESSED_DIR_PATH, "test_NF-TON-IoT-
        ↪ v2.csv.csv"))
```

9 CÓDIGO FONTE PARA TREINAMENTO DO MODELO CENTRALIZADO

9.1 CÓDIGOS PARA TREINAMENTO DA REDE NEURAL CENTRALIZADA

```
#Importacao das bibliotecas

import os
import codecs
from random import seed
from random import randint
import pandas as pd
from sklearn import preprocessing
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder, OneHotEncoder,
    ↪ StandardScaler,MinMaxScaler
from sklearn.model_selection import train_test_split
import json
import math
import datetime
from tensorflow_privacy.privacy.optimizers import dp_optimizer
from tensorflow_privacy.privacy.analysis import
    ↪ compute_dp_sgd_privacy
from tensorflow_privacy.privacy.optimizers.dp_optimizer import
    ↪ DPGradientDescentGaussianOptimizer, DPAdamGaussianOptimizer
from tensorflow_privacy.privacy.optimizers.dp_optimizer_keras import
    ↪  DPKerasAdamOptimizer, DPKerasSGDOptimizer

#Definicao de constantes

DIR_PATH = "/content/drive/MyDrive/datasets/"
MODEL_PATH = "/content/drive/MyDrive/NF-TON-IOTv2/centralizado"
```

```
MODEL_NAME = "NF-TON-IoT-v2.central.h5"
MODEL_NAME_F = "NF-TON-IoT-v2-{alg}-{model_id}-{epoch}.h5"
LOG_FILE_NAME = "NF-TON-IoT.log"
PROCESSED_DIR_PATH = f"{DIR_PATH}NF-ToN-IoT-v2/"
model_path = os.path.join(MODEL_PATH, MODEL_NAME);
log_path = os.path.join(MODEL_PATH, LOG_FILE_NAME);
json_log_path = os.path.join(MODEL_PATH, "json_log_path_priv.json")
json_history_path = os.path.join(MODEL_PATH, "
    ↪ json_history_path_priv.json")
test = os.path.join(PROCESSED_DIR_PATH, "test_NF-TON-IoT-v2.csv")
train = os.path.join(PROCESSED_DIR_PATH, "train_NF-TON-IoT-v2.csv")

#Carregamento dos datasets

df_train = pd.read_csv(train, skipinitialspace=True)

df_test = pd.read_csv(test, skipinitialspace=True)

#Tratamentos das features rotulos das classes

if "Attack" in df_train.columns:
  y_train = df_train['Attack'].apply(lambda x: classes[x] if
      ↪ isinstance(x,str) else x)
  X_train = df_train.drop('Attack', axis=1)
  del df_train


if "Attack" in df_test.columns:
  y_test = df_test['Attack'].apply(lambda x: classes[x] if
      ↪ isinstance(x,str) else x)
  X_test = df_test.drop('Attack', axis=1)
  del df_test

#Metodos de arquitetura da rede

def build_nn_model_Adam(learning_rate = 1e-2):
  optimizer = tf.optimizers.Adam(learning_rate=learning_rate)
  hiden_units_num = int((len(X_train.columns) + len(classes))/2)
  model = the_model()

  model.compile(loss='sparse_categorical_crossentropy',
                  metrics=['sparse_categorical_accuracy'],
```

```
69                    optimizer=optimizer)
70
71    return model
72
73  def build_nn_model_DP_Adam(learning_rate = 1e-2):
74    optimizer = DPKerasAdamOptimizer(
75      l2_norm_clip=0.1,
76      noise_multiplier=0.5,
77      learning_rate=learning_rate,
78      num_microbatches=BATCH_SIZE)
79
80    model = the_model()
81
82    loss = tf.keras.losses.SparseCategoricalCrossentropy(reduction=
        ↪ tf.losses.Reduction.NONE)
83
84    model.compile(loss=loss,
85                    metrics=['sparse_categorical_accuracy'],
86                    optimizer=optimizer)
87
88    return model
89
90  #Metodos auxiliares
91
92  def train_model(model, batch_size = 250, epochs = 1):
93    return model.fit(X_train, y_train, epochs=epochs, batch_size=
        ↪ batch_size, validation_split=0.2)
94
95  def splits(instance, X,y, validation=True, num_splits = 0):
96    num_clients = NUM_SPLITS if num_splits == 0 else num_splits
97    partition_size = math.floor(len(X) / num_clients)
98    idx_from, idx_to = int(instance) * partition_size, (int(instance)
        ↪ + 1) * partition_size
99    x_train_cid = X[idx_from:idx_to]
100   y_train_cid = y[idx_from:idx_to]
101
102   x_val_cid,y_val_cid = [],[]
103   return x_train_cid, y_train_cid,x_val_cid,y_val_cid
```

9.2 TREINAMENTO DA REDE CENTRALIZADA SEM DP

```
#Execucao dos treinamentos dos modelos Centralizado sem DP e salvamento dos modelos e dos resultados das avaliacoes

global optimization_alg
global m_index
global batch_size
NUM_SPLITS = 10

BATCH_SIZE = 1000
EPOCHS = 10
LEARNING_RATE = 0.25

json_history_log = open(json_history_path, mode='a', buffering=1)
optimization_alg = 'Adam'
for _ in range(1, 11):
  model = build_nn_model_DP_Adam(LEARNING_RATE)
  for i in range(0, 10):
    m_index = i
    x__train, y__train, *_ = splits(i, X_train, y_train, validation=False)
    history = model.fit(x__train, y__train, epochs=EPOCHS,batch_size=BATCH_SIZE, validation_split=0.2)
    loss, accuracy = model.evaluate(X_test, y_test, batch_size=BATCH_SIZE, verbose=2)
    json_log = open(json_log_path, mode='a', buffering=1)
    date_time = datetime.datetime.now()
    date_time = date_time.strftime(' %d/%m/%Y %H:%M:%S.%f')[:-3]
    json_log.write(json.dumps({"instance": i, "loss": str(loss), "accuracy": str(accuracy), "datetime": date_time}) + '\n')
    json_log.close()
    json_history_log.write(json.dumps({'index': m_index, 'alg': optimization_alg, 'history': json.dumps(history.history)}) + '\n')
```

9.3 TREINAMENTO DA REDE CENTRALIZADA COM DP

```
#Execucao dos treinamentos do modelo Centralizado com DP e salvamento dos modelos e dos resultados das avaliacoes

global optimization_alg
global m_index
global batch_size
NUM_SPLITS = 10

BATCH_SIZE = 1000
EPOCHS = 10
LEARNING_RATE = 0.25

json_history_log = open(json_history_path, mode='a', buffering=1)
optimization_alg = 'DP-Adam'
for _ in range(1, 11):
  model = build_nn_model_DP_Adam(LEARNING_RATE)
  for i in range(0, 10):
    m_index = i
    x__train, y__train, *_ = splits(i, X_train, y_train, validation=False)
    history = model.fit(x__train, y__train, epochs=EPOCHS,batch_size=BATCH_SIZE, validation_split=0.2)
    loss, accuracy = model.evaluate(X_test, y_test, batch_size=BATCH_SIZE, verbose=2)
    json_log = open(json_log_path, mode='a', buffering=1)
    date_time = datetime.datetime.now()
    date_time = date_time.strftime(' %d/%m/%Y %H:%M:%S.%f')[:-3]
    json_log.write(json.dumps({"instance": i, "loss": str(loss), "accuracy": str(accuracy), "datetime": date_time}) + '\n')
    json_log.close()
    json_history_log.write(json.dumps({'index': m_index, 'alg': optimization_alg, 'history': json.dumps(history.history)}) + '\n')
```

10 CÓDIGO FONTE PARA TREINAMENTO DOS MODELOS FEDERADOS

10.1 MÉTODOS PARA TREINO DA REDE NEURAL FEDERADA

```
1
2  #Importacao das libs
3
4  import os
5  import codecs
6  from random import seed
7  from random import randint
8  import pandas as pd
9  from sklearn import preprocessing
10 import numpy as np
11 from sklearn.metrics import confusion_matrix
12 import seaborn as sns
13 import tensorflow as tf
14 import matplotlib.pyplot as plt
15 import matplotlib.pyplot as plt
16 from sklearn import preprocessing
17 from sklearn.preprocessing import LabelEncoder, OneHotEncoder,
       ↪ StandardScaler,MinMaxScaler
18 from sklearn.model_selection import train_test_split
19 import json
20 import math
21 import datetime
22 import flwr as fl
23 from flwr import common
24 from tensorflow_privacy.privacy.optimizers import dp_optimizer
25 from tensorflow_privacy.privacy.analysis import
       ↪ compute_dp_sgd_privacy
26 from tensorflow_privacy.privacy.optimizers.dp_optimizer import
       ↪ DPGradientDescentGaussianOptimizer, DPAdamGaussianOptimizer
27 from tensorflow_privacy.privacy.optimizers.dp_optimizer_keras import
       ↪  DPKerasAdamOptimizer, DPKerasSGDOptimizer
28 from random import randrange
29
```

```
30 os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
31
32
33 #Constantes
34
35 DIR_PATH = "/content/drive/MyDrive/datasets/"
36 MODEL_PATH = "/content/drive/MyDrive/NF-TON-IOTv2/fed"
37 MODEL_NAME = "NF-TON-IoT-v2_Adam_i{instance}_r{round}.dp-fed.h5"
38 LOG_FILE_NAME = "NF-TON-IoT.log"
39 PROCESSED_DIR_PATH = f"{DIR_PATH}NF-ToN-IoT-v2/"
40 model_path = os.path.join(MODEL_PATH, MODEL_NAME);
41 log_path = os.path.join(MODEL_PATH, LOG_FILE_NAME);
42 json_log_path = os.path.join(MODEL_PATH, "
      ↪ json_log_path_fed_priv.json")
43 json_history_path = os.path.join(MODEL_PATH, "
      ↪ json_history_path_fed_priv.json")
44 test = os.path.join(PROCESSED_DIR_PATH, "test_NF-TON-IoT-v2.csv")
45 train = os.path.join(PROCESSED_DIR_PATH, "train_NF-TON-IoT-v2.csv")
46
47 #Carregamento dos datasets em memria
48
49 df_train = pd.read_csv(train, skipinitialspace=True)
50
51 df_test = pd.read_csv(test, skipinitialspace=True)
52
53 #Tratamento das classes de ataque
54
55 if "Attack" in df_train.columns:
56   y_train = df_train['Attack'].apply(lambda x: classes[x] if
      ↪ isinstance(x,str) else x)
57   X_train = df_train.drop('Attack', axis=1)
58   del df_train
59
60
61 if "Attack" in df_test.columns:
62   y_test = df_test['Attack'].apply(lambda x: classes[x] if
      ↪ isinstance(x,str) else x)
63   X_test = df_test.drop('Attack', axis=1)
64   del df_test
65
66 #Metodos de criacao da arquittura da rede e treinamento
67
```

```
68 def build_nn_model_Adam(learning_rate = 1e-2):
69   optimizer = tf.optimizers.Adam(learning_rate=learning_rate)
70   hiden_units_num = int((len(X_train.columns) + len(classes))/2)
71   model = the_model()
72 
73   model.compile(loss='sparse_categorical_crossentropy',
74                 metrics=['sparse_categorical_accuracy'],
75                 optimizer=optimizer)
76 
77   return model
78 
79 
80 def build_nn_model_DP_Adam(learning_rate = 1e-2):
81   optimizer = DPKerasAdamOptimizer(
82     l2_norm_clip=L2_NORM_CLIP,
83     noise_multiplier=NOISE,
84     learning_rate=learning_rate,
85     num_microbatches=BATCH_SIZE)
86 
87   model = the_model()
88 
89   loss = tf.keras.losses.SparseCategoricalCrossentropy(reduction=
       ↪ tf.losses.Reduction.NONE)
90 
91   model.compile(loss=loss,
92                 metrics=['sparse_categorical_accuracy'],
93                 optimizer=optimizer)
94 
95   return model
96 
97 def train_model(model, batch_size = 250, epochs = 1):
98   return model.fit(X_train, y_train, epochs=epochs, batch_size=
       ↪ BATCH_SIZE, validation_split=0.2)
99 
100 
101 #Mtodos para o FlowerClient e outros mtodos auxiliares
102 
103 class FlowerClient(fl.client.NumPyClient):
104     def __init__(self, model, x_train, y_train, x_test, y_test,
        ↪ v_split = .1, client_id = 0) -> None:
105       self.model = model
106       self.x_train, self.y_train = x_train, y_train
```

```
107         self.x_test, self.y_test = x_test, y_test
108         self.history = None
109         self.v_split = v_split
110         self.batch_size=BATCH_SIZE
111         self.client_id = client_id
112         self.epochs = EPOCHS
113         self.log_file_name = f"dnn_CIC__c{NUM_CLIENTS}_r{ROUNDS}_e{
                ↪ EPOCHS}_ci{self.client_id}"
114         self.csv_logger = tf.keras.callbacks.CSVLogger(log_path,
                ↪ separator=',', append=True)
115 
116     def get_parameters(self, config):
117         return self.model.get_weights()
118 
119     def fit(self, parameters, config):
120         stop_early = tf.keras.callbacks.EarlyStopping(monitor='
                ↪ val_loss', patience=3)
121         self.model.set_weights(parameters)
122         history = self.model.fit(self.x_train, self.y_train, epochs=
                ↪ EPOCHS,batch_size=BATCH_SIZE, validation_split=0.2,
                ↪ verbose=2, callbacks=[stop_early])
123         results = {
124             "loss": history.history["loss"][0],
125             "accuracy": history.history["sparse_categorical_accuracy
                    ↪ "][0],
126             "val_loss": history.history["val_loss"][0],
127             "val_accuracy": history.history["
                    ↪ val_sparse_categorical_accuracy"][0],
128         }
129         return self.model.get_weights(), len(self.x_train), results
130 
131     def evaluate(self, parameters, config):
132         self.model.set_weights(parameters)
133         loss, acc = self.model.evaluate(self.x_test, self.y_test,
                ↪ batch_size=BATCH_SIZE, verbose=2)
134         return loss, len(self.x_val), {"accuracy": acc}
135 """
136     def fit(self, parameters, config):
137         self.model.set_weights(parameters)
138         self.history = self.model.fit(self.x_train, self.y_train,
                ↪ batch_size=self.batch_size, epochs=self.epochs,
                ↪ validation_split=self.v_split, verbose=2)
```

```
        # Return updated model parameters and results
        parameters_prime = self.model.get_weights()
        num_examples_train = len(self.x_train)

        results = {
            "loss": self.history.history["loss"][0],
            "accuracy": self.history.history["accuracy"][0],
            "val_loss": self.history.history["val_loss"][0],
            "val_accuracy": self.history.history["val_accuracy"][0],
        }


        return parameters_prime, num_examples_train, results
"""



def name_global_model(iround = None):
  cround = ROUNDS
  global optimization_alg
  if iround is not None:
    cround = iround

  return f"dnn_NF-TON-IoT-V2__c{NUM_CLIENTS}_r{cround}_e{EPOCHS}
     ↪ _global.h5"

def splits(instance, X,y, validation=True, num_clients = 0):
  num_clients = NUM_CLIENTS if num_clients == 0 else num_clients
  partition_size = math.floor(len(X) / num_clients)
  idx_from, idx_to = int(instance) * partition_size, (int(instance)
     ↪ + 1) * partition_size
  x_train_cid = X[idx_from:idx_to]
  y_train_cid = y[idx_from:idx_to]

  x_val_cid,y_val_cid = [],[]
  if(validation):
    # Use 10% of the client's training data for validation
    split_idx = math.floor(len(X) * 0.9)
    x_train_cid, y_train_cid = x_train_cid[:split_idx], y_train_cid[:
       ↪ split_idx]
    x_val_cid, y_val_cid = x_train_cid[split_idx:], y_train_cid[
       ↪ split_idx:]
```

```
177
178   return x_train_cid, y_train_cid,x_val_cid,y_val_cid
179
180 def create_dense_model(learning_rate = 1e-2):
181   global optimization_alg
182   model = None
183   if optimization_alg == 'Adam':
184     model = build_nn_model_Adam(learning_rate)
185   if optimization_alg == 'SGD':
186     model = build_nn_model_SGD(learning_rate)
187   if optimization_alg == 'Adagrad':
188     model = build_nn_model_Adagrad(learning_rate)
189   if optimization_alg == 'DP-Adam':
190     model = build_nn_model_DP_Adam(learning_rate)
191
192   return model
193
194 def register_results(model, X_test_split, y_test_split, instance,
      ↪ round):
195   y_pred = np.argmax(model.predict(X_test_split), axis=1)
196   r_df = pd.DataFrame()
197   r_df['y_test'] = y_test_split
198   r_df['y_predict'] = y_pred
199   r_df['instance'] = instance
200   r_df['round'] = round
201   path = os.path.join(MODEL_PATH, f"results_fed_dp_adam.csv")
202   r_df.to_csv(path, mode='a', header=not os.path.isfile(path))
203
204 def get_NF_TON_IoT_V2_model_and_data(learning_rate = 1e-2):
205     model = create_dense_model(learning_rate)
206
207     # Load data partition (divide MNIST into NUM_CLIENTS distinct
          ↪ partitions)
208     train,test = (X_train,y_train),(X_test, y_test)
209     return model, train, test
210
211 def client_fn(cid: str) -> fl.client.Client:
212     # Create model
213     model, (x__train, y__train), (x__test, y__test) =
          ↪ get_NF_TON_IoT_V2_model_and_data()
214
```

```
215     # Load data partition (divide MNIST into NUM_CLIENTS distinct
            ↪ partitions)
216     x_train_cid, y_train_cid, *_ = splits(cid, x__train, y__train,
            ↪ validation=False)
217 
218     x_test_cid, y_test_cid, *_ = splits(cid, x__test, y__test,
            ↪ validation=False)
219 
220     # Create and return client
221     return FlowerClient(model, x_train_cid, y_train_cid,x_test_cid,
            ↪ y_test_cid, client_id=cid)
222 
223 def get_eval_fn(model, x_test, y_test, instance = None):
224     """Return an evaluation function for server-side evaluation."""
225 
226     x_val, y_val = x_test, y_test
227 
228     # The `evaluate` function will be called after every round
229     def evaluate(round, parameters, _):
230         model.set_weights(parameters) # Update model with the latest
                ↪ parameters
231         model_path = os.path.join(MODEL_PATH, MODEL_NAME.format(
                ↪ instance=instance, round=round))
232         model.save(model_path)
233 
234         loss, accuracy = model.evaluate(x_val, y_val, batch_size=
                ↪ BATCH_SIZE, verbose=2)
235 
236         json_log = open(json_log_path, mode='a', buffering=1)
237         date_time = datetime.datetime.now()
238         date_time = date_time.strftime(' %d/%m/%Y %H:%M:%S.%f')[:-3]
239 
240         json_log.write(json.dumps({"instance": instance, "round": str(
                ↪ round), "loss": str(loss), "accuracy": str(accuracy), "
                ↪ datetime": date_time}) + '\n')
241         json_log.close()
242         x__test, y__test, *_ = splits(randrange(10), X_test, y_test,
                ↪ validation=False, num_clients=10)
243         register_results(model, x__test, y__test, instance, round)
244 
245         return loss, {"accuracy": accuracy}
246 
```

```
247
248     return evaluate
```

10.2 TREINAMENTO DAS REDES NAURAIS FEDERADAS SEM DP

```
1
2  global optimization_alg
3  global m_index
4  global batch_size
5
6  NUM_CLIENTS = 100
7  EPOCHS = 10
8  ROUNDS = 10
9  L2_NORM_CLIP = 1.5
10 NOISE = 0.5
11 BATCH_SIZE = 1000
12
13 optimization_alg = 'Adam'
14    for i in range(1, 11):
15   model, (x__train, y__train), (x__test, y__test) =
        ↪ get_NF_TON_IoT_V2_model_and_data()
16   strategy=fl.server.strategy.FedAvg(
17        fraction_fit=0.1, # Sample 10% of available clients for
             ↪ training
18        fraction_evaluate=0.1, # Sample 5% of available clients for
             ↪ evaluation
19        evaluate_fn=get_eval_fn(model, x__test, y__test, i),
20        min_fit_clients=10, # Never sample less than 10 clients for
             ↪ training
21        min_evaluate_clients=10, # Never sample less than 5 clients
             ↪ for evaluation
22        min_available_clients=int(NUM_CLIENTS * 0.75), # Wait until at
             ↪  least 75 clients are available
23   )
24
25 # Start simulation
26   fl.simulation.start_simulation(
27      client_fn=client_fn,
28
```

```
29      num_clients=NUM_CLIENTS,
30      config=fl.server.ServerConfig(num_rounds=ROUNDS), # Just three
            ↪ rounds
31      strategy=strategy,
32  )
```

10.3 TREINAMENTO E DAS REDES NEURAIS FEDERADAS COM DP

```
1     global optimization_alg
2 global m_index
3 global batch_size
4
5 NUM_CLIENTS = 100
6 EPOCHS = 10
7 ROUNDS = 10
8 L2_NORM_CLIP = 1.5
9 NOISE = 0.5
10 BATCH_SIZE = 1000
11
12 optimization_alg = 'DP-Adam'
13 for i in range(1, 11):
14   model, (x__train, y__train), (x__test, y__test) =
        ↪ get_NF_TON_IoT_V2_model_and_data()
15   strategy=fl.server.strategy.FedAvg(
16       fraction_fit=0.1, # Sample 10% of available clients for
            ↪ training
17       fraction_evaluate=0.1, # Sample 5% of available clients for
            ↪ evaluation
18       evaluate_fn=get_eval_fn(model, x__test, y__test, i),
19       min_fit_clients=10, # Never sample less than 10 clients for
            ↪ training
20       min_evaluate_clients=10, # Never sample less than 5 clients
            ↪ for evaluation
21       min_available_clients=int(NUM_CLIENTS * 0.75), # Wait until at
            ↪  least 75 clients are available
22   )
23
24 # Start simulation
25   fl.simulation.start_simulation(
```

```
26      client_fn=client_fn,
27
28      num_clients=NUM_CLIENTS,
29      config=fl.server.ServerConfig(num_rounds=ROUNDS), # Just three
            ↪ rounds
30      strategy=strategy,
31  )
```

11 CÓDIGO FONTE PARA ATAQUES DE INFERENCIA DE MEMBROS

11.1 CÓDIGOS PARA GERAR OS DOIS CONJUNTOS DE DADOS (TESTE E TREINO)

```
#Importacoes de bibliotecas

import argparse
import os

# Make TensorFlow log less verbose
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
import sys
import tensorflow as tf
import numpy as np
from matplotlib import pyplot as plt
from art.attacks.inference.membership_inference import
    ↪ MembershipInferenceBlackBox
from art.utils import to_categorical
import tensorflow.python.ops.numpy_ops.np_config as np_config

import os
import codecs
from random import seed
from random import randint
import pandas as pd
from sklearn import preprocessing
import numpy as np
from sklearn.metrics import confusion_matrix
import seaborn as sns
import tensorflow as tf
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder, OneHotEncoder,
    ↪ StandardScaler,MinMaxScaler
from sklearn.model_selection import train_test_split
```

```
import json
import math
import datetime

from tensorflow_privacy.privacy.optimizers.dp_optimizer_keras import
    ↪ DPKerasAdamOptimizer
np_config.enable_numpy_behavior()

from sklearn.preprocessing import LabelEncoder, OneHotEncoder,
    ↪ StandardScaler,MinMaxScaler
import privacy_evaluator.models.torch.dcti.dcti as torch_dcti
import privacy_evaluator.models.tf.dcti.dcti as tf_dcti

from privacy_evaluator.datasets.tf.cifar10 import TFCIFAR10
from privacy_evaluator.datasets.torch.cifar10 import TorchCIFAR10

from privacy_evaluator.classifiers.classifier import Classifier

from privacy_evaluator import validators as vl
from privacy_evaluator.attacks.membership_inference.black_box import
    ↪ MembershipInferenceBlackBoxAttack
from
    ↪ privacy_evaluator.attacks.membership_inference.black_box_rule_based
    ↪ import MembershipInferenceBlackBoxRuleBasedAttack
from
    ↪ privacy_evaluator.attacks.membership_inference.label_only_decision_bounda
    ↪ import MembershipInferenceLabelOnlyDecisionBoundaryAttack
from privacy_evaluator.attacks.membership_inference import
    ↪ MembershipInferenceAttackAnalysis

from privacy_evaluator.attacks.membership_inference import
    ↪ MembershipInferenceAttackOnPointBasis
from privacy_evaluator.attacks.membership_inference import
    ↪ MembershipInferencePointAnalysis

from
    ↪ privacy_evaluator.attacks.membership_inference.data_structures.attack_inp
    ↪ import AttackInputData
from
    ↪ privacy_evaluator.attacks.membership_inference.data_structures.slicing
    ↪ import Slicing

```

```
#Variaveis de configuracao

DIR_PATH = "/content/drive/MyDrive/datasets/"
MODEL_PATH = "/content/drive/MyDrive/NF-TON-IOTv2/model/fed"
MODEL_NAME = "NF-TON-IoT-v2.central.h5"
MODEL_NAME_F = "NF-TON-IoT-v2-{alg}-{model_id}-{epoch}.h5"
LOG_FILE_NAME = "NF-TON-IoT.log"
PROCESSED_DIR_PATH = f"{DIR_PATH}NF-ToN-IoT-v2/"
model_path = os.path.join(MODEL_PATH, MODEL_NAME);
log_path = os.path.join(MODEL_PATH, LOG_FILE_NAME);
json_log_path = os.path.join(MODEL_PATH, "json_log_path_priv2.json")
json_history_path = os.path.join(MODEL_PATH, "
    ↪ json_history_path_priv2.json")
test = os.path.join(PROCESSED_DIR_PATH, "test_NF-TON-IoT-v2.csv")
train = os.path.join(PROCESSED_DIR_PATH, "train_NF-TON-IoT-v2.csv")

df_train = pd.read_csv(train, skipinitialspace=True)

df_test = pd.read_csv(test, skipinitialspace=True)

model_path = os.path.join(MODEL_PATH,"NF-TON-IoT-v2-FED-DP-DP-Adam-
    ↪ r10-20220922-045513.h5")

federado = tf.keras.models.load_model(model_path, compile=False)

compile_model(federado)

global optimization_alg
global m_index
global batch_size
NUM_SPLITS = 10

BATCH_SIZE = 1000
EPOCHS = 10
LEARNING_RATE = 0.25

optimization_alg = 'DP-Adam'


model = build_nn_model_DP_Adam()

```

```
(x_train, y_train), (x_test, y_test) = (X_train, y_train), (X_test, y_test)
"""
trainer = aux.CentralizedTrainer(model, train, test, epochs=epochs, batch_size=batch_size,
                                 validation_split=0.1, categorical=True)

trainer.model.save("centralizado100epocascombined.h5")

target_model = trainer.model
"""
def blackbox_attack(target_model):
    classifier = get_classifier(target_model)

    attack = MembershipInferenceBlackBoxAttack(classifier)

    attack.fit(x_train[:100], y_train[:100], x_test[:100], y_test[:100])
    attack.attack(x_train[:100], y_train[:100])

    output = attack.attack_output(
        x_train[:100],
        y_train[:100],
        x_train,
        y_train,
        x_test,

        y_test,
        np.ones((100,))
    )
    print(output.to_json())

    analysis = MembershipInferenceAttackAnalysis(
        MembershipInferenceBlackBoxAttack,
        AttackInputData(
            x_train[:100],
            y_train[:100],
            x_test[:100],
            y_test[:100]
        )
    )

```

```
137     slicing = Slicing(
138         entire_dataset=True,
139         by_class=False,
140         by_classification_correctness=False
141     )
142 
143     result = analysis.analyse(
144         classifier,
145         np.concatenate((x_train[:100], x_test[:100])),
146         np.concatenate((y_train[:100], y_test[:100])),
147         np.concatenate((np.ones(len(x_train[:100])), np.zeros(len(
            ↪ x_test[:100])))),
148         slicing
149     )
150 
151     print("\n".join((str(r) for r in result)))
152 
153 
154 def get_classifier(target_model):
155     classifier = Classifier(target_model,
        ↪ tf.keras.losses.SparseCategoricalCrossentropy(reduction=
        ↪ tf.losses.Reduction.NONE), len(classes), (X_train.columns,)
        ↪ )
156     return classifier
157 
158 
159 def rule_based_attack(target_model):
160     classifier = get_classifier(target_model)
161     attack = MembershipInferenceBlackBoxRuleBasedAttack(classifier)
162     attack.attack(x_train, y_train)
163     output = attack.attack_output(
164         x_train[:100],
165         y_train[:100],
166         x_train,
167         y_train,
168         x_test,
169         y_test,
170         np.ones((len(y_train[:100]),))
171     )
172 
173     print(str(output))
174     output.to_dict()
```

```
175
176     analysis = MembershipInferenceAttackAnalysis(
177         MembershipInferenceBlackBoxRuleBasedAttack,
178         AttackInputData(
179             x_train[:100],
180             y_train[:100],
181             x_test[:100],
182             y_test[:100]
183         )
184     )
185
186     slicing = Slicing(
187         entire_dataset=True,
188         by_class=False,
189         by_classification_correctness=False
190     )
191
192     result = analysis.analyse(
193         classifier,
194         np.concatenate((x_train[:100], x_test[:100])),
195         np.concatenate((y_train[:100], y_test[:100])),
196         np.concatenate((np.ones(len(x_train[:100])), np.zeros(len(
            ↪ x_test[:100])))),
197         slicing
198     )
199
200     print("\n".join((str(r) for r in result)))
201     print("\n result: ".join((str(r) for r in result)))
202
203
204 def label_only_decision_boundary(target_model):
205
206     target_model = get_classifier(target_model)
207     attack = MembershipInferenceLabelOnlyDecisionBoundaryAttack(
            ↪ target_model)
208     attack.fit(x_train[:100], y_train[:100], x_test[:20], y_test
            ↪ [:20], max_iter=5, max_eval=5, init_eval=5)
209     attack.attack(x_train[500:520], y_train[500:520])
210     output = attack.attack_output(
211         x_train[500:520],
212         y_train[500:520],
213         x_train[:520],
```

```
        y_train[:520],
        x_test[:520],
        y_test[:520],
        np.ones((20,))
    )

    print("output: " + str(output))


    analysis = MembershipInferenceAttackAnalysis(
        MembershipInferenceLabelOnlyDecisionBoundaryAttack,
        AttackInputData(
            x_train[:100],
            y_train[:100],
            x_test[:100],
            y_test[:100]
        )
    )

    slicing = Slicing(
        entire_dataset=True,
        by_class=False,
        by_classification_correctness=False
    )

    result = analysis.analyse(
        target_model,
        np.concatenate((x_train[:100], x_test[500:600])),
        np.concatenate((y_train[:100], y_test[500:600])),
        np.concatenate((np.ones(len(x_train[:100])), np.zeros(len(
            ↪ x_test[:100])))),
        slicing
    )

    print("\n result: ".join((str(r) for r in result)))


#centralizado = tf.keras.models.load_model("
    ↪ centralizado100epocas150KT.3-V1.h5",
                                #custom_objects={'DPOptimizerClass':
                                    ↪  DPKerasAdamOptimizer})
```

```
253
254 model_path = os.path.join(MODEL_PATH,"NF-TON-IoT-v2-FED-DP-DP-Adam-
      ↪ r10-20220922-045513.h5")
255
256 federado = tf.keras.models.load_model(model_path, compile=False)
257
258 #Execucao dos testes de blackbox membership inference
259
260 blackbox_attack(centralizado_sem_dp)
261
262 blackbox_attack(centralizado_com_dp)
263
264 blackbox_attack(federado_sem_dp)
265
266 blackbox_attack(federado_com_dp)
267
268 #Execucao dos testes de rule based blackbox membership inference
269
270 rule_based_attack(centralizado_sem_dp)
271
272 rule_based_attack(centralizado_com_dp)
273
274 rule_based_attack(federado_sem_dp)
275
276 rule_based_attack(federado_com_dp)
```

12 CÓDIGO FONTE PARA ATAQUES DE INVESÃO DE MODELO

```
import warnings
warnings.filterwarnings('ignore')
import tensorflow as tf
tf.compat.v1.disable_eager_execution()
import numpy as np
import os
import codecs
import pandas as pd
from sklearn import preprocessing
import numpy as np
from sklearn.metrics import confusion_matrix
from sklearn import metrics, neighbors
import math
import seaborn as sns
from sklearn import preprocessing,decomposition
from sklearn.preprocessing import LabelEncoder, OneHotEncoder,
    ↪ StandardScaler,MinMaxScaler
from art.estimators.classification import KerasClassifier
from tensorflow_privacy import DPKerasAdamOptimizer
from art.attacks.inference.membership_inference import
    ↪ MembershipInferenceBlackBoxRuleBased,
    ↪ MembershipInferenceBlackBox, LabelOnlyDecisionBoundary
from art.evaluations.security_curve import SecurityCurve
from art.utils import load_dataset,load_mnist
from google.colab import data_table
from art.attacks.inference.model_inversion import MIFace
import plotly.express as px
import plotly.graph_objs as go
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
    ↪  as LDA
import logging
import os
from typing import List, Optional, TYPE_CHECKING

import numpy as np
```

```
from scipy.optimize import least_squares

from art.attacks.attack import ExtractionAttack
from art.estimators.estimator import BaseEstimator,
    ↪ NeuralNetworkMixin
from art.estimators.classification.classifier import ClassifierMixin
from art.estimators.classification.keras import KerasClassifier
from art.estimators.classification.blackbox import
    ↪ BlackBoxClassifier

if TYPE_CHECKING:
    from art.utils import CLASSIFIER_TYPE

NUMPY_DTYPE = np.float64 # pylint: disable=C0103

logger = logging.getLogger(__name__)

DIR_PATH = "/content/drive/MyDrive/datasets/"
MODEL_PATH = "/content/drive/MyDrive/NF-TON-IOTv2/art"
MODEL_NAME = "NF-TON-IoT-v2_DPKerasAdamOptimizer.central.h5"
PROCESSED_DIR_PATH = f"{DIR_PATH}NF-ToN-IoT-v2/"
model_path = os.path.join(MODEL_PATH, MODEL_NAME);
test = os.path.join(PROCESSED_DIR_PATH, "test_NF-TON-IoT-v2.csv")
train = os.path.join(PROCESSED_DIR_PATH, "train_NF-TON-IoT-v2.csv")

df_train = pd.read_csv(train, skipinitialspace=True)

df_test = pd.read_csv(test, skipinitialspace=True)

def convert_to_binary(y_set, positive_label = 0):
    if not isinstance(y_set, np.ndarray):
        targ_tmp = np.asarray(y_set)
    else:
        targ_tmp = y_set

    return np.where(targ_tmp != positive_label, 0, 1)
```

12.1 ATAQUES DE INVERSÃO DE MEMBROS

```
    BATCH_SIZE = 1000
MODEL_PATH = "/content/drive/MyDrive/NF-TON-IOTv2/priv_eval"
model_path = os.path.join(MODEL_PATH,"NF-TON-IoT-v2_Adam_i{instance}
    ↪ _r1.dp-fed.h5")
TEST_SIZE = 100
instances = [0]
x_init_white = np.zeros((TEST_SIZE,41))
y = np.random.randint(10, size=TEST_SIZE)
for i in instances:
    federado = tf.keras.models.load_model(model_path.format(instance=
        ↪ i))
    #compile_fed_model(federado, noise=i,l2=1)
    classifier = KerasClassifier(model=federado, use_logits=False)
    attack = MIFace(classifier, max_iter=10000, threshold=1.)
    x_infer_from_grey = attack.infer(x=x_init_white,y=y)
    pd.DataFrame(x_infer_from_grey).to_csv(os.path.join(MODEL_PATH,f'
        ↪ MIFace_{i}.csv'))
```

12.2 CONVERSÃO PARA PCA E GERAÇÃO DAS FIGURAS

```
    BATCH_SIZE = 1000
MODEL_PATH = "/content/drive/MyDrive/NF-TON-IOTv2/priv_eval"
model_path = os.path.join(MODEL_PATH,"NF-TON-IoT-v2_Adam_i{instance}
    ↪ _r1.dp-fed.h5")
TEST_SIZE = 100
instances = [.5,5,21,55,89]
x_init_white = np.zeros((TEST_SIZE,41))
y = np.random.randint(10, size=TEST_SIZE)
for i in instances:
    exemples = pd.read_csv(os.path.join(MODEL_PATH,f'MIFace_{i}.csv')
        ↪ )
    pca = decomposition.PCA(n_components=2)
    X_pca = pca.fit_transform(exemples)
    federado = tf.keras.models.load_model(model_path.format(instance=
        ↪ i), compile=False)
    compile_fed_model(federado, noise=i,l2=1)
    y_pca = np.argmax(federado.predict(exemples.iloc[:,1:]),axis=1)
    (X_sample, y_sample) = dataset_subsample(X_train,y_train,len(
        ↪ y_pca))
```

```
    pca_x = pca.fit_transform(X_sample)
    X_pca_sample = pd.DataFrame(MinMaxScaler().fit_transform(
        ↪ pca.transform(X_sample)), columns=['1 componente','2
        ↪ componente'])
    X_pca_sample['source'] = 'Dataset original'
    X_pca_sample['y'] = y_sample.to_numpy()
    X_pca = pd.DataFrame(MinMaxScaler().fit_transform(X_pca), columns
        ↪ =['1 componente','2 componente'])
    X_pca['source'] = 'Dataset gerado com MI'
    X_pca['y'] = y_pca
    frames = [X_pca_sample,X_pca]
    result = pd.concat(frames)
    fig = px.scatter(result,x='1 componente',y='2 componente', symbol
        ↪ ='source', color='source', title=f'={i}')
    fig.update_layout(
    legend_orientation = 'h',
    font=dict(
        size=12,
        family="sans-serif"
    ),
    legend=dict(
        title='',
                    yanchor="bottom",
                xanchor='left',
                y=1,
                x=0,
                orientation="h",
    ),
    autosize=False,
    width=500,
    height=500,
)
    fig.update_xaxes(showticklabels=False)
    fig.update_yaxes(showticklabels=False)
    fig.show()
```