Mariana França Coelho

FEDERATED LEARNING APLICADA EM DISPOSITIVOS COM TECNOLOGIA INTERNET OF THINGS

UNIVERSIDADE D COIMBRA
1 2 9 0

1 2  9 0

UNIVERSIDADE D
COIMBRA

Mariana França Coelho

FEDERATED LEARNING APLICADA EM DISPOSITIVOS COM TECNOLOGIA INTERNET OF THINGS

VOLUME 1

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia e Ciência de Dados, pelo Professor Doutor Tiago José dos Santos Martins da Cruz e Professor Doutor Pedro Henriques Abreu e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro de 2022

Faculdade de Ciência e tecnologia

Departamento de Engenharia Informática

Federated Learning aplicada em dispositivos com tecnologia *Internet of Things*

Mariana França Coelho

Dissertação no âmbito do Mestrado em Engenharia Informática, especialização em Engenharia e Ciência de Dados, orientada pelo Professor Doutor Tiago José dos Santos Martins da Cruz e Professor Doutor Pedro Henriques Abreu e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Setembro 2022

UNIVERSIDADE D
COIMBRA

This page is intentionally left blank.

Agradecimentos

O meu profundo agradecimento a todos que, de alguma forma, fizeram parte da minha caminhada até aqui, em especial:

Aos meus orientadores, o Professor Tiago Cruz e o Professor Pedro Abreu, pela disponibilidade e dedicação investida durante todo meu percurso para desenvolvimento da dissertação. Obrigada pelo constante *feedback*, pela liberdade na busca de conhecimento sobre o tema, lições e entendimentos compartilhados. Alongo o meu agradecimento ao Professor Paulo Carvalho que sanou todas as minhas questões, estendendo a mão sempre que necessário.

Aos meus pais e irmã, Cláudia, Luiz Eduardo e Luíza, agradeço pelo constante e consistente suporte para meu aperfeiçoamento pessoal e profissional, e pelo exemplo indescritível de seres humanos, que me moldaram a ser quem sou.

Agradeço a todos os meus familiares, principalmente à minha avó Augusta, que me ensinou que Portugal é casa, e que a herança familiar é o nosso bem mais precioso. Obrigada aos meus bisavós, Máximo e Candida, pelo desejo em me ver estudar na Universidade de Coimbra, um sonho compartilhado que se tornou realidade. Estendo a dedicação à Amora, o meu amor de quatro patas.

À Georgiana, que, com sua presença constante, me inspirou a ser minha melhor versão. Me deu luz e paz para trilhar um novo caminho que, juntas, vamos percorrer. Obrigada por fazer tudo ser mais fácil.

A todos os amigos brasileiros e portugueses pelo apoio durante esta fase tão importante. Principalmente à Mariana Gianotti que sempre se fez presente, independentemente da distância física.

This page is intentionally left blank.

Abstract

Federated Learning has gained momentum recently because it is highly scalable and can host different devices with heterogeneous data. It also allows collaboration between multiple entities without exposing data. Devices, which have Internet of Things technology, are increasingly present in the daily lives of humans, demonstrating the need for high information processing. From this, we focus on exploring and understanding the implementation of Federated Learning with different frameworks relative to the range of Internet of Things devices. The experiments were performed using the framework Flower to evaluate the parameters. With this, the performance of the PassiveAggressive, Perceptron, SVM, and Logistic Regression models were tested for all UNSW-NB15 (2015), CIC-IDS2017 (2017), TON_IoT (2019), WUSTL-IIOT-2021 (2021) datasets, which provide information of the presence of cyber attacks. We sought to explore the simulated environment in applications of a binary, and multiclass classifier, from Federated Learning techniques, addressing each of the steps and understanding the primary evaluation metric for decision making, f1-score. We run combinations between the presented models and dimensionality reduction techniques to calculate the metric. We present the results obtained from the simulations and the challenges encountered during the steps, concluding that in almost all tests, we obtained better results with the PassiveAggressive model, besides the dimensionality reduction from the importance of the characteristics. For each test performed, we reached an information gain among clients of at least 2% f1-score, denoting the effectiveness of the simulated environment. For future work, we suggested applying scenarios using unbalancing the number of samples distributed to all clients.

Keywords

Federated Learning, Internet of Things, Machine Learning, Cyber security, Edge Computing

This page is intentionally left blank.

Resumo

O Aprendizado Federado vem ganhando força nos últimos anos, principalmente por ser altamente escalável, podendo abrigar diferentes dispositivos, com dados heterogêneos. Também permite colaboração entre múltiplas entidades, sem a exposição de dados. Os dispositivos, que possuem tecnologia Internet das Coisas, estão cada dia mais presente no dia a dia dos seres humanos, demonstrando a necessidade de um alto processamento de informações. A partir disto, nos concentramos em explorar e entender a implementação do Aprendizado Federado com diferentes estruturas, relativo à gama de dispositivos de Internet das Coisas. As experiências realizadas utilizam o *framework Flower* para avaliar os parâmetros. Com isso, o desempenho dos modelos *PassiveAggressive*, *Perceptron*, *SVM* e Regressão Logística foram testados, para todos os conjuntos de dados UNSW-NB15 (2015), CIC-IDS2017 (2017), TON_IoT (2019), WUSTL-IIOT-2021 (2021), os quais fornecem informações da presença de ciber ataques. Buscou-se explorar o ambiente simulado em aplicações de um classificador binário, e multiclasse, a partir de técnicas de Aprendizagem Federada, abordando cada uma das etapas, e criando um entendimento da métrica de avaliação principal para tomada de decisão, *f1-score*. Para calculo da métrica, executamos combinações entre os modelos apresentados e técnicas de redução de dimensionalidade. Apresentamos cada um dos resultados obtidos a partir das simulações, bem como os desafios encontrados durante as etapas, concluindo que em quase todos testes, obtivemos melhores resultados com o modelo *PassiveAggressive*, além de redução da dimensionalidade a partir da importância das características. Por cada teste feito, chegamos a um ganho de informação entre os clientes de no mínimo 2% *f1-score*, denotando a efetividade do ambiente simulado. Para trabalhos futuros foi sugerido a aplicação de cenários utilizando o desbalanceamento de número de amostras distribuídos por todos cliente.

Palavras-Chave

Aprendizagem Federada, Internet das Coisas, Aprendizagem Computacional, Ciber segurança, Computação de Bordas

This page is intentionally left blank.

Conteúdo

This page is intentionally left blank.

Lista de Acrónimos

AI Artificial Intelligence. 1, 26, 27

ANN Artificial Neural Network. 1, 14, 17, 18

CAV Connected and Automated Vehicles. 32

CFL Centralized Federated Learning. 27, 31, 32

CNN Convolutional Neural Network. 32

DDOS Distributed Denial of Service. 23

DFL Decentralized Federated Learning. 27

DNN Deep Neural Network. 31, 35

DOS Denial of Service. 23

DT Decision Tree. 14

FD Federated Distillation. 31

Fed Avg Federated Averaging. 1, 4, 7, 11, 25–28, 30–35, 47, 50–52, 56, 71

FL Federated Learning. xiv, 1–4, 6–9, 11, 14, 16, 22–28, 30–34, 37, 39, 40, 42, 44, 47, 50, 52, 65–68, 71, 72

FL-PQSU Federated Learning with Pruning, Quantization and Selective Updating. 31

FTL Federated Transfer Learning. xiv, 27, 31, 33

GRU Gated Recurrent Units. 34, 35

HFL Horizontal Federated Learning. xiv, 26, 27

IDE Integrated Development Environment. 43

IDS intrusion detection systems. 32

IID Independent and identically distributed. 33, 72

IIoT Industrial Internet-of-Things. 3, 6, 23

IoT Internet-of-Things. 2, 3, 6, 8, 9, 11, 22–24, 28, 30, 32–34, 41, 71

KDD Knowledge-discovery in databases. xiv, 11–13

kNN K-nearest neighbors. 14, 31

LSTM Long short-term memory. 30, 33–35

ML Machine Learning. xiv, xvii, 1–4, 6–9, 11–14, 16, 19–21, 24–27, 30, 31, 33–35, 37, 39, 40, 42, 44–47, 50–52, 60, 67, 71

Non-IID Nonindependent and Identically Distributed Data. 30, 31, 33, 72

OFL Online federated learning. 32, 33

OTL Online transfer learning. 33

PCA Principal component analysis. 21, 41, 45, 51, 68, 69

RF Random forest. 14, 21, 31, 41, 45

RNN Recurrent Neural Networks. 34

SGD Stochastic Gradient Descent. xiv, 15–19, 31, 32

SMOTE Synthetic Minority Oversampling Technique. 21, 45–47

SVM Support Vector Machine. xiv, 14, 16, 17, 34, 35, 51, 63, 65, 68, 69, 72

VFL Vertical Federated Learning. xiv, 27

XSS Cross-site Scripting. 24

This page is intentionally left blank.

Lista de Figuras

This page is intentionally left blank.

Lista de Tabelas

This page is intentionally left blank.

Capítulo 1

Introdução

Hoje em dia, a coleta e armazenamento de dados se torna cada vez mais necessária, para que seja possível viabilizar uma melhor experiência para o usuário. Para que seja viável, se faz imprescindível a curiosidade do ser humano em retirar o máximo de performance das novas tecnologias, para que viabilize de forma ágil a execução de tarefas automáticas. Com isso, surge a necessidade de utilizar Artificial Intelligence (AI), a qual se relaciona com a capacidade de máquinas exercerem atividades inteligentes, podendo executar tarefas com objetivo de auxiliar no desenvolvimento e processamento dos dados de forma eficiente. À vista disto, se faz necessário que a máquina retenha o máximo de informação dos dados disponíveis, para que se execute uma extração de alta performance, utilizando assim, técnicas de Machine Learning (ML). As quais, vem se espalhando por alcançar uma alta capacidade de extrair cruciais informações em diferentes cenários.

A diversa gama de informações, pode ser proveniente de grandes fontes de dados, o que recai em uma busca para viabilizar a extração da informação, se fazendo necessária a aplicação de modelos de de ML para este fim. Artificial Neural Network (ANN) é tipo de modelo de ML bastante requisitado, que fornece uma dinâmica estruturação de processos, que permitem uma redução da complexidade provenientes dos dados, providenciando a coleta de informação em larga escala de forma mais ágil. Porém, a coleta de dados não é uma tarefa fácil, visto que muitas vezes se faz necessário armazenar dados em diferentes locais, isso se da, pela limitação dos espaços físicos de armazenamento. Outro quesito relevante, considerando a coleta e armazenamento tradicional dos dados, é a questão direcionada à privacidade da informação do usuário, muitas vezes retem-se informações sensíveis, as quais podem expor o usuário, como endereço, condição financeira, entre outras informações que combinadas podem ser consideradas sensíveis. Resultando em um processo ainda mais complexo, por ter que garantir a segurança à privacidade dos usuários e, ao mesmo tempo, permitir uma extração das informações por meio da aplicação de ML.

Com isso, emerge um conceito que está a ser bastante cotado que viabiliza o treino de modelos de ML de forma colaborativa, ou seja, todos usuários do ambiente conseguem cooperar para a obtenção da informação. Contando também com uma não exposição de dados, por não necessitar centralizar todas as informações dos usuários, mas sim, solicitando que cada integrante da rede, utilize suas próprias amostras de dados para criar um modelo baseado em suas informações, chamando essa etapa de treino local. Por fim, envia-se os parâmetros para um servidor, que por sua vez utiliza técnicas de agregação, como por exemplo Federated Averaging (Fed Avg), para a obtenção de um modelo global, essa técnica é chamada de Federated Learning (FL).

FL diz respeito a uma aplicação de ML de forma descentralizada, o que permite a inserção

de novos usuários no ambiente sem grandes dificuldades, visto que não se faz necessário etapas de armazenamento dos dados centralizados, só se fazendo necessário o compartilhamento com o servidor de parâmetros de modelos treinados. Com essa grande evolução em relação a escalabilidade, a utilização do FL ganha cada vez mais visibilidade nas suas diferentes aplicações como abordado por Nguyen et al. [1].

A aplicação tradicional do FL é retratada em artigos relacionados com relevância, por viabilizar a escalabilidade, como por exemplo aplicado em casos na área da saúde, cidades inteligentes, transportes autônomos, entre outros. Esses dispositivos inteligentes são objetos físicos conectados entre si, que compartilham em alta velocidade uma grande quantidade de dados com baixa latência, o que leva a conexão entre computadores a nível mundial. O desdobramento da evolução de tecnologia permite que todos tipos de objetos físicos se conectem a partir de uma rede, com capacidade de reunir e de transmitir dados, denominando esse sistema Internet-of-Things (IoT). Tratando-se desta tecnologia é importante ressaltar a necessidade de armazenamento para reter a quantidade massiva de informação, pois, por norma quanto mais informação tivermos, melhor performance dos modelos.

Para a retenção da informação ocorrer, tradicionalmente, utiliza-se um mecanismo que requer a utilização de um armazenamento central, onde se agrupam todos os dados provenientes de todos dispositivos da rede. A partir disso executam técnicas de ML, voltadas para treinar e criar um modelo genérico que pode ser aplicado à todos dispositivos. Esse sistema pode ser chamado de ML centralizado, que por promover a unificação da informação, pode acarretar a situações perigosas, principalmente em qualquer tipo de fragilidade do sistema, que pode afetar diretamente o servidor central.

Com isso, denota-se o grande espaço com aplicações de cenários de estruturas descentralizadas FL. Visto que, cada vez mais surgem novas aplicações para cenários *Big Data* dos dispositivos com tecnologia IoT. A aplicação do FL surge como uma possibilidade de suprir as necessidade de grandes servidores centrais para armazenamento e processamento, visto que se baseia em compartilhamento de informações de forma a não reter dados dos usuários. A informação passa a ser armazenada, processada e analisada, utilizando *Edge Computing*, que permite o processamento dos dados na própria fonte onde são gerados ou, o mais próximo possível da mesma, acarretando em uma redução da latência e do consumo energético. Viabilizando uma conexão mais rápida e eficiente, com maior fluidez da informação.

Neste contexto de *Edge Computing*, em que os dados são transmitidos entre a rede local (armazenamento interno do aparelho) e a nuvem (armazenamento de forma virtual dos dados) pelo *Edge Device/ Server*, que por sua vez, é responsável por orquestrar e avaliar os resultados provenientes da informação recebida dos dispositivos, utilizando a *Internet* para a troca de informação. Essa conexão via *Internet* pode ser afetada por um agente externo, graças a possíveis fragilidades entre o dispositivo que envia a informação até o receptor. O mau funcionamento da conexão pode expor todo o processo a ciber ataques, ou seja, expô-los a uma atividade ilegal de um software informático, promovendo uma invasão de forma maliciosa, podendo infectar o computador de um usuário legítimo e prejudicá-lo.

Os ataques denotam a necessidade de um sistema robusto, que consiga detectar, com antecedência o componente maligno, para ser possível combater o ataque, por exemplo com a interrupção dos serviços que fornecem, ou recebem uma informação do servidor para ser reiniciado, de acordo com a identificação da atividade maliciosa. O que pode ser considerado em tema de alta relevância em cenários IoT, pois a evolução leva a exatidão e rapidez da informação. Com isso, atrela-se grandes necessidades de investimentos em técnicas para detecção e proteções garantindo uma maior credibilidade entre o usuário

e, a empresa que fornece qualquer tipo de serviço, para isso existe a definição de ciber segurança, que promove a proteção de sistemas de computador contra possíveis ataques. O que leva a aplicação de FL como ML descentralizado como uma boa solução em relação a escalabilidade, fidelidade da informação e principalmente promover uma maior segurança para a rede dos *nodes* que muitas vezes possui muita informação sensível dos usuários.

1.1 Motivação

Para ser possível construir uma ferramenta de ML poderosa, é importante que seja coletado o maior número de amostras para viabilizar uma melhor adaptação do modelo, a possíveis descobertas de padrões. A aplicação tradicional, diz respeito ao uso de um servidor centralizado para reter os dados, e retirar aprendizado. Podendo gerar um risco de violação ou vazamento dos dados, já que é responsável por armazenar unificadamente. O risco de exposição dos dados, é considerado um tema muito atual e relevante, visto que um ataque maligno que afete diretamente o servidor, se for bem sucedido, poderá resultar em livre acesso para afetar, apagar, alterar, copiar, entre outras ações que podem vir a causar danos a todos os dispositivos conectados na rede.

A informação dos dispositivos, pode ser entendida como de caráter sensível, ou seja, se exposta, pode causar um colapso de uma organização e, principalmente a sua reputação com seus clientes. Devendo se atender à regulamentação para a União Europeia prevista em *General Data Protection Regulation (GDPR)* [2], criada em Maio de 2018. Demonstrando que novas descobertas devem buscar garantir a não exposição dos clientes, com isso, surge a oportunidade de aplicar o conceito de FL, uma vez que, pode ser uma melhoria em comparação a aplicação convencional de ML, já que prevê que os dados dos utilizadores não deixem seus dispositivos. Entretanto, deve-se avaliar a qualidade da informação enviada ao servidor, visto que, não deve viabilizar a reconstrução da informação a fim de obter acesso a dados sensíveis.

A criação de um ambiente descentralizada em questão, explora a aplicação de FL que é o tema foco da dissertação. Alguns dos principais projetos que deram maior visibilidade ao tema, podem ser encontrados na literatura. Primeiramente podemos identificar a aplicação por parte da Google que utiliza FL no seu teclado *Gboard*, buscando fornecer uma personalização de previsão da próxima palavra, baseando-se na informação do usuário e, também contando com a colaboração dos outros dispositivos [3, 4]. Também encontra-se aplicações por parte da Apple, para auxiliar na classificação e, entendimento das diferentes pronúncias do seu agente pessoal *Siri* com a frase "Hey, Siri"[4]. Em outros cenários, pode-se encontrar FL em temáticas industriais, como em aplicações Industrial Internet-of-Things (IIoT) [5], onde explora-se na *Survey*, lições relevantes com as aplicações industriais, abordando principalmente grandes oportunidades na temática de ciber segurança, onde apresenta Alazab et al. [6] que aborda desafios, possíveis aplicações futuras e o que já existe de mais atualizado na área, denotando que estamos a lidar com um tema promissor. Com isso, percebe-se que grandes empresas estão investindo em FL aplicado à cenários de *smart devices, smart city*, o que denota que a temática IoT possui sua relevância.

Por fim, denota-se uma grande motivação em ganhar conhecimento de possíveis ataques num contexto de ciber segurança. Promovendo o entendimento da qualidade da troca de informação, principalmente devido a colaboração entre os dispositivos, já que exploraremos um cenário que se trata de um ambiente distribuído aplicando FL. Com isso construiremos um sistema inteligente que, permita a classificação da existência de ataque, e também, determinar o tipo de ataque, se aplicável. Para a criação da simulação utilizaremos a

aplicação mais tradicional de FL que diz respeito a agregação dos modelos por parte do *Edge device/server*, utilizando o Fed Avg [7] como principal técnica.

1.2 Casos de Uso

Visto que a motivação da dissertação está direcionada com a identificação de ataques em ambiente distribuído usando técnicas de FL, faz-se necessário o entendimento dos diferentes casos de uso do presente trabalho, fornecendo o entendimento da coleta dos dados, seguido da explicação dos *datasets* propostos, tendo em vista as mais recentes aplicações de cenários de ciber ataques, que ajudarão na construção de uma *pipeline* robusta para o cenário de simulação.

Casos de Uso

Considerando um cenário de *Cybersecurity*, selecionamos alguns dos principais e, mais recentes fontes de dados públicas, considerando também, a sua relevância em aplicações de ML. Todas coletas se baseiam em capturas de tráfego de rede com registros de atividades de ataques de diversos tipos que entraremos em detalhes durante a exemplificação dos *datasets*. A referência de limpeza e pré-processamento dos dados, além da escolha dos *dataset* se deu baseada no artigo Frazão et al. [8].

Para viabilizar um melhor entendimento de cada cenário, ilustraremos e explicaremos brevemente sobre cada coleta das amostras. Visto que dentre todos *datasets* escolhidos existem dados coletados em formatos diferentes, porém, para auxiliar o entendimento escolhemos focar nos dados armazenados no formato **.csv file**. Isso fez-se necessário pela dificuldade da identificação da gama de ataques presente nos *datasets* e, os dados já em formato **.csv** já são disponibilizados com um prévio tratamento, ou seja, com a devida *label*, já informando se aquele registro é um tipo de ataque ou não, e se for ataque qual o seu tipo.

Dataset 1: UNSW-NB15

Temos em Figura 1.1 uma adaptação da coleta dos dados feita por Moustafa and Slay [9] onde temos o *Traffic Generator* que está conectado e distribuindo tráfego normal para o *Server* 1 e *Server* 3, por outro lado, o *Server* 2 é responsável por gerir atividades maliciosas no tráfego de rede. Estabelecendo a conexão com tráfegos de rede pública e privada, se distinguindo por endereços de IP diferentes. Os servidores estão conectados aos *hosts* em dois *routers* 1 e 2 conectados à dois endereços de IP cada. No que lhe concerne, conectados a *firewall* que é responsável por filtrar e examinam as informações provenientes da conexão à *Internet*, podendo assim, receber tráfego normal e atacado.

Por fim, o *router* 1 fornece as capturas de tráfego para a base de dados de *PCAP files*, enviando tráfegos normais e com ataques. A partir desse ambiente, foi possível criar uma automatização que conseguisse extrair 49 *features* com configurações de nove *attacks* diferentes e tráfego normal. Essa base de dados fica armazenada no formato de *csv file* (https://research.unsw.edu.au/projects/unsw-nb15-dataset). Foram utilizados os arquivos com denominação: *UNSW-NB15_1.csv, UNSW-NB15_2.csv, UNSW-NB15_3.csv andUNSW-NB15_4.csv*.

Na coleta, tem-se a presença de diferentes classes, as quais abordam alguns ciber ataques, pode-se considerar que a distribuição de maior para o menor número de amostras se dá na ordem: *Normal*, *Generic*, *Exploits*, *Fuzzers*, *DoS*, *Reconnaissance*, *Analysis*, *Backdoors*,

Shellcode, *Worms*.

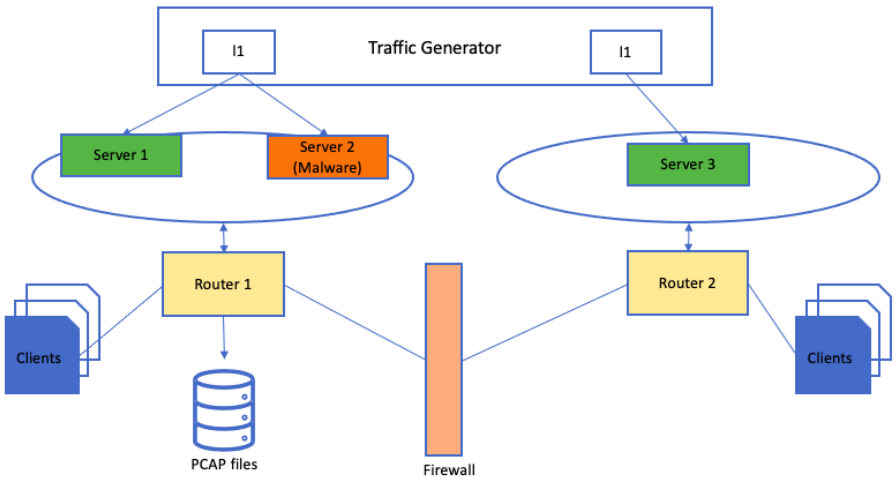

Figura 1.1: Arquitetura do processo de coleta de capturas de tráfego de rede *UNSW-NB15* data. (Adaptado de [9])

Dataset 2: CIC-IDS2017

Para o *dataset* em questão foi feita a coleta diretamente dos dados originais, abordada por Sharafaldin et al. [10]. As capturas foram coletadas de um sistema mais complexo, se comparado a *UNSW-NB15*, com uma quantidade mais expressiva de amostras, com uma coleta da simulação durante cinco dias. A *testbed* possui uma infra-estrutura com uma rede direcionada ao *Victim-Network* e *Attack-Network* utilizando para sua estruturação alguns sistemas operativos, são eles *Windows*, *Linux* e *acintosh*. Neste caso temos *firewall* com IPs públicos e privados. *Attack-Network* possui um *router* e um *switch* conectadas a quatro computadores, e a *Victim-Network* possui três servidores, um *firewall* dois *switches* e dez computadores conectados. A captura de tráfego de toda estrutura se deu em uma porta na rede vítima, que foi armazenado no formato de *PCAP files*. A partir da coleta permitiu a limpeza e conversão em oitenta *features* com a geração de diferentes testes com a presença de variados ciber ataques. Para auxiliar a utilização dos dados utilizamos os disponibilizados com a devida *label*, com isso, utilizamos diferentes *files* que estão no formato *.csv* (https://www.unb.ca/cic/datasets/ids-2017.html), são elas: *Friday-WorkingHours-Afternoon-DDos.pcap_ISCX.csv*,

Friday-WorkingHours-Afternoon-PortScan.pcap_ISCX.csv,

Friday-WorkingHours-Morning.pcap_ISCX.csv,

Monday-WorkingHours.pcap_ISCX.csv,

Thursday-WorkingHours-Afternoon-Infilteration.pcap_ISCX.csv,

Thursday-WorkingHours-Morning-WebAttacks.pcap_ISCX.csv,

Tuesday-WorkingHours.pcap_ISCX.csv,

Wednesday-workingHours.pcap_ISCX.csv.

Na coleta, tem-se a presença das classes: *Benign*, *DoS Hulk*, *PortScan*, *DDoS*, *DoS GoldenEye*, *FTP-Patator*, *SSH-Patator*, *DoS slowloris*, *DoS Slow-httptest*, *Bot*, *Web Attack Brute Force*, *Web Attack XSS*, *Infiltration*, *Web Attack Sql Injection*, *Heartbleed*.

Dataset 3: ToN-IoT

Para o *dataset* tivemos como base a sua extração original que se configura em uma *testbed* [11]. Se baseia na estrutura de um sistema IoT com três *layers of edge, fog* e *cloud*. O dinamismo das três camadas, incluindo os sistemas físicos e simulados, é gerido de forma flexível pelas tecnologias da *SDN* e *NVF*. A solução *NSX-VMware* utilizada para providenciar a aplicação de *SDN*, permitindo assim a criação de redes sobrepostas com as mesmas capacidades das redes físicas. A *VMware* permite a criação e gestão de várias máquinas virtuais que funcionam em simultâneo para oferecer os serviços IoT/IIoT e de rede. A plataforma *NSX vCloud NFV* permite a concepção de uma rede dinâmica de teste IoT/IIoT do ToN_IoT com criação e controle de várias *VMs* para ataques e operações normais, permitindo as comunicações entre *edge, fog* e *cloud layers*. Foram capturadas capturas de tráfego no formato *PCAP* e transformadas em *csv files* (https://research.unsw.edu.au/projects/toniot-datasets). Os arquivos disponíveis, foram extraídos são no formato 'Network_dataset_X.csv' no lugar do *X* temos valores de {1 a 23}, com a presença de 45 *features* coletadas, bem como o *label* informando se aquele registro é um ataque ou não, com seu devido tipo. Encontra-se a presença das classes: *Scanning*, *DDoS*, *DoS*, *XSS*, *Password*, Normal, *Backdoor*, *Injection*, *Ransomware*, *Mitm*, os quais serão explicados mais a frente

Dataset 4: WUSTL-IIOT-2021

Por fim selecionamos um último *dataset* que foi extraído mais recentemente explicado em [12]. Que possui objetivo de emular um sistema industrial o real possível e, permitir a possibilidade de realizar verdadeiros ciber ataques. Pode ser encontrada a ilustração resumida da *testbed* na Figura 1.2. Consisamente pode-se entender o processo industrial como um sistema IIoT que supervisiona o nível e distribuição da água do *Water tank*. Este banco de ensaio inclui componentes como *History Logs*, *HMI*, e *PLC*. Tendo três sensores, dois sensores de nível de água e um sensor de controle da turbina analógico. Existe um alarme de turbidez de três luzes conectado no PLC, uma válvula conectada na *Pump 2*, e duas bombas de água *Pump* 1 e 2 que recebem os comandos do PLC. Também se controla de forma manual quando necessário parar todo o sistema.

Foi recolhido um total de 2,7 GB de dados, em cerca de 53 horas de simulação. Os dados foram armazenados no formato *PCAP* e transformados em *csv files*. Na etapa seguinte foi feito um pré-processamento e limpeza do conjunto de dados (removendo linhas com *missing values*, *errors* e *outliers*. Foi disponibilizado os dados resumidos, que pode ser entendido e ter mais informações em Zolanvari [13] (https://ieee-dataport.org/documents/wustl-iiot-2021) com um total de 41 *features*. Encontra-se no *dataset* a presença das seguintes classes: *Normal*, *DoS*, *Reconnaissance*, *Command Injection*, *Backdoor*.

1.3 Objetivos e questões de Investigação

O foco deste trabalho incide em implementar e analisar a viabilidade de um ambiente distribuído utilizando um computador com recursos limitados. Fornecendo resultados de forma simples, utilizando estratégias de FL focada em aplicação em cenários de ciber segurança. Para a criação da *framework* se faz necessário a utilização de uma estrutura que forneça suporte para a implementação de modelos de ML em dispositivos em um ambiente federado, com isso optaremos pela *framework Flower (Flwr)*.

Para a etapa de aplicações teste, usaremos algumas bases de dados recentes encontradas *Open Source*, para que se viabilize a replicação de forma simples, os dados são explicados em [9, 10, 11, 13], e já tivemos uma breve introdução sobre a coleta de cada um na Seção 1.2.

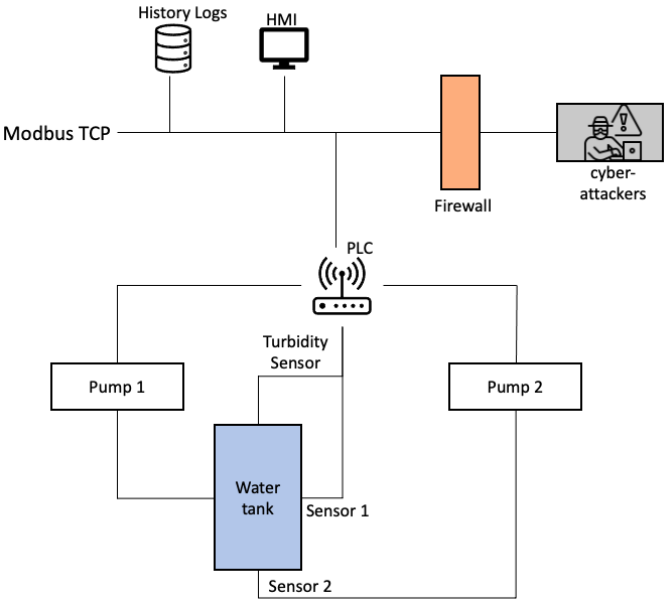

Figura 1.2: Arquitetura do processo de coleta de capturas de tráfego de rede *WUSTL-IIOT-2021* data. (Adaptado de Zolanvari et al. [12])

Entende-se que se por alguma razão não tivermos dados suficientes, iremos os tentar gerar novas amostras, para moldar a distribuição entre as classes. Pode-se executar essa tarefa, com técnicas de introdução de réplicas de exemplos da classe minoritária, bem como possivelmente técnicas de normalização dos dados.

Provendo ao fim, a metodologia de atuação para cada *dataset*, o seu processamento e quantidade de ciber ataques selecionada, de acordo com a quantidade de amostras significativas de cada ataque que os diferentes *datasets* trazem. Dividindo a aplicação em cenários com casos de classificação binária, ou seja, existir ataque ou não, e classificações multi-classe, permitindo a informação de qual ataque específico estamos lidando. Com isso, o ambiente simulado de FL explorará os pontos positivos da aplicação e possíveis problemas com a capacidade de processamento.

Para termos um guia, os resultados esperados com a dissertação focaremos em algumas questões cruciais, que são:

1. Como cada etapa do ambiente proposto se relaciona?

2. Como o algoritmo de agregação Fed Avg performa? Qual a evolução para os clientes devido a colaboração?

3. Qual o ganho de informação para os dispositivos que inicialmente (*round* 0) possuem a pior *performance* na métrica de avaliação (*f1-score)*?

4. Como se da a evolução das métricas de avaliação, considerando um envio de novos dados a cada *round*, para os cenários de classificação binários, e nos de multi-classe? Qual o melhor modelo ML para cada fonte de dados utilizado?

5. Considerando os cenários multi-classe, quais foram os ciber ataques de maior complexidade na identificação? Porquê?

1.4 Planejamento

Para esta seção iremos descrever as atividades realizadas no primeiro semestre da dissertação, seguido de justificações que fizeram o plano ser alterado. Desta forma iremos apresentar um gráfico de *Gantt* do trabalho feito no primeiro e segundo semestre da dissertação. Entende-se que o gráfico é responsável por denotar o tempo investigo em cada uma das tarefas propostas.

Primeiro semestre

Para o primeiro semestre, pode-se dizer que se baseou em entendimento do tema FL e diferentes variações de cenários de aplicação. Foi criado uma simulação simples para avaliação de um *dataset* voltado para um cenário de IoT. Podemos identificar as atividades executadas no primeiro semestre na Figura 1.3. Na figura temos em azul ao tempo esperado para cada atividade, e em roxo a extensão da atividade até o período que se fez necessário.

Durante o primeiro semestre, se fez necessário estender a duração de algumas atividades, como, por exemplo: a atividade inicial que diz respeito ao entendimento com o ambiente em que o FL se insere, como se trata de uma técnica que ainda não tinha sido explorada por mim, acabou levando mais 10 dias do que o planejado; levou-se mais tempo também para a primeira replicação de um caso de uso simples, por necessitar de conhecimentos prévios que ainda não eram conhecidos; A etapa de escolha de um teste preliminar, foi um desafio, por ser baixo o número de exemplificações simples sobre o tema, FL; para o ensaio preliminar, fizemos uma análise de um cenário com dispositivos IoT, considerando um número pequeno de amostras, por na altura estarmos com grandes problemas em questão do recurso computacional, o que levou a demora de mais de 8 dias do que planejado.

TAREFA	TIPO DE TAREFA	PROGRESSO	INÍCIO	TÉRMINO
1 Semestre				
Tarefa 1	Ambientação estado da arte, entendimento sobre FL	100%	15/9/21	15/10/21
Tarefa 2	Ambientação com tecnologias de orquestração e gestão do ambiente distribuído	100%	16/10/21	31/10/21
Tarefa 3	Ambientação com um ambiente simples	100%	1/11/21	1/12/21
Tarefa 4	Escolha de um caso preliminar para testes iniciais	100%	2/12/21	17/12/21
Tarefa 5	Ensaio preliminar utilizando um cenário IoT para aplicação do FL	100%	18/12/21	2/1/22
Tarefa 6	Escrita entrega preliminar	100%	25/12/21	24/1/22

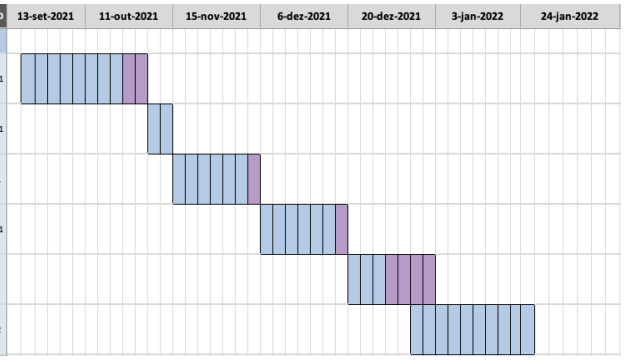

Figura 1.3: Planejamento do primeiro semestre, usando gráfico de *Gantt*

Segundo semestre

Para o segundo semestre, tivemos mais energia gasta à dissertação em relação à criação do ambiente e diversas técnicas que viabilizam a melhor aplicação de cenários, aplicação de uma *pipeline*. A qual se fez necessário entendimento e aplicação de ML centralizada e após finalizada, adaptada para um cenário de FL utilizando como base a *framework* proveniente de uma biblioteca disponível pelo Python, chamada *Flower*. Por fim, reorganizamos o relatório preliminar e adicionamos toda a informação e resultados obtidos com a simulação. Podemos identificar as atividades executadas no segundo semestre na Figura 1.4.

Durante o segundo semestre, a maioria dos percalços que surgiram foram: em primeiro

momento tentamos explorar um caso de uso, porém ainda não estava processado, ou seja, estavam com dados sem suas devidas *labels*, o que dificultaria muito o trabalho, acabou-se optando por recolher fontes públicas para o trabalho, que engobassem a temática IoT com informações de ciber ataques; outra questão foi o entendimento da *framework Fower (Flwr)* levando a busca por informações e entendimento principalmente na passagem dos parâmetros na etapa de agregação; ao criamos o ambiente tomando para avaliar os *datasets* selecionados, tivemos que adaptar diversos processos, que precisavam ser executados de forma constante nos clientes, além de garantir que o cliente passe adiante para o servidor seus melhores parâmetros coletados até o momento.

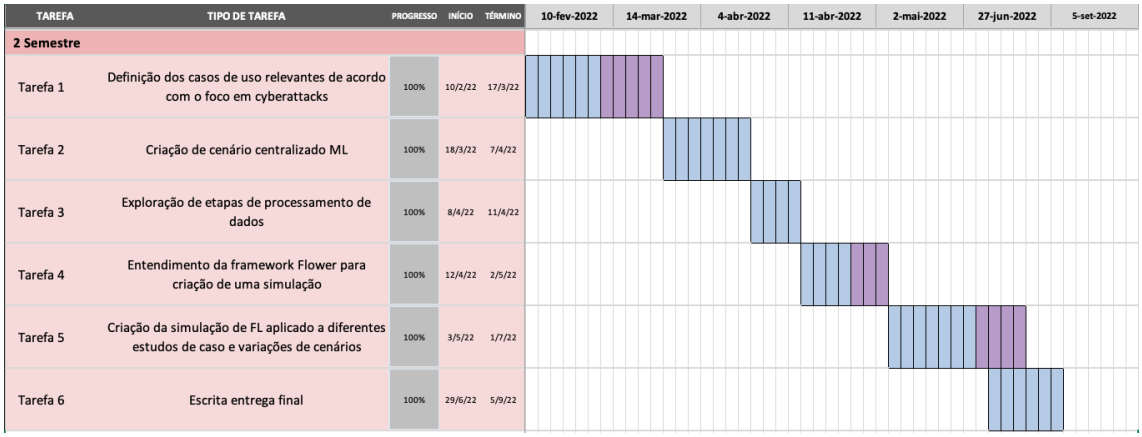

Figura 1.4: Planejamento do segundo semestre, usando gráfico de *Gantt*

1.5 Estrutura do Documento

Os restantes conteúdos desta dissertação estão organizados da seguinte forma: **Capítulo 2** apresentamos os conceitos principais para apoiar o presente trabalho, com foco em diversos entendimentos no quesito IoT, FL e *cybersecurity*. Seguido do **Capítulo 3**, com a revisão de artigos sobre os temas mais importantes, bem como tipos de modelos de ML mais utilizados, seguidos do entendimento da melhor métrica de avaliação, de acordo com o cenário aplicado. A seguir, **Capítulo 4** descreveremos a metodologia da simulação criada, com a exploração dos dados dos estudos de caso escolhidos. Com isso, temos o **Capítulo 5** que aborda nos diferentes cenários criados, os resultados obtidos. Por fim, temos a conclusão que de forma concisa debate as questões relevantes de investigações e trabalho futuro.

This page is intentionally left blank.

Capítulo 2

Fundamentos Teóricos

Neste capítulo, abordaremos primeiramente o conceito KDD (Seção 2.1) especificando cada uma de suas etapas, detalhando as mais relevantes para a dissertação. Explorando cada um dos modelos de ML, que serão utilizados com sua respectiva teoria. Seguindo, iremos adentrar em alguns aspectos relevantes para obtermos uma classificação de qualidade (Seção 2.2), bem como, técnicas a serem aplicadas aos dados, com objetivo de alcançar um melhor resultado de métricas de avaliação. Logo após, iremos adentrar no conceito de IoT (Seção 2.3), detalhando mais a fundo alguns de diferentes agentes (*Malware*) que podem vir a executar uma atividade ilegal e entre a comunicação dos dispositivos inteligentes, focalizando nos ciber ataques presentes nas *testbeds* dos *datasets* selecionados (Seção 1.2). Por fim, temos a definição de FL, como também de seu algoritmo de agregação mais popular da literatura, Fed Avg. Para viabilizar a percepção da estrutura selecionada, voltada para apoiar o ambiente federado, teremos a introdução da *framework Flower (flwr)*, que será utilizada para as simulações.

No contexto atual de novas tecnologias e conhecimento, temos que o aprendizado, diz respeito a receber informações e reter lições em um domínio particular, esse tema se relaciona diretamente com uma atividade própria dos humanos. Os seres racionais possuem a capacidade de aprender com seus erros e com aprendizados que são submetidos durante sua trajetória de vida. Com isso, emerge o conceito de aprendizagem de máquina (ML) que está diretamente ligada à habilidade de uma máquina receber novos conhecimentos ou até aperfeiçoar algum entendimento anterior, de acordo com o aprendizado retido proveniente das novas informações. Os conhecimentos recebidos, estão normalmente relacionados com estudos científicos de algoritmos, e modelos estatísticos de acordo com uma tarefa específica, pode ser chamado de ML. Esse aprimoramento do entendimento pode receber dados provenientes de diversas fontes, e considerando o cenário em que a dissertação se insere, temos dados coletados provenientes de dispositivos ligados à *Internet*, conhecidos como IoT. E estão ganhando cada vez mais espaço, com os avanços tecnológicos, por serem capazes de reunirem, e transmitir dados em tempo real.

Dito isso, é importante começarmos a entender mais sobre o processo da extração da informação, proveniente de diferentes fontes de dados. Para isso, aplica-se à base de dados, diferentes processos, para que seja possível extrair propriamente conhecimento de interesse aos especialistas.

2.1 Knowledge Discovery in Databases (KDD)

Para que se possa melhor compreender cada uma das etapas pertencentes ao processo de extração da informação. Pode-se explorar mais a fundo o conceito KDD, que é composto por cinco grandes etapas, as quais podem ser visualizadas na Figura 2.1.

1. Seleção dos dados: Corresponde ao entendimento do contexto do problema e escolha do foco.

2. Pré-processamento: Composto de técnicas para limpeza dos dados, para sanar alguns problemas que possam ter vindo nos dados originais, bem como erros, dados faltantes ou duplicados e, possivelmente com algum tipo de ruído, ou seja, que por alguma razão possam vir a ter sido corrompidos alterando a real informação. Para ao fim dos processos, alcançar um conjunto de dados que possa representar o objetivo definido na etapa 1.

3. Transformação: Redução da dimensionalidade dos dados e orquestrar onde os armazenar.

4. *Data Mining*: Etapa responsável por explorar e analisar Dados, para ao final alcançar e reconhecer os padrões.

5. Avaliação e Interpretação: Entendimento e avaliação dos resultados obtidos na etapa anterior, onde alcançamos por fim o conhecimento desejado.

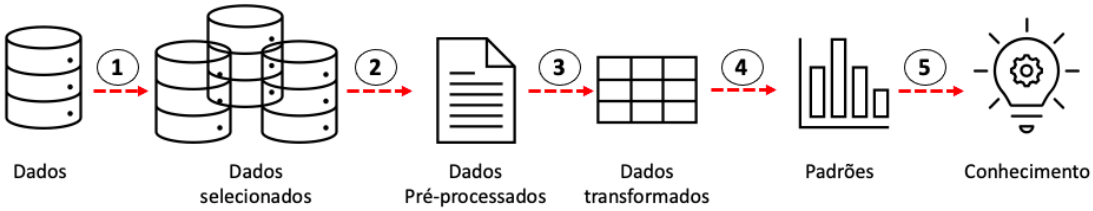

Figura 2.1: Etapas do processo de KDD sendo elas: 1 Seleção, 2 Pré-processamento, 3 Transformação, 4 *Data Mining*, 5 Interpretação e Avaliação (Adaptado de Fayyad et al. [14]).

A partir do entendimento do processo KDD iremos focar em algumas das etapas e, explorar técnicas que se fazem necessárias para o contexto da dissertação. Iniciaremos com a etapa de pré-processamento, seguido de *Data Mining* complementada com explicações mais a fundo de modelos de ML e suas principais métricas de avaliação.

2.1.1 Pré-processamento

Etapa responsável pela limpeza e organização dos dados, com isso, algumas técnicas devem ser utilizadas para garantir que a partir desse passo tenhamos dados já organizados e prontos para dar continuidade no processo, proposto na Figura 2.1.

A partir dessa etapa podemos remover dados com erros, por exemplo, quando se possui um projeto que já se tenha um entendimento prévio das *features*, pode ser necessário apagar algum registro mal coletado, bem como fora de possíveis limites.

A técnica de verificação de dados faltantes pode ser lidada de várias formas diferentes, se tivermos uma base de dados com poucas amostras, ou seja, toda e qualquer informação ali dentro é altamente necessária, pode-se aplicar algumas técnicas de adicionar dados de forma aleatória, utilizando outras amostras mais similares como base. Outra técnica é, adicionar valores com base em valores estatísticos como média/moda/mediana, temos que nos atentar somente ao tipo de dados de cada *feature* e, muitas vezes aplicar técnicas específicas. Pode ser usado também, uma regressão para identificar de acordo com uma previsão, qual deveria ser o valor daquela amostra específica, existem outras técnicas que foram exploradas por Patrician [15].

A partir da limpeza dos dados, é muitas vezes necessário que se faça uma normalização dos mesmos porém, não é possível normalizar tipos de dados categóricos. Para isso, devemos aplicar uma técnica que garanta que os dados categóricos sejam representados, ou seja, convertemos cada valor categórico em uma nova coluna categórica, e atribui-se um valor binário de 1 ou 0 a essas colunas. Cada valor inteiro é representado como um vetor binário. Esse processo de tratamento das *features* categóricas é chamado de *One-Hot Encoding*, usado para auxiliar em uma melhor aplicação dos modelos de ML, bem como, na precisão de classificação de um modelo. Podemos acompanhar o processo de forma visual na Figura 2.2.

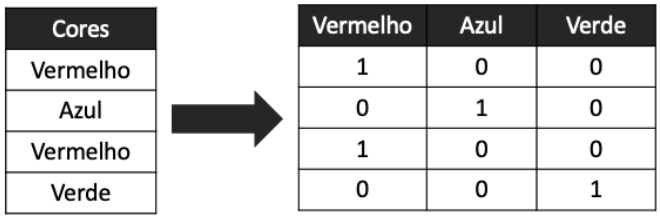

Figura 2.2: Aplicação da técnica *One-Hot Encoding*.

2.1.2 *Data Mining*

Para iniciar essa seção é importante ressaltar que *Data Mining* faz parte de um processo de extração de informação, e esta presente na quarta etapa deste ciclo, chamado KDD explicado por Fayyad et al. [14] e contextualizado na Seção 2.1.

Data Mining é responsável por identificar padrões, a partir dos dados coletados. Isso se dá, utilizando técnicas inteligentes para extrair os padrões, podendo ser técnicas de classificação, *clustering*, entre outros. Dito isso, pode-se entender que algumas aplicações de ML [16, 17] e são brevemente explicados abaixo. Deve-se ter atenção que para a dissertação iremos explorar uma classificação, ou seja, lidaremos com um cenário de *Supervided learning*.

1. *Supervised learning:*
   Usado quando se possui conjuntos de dados, *dataset* com um exemplo de saída, onde o modelo aprende a partir de encontrar os padrões dos dados. Isto auxiliar o modelo a aprender e, portanto, a fornecer facilmente o resultado do problema. Podendo ser utilizado num cenário de Classificação e de problemas de Regressão. A grande diferença dentre eles é que, na Regressão, procura-se estimar um valor numérico e, não uma classificação estima-se uma observação.

2. *Unsupervised learning:*
   Quando se lida com conjunto de dados que não estão com suas devidas *labels*, ou

seja, dados não estão categorizados, se faz necessário aplicar técnicas para encontrar padrões e, fazer algumas previsões sobre os possíveis resultados. Pode-se visualizar seus padrões aplicando técnicas de *Clustering* num sistema de recomendação, ou redução da dimensionalidade num cenário de *Big Data*.

3. *Reinforcement learning:*
   Possibilita máquinas a reterem aprendizados e ensinamentos, através de maximizar a noção de recompensa cumulativa, a partir da interação com o ambiente por si só. Podem ser utilizados em tempo real, como em carros autônomos, ou tarefas de aprendizagem como aspirador robô que se planeja para limpar a casa.

Modelos de *Machine Learning*

Como dito anteriormente, iremos abordar um cenário de classificação, ou seja, utilizando aplicações voltadas para *Supervised learning*. Pode-se considerar que nossos *datasets* são organizados com suas devidas *labels*, ou seja, cada registro terá sua classe atribuída. A partir disso, utilizaremos as informações de cada dispositivo, buscando perceber o ganho devido a colaboratividade dos mesmos, aplicando um cenário descentralizado FL. Com isso, exploraremos algoritmos de ML para viabilizar a classificação.

Para perceber algumas técnicas de ML, temos a Tabela 2.1, que apresenta os breves conceitos de modelos Caruana and Niculescu-Mizil [18]. A partir da explicação dos mesmos, daremos maior ênfase e entraremos no detalhe dos selecionados para a dissertação, por fim, descreveremos diferentes métricas de avaliação.

Modelo	Conceito
K-nearest neighbors (kNN)	Responsável pelo cálculo do vizinho mais próximo para reconhecer e fornecer os padrões.
SVM	Busca separar as classes mais "diferentes"encontrando o hiperplano que melhor as diferencie.
Decision Tree (DT)	Forma de expressar as mesmas regras que são obtidas quando se constrói uma tabela, mas em formato de árvore.
Random forest (RF)	Construção de uma infinidade de árvores de decisão correspondente à tarefas.
Logistic Regression	Utilizado para prever a probabilidade de uma variável alvo.
ANN	Conjunto de neurônios que podem ou não ser ativados dependendo do *input* da entrada. A informação adquirida passa para os outros neurônios, até finalmente, um neurônio de saída ser ativado.

Tabela 2.1: Modelos de ML *Supervised Learning* com os respectivos conceitos

Tratando-se de aplicações *Supervised learning* podemos entender que a sua *loss function* se baseia em, o quão boa a previsão alcançou, se for comparada a *label* correta. Os parâmetros focam em minimizar a *loss function*.

O processo para validação das previsões se dão a partir de treino dos dados e, utilização de uma parte ainda não vista para testar o modelo treinado. Com isso, alguns modelos usam a aplicação de um método denominado como *Gradient Descent* [19]. O qual se baseia em, um algoritmo que de forma iterativa, segue em direção a buscar melhores resultados.

Primeiramente, iniciado de forma arbitrária, e a cada iteração a derivada parcial da *loss* é calculada, bem como os parâmetros do modelo são computados. Com isso, a cada nova iteração iremos nos aproximando para o cenário ideal, buscando uma classificação com 100% de exatidão.

O conceito de seguir o gradiente para o cenário de otimização será utilizado por todos modelos da dissertação, por viabilizar a cada iteração o ajustamento dos parâmetros. Porém, utilizaremos SGD, que por sua vez performa em partes os dados, ou seja, a partir de um ponto aleatoriamente selecionado, segue-se para um ponto de otimização arbitrário, para isso normalmente é aplicado em modelos que aceitam partes do *dataset* original para treino de forma incremental. Utilizando o conceito de *batch* que se refere a coleção de dados, selecionada de forma aleatória para treino, que será utilizada na ronda. Pode-se entender que *epoch* se relaciona a quantidade de vezes que treinaremos o conjunto de dados e, que são selecionados a cada *batch*, visando a melhor combinação de parâmetros, baseado na menor *loss function*.

Para se compreender a diferença entre *Gradient Descent* e SGD, podemos considerar um cenário em que temos mil amostras de dados. Se aplicarmos um *Gradient Descent* tradicional, teremos que utilizar as mil amostras de dados a cada *epoch*, até alcançarmos o mínimo. O que faz com que se estivermos a trabalhar com um grande número de dados, torna-se algo computacionalmente inviável. O que pode ser elucidado ao aplicarmos SGD pois, utilizamos uma parte dos dados a cada rodada, essa coleta de dados é uma parte selecionada de forma aleatória dos dados de treino, previamente misturados.

Então, pode-se entender que SGD, a cada iteração providencia uma *loss function*. Por outro lado, o *Gradient Descent* se dá pela soma de todas as *loss function* para todas as amostras. Uma aplicação de *Gradient Descent* é *Batch Gradient Descent* [19], que pode ser entendido por, tomar a média de todos gradientes medidos a partir dos dados de treino e usar essa média para atualizar os parâmetros. Isso diz respeito a um passo do cálculo do *Gradient Descent* dentro de um *epoch*.

Para ilustrar as diferenças abordadas por Ruder [19] podemos identificar a Figura 2.3 que ao utilizar todos os dados em cada *epoch* temos a evolução do *Batch Gradient Descent* de forma contínua, porém, se torna muito dispendioso. Por outro lado, temos uma aplicação do SGD que não segue propriamente o gradiente de forma linear porém, busca melhorar o passo anterior de forma mais rápida, por considerar somente uma parte do *dataset* de treino. Por fim, podemos perceber que ao utilizarmos o SGD pode-se dizer que podemos alcançar bons resultados e principalmente um *trade-off* entre tempo de processamento dos dados de treino e alcançar um resultado de alta performance.

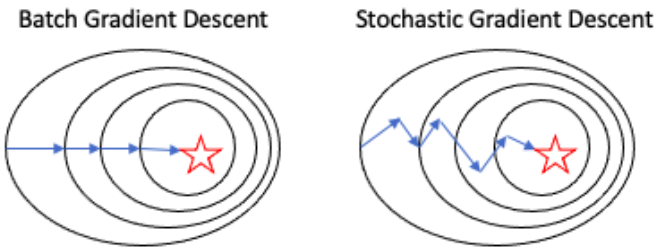

Figura 2.3: Uma exemplificação da aplicação de *Batch Gradient Descent* à esquerda e SGD à direita.

Para a dissertação iremos utilizar variações de aplicações de SGD com diferentes algoritmos porém, buscando através do gradiente uma menor *loss function*. Isso se dá pela aplicação

do SGD permitir que a cada iteração nova, possamos decidir se iremos seguir a partir de parâmetros pré-definidos ou, se iremos começar um processo sem informação prévia, ou seja, valores vazios para os parâmetros.

Primeiramente, é importante percebermos alguns termos técnicos em o que consiste uma classificação. O poder e o entendimento que o processamento dos dados pode fornecer, dito isso, temos que um determinado *dataset* que é um conjunto de dados já com sua *label*, em formato matemático se entende por $D = (x_1, y_1), ..., (x_n, y_n)$ onde x_n corresponde as *features* que serão avaliadas e que de acordo com a sua combinação nos fornece um *output*, o qual diz respeito a uma determinada classe aquela amostra faz parte. Se for considerado um cenário binário, podemos dizer que y varia entre 0 e 1, temos que normalmente x corresponde à um conjunto de *features* que são independentes entre si.

A partir do conhecimento dos dados, podemos aplicar diferentes modelos de classificação, alguns direcionados à distinção das classes e, outro diz respeito a probabilidade associada à cada classe $P(y|x)$. Dreiseitl and Ohno-Machado [20] fornece um complemento de informações importantes sobre cada modelo. Outra fonte recente sobre tipos de classificadores, Kiranmayee et al. [21]. Com o entendimento prévio, iremos agora entrar no esclarecimento de como os quatro modelos selecionados funcionam, bem como sua possível aplicação em cenários de FL.

Support Vector Machine (SVM):

É um dos algoritmos de ML mais comumente aplicados a problemas de classificação. Pode se fazer entender ao fornecermos um conjunto de amostras para treino, cada dado possui o seu *label* que pode ser um cenário binário ou multi-classe como pode ser visto na Figura 2.4 respectivamente representado.

Podemos encontrar a utilização do SVM em diversos cenários de classificação, porém, mais recentemente vem sendo explorados na temática abordando FL [22, 23, 24]. Podemos identificar que existem algumas aplicações em cenários simulados. A principal aplicação do modelo é utilizando *SGD-based* o que se refere a aplicação de métodos de otimização. Ao utilizarmos essa metodologia, pode-se entender que, após o treino local dos clientes, ele troca com o servidor parâmetros que dizem respeito a valores do gradiente, e será feito a agregação a partir dos parâmetros de todos clientes. Para computar o SGD em um cenário de utilização do SVM, temos que optar pela *loss function: hinge*.

Para ilustrar o funcionamento do modelo SVM em um cenário binário, tendo somente duas opções de classes, como pode ser visto na Figura 2.4 do lado esquerdo. Temos duas classes verde e laranja, os pontos no espaço representam cada amostra, que é mapeada de maneira que os exemplos de cada categoria sejam divididos por um espaço maior possível, quanto maior a distância, entre as classes, melhor podemos dizer que o modelo está. Ao considerarmos a inserção de novos dados amostrais, faz-se um mapeamento e as distribuímos de acordo com a melhor representação, baseada nos conhecimentos obtidos com os dados de treino. Essa previsão denota a qual classe cada elemento é baseado em qual o lado do espaço eles são colocados.

Em um cenário multi-classe, que pode ser identificado ao lado direito na Figura 2.4 podemos ter o mesmo raciocínio, porém a separação deve ser a maior possível entre todas as classes presentes. Com isso, passamos a ter um conjunto de hiperplanos para a separação das mesmas.

Vantagens na aplicação do *svm* é por conseguir facilmente desconsiderar *outliers*, por ser focado na separação das classes presentes. Por outro lado, existe desvantagem ao seu uso, que é abordada por Dreiseitl and Ohno-Machado [20] que se dá pelo modelo em questão

não se basear na probabilidade da amostra pertencer a uma classe.

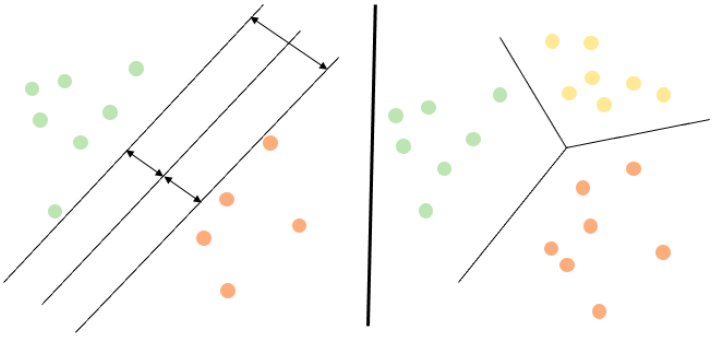

Figura 2.4: A esquerda temos um exemplo de aplicação do SVM para uma classificação binária e a direita uma multi-classificação.

Logistic Regression:

Pode ser entendido como a regressão voltada para um *output* categórico. As técnicas conhecidas em regressão podem ser encontradas para dados contínuos porém assumindo o nome de regressão linear. Na Figura 2.5 podemos identificar uma classificação binária e um exemplo de multi-classe respectivamente.

Com uma ideia de atribuir a cada classe uma probabilidade associada ou seja, se binário podemos usar 0 e 1, com isso, utilizamos a parte dos dados de treino do modelo para alimentar o nosso classificador. Com isso, ao prever as probabilidades das amostras de testes, teremos valores entre 0 e 1 e atribuímos a classe que mais se aproximar daquela amostra.

Essa atribuição de probabilidade se dá por uma *cost function* chamada *sigmoid*. Quanto maior a separação das classes, mais fácil o classificador dará um *output* sem erro. Para a aplicação do modelo *Logistic Regression* utilizando SGD focado em otimização, temos que optar por uma *loss function: log_ loss*.

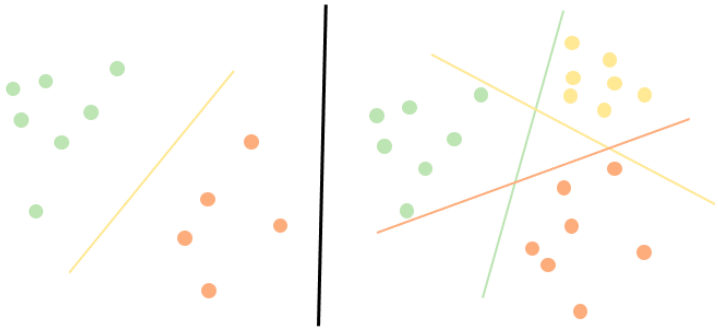

Figura 2.5: A esquerda temos um exemplo de aplicação do *Logistic Regression* para uma classificação binária e a direita um cenário de multi-classificação.

Artificial Neural Network:

A utilização de ANN em cenários com grandes quantidades de dados vem sendo cada vez mais relevante, por possuir alta performance. Conseguindo retirar conclusões mais precisas por sua maior complexidade em reter informação no seu neurônio. Porém, deve-se considerar a complexidade ao formarmos uma rede neural e o recurso computacional que sua aplicação requer.

Tendo em conta o objetivo da dissertação de trazer a aplicação do ambiente federado, utilizando um computador com baixo poder computacional, se comparado com *high-end laptops* e *desktops* com CPUs e GPUs mais rápidos. Baseando-nos em aplicações mais recentes da literatura, onde denota-se desafios do ambiente base *Flower (flwr)*, optou-se por forcar em aplicações simples, utilizando *Perceptron* e *Passive Aggressive*.

Perceptron:

Conhecido por ser um classificador linear, que se assemelha de uma forma geral a uma aplicação do *Logistic Regression*, ou seja, um algoritmo de classificação que faz as suas previsões com base numa função de previsão linear combinando um vetor de *weights* com o vector de *features*.

Classificador normalmente utilizado em cenários binários, porém também utilizado como multi-classificadores. É responsável por decidir se um *input*, que é representado por um vetor de *features*, pertence ou não a alguma classe específica.

Perceptron é uma representação simples de uma aplicação de ANN, com isso, ilustramos uma representação na Figura 2.6. Identificamos os quadrados (*inputs*) correspondentes a combinação das *features* dessa classificação, o *Hidden Layer* que se apresenta na aplicação de um *Multi-Layer Perceptron*, e é responsável por através dos *weights*, determinar o possível *output* que aquela amostra se aproxima. Na aplicação mais simples do *Perceptron*, não possuímos o *Hidden Layer*, somente o *input* os *weights* e com a combinação da informação, obtemos um *output*.

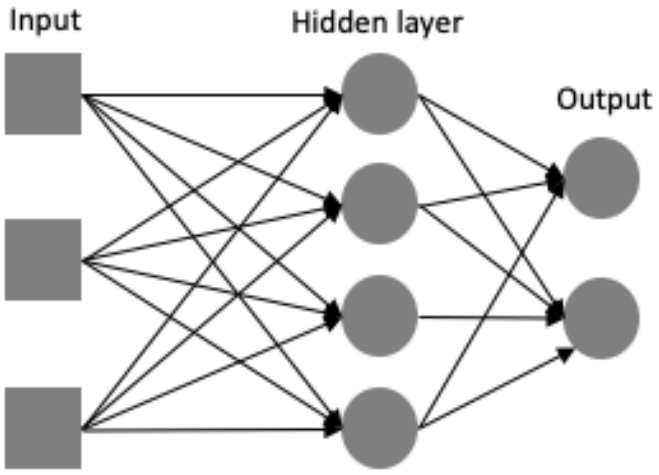

Figura 2.6: Podemos verificar o Input conectado à um *Hidden Layer* e que por fim atribui um *Output*.

Ao considerarmos a possibilidade de uma multi-classificação temos que x representa o *Input* com isso é atrelado a um *output* y e o *Hidden Layer* responsável por mapear todas as possibilidades de combinações *input/output* para chegarmos a uma dimensão finita de possibilidades. A partir disso, temos que o vetor das *features* é multiplicado pelo vetor dos *weights*, para ao fim chegar ao *output* considerando todas as possibilidades de cada uma das classes. Para a utilização do *Perceptron* utilizaremos o algoritmos baseado em aplicação do SGD com sua *loss function = perceptron*.

Passive Aggressive Classifier:

Faz parte dos algoritmos que fazem *updates online*, ou seja, funciona como um processo

de validação constante da previsão, respondendo como *Passive* para classificações corretas e, *Aggressive* para qualquer erro de cálculo. Como podemos identificar a representação na Figura 2.7 temos a ilustração do modelo se auto corrigindo e, buscando alcançar a correta divisão das classes, sem cometer erros.

Para a utilização do *Passive Aggressive* utilizaremos o algoritmos baseado em aplicação do SGD com sua *loss function = epsilon_insensitive*.

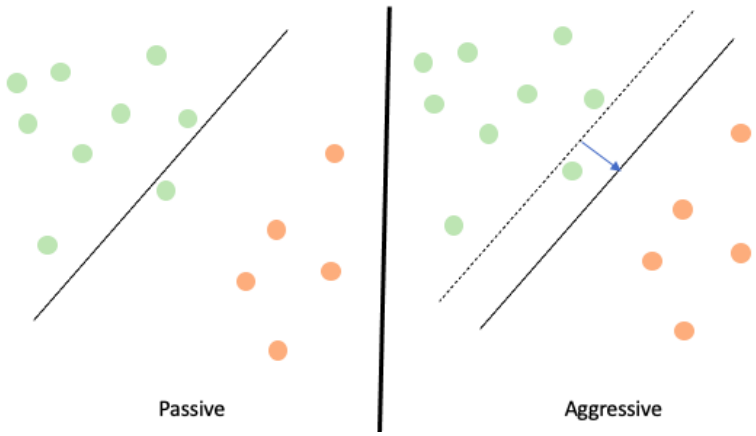

Figura 2.7: Temos a aplicação do modelo *Passive Agressive*, do lado esquerdo temos o cenário *Passive* que configura que o classificador não cometeu erros, e do lado direito temos *Aggressive* que configura um cenário de corrigir o erro do classificador.

A partir dos modelos detalhados acima, temos quatro aplicações baseadas na utilização do SGD com sua devida variação em relação a *loss function*. A partir dos modelos, é de extrema importância explorarmos as métricas de avaliação, as quais serão nossa base para determinar o sucesso ou não de determinado modelo na classificação.

Métricas de avaliação:

ML é um campo que dependendo do contexto onde é aplicado, pode-se utilizar diferentes métricas para medir o desempenho dos algoritmos. Apresentamos aqui as métricas mais comuns para avaliar seus cenários.

Antes de compreender as fórmulas da métrica de avaliação, é importante compreender que considerando um problema binário de classificação, podemos ter resultados previstos corretamente e incorretos. Se o modelo prevê que é um cenário real e é a sua verdadeira classe, então temos um *TP*. Pelo contrário, se o modelo prevê o resultado positivo de forma incorreta, temos um *FN*. *TN* quando o modelo prediz corretamente um resultado negativo. Finalmente, se o modelo prevê um resultado negativo de forma incorreta, obtemos um *FP*. Para exemplificar a classificação das previsões, temos a Figura 2.8. A partir da matriz de confusão, podemos aplicar algumas diferentes fórmulas matemáticas para avaliar o cenário em que se insere a classificação, temos algumas delas na Tabela 2.2.

2.2 Processamento dos dados

Ao considerarmos um cenário de aplicação *Supervided Learning*, pode-se utilizar algumas técnicas para auxiliar na obtenção de um melhor desempenho. Com isso, apresentamos na Figura 2.9 uma ilustração de um processo para a execução de uma classificação com armazenamento de dados centralizado.

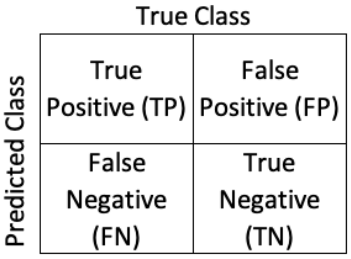

Figura 2.8: Matriz de confusão para um problema de classificação

Fórmula	Conceito
$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$	Fração de previsões que o modelo acertou.
$Precision = \frac{TP}{TP+FP}$	Fração de previsões corretas, pelo total de observações positivas previstas
$Recall = Sensitivity = \frac{TP}{TP+FN}$	Fração de previsões que o algoritmo previu como positivo, e era de classe positiva
$F1 - score = \frac{2*Precision*Recall}{Precision+Recall}$	Média ponderada entre *Precision* e *Recall*
$Specificity = \frac{TN}{FP+TN}$	Fração para medir a taxa de verdadeiros negativos
ROC-curve = (True Positive Rate / False Positive Rate)	*Trade-off* entre *sensitivity* e *specificity*

Tabela 2.2: Métricas de avaliação para os modelos de ML com os seus conceitos

Na figura, pode-se perceber que primeiramente, deve-se decidir o foco do problema, a partir disso, é fundamental que busque fontes de dados com o máximo de informação relevante para o problema em questão. esses dados são chamados de *Raw Data*, pois ainda não se aplicou nenhum processo para melhoria e limpeza do mesmo. A partir disso, temos a etapa **1** *Data Preparation* que engloba alguns processos já citados, como por exemplo remoção de registros com erro, correção dos dados e fundamentalmente, remoção de dados faltantes, a partir dessa etapa temos já os dados preparados para se aplicar a normalização das *features* se os dados não possuírem uma escala significativa e comparável.

A etapa **2** *Data distribution verification* é fundamental em problemas de classificação com desbalanceamento dos dados, visto que, aprendizagem de máquina depende da representatividade das classes no conjunto de dados, pois, isso impacta diretamente nas previsões. Técnica fundamental em sistemas que possuem registros de *intrusion detection*. Logo, para evitar um mau classificador, podemos utilizar algumas técnicas são elas *Oversampling Techniques* conhecido por duplicar registros ou sintetizam novos exemplos, tendo por base a classe minoritária. Temos também *Undersampling Techniques* que por sua vez é utilizado para eliminar ou selecionar um subconjunto de exemplos da classe maioritária. Por fim,

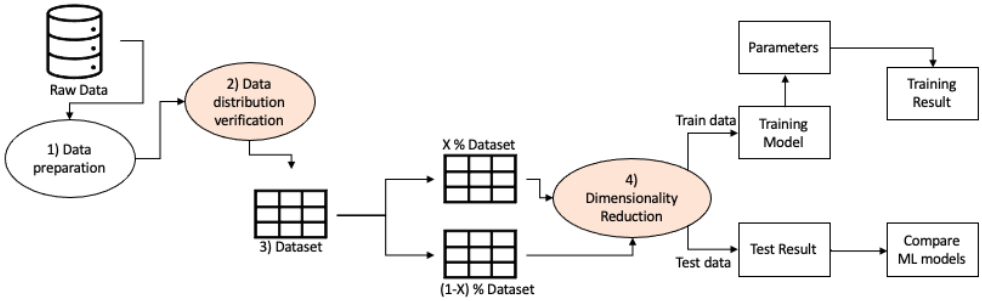

Figura 2.9: Ilustração de passos para uma classificação centralizada

temos *Combinations of Techniques* que combina técnicas dos métodos anteriores para obter um melhor resultado global.

Com o entendimento de diferentes técnicas, pode-se dizer que para a dissertação utilizou-se do Synthetic Minority Oversampling Technique (SMOTE) que é conhecida como a técnica de *Oversampling* mais popular e talvez a mais bem sucedida. Consiste em criar exemplos de forma sintética, baseados nos exemplos mais próximo do espaço das *features*. A nova amostra, é criada baseado nos k vizinhos mais próximos da classe minoritária. Pode ser encontrado com mais detalhes em Chawla et al. [25].

A etapa **3** consiste um dividir os dados de forma balanceada entre treino e teste. Tendo em consideração que a soma percentual deva dar 1. Como podemos ver no exemplo da Figura 2.9 temos a divisão entre $x\%$ dados de treino e $(1-x)\%$ dados de teste para serem usados pelo modelo. Normalmente usa-se x entre 60 e 75 %. A divisão das classes é distribuída de forma balanceada de acordo com a distribuição do *dataset* inicial.

Na etapa **4** *Dimensionality Reduction* executamos a transformação de dados de um espaço de alta dimensão em um espaço de baixa dimensão de forma que a representação de baixa dimensão retenha algumas propriedades significativas dos dados originais. Fundamental para *datasets* com grande número de *features*, porém, devemos ressaltar o problema que se insere ao aplicarmos essas técnicas é a perda de informação, ou seja, podemos reduzir a eficacia do nosso classificador.Com base no artigo Alhowaide et al. [26] podemos observar algumas técnicas mais comuns para a redução de dimensionalidade, são elas Principal component analysis (PCA) e aplicar *Feature selecion* utilizando RF. Para o melhor entendimento iremos explicar um pouco mais a fundo como cada redução ocorre.

Feature selection, corresponde a aplicação de um modelo de ML com o objetivo de incluir apenas as *features* mais relevantes, com isso dito, podemos entender que a aplicação de alguns modelos de ML que por si só já conseguem informar a ordenação das *features* mais relevante para tal caso, podemos também usar algum tipo de *threshold* para selecionarmos as features que alcancem um valor de importância na distribuição.Podemos também aplicar e validar a importância das *features* de acordo com a sua correlação entre elas, entre outros métodos.

Também considerando a redução de dimensionalidade, temos PCA, conhecido por ser uma prática voltada para reduzir a complexidade proveniente dos dados, encontrando correlações entre as amostras, que por muitas vezes não possuem correlação, chamando-as de componentes principais. O número de componentes, é menor ou igual ao número de *features*. É sensível a escala relativa das *features* originais. Pode-se encontrar mais em Malhi and Gao [27] que aborda *defect classification*.

A partir dos processos citados, teremos dados organizados e limpos, prontos para as etapas

de treino e teste, para avaliar nosso classificador.Pode-se considerar a aplicação de diferentes modelos para testar nossos dados, um de cada vez é utilizado para treinar um modelo, após o resultado, obtemos parâmetros e, com isso, a partir dos dados de treino podemos validar os resultados obtidos, verificar as métricas de avaliação que estão listadas na Tabela 2.2. Por fim, fazemos as comparações dos resultados entre todas as possibilidades de combinações da etapa **4**, ficando assim com uma tabela de resultados obtidos.

Como dito na etapa **1**, é importante considerarmos toda e qualquer informação disponível, e devido a evolução da tecnologia, pode-se buscar as mais variadas informações em dispositivos inteligentes. Os quais, armazenam diferentes tipos de dados. Com isso, vale explorarmos mais a fundo sobre a tecnologia de conexão à *Internet*, que faz com que os dispositivos sejam 'inteligentes' e com receptividade de novas informações quase em tempo real. Para isto exploraremos a tecnologia IoT dos dispositivos.

2.3 Internet of Things

Definição na literatura [28] de "Uma rede aberta e abrangente de objetos inteligentes que têm a capacidade de se auto-organizar, partilhar informações, dados e recursos, reagir e agir face a situações e mudanças no ambiente".

Que pode ser definida como a rede de redes, onde podemos ter dispositivos finais como uma vantagem, e é um dispositivo informático, que pode ser responsável pela integração, e não um dispositivo do utilizador. Este *edge device* pode tratar de dados, sem a interação humana, por exemplo, monitor cardíaco para pacientes, ele ou ela não está diretamente ligado à geração de dados, através da ligação à Internet. Usando IoT é possível monitorizar o desempenho do negócio, podendo conectar-se diretamente a um dispositivo, que possua toda a informação. Isto significa que, é capaz de identificar um problema em tempo real. O que torna a resolução do problema mais fácil, e mais rápida, não necessita da atenção total de um humano. Pode ser considerado como uma melhor comunicação de humano-para-humano e, de humano-para-dispositivos.

Para entrarmos a fundo na temática de identificação de um problema em tempo real, temos que perceber que existem algumas formas de criarmos e, captarmos os dados para alimentar um servidor, podendo assim, criar uma rede de alimentação dos dados. Essas redes podem ser de diferentes formatos, temos algumas ilustradas na Figura 2.10. Pode-se entender que *Centralized Approach* diz respeito a coleta de dados, que foi enviada por cada *IoT device* com o devido armazenamento por parte do servidor central, o qual tem como responsabilidade, executar toda *pipeline* de forma exclusiva, o que exige um grande poder computacional, visto a quantidade de dados produzidos por dispositivos com tecnologia IoT. Para solucionar o problema com grandes armazenamento de dados, podemos utilizar o *Distributed Approach*, fazendo com que cada objeto da rede, tenha seu próprio treino interno, diminuindo o poder computacional em não necessitar do armazenamento dos dados em um servidor central porém, se perde o ganho de informação que os diferentes dispositivos poderiam compartilhar, ou seja, sem colaboratividade. Por outro lado, surge o FL que nasce com a proposta de promover o não compartilhamento dos dados, ou seja, treinos locais, dentro de cada *node* (dispositivo), permitindo o compartilhamento somente dos parâmetros a partir do treino local, ou seja, de forma exclusiva em cada *node*. Enviando para o *central node* (Servidor) onde será feita a *Aggregation* dos parâmetros provenientes de cada *node*. O cenário apresentado nas imagens, denotam a presença de diferentes dispositivos com tecnologia IoT, somente com efeito de ilustrar os diferentes ambientes que podem ser utilizados, em cenários como *smart homes*, *smart city*, *tele-communication*,

smart grid, entre outros.

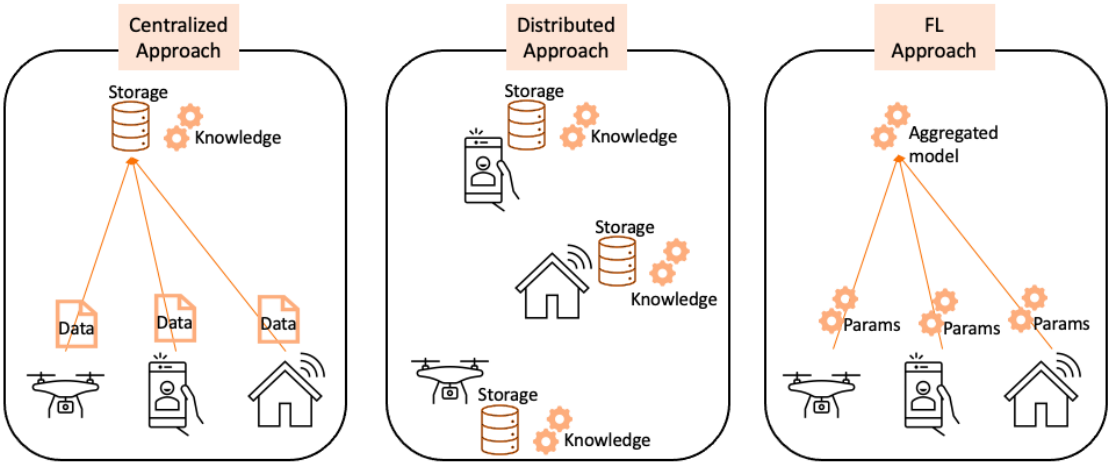

Figura 2.10: Esquema comparativo entre ambientes centralizados, descentralizados e FL (Adaptado de Campos et al. [29]).

Dos cenários apresentados, focaremos na aplicação dos dispositivos inteligentes no *FL Approach*. Permitindo a troca de informação dos *edge devices/servers*, que são compostos de hardware e software, e são orquestrados por uma estrutura chamada *Edge Computing*. Esses *edges* podem ser responsáveis por executar a agregação dos parâmetros num contexto de FL. Proporcionando menor latência e, reduzindo os custos de transmissão, o que minimiza a necessidade de os dados serem processados num centro de dados remoto. Tendo em conta a não necessidade de centralizar os dados, pode-se considerar um ambiente mais robusto, garantindo uma maior segurança para cada *node*.

Com isso, para testar a robustez do sistema, pode-se promover ciber ataques a essa estrutura descentralizada, de forma a promover o aprendizado para detecção da anomalia e, auxiliar na identificação de uma atividade suspeita. Na literatura existem alguns tipos de ataques [30, 31] com isso, focaremos nos ciber ataques abordados pelos *datasets* selecionados na dissertação. Para os dividirmos, focaremos em qual aspecto de segurança o ataque se relaciona, iremos dividiremos em quatro aspectos [12], são eles: integridade, disponibilidade, confidencialidade e, autorização onde comprometem a segurança do ambiente.

- **Integridade**:
  - ***Injection*** [32] ocorre quando o ataque injeta ou insere alguma informação falsa dos dados do cliente para alguma aplicação, como por exemplo *SQL injection attack* que explora aplicações do tipo *PHP* e *ASP*. Para o *dataset* em questão utilizou-se diversas ofensivas com um endereço de IP para injetar *inputs* contra *web application* e, as *virual machines* dos dispositivos IoT. Por fim, pode-se dizer que executarão comandos como *SQL injection*, *client-side injection*, quebra de autenticação e gestão dos dados, gerando uma fuga involuntária de dados.

- **Disponibilidade**:
  - **Denial of Service (DOS)** [10, 32] diz respeito a atividades falsas de *flooding* que tentam corromper os recursos IoT/IIoT. Para executar o ataque aplicou-se um *hack* em elementos vulneráveis da *testbed*. Utilizando Python *scripts* com a biblioteca *Scapy*.
  - Outro ataque similar como esse, é o **Distributed Denial of Service (DDOS)**

[10, 32] que também foi criado com Python *scripts* porém em maior escala, também atacando vulnerabilidades da *testbed*.

- **Confidencialidade**:
  - **Cross-site Scripting (XSS)** [32] técnica maliciosa de injetar aplicações *web* normais e de confiança, bem como páginas de serviços IoT. Esse ataque foi gerado a partir de *scripts* de Python, utilizando *toolkit* conhecido como *XSSer* direcionando o ataque para *web-based applications*.
  - ***Reconnaissance*** [12] para o cenário aplicado, pode-se dizer que se trata de um ataque passivo, visto que o atacante é silencioso e não injeta nenhuma informação que o possa expor. Pode-se entender como um ataque não severo, porém existe a exposição da informações privadas, podendo ser acessada por uma pessoa não autorizada. Durante o artigo é considerado um ataque com alta dificuldade de ser detectado. Também conhecido como ***Scanning*** [32] e ***PortScan*** [10], o atacante busca *open ports* e o endereço de IP ativo, para infiltrar no sistema da vitima. Podem ser gerados a partir das ferramentas *Nessus* e *Nmap* para promover a busca de vulnerabilidades na vítima.
  - ***Password cracking*** [32] técnica de *hacking* podendo ser *brute-force* ou *dicitionary attacks*, que depende da quantidade de combinações de diferentes senhas testadas, até descobrir a correta. Usado para violar o funcionamento do sistema operacional do usuário, serviços IoT e *web applications*. Utilizou-se *hydra* e *cewl toolkits* para gerar senhas e possibilitar o *hacking* de *nodes* vulneráveis.
  - ***Generic*** [9] vai de encontro com todos tipos de ataques em *block-ciphers*. se relaciona com blocos de dados e chaves de encriptação, o ataque acontece diretamente conectado à troca de mensagem a longa é dividida numa série de blocos de mensagens sequenciais, e a cifra opera nestes blocos um de cada vez.

- **Autorização**:
  - ***Backdoor*** [32] é uma prática maliciosa onde o atacante busca acesso ao sistema e a informações da vitima. A partir do sucesso da conexão com a vitima tem como objetivo roubar informações pessoais e bancárias, instalar outro *malware*. Foi criado utilizando *Metasploit framework* e executando *bash scripts*.

2.4 Federated Learning

O conceito de FL também conhecido por aprendizagem colaborativa, surge em 2016 [33]. Com o foco principal de criar um processo de ML através de múltiplos dispositivos ou servidores descentralizados (*edge device/server*) com os dados do usuário armazenados internamente, sem ter de expô-los.
Com o FL conseguimos combater diretamente a necessidade de ter sistemas com servidores centralizados, garantindo uma maior segurança, uma vez que funciona de forma descentralizado, o que garante que cada *edge* partilhe apenas o modelo já treinado, mantendo a informação sensível dentro de cada *node*. Pode ser visto como um dispositivo inteligente, o *edge device*, é responsável por agregar os modelos provenientes de cada um dos *nodes*. Isto reduz a necessidade de armazenar grandes quantidades de dados e permite assim uma maior agilidade na obtenção da informação, uma vez que, não será necessário centralizar os dados para o seu processamento. Temos alguns possíveis problemas como o consumo de energia no *node*, perdas de conexão durante uma ronda de treino.

2.4.1 Entendimento *Federated Averaging* e notações matemáticas

Conceito introduzido por McMahan et al. [7], o algoritmo mais conhecido como Fed Avg, mais comum nos dias de hoje, quando se considera a abordagem descentralizada de aplicação do FL. O FL fornece uma estratégia descentralizada que pode utilizar um modelo para treinar cada *node* em si, criando o modelo local a partir dos dados locais, e depois enviá-lo para o servidor central, para inicial a etapa de agregação, considerando o peso de cada *node*, criando o modelo global. A fim de fazer atualizações ao modelo global, é necessário aplicar um algoritmo de agregação e produzir um novo modelo global, para que isso aconteça utilizaremos a aplicação de *Fed Avg*.

Para compreender o algoritmo, podemos pensar que temos um ambiente que inclui quatro *nodes* (cada um pode ser considerado como cliente), neste cenário podemos ver a representação na figura Figura 2.11. Quando uma rodada se inicia, considera-se t o horário de início que o servidor central (responsável pela agregação) possui todos os *nodes* acessíveis C para iniciar o processo. O servidor é responsável por enviar o modelo global atualizado w_t treinado na a cada rodada. Caso de maior atenção na primeira rodada do Fed Avg, visto que cada *node* recebe instruções de um modelo inicializado, ou seja, o modelo de ML escolhido para o treino e seus devidos parâmetros, além disso, é importante já inicializarmos os *outputs* ou seja, os parâmetros de saída são inicializados respeitando o número de classes e *features* do *dataset*.
Os clientes são representados por S_t, cada cliente k treina localmente utilizando os seus próprios dados P_k resultando em w_{t+1}^k a combinação de parâmetros do modelo local, é enviado para o servidor central que agrega e calcula a média dos parâmetros do modelo recebidos. Gerando o modelo global $w_{t+1} = \sum_{k \in S_t} \frac{n_k}{n_t} w_{t+1}^k$ onde n_t é o número total de amostras de um *node*, e n_k são as amostras no cliente k. Quando o processo do modelo global está completo, é enviado os parâmetros globais, de volta para todos os *nodes* que participam na rede. Além disso, podemos optar por selecionar somente uma fração de *nodes* para participar da rodada. Os selecionados C para cada ronda, B o tamanho do lote local, E o número de vezes que cada *node* treina sobre o seu conjunto de dados, e a taxa de aprendizagem η. O algoritmo completo com maior detalhamento pode ser encontrado [34]. Para a próxima rodada é importante ressaltar que os *nodes* selecionados, iniciam os seus modelos com os parâmetros provenientes do modelo global, obtido na rodada anterior. E segue-se a lógica explicada, de cada *node* treinar seu novo modelo local, e após o treino, o enviar para o servidor que se encarrega da nova agregação dos parâmetros.

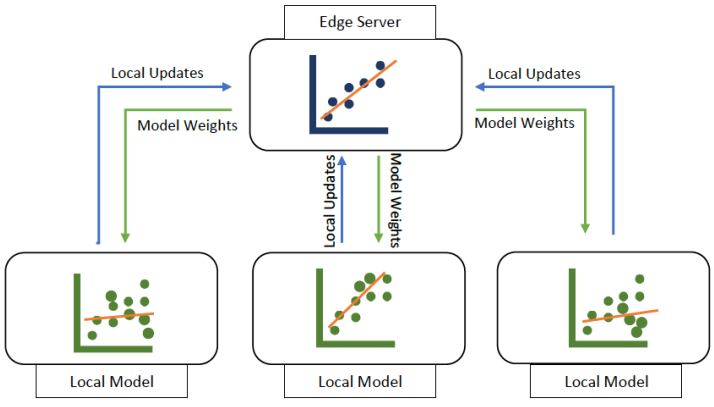

Figura 2.11: Exemplo simplificado de FL (Adaptado de McMahan et al. [7])

2.4.2 Federated Learning Flower framework

Como o objetivo da dissertação é criarmos uma simulação de um cenário descentralizado, utilizando *datasets* baseados em ciber ataques. Optou-se por utilizar o *Flower* [35] como *framework* base, para conseguirmos criar um ambiente simulado, seguindo os princípios do FL sem expor dados e garantindo a segurança e integridade dos dados dos *nodes*.

Flower surgiu em meados de 2020, pode ser encontrado como biblioteca em diversas linguagens de programação, são elas, Python, C++, Java, entre outras. Criado com objetivo de fornecer à investigadores experimentar e implementar rapidamente novas ideias em cima de uma ideia fiável, por ser altamente personalizável, podendo ser utilizada instruções e organizações pré-feitas por desenvolvedores da biblioteca, como também, adaptar a aplicação à diversos cenários.
Fornece apoio para estender as implementações de FL à dispositivos móveis e com conexão *wireless*, com recursos heterogêneos de computação, memória e comunicação. Pode-se ser simulada em servidores *cloud* distribuídos, ou mesmo numa única máquina.

Possui a ideologia de uma fácil escalabilidade baseada no FL permitindo que aproveite-se tanto uma grande variação de número de clientes conectados como um grande número de clientes selecionados para cada rodada do FL.A sua *framework* se subdivide em algumas etapas são elas:

- ***Flower Client*** a interface do cliente permite que *Flower* orquestre o processo do FL visto que cada cliente recebe um código para saber se por exemplo irá fazer parte dos clientes selecionados para executar o treino local ou para fornecer a métrica de avaliação própria, coletada após o treino local.
- ***Flower Server*** é responsável por identificar e garantir que a conexão com os clientes esteja correta, que os clientes executaram os passos que deveriam, garantir que performem as instruções quando o cliente foi requisitado. Responsável por performar a atualização do modelo global. Para a *framework* é importante ressaltar que podemos considerar diversos clientes, porém somente um servidor.
- ***Federation Strategy*** etapa facilmente personalizada, porém já pode-se encontrar o algoritmo Fed Avg e algumas de suas variações implementadas, bem como outras estratégias de agregação [35].

2.4.3 *Data partitioning*

Uma parte importante dos problemas no modelo AI é a recolha das fontes de dados desejadas, compreendendo as suas características e também diferentes possibilidades de amostras, o que pode levar a um melhor resultado da métrica de avaliação *accuracy*, do modelo de ML. Em relação ao modelo FL existem diferentes arquiteturas que podem ser utilizadas em relação à *Data partitioning*, as mais comuns são [36, 37] e ilustradas em Figura 2.12.

Horizontal Federated Learning (HFL)

HFL ou aprendizagem federada baseada em amostras, pode ser interpretada como duas fontes de dados diferentes, podendo não partilhar espaço de amostra (IDs), mas partilhar o mesmo espaço de *features*. Tendo as mesmas *features* nos dados, é possível ter o mesmo modelo AI para a sua etapa de treino dos dados. Nesta abordagem de FL, cada *node* treina o seu modelo localmente e, depois envia-o para a agregação. Os dados podem ser enviados usando algumas técnicas de encriptação e mascaramento, provendo uma segurança ao

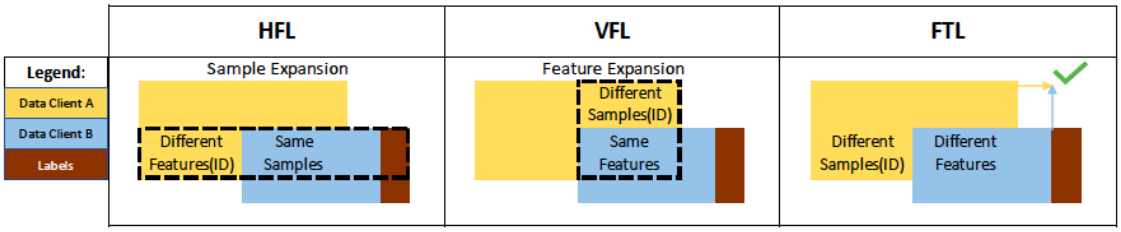

Figura 2.12: Exemplo de HFL, VFL, FTL Subseção 2.4.3 (Adaptado de Zhang et al. [37])

utilizador. A agregação do servidor é responsável pela formação do modelo global e depois de envia-lo a cada utilizador, preparado-os para a próxima ronda de aprendizagem. O processo replica-se até atingir a desejável *Accuracy* ou a *loss function* convergir. Um exemplo é o Gboard, de Hard et al. [3], que utiliza usuários de Android, com o objetivo de gerar a sugestão de *top-n* palavras, o que corresponde a sugestão de palavras (n) mais coerentes, baseados no contexto anterior. Para tal, utilizou-se o algoritmo Fed Avg incorporado na FL *framework*, que proporciona segurança contra o servidor central.

Vertical Federated Learning (VFL)

VFL ou *feature-based* FL se aplica ao combinar partes do cliente A e cliente B, entidades de serviços diferentes, detentores do mesmo espaço de amostra (ID), mas diferem no espaço de *feature*. Com isso, torna impossível ter o mesmo modelo AI para a etapa de treino dos dados. Cada *node* treina localmente os seus dados, combinando as suas amostras, normalmente utiliza algumas técnicas de encriptação. Um exemplo de VFL pode ser identificado quando se tem duas empresas diferentes na mesma cidade, uma da área de comércio eletrônico e, a outra um banco. Podendo compartilhar de mesma amostra de clientes, considere que ambas têm o endereço do utilizador. O banco pode ter os registos do seu saldo bancário, e os possíveis empréstimos que este cliente possa ter recolhido, por outro lado, o comércio eletrônico armazena o perfil de compras e pesquisas do cliente. Com o interesse mútuo de perceber o comportamento dos clientes de acordo com a junção das informações que cada entidade detêm. Podem executar rodadas de VFL, o que permite a colaboratividade entre as entidades, neste caso o banco e o comércio eletrônico, sem a necessidade de exposição de dados sensíveis dos clientes.

Federated Transfer Learning (FTL)

Diz respeito a expansão do cenário de VFL, neste caso com mais clientes, mas o conjunto de dados raramente são do mesmo espaço de amostra (ID). Esta cobertura de todos os espaços de amostra, envolve uma representação comum que minimiza erros de previsão. Um exemplo que se aplica FTL num cenário de classificação em Ju et al. [38] combinando vários pequenos conjuntos de dados para treino com um conjunto de modelos de ML, no final a combinação das informações levou a uma melhoria de 6% de *accuracy* em relação ao trabalho anterior, o que foi constatado como um ganho de alta performance.

2.4.4 Estrutura de redes

FL está dividido em diferentes estruturas de rede e pode ser entendido como Centralized Federated Learning (CFL), Decentralized Federated Learning (DFL).

Centralized Federated Learning (CFL): que é um cenário mais comum de aplicação de *FL-IoT*, pois considera, um ambiente onde todos os *node* são utilizados. Ao iniciar o processo, cada *node* faz o seu treino local e envia-o para o servidor central que é responsável

pela agregação, dos seus modelos usando algum algoritmo de agregação para obter um modelo global. Por exemplo, utilizando Subseção 2.4.1 que aplica o algoritmo de ponderação Fed Avg.

Decentralized Federated Learning (DFL): neste ambiente não se utiliza um servidor central para orientar o processo de treino. Todos os clientes são ligados por ligação *peer-to-peer*. O quer diz respeito a um cenário onde os utilizadores se comunicam com os seus vizinhos propiciando uma agregação diretamente de usuário para usuário.

2.4.5 *Federated Optimization*

É semelhante à otimização distribuída mas, em relação ao cenário FL, que pode possuir propriedades diferentes.

1. *IID data:* Quando o conjunto de dados está bem equilibrado entre todos os *node*. Neste caso, tende a ser mais fácil modelar o comportamento dos dados no grupo de utilizadores, é Independente e distribuída de forma idêntica as variáveis aleatórias.

2. *Non-IID data:* Por outro lado, num cenário IoT, normalmente clientes diferentes não têm qualquer semelhança de *features*. Assim, não tendo características comuns, torna mais difícil para o treino do modelo considerar qualquer comportamento específico de algum utilizador. O conjunto de dados local do utilizador não será representativo no que diz respeito a distribuição geral da população.

3. *Unbalanced:* Do mesmo modo, considerando um cenário IoT, é comum termos alguns utilizadores com maior tempo de uso do dispositivo do que outros, e para finalidades diferentes. Tendo assim, uma grande variedade de dados para de treino local.

4. *Massively distributed:* Espera-se que tenha muito mais *node* na etapa de agregação do modelo global do que o número de diferentes *features* como input. É interessante criar escalabilidade ao considerar cenários IoT, novos dispositivos integrando o ecossistema.

5. Comunicações limitadas: Dispositivos IoT estão frequentemente desligados devido à falta de bateria ou à falta de conexão com a rede de wi-fi, o que faz com que o passo de seleção dos *node* esteja destinado a falhar.

This page is intentionally left blank.

Capítulo 3

Trabalhos Relacionados

Explorando os trabalhos relacionados, podemos encontrar diferentes abordagens pertinentes no cenário. Faremos um apanhado visando ressaltar o que a literatura já cobriu, com possíveis cenários de aplicações. Com isso, iremos explicar um pouco mais a fundo algumas das novas tendências e, descobertas em aplicações FL, adentrando em diversas aplicações, que o utilizam como solução para alguns dos desafios em reter informação em um cenário IoT (Seção 3.1). A partir disso, iremos focar na aplicação de FL em cenários de ciber segurança (Seção 3.2) demonstrando diferentes tipos de dados com detecção de anomalias (Seção 3.3), explorando os diferentes *datasets* utilizados, bem como melhorias no processo pela variação de formatos de agregação propostas. Por fim, temos a Seção 3.4 onde resumiremos as informações encontradas.

3.1 Aplicações de cenários de Federated Learning

McMahan et al. [7] propôs uma das primeiras abordagens de FL, trazendo à importância da sua aplicação, como foco em sanar problemas voltados para a privacidade dos dados. Com isso, sugere a aplicação da abordagem descentralizada de FL. Considerando informações provenientes de *mobile devices*, utilizando assim uma arquitetura simples de agregação local, que funciona a partir de *updates* provenientes da informação vinda de todos *nodes*. Para testar sua abordagem, utiliza quatro diferentes *datasets* explorando a presença das classes de forma desbalanceada, também conhecido como Nonindependent and Identically Distributed Data (Non-IID). Utiliza o algoritmo Fed Avg para sua etapa de agregação dos parâmetros, para a criação do modelo global, utilizou variações de parâmetros ao implementar Long short-term memory (LSTM). Considerando a parte experimental, traz aplicações em cenários de classificação de imagens, e modelação de linguagem. É possível concluir os benefícios utilizando Fed Avg, que apresenta uma arquitetura robusta, a partir de uma arquitetura descentralizada, a qual limita a possibilidade de um ataque afetar todos os clientes de uma vez só. Por fim, consegue proporcionar uma melhor métrica de avaliação, *accuracy* mesmo não expondo os dados do utilizador. Propondo para trabalho futuro, analisar contextos de privacidade e segurança da aplicação de FL.

Chandiramani et al. [39] se focou em um cenário de classificação de imagens, em busca de aplicações que fornecesse uma maior privacidade para o usuário. Para isso utilizou o conjunto de dados *Fashion-MNIST* [40]. O objetivo é comparar a segurança entre a aplicação de ML descentralizado e um cenário FL descentralizado, pode-se entender que são cenários similares, a maior diferença está relacionada com no FL temos um *node* ou servidor que promove a agregação e cada participante executa seu treino de forma independente

uns dos outros, por outro lado o ML descentralizado possui grande similaridade com ML centralizado, com somente um armazenamento descentralizado. Durante seu processo de treino, aplicou-se Deep Neural Network (DNN). Tendo como objetivo fornecer um entendimento do ganho de informação proveniente da colaboração que a aplicação de FL fornece. Utilizou-se de métrica de avaliação, a *accuracy*. O artigo constata que as aplicações e testes, obtiveram uma *accuracy* semelhante. Porém, tendo em conta o consumo de tempo do processo, FL obteve menos da metade do tempo que uma aplicação de ML descentralizada. Por fim, não relata trabalho futuro.

Wu et al. [41] propõe uma *framework* personalizada para lidar com dispositivos heterogêneos, com heterogeneidade estatística e heterogeneidade de modelos. O artigo, visa comparar a performance de aplicações mais comuns de ML centralizado, são elas kNN e RF, com CFL. Esta última abordagem utiliza-se de uma estrutura descentralizada que realiza a agregação de parâmetros a partir do algoritmo Fed Avg. O artigo propõe uma arquitetura chamada *PerFit*, que é baseada em *cloud*, e uma aplicação de FL personalizada, utilizando cenários de FTL (Subseção 2.4.3) e Federated Distillation (FD) (Subseção 2.4.4). Para testar a arquitetura, foi utilizado *MobiAct* [42], que é um *dataset* que armazena dados de comportamento humano utilizando o *smartphone*, detectando e armazenando dados que dizem respeito à acelerômetro e sensores de giroscópio. Para sua análise, avaliou-se o *trade-off* entre a *accuracy* e a necessidade de recurso computacional. Conclui-se que os métodos personalizados de FL obtiveram a melhor precisão (mais de 90%) enquanto os outros cerca de 80%. O cálculo do gasto da comunicação mostrou que ao utilizar FTL não se faz viável, se comparado à FD. Por fim, não aborda trabalho futuro, somente reforça a efetividade do *PerFit*.

Xu et al. [43] propõe, uma abordagem centrada na *cloud* com a utilização dos *edge devices* para cenários de *Big Data*, colocar em risco a privacidade do utilizador e, o custo de comunicação da rede. Sugere a *framework* Federated Learning with Pruning, Quantization and Selective Updating (FL-PQSU) com objetivo de superar a proposta do Fed Avg, por permitir a seleção dos *nodes* mais relevantes para serem considerados na criação do modelo global. Para testar a *framework*, sugere a simulação de um problema de classificação de imagens utilizando os conjuntos de dados *MNIST* [44] e *CIFAR10* [45]. Para sua estratégia, foi feito uma etapa de *pooling*, que é responsável por remover o que for redundante e tornar a DNN de forma mais rápida. A quantificação de dados, reduz o tamanho do modelo e melhora a eficiência computacional. Aplicando a *framework* proposta, os resultados obtidos a partir da métrica de avaliação *accuracy*, preservam as vantagens da aplicação do FL, enquanto que reduzem o recursos computacionais, armazenamento e comunicação. Propõe para trabalho futuro a exploração dos mesmos conjuntos de dados, mas utilizando a distribuição de dados Non-IID.

Damaskinos et al. [46] buscou abordar em sua análise algumas limitações de aplicações tradicionais de FL utilizando a agregação Fed Avg. O seu maior foco era denotar o gasto energético da bateria, e impacto nos *mobile devices*, quando estamos lidando com um cenário de atualizações frequentes dos modelos globais. Com isso, sugere a aplicação de *Online* FL utilizando dados provenientes de celulares, e que fornecem em tempo real novas atualizações. Para teste utilizou-se de *datasets* para um cenário de classificação, são eles: *MNIST* [44], *CIFAR-100* [47], *E-MNIST* [48] o algoritmo de agregação *AdaSGD* que é uma adaptação do SGD utilizando a proposta de *online learning*. Para trazer teste mais relevantes, utilizou-se de distribuição de dados Non-IID, aplicando o modelo *Passive Aggressive*. Provou que o consumo de bateria do cliente era de aproximadamente 0.036% de uma carga normal, bem como supera a aplicação tradicional em 3.6x na métrica de avaliação *accuracy*. Além de ter um tempo de convergência muito mais eficaz, aproximadamente 18.4% mais rápido. Para trabalho futuro buscará técnicas para limitar a adição de ruido

ao utilizar *AdaSGD*.

3.2 *Federated Learning* aplicado à temática dos ciber ataques

Temos a avaliação de privacidade relacionado com ataques de ciber segurança no Connected and Automated Vehicles (CAV) feita por Mestoukirdi et al. [49]. No artigo, é considerado a heterogeneidade dos dados através de diversos dispositivos IoT. Com isso, foi definido o objetivo principal de abordar essa heterogeneidade, a qual leva a um investimento maior no tempo de convergência para o treino e a capacidade do modelo viabilizar uma generalização. Para abordar o tema, sugere uma estrutura personalizada da aplicação de FL, criando assim, múltiplas regras de agregação centradas no utilizador, chamado-as de *user-centric aggregation*. Propõe a utilização de dois conjuntos de dados, problema de reconhecimento de caracteres/dígitos, *E-MNIST* [48] e um problema de classificação de imagens utilizando o *CIFAR-10* [45], utilizou a aplicação de uma Convolutional Neural Network (CNN) com baixa complexidade, conhecida como *LetNet-5*. A heterogeneidade de dados foi testada em diferentes cenários: Fed Avg, aprendizagem local, CFL e *FedFOMO*, essa última foi a sua proposta, que é voltada à tomada de decisão de quais *nodes* serão selecionados para a agregação, optando pelos mais relevantes. Os modelos personalizados têm um desempenho superior com maior *accuracy*, sendo ao mesmo tempo mais eficientes em termos de comunicação em comparação com as aplicações tradicionais. O tema para extensão do trabalho para o futuro não foi abordado.

Temos algumas novas informações de cenários de ciber ataques, que ajudam a compreender as novas tendências encontradas no mundo, explorando aparelhos com tecnologia IoT e com isso a necessidade de novos formatos para extração da informação. Entraremos em maiores detalhes de intrusion detection systems (IDS) em [17, 29].

Campos et al. [29] aborda, alguns tópicos fundamentais para a detecção de um fator externo, com isso aborda a existência de diversos ataques e *datasets* utilizados *Bot-IoT* [50], *N-BaIoT* [51], *MedBIoT* [52], *IoTID20* [53], *ToN_IoT* [32]. Para exemplificar utiliza-se de *PCAP files* do *dataset ToN_IoT* para promover um estudo de caso. Utilizando cenários multi-classe com a presença do desbalanceamento entre os ataques. Tendo como modelo base *Logistic Regression* proveniente da implementação do SGD utilizando a biblioteca *scikit-learn* em conjunto com a linguagem de programação Python. Para a etapa de agregação utiliza-se Fed Avg e *Fed+*, que por sua vez relaxa a exigência de forçar todas os *nodes* a convergir para um modelo único, o que auxilia num cenário heterogêneos de dados. Com isso, providência uma simulação com 300 rodadas e a presença de 10 *nodes*. A sua conclusão é que, nas primeiras rodadas do FL consegue demonstrar grandes evoluções em relação a métrica de avaliação *accuracy*, porém até a rodada 50 a métrica se estabiliza em todos cenários. O ganho de informação devido a colaboratividade dos *nodes*, é de aproximadamente 5% no modelo global em relação a agregação inicial. Acabando por concluir que o algoritmo base Fed Avg consegue alcançar alta performance tanto nos cenários balanceados como não balanceados. Para trabalho futuro, estudará a possibilidade de criar um cenário real para testes, além de aplicações personalizadas de cenários FL.

Ao avançar as buscas pelo tema, encontramos uma *survey* recente feita por Dai and Meng [17] que aborda temas como Online federated learning (OFL) direcionado a aplicação do cenário de FL porém no formato tradicional o treino local dos clientes ocorre de forma *offline* e a agregação começa com todas as informações coletadas, porém em OFL temos um maior dinamismo por a informação dos clientes chegarem de forma *online*, ou seja permitindo que os clientes treinem seus modelos em momentos diferentes, ou seja, com

atualização de novos dados em tempo real. Outro tema é Online transfer learning (OTL), são cenários de FTL porém com envio de dados em tempo real. Relatando a aplicação de dois modelos em um cenário de *online machine learning*, são eles, *Perceptron* e *Passive-Aggressive*. Os quais entram na aplicação de OFL em cenários de classificação, tratados como modelos ainda pouco explorados e, com grandes perspectivas para simulações de FL com *datasets* tradicionais como *MNIST* [44], *CIFAR-10* [45], entre outros.

Chen et al. [54] buscou abordar complicações ao utilizar Fed Avg para performar a agregação, como por exemplo, não obter alta performance com clientes heterogêneos, graças a variedade em relação a quantidade de dados distribuídos pelos *edge devices*, com capacidade de performar com diferentes latências e configurações. Além de não lidar diretamente com a indisponibilidade de algum dispositivo, levando a não contribuir na criação do modelo federado. Com isso, utiliza-se, da aplicação de uma rede neural, LSTM. Colocando em prática cenários de *ofl* com objetivo de identificar a evolução da classificação com o decorrer dos *rounds* com a adição de novos dados. Utilizou os *datasets FitRec* [55] que provem de dados esportivos de jogadores, bem como *Fashion-MNIST* [40] com dados de imagens de roupas, os quais se encaixam em cenários de classificação. Por fim, conclui-se que ao utilizar a realimentação dos *nodes* consegue obter melhores resultados, mesmo selecionando de forma aleatória quais os *nodes* participam de cada rodada. Obtendo assim, melhores métricas de avaliação *f-score* com o algoritmo *ASO-Fed*, que foi proposto, que performa seu modelo local e de forma contínua compartilha com o *server* seus parâmetros, para uma constante adaptação dos parâmetros. Obtendo assim, comparativamente com um servidor centralizado, um menor requerimento de recurso computacional. Para o desenvolvimento futuro é esperado investir em desenvolver uma simulação eficiente com *updates* não sincronizados.

Para explorar a temática IoT, em aplicação de simulações de FL, utilizando um contexto de *Smart grids*, temos Zhao et al. [56], que aborda a detecção e controle de alarme que, é alvo de ataques do tipo *Data Injection*. Sendo o pioneiro na temática *power electronics* em uma aplicação descentralizada de *fl*.Criando uma *framework* simplificada, baseada no ambiente da biblioteca *Flower* [35] conectando a biblioteca *PyTorch* [57] para execução de multi-processamentos de forma sincronizada. Para sua aplicação utiliza-se de redes neurais LSTM. Criação do *dataset* de forma simulada gerando um pouco mais de 80 mil dados, dentro deles 27 mil correspondem a um sistema com funcionamento normal, e 53 mil com ataques. As métricas de avaliação testadas foram *loss*, *accuracy*, *precision*, *F1-score* e *recall*. Utilizando somente quatro clientes na simulação com diferentes distribuição de dados Independent and identically distributed (IID) e Non-IID. A aplicação FL detém de um desempenho superior a uma aplicação centralizada, em todos cenários de distribuição dos dados no quesito de poder computacional. O poder computacional de envio dos parâmetros é em média 0.34% tendo maior custo interno dos *devices* por outro lado o ML centralizado fica com 100% do consumo no envio dos dados para o servidor para serem processados. Por fim, não entra na temática de planos para trabalhos futuros.

3.3 *Federated Learning* explorado em cenários de detecção de anomalias

Qin et al. [58] evidencia uma aplicação voltada para um modelo que promove a seleção de clientes antes de iniciar a etapa de agregação. Utiliza-se de algoritmos *neural network-based*, que é explorado em dados descentralizados de diferentes *nodes*, propondo a redução da latência com a utilização de *edge devices*, providenciando uma seleção dos clientes que farão parte de cada rodada de agregação. Abordando um problema de classificação, que

têm como objetivo a atualização contínua da informação, de forma *online* para que seja possível provisionar a presença de anomalias. É levado em conta a utilização dos modelos de ML SVM e *Logistic Regression*. Para a etapa de seleção dos clientes mais relevantes, baseia-se em redes neurais. Para o teste da *framework* proposta, utiliza o *dataset MNIST* [44] que possui uma classificação com presença de 10 classes. Promovendo a comparação entre, a seleção de clientes de forma personalizada chamada de *"score-based"*, e a não seleção dos clientes, que é a proposta inicial de aplicação de cenários FL utilizando Fed Avg como agregação. Para métrica de avaliação principal utiliza-se o *F-score*, concluindo que ao selecionar o cliente ultrapasse a métrica de avaliação alcançando aproximadamente melhoria na avaliação de 94% para 94,3%. Com a maior contribuição focada em providenciar uma maior escalabilidade, por reduzir a necessidade do recurso computacional. Para trabalho futuro irão buscar explorar o *dataset FEMNIST* [59] por possuir maior complexidade, com 62 classes.

Mothukuri et al. [60] apresenta uma solução para o problema de detecção da anomalia, que não necessite da utilização de um servidor centralizado para armazenar e processar os dados. A *framework* se baseia em FL para proativamente identificar pelo sistema um intruso, na conexão dos dispositivos IoT. Utilizou-se de aplicações de LSTM e Gated Recurrent Units (GRU) que são variações de Recurrent Neural Networks (RNN). O *dataset* escolhido foi *Modbus-based* [8], com a aplicação de um cenário industrial. Como resultado da comparação entre FL e a aplicação de ML centralizado, obteve-se um ganho na métrica *accuracy* de aproximadamente 86% para 90%. Para trabalho futuro, visiona testar dados em tempo real.

3.4 Resumo

Para consolidar as informações provenientes dos artigos selecionados, criamos a Tabela 3.1, para comparar diferentes abordagens de FL em dispositivos IoT, nas suas diferentes aplicações e, *datasets* testados.

Em suma, foi lido diversas aplicações na área de FL e IoT, com isso, nos leva a perceber a alta demanda e a diversidade de aplicações propostas mas, o que chama atenção é, que por ser uma área recente, os artigos mais focam em apresentar uma sugestão de aplicação em cenários simulados, utilizando fontes de dados como e.g. o *Tensorflow* mostrando a escassez de diferentes etapas de processamento.

Tabela 3.1: Resumo dos trabalhos relacionados

Trabalho relacionado	**Modelo de ML**	**Algoritmo de agregação**	**Métrica avaliação**
McMahan et al. [7]	LSTM	Fed Avg	*Accuracy*
Chandiramani et al. [39]	DNN	Fed Avg	*Accuracy*, Recurso computacional
Wu et al. [41]	LSTM	Fed Avg e *PerFit*	*Accuracy*, Recurso computacional
Xu et al. [43]	DNN	Fed Avg e *Flpqsu*	*Accuracy*, Recurso computacional
Damaskinos et al. [46]	*Passive Aggressive*	Fed Avg e *AdaSGD*	*Accuracy*, Recurso computacional
Mestoukirdi et al. [49]	*LetNet-5*	Fed Avg e *Fed-FOMO*	*Accuracy*, Recurso computacional
Campos et al. [29]	*Logistic Regression*	Fed Avg e *Fed+*	*Accuracy*
Dai and Meng [17]	*Perceptron, Passive Aggressive*	N/A	N/A
Chen et al. [54]	LSTM	Fed Avg e *ASO-Fed*	*F-score*, Recurso computacional
Zhao et al. [56]	LSTM	Fed Avg	*Accuracy, Precision, F-score* e *Recall*
Qin et al. [58]	SVM, *Logistic Regression*	Fed Avg	*F-score*
Mothukuri et al. [60]	LSTM e GRU	Fed Avg	*Accuracy*

This page is intentionally left blank.

Capítulo 4

Metodologia

Nesta seção, caracterizamos o trabalho que foi feito durante a dissertação. Tendo como objetivo criar protótipos de aplicação de cenário FL em diferentes *datasets* que estão disponíveis *Open Source*. Buscamos por fontes de dados distintas e relevantes no cenário, que pudessem constituir um classificador com uma gama variada de informações. Esses dados foram coletadas em diferentes anos, são eles, 2015, 2017, 2019, 2021. Toda a estratégia explicada será de grande importância na etapa de Resultados, para isso, devemos relembrar que os *datasets* que estão a ser utilizados, já foram introduzidos na Seção 1.2.

4.1 Tipos de Amostras

Para se compreender melhor cada etapa da arquitetura de criação do classificador, deve-se primeiramente identificar a formação dos *datasets* e principalmente o tipo de ciber ataques. Para essa etapa é fundamental o entendimento da distribuição das classes dentro do *dataset*, viabilizando um entendimento das classes minoritárias e se, pelo menos, existe uma presença, nem que pequena, de representatividade. Com isso, optamos por manter as classes que possuíam mais de 1% das amostras. Pois, se considerarmos o cenário de ciber ataques, pode-se encontrar diversos tipos de dados, mas a maior parte das amostras coletadas representa o sistema com funcionamento normal, o que faz com que a identificação da anomalia tenha baixa representatividade e dificuldade de ser percebida.

Iremos iniciar promovendo um entendimento dos *datasets* abordando a presença de cada ataque, e a sua relevância, em relação as outras amostras, pois, iremos abordar os *datasets* menos representativos de forma somente binária, e os mais complexos de uma forma a perceber qual o tipo de ataque, conseguindo assim, viabilizar uma classificação multi-classe. Com isso, temos primeiramente o *dataset UNSW-NB15* [9] (2015) que é bastante utilizado para estudos de ciber segurança, principalmente em cenários ML centralizado. Para compreender melhor sobre sua distribuição de classes, temos a Tabela 4.1.

Pode-se identificar na tabela que as duas primeiras classes *Normal* (representado por 0) e *Generic* (representado por 1). Juntos caracterizam mais de 90% das amostras totais coletadas, que foram 2,540,047.

Para o *dataset* criado em 2017, optamos por outra fonte de dados, *CIC-IDS2017* [10], utilizado em cenários de detecção de invasores. Com isso, a presença de diferentes tipos de ataques, já se denota com uma distribuição e quantidade de amostras mais relevante. Podemos identificar a distribuição na Tabela 4.2, onde optamos por selecionar, os quatro

Tipo	Número de Amostras	Cenário Binário
Normal	2,218,761	0
Generic	215,481	1
Exploits	44,525	-
Fuzzers	24,246	-
DoS	16,353	-
Reconnaissance	13,987	-
Analysis	2,677	-
Backdoors	2,329	-
Shellcode	1,511	-
Worms	174	-

Tabela 4.1: Tipos de ataques encontrados no *dataset UNSW-NB15* (colocamos em sublinhado os utilizados na dissertação).

primeiros ataques com maior representatividade para utilizarmos. Representando mais de 90% das amostras, iremos utilizar *Benign* representado por 0, que nada mais é que o sistema em atividade normal, e o funcionamento do sistema atacado, teria a presença de *DoS Hulk*, *PortScan* e *DDoS*, representados por 1,2,3 respectivamente em um cenário multi-classe. No caso binário, uniremos os ataques e os representaremos como 1, e sistema normal 0.

Tipo	Número de Amostras	Cenário Multi	Cenário Binário
Benign	2,359,087	0	0
DoS Hulk	231,072	1	1
PortScan	158,930	2	1
DDoS	41,835	3	1
DoS GoldenEye	10,293	-	-
FTP-Patator	7,938	-	-
SSH-Patator	5,897	-	-
DoS slowloris	5,796	-	-
DoS Slow-httptest	5,499	-	-
Bot	1,966	-	-
Web Attack Brute Force	1,507	-	-
Web Attack XSS	652	-	-
Infiltration	36	-	-
Web Attack Sql Injection	21	-	-
Heartbleed	11	-	-

Tabela 4.2: Tipos de ataques encontrados no *dataset CIC-IDS2017* (estão destacados os utilizados na dissertação).

Para o *dataset* coletado em 2019, utilizamos uma fonte abordada em diversas *surveys* [29], o *ToN-IoT* que reune um conjunto de dados de diferentes *smart devices*. Essa coleção de dados foi distribuída em 9 ataques e um funcionamento normal do sistema. Na Tabela 4.3 podemos identificar os tipos escolhidos, podendo-se perceber que neste caso também trabalharemos com dois cenários, testes multi-classe (identificação de qual o tipo de ataque) e teste binário (identificação se há ou não ataque).

Por último, temos o *dataset* Zolanvari [13] que é o mais recente dentre os nossos testes, foi coletado em 2021. Presente na *survey* [61], onde é citado como pouco explorado, porém é sugerido como promissor. É posicionado como um *dataset* com aplicações de

Tipo	Número de Amostras	Cenário Multi	Cenário Binário
Scanning	7,140,161	6	1
DDoS	6,165,008	1	1
DoS	3,375,328	2	1
XSS	2,108,944	7	1
Password	1,718,568	5	1
Normal	796,380	4	0
Backdoor	508,116	0	1
Injection	452,659	3	1
Ransomware	72,805	-	-
Mitm	1,052	-	-

Tabela 4.3: Tipos de ataques encontrados no *dataset ToN-IoT* (estão destacados os utilizados na dissertação)

ML centralizado, porém sem estudos em relação à sua possibilidade em utilização em um cenário descentralizado FL. Com isso, o selecionamos para explorarmos se conseguimos trazer melhorias com uma proposta nova de aplicação, baseada em FL.

O *dataset* possui poucas amostras em sua coleta, por isso, lidaremos com um solução em um caso binário. Podemos identificar a distribuição dos dados na Tabela 4.4, que denota a presença de mais de 92% das amostras do tipo normal, ou seja, as amostras que denotam a presença de ataques com pouca representatividade, para isso, iremos utilizar todas as amostras, porém lidaremos somente com os ataques de forma agrupadas, ou seja, voltado para a identificação de existir ou não ataque. para classificarmos se existe ou não ataque, visto a escassez de amostras de ataque, vide Tabela 4.4.

Tipo	Número de Amostras	Cenário Binário
Normal	1,107,448	0
DoS	78,297	1
Reconnaissance	8,232	1
Command Injection	270	1
Backdoor	218	1

Tabela 4.4: Tipos de ataques encontrados no *dataset WUSTL-IIOT-2021* (estão destacados os utilizados na dissertação).

4.2 Organização e Preparação dos dados

Inicialmente, para se entender os passos da infra-estrutura de simulação de FL criada, a qual utilizará os quatro *datasets* selecionados, com o objetivo de criar um classificador de alta performance. Com base em explorar técnicas utilizadas por outros pesquisadores, e buscando entender qual a melhor combinação de técnicas de processamento, levando em consideração cada uma das configurações dos *datasets*.

Primeiramente, iremos descrever algumas das estruturas utilizadas, bem como as diferentes etapas podem levar a diferentes resultados. Onde as etapas, estão muitas vezes agrupados em diferentes formas de preprocessamento, preparação dos dados, e da aplicação de diferentes testes com modelos de ML, com foco em, testar a nossa abordagem descentralizada. Para isso, deve-se compreender um pouco da coleta dos dados, e a divisão que

faremos para testar abordagens, de forma que seja possível identificar a presença de ataque (cenário binário) e outra identificando o tipo de ataque presente (cenário binário).

Com isso, adotamos uma estruturação baseada em outros artigos da academia, que abordam o comparativo entre ML centralizado e FL. Temos a representação na Figura 4.1 onde temos a divisão de oito grandes etapas.

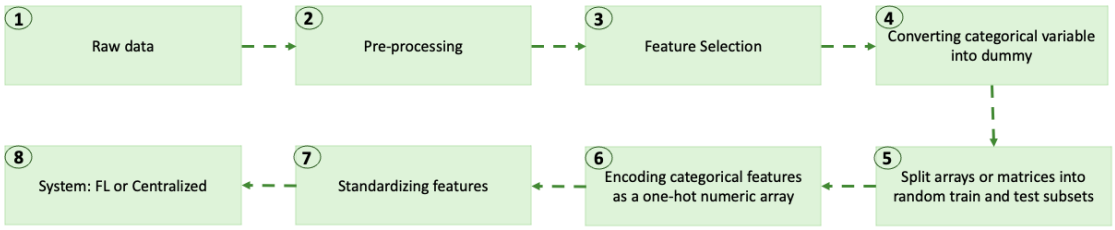

Figura 4.1: Arquitetura para ML centralizado

1 *Raw data*

Primeiramente deve-se decidir qual *dataset* iremos utilizar no processamento, considerando todas *features* pertinentes. Com isso, dependendo da complexidade dos dados e da distribuição das amostras, pode-se optar por lidar com um classificador de forma binária, multi-classe ou até mesmo abordar os dois de forma separada, porém utilizando a mesma fonte de dados. Para isso, trazemos a Tabela 4.5 que divide as explorações que faremos, explicitando o tamanho dos arquivos que serão utilizados, número de *features* e principalmente como aplicaremos o classificador.

Portanto, dividiremos em dois grupos, um que concluirá se existe a presença ou não de ataque (classificador binário), e será aplicada para todos *datasets* e, o outro será voltado em identificar o tipo de ataque, aplicados somente para os *datasets CIC-IDS2017* e *ToN-IoT*. Isso se deu, pela sua variedade representativa nos dados (classificador multi-classe).

Dataset	**Binário**	**Multi-classe**	***Features***	**Tamanho (MB)**
UNSW-NB15	✓	✗	48	579,0
CIC-IDS2017	✓	✓	80	854,1
ToN-IoT	✓	✓	45	3260,0
WUSTL-IIOT-2021	✓	✗	41	313,3

Tabela 4.5: Resumo dos *datasets* e seus respectivos possíveis testes, número de *features* e tamanho do armazenamento.

2 Pré-processamento

Para a etapa de pré-processamento, é importante se entender cada particularidade da fonte de informação, visto que cada *dataset* aborda diferentes tipos de ataques, e consequentemente diferentes coletas. Foram aplicadas técnicas específicas para limpeza dos dados, mas estão resumidamente listadas abaixo processos necessários a serem feitos em todos *datasets*.

- Remoção de erros: Considerando única e exclusivamente a partir do entendimento dos dados, proveniente dos artigos onde retiramos os *datasets*. Alguns processos foram feitos como a correção no tipo da *feature* por erros de poucas amostras, as quais foram removidas. Foi corrigido tipos de ataques que tinham escrita diferente, mas eram da mesma categoria, por exemplo *backdoors* e *backdoor*.

- *Missing Values*: removemos as linhas de amostras que possuíam dados faltantes.

Vale ressaltar que a maioria dos *datasets* não se faz necessário grandes técnicas de processamentos iniciais, já que se encontravam pré-processados. Isso ocorre por os disponibilizarem em dois formatos, *pcap files* que fornecem dados sem pré-processamento, e outro pelo qual optamos por utilizar, no formato *csv*, por já se tratar de amostras com sua devida *label*.

3 *Feature Selection*

Deve-se perceber se há necessidade de aplicar técnicas de redução de dimensionalidade, que de uma forma geral promoveria uma seleção das *features*. Partindo do objetivo de agilizar o processamento dos dados. Se optamos por aplicar, trazemos duas opções, são elas PCA ou *Feature Importance*.

Para aplicar o PCA, é necessário definir alguns parâmetros, tais como, número de componentes que iremos ficar no final. Durante essa dissertação utilizaremos *n_ components*=4. Partindo do entendimento de que, ao aplicarmos essa técnica, perdemos informação, porém nos permite obter uma grande aceleração no treino do nosso modelo, pela redução da complexidade. A partir de testes, pode-se acompanhar se foi possível manter a integridade da informação, e com isso, optar por um número de componentes que explique a maioria dos dados.

Outra opção é, aplicar *Feature Importance* que nesse caso se optou por utilizar um treino do modelo RF usando *n_ estimators=100*. Com o modelo treinado permite-nos criar uma tabela das *features* mais importantes para o modelo. Decidimos uma *threshold* da importância > 0.0001 para utilizar somente as *features* com maior relevância. Temos na Tabela 4.6 a análise dos *datasets ToN-IoT* e *NB-15*, e Tabela 4.7 para *WUSTL-IIOT-2021* e *CIC-IDS2017*, as 25 *features* mais relevantes utilizando a aplicação da *Feature Importance*.

4 Converter variáveis categóricas para numéricas

Esta é uma etapa mais focada nos cenários de multi-classificadores, para garantirmos que os ataques apareçam no formato numérico e não o seu nome. Com isso, utilizamos a aplicação de *Label Encoder* ficando somente com número invés do nome dos ataques. Podemos identificar a aplicação da técnica nas tabelas, Tabela 4.3 e Tabela 4.2. Podemos identificar, na coluna "Cenário Multi", os respectivos números atribuídos para cada um dos ataques utilizados, tendo a presença de quatro classes no *dataset CIC-2017* e oito no *TON-IoT*.

5 *Split Train/Test*

Para a divisão dos dados foi utilizado a biblioteca *scikit-learn* para garantir que as amostras tenham distribuição o mais próximo da proporção dos dados originais. Utilizando a percentagem de 70% das amostras para treinar, e 30% para a etapa dos testes.

6 *One-Hot Encoding*

Considerando um cenário IoT temos uma gama de dados imensa, e com isso se faz necessário nos atentarmos que as *features* com dados categóricos precisam ser tratadas. Para isso, geramos novas *features* baseadas em valores únicos de todas as categorias possíveis.

Esse processo promove a criação de novas *features*, porém permite que lidemos com um problema mais simplificado, considerando que teremos somente valores numéricos em cada nova *feature* criada.

7 Normalização

(a) *ToN-IoT*

features	**Importância**
src_ip	0.379
dst_ip	0.1074
src_ip_bytes	0.0946
src_pkts	0.0773
conn_state	0.0763
proto	0.0569
dst_port	0.0544
dst_ip_bytes	0.0295
src_port	0.0245
dst_pkts	0.0202
dns_qtype	0.0188
service	0.0121
dns_AA	0.0111
dns_RD	0.0108
duration	0.0078
dns_query	0.0063
dns_rejected	0.0042
dns_qclass	0.0025
dns_RA	0.0024
dst_bytes	0.0019
dns_rcode	0.0009
src_bytes	0.0008

(b) *NB-15*

features	**Importância**
sbytes	0.1623
ct_state_ttl	0.1507
sttl	0.1432
smeansz	0.1219
sload	0.0699
state_INT	0.049
ct_dst_src_ltm	0.0312
ct_srv_dst	0.0294
dbytes	0.0266
dttl	0.0259
dpkts	0.0194
dload	0.0187
dmeansz	0.0169
ct_srv_src	0.0165
dur	0.0159
service_dns	0.0152
sintpkt	0.0079
dintpkt	0.0077
dsport	0.0075
ct_src_dport_ltm	0.0075
ct_src_ltm	0.007
ct_dst_sport_ltm	0.0059
swin	0.0054
ct_dst_ltm	0.0053
dloss	0.0053

Tabela 4.6: Resumo das *Features* mais importantes presentes em cada *dataset* considerando o contexto binário

Etapa fundamental para garantir que o classificador receba *inputs* de unidades diferentes, e consiga padronizar o peso das escalas. Se faz necessário por algoritmos de ML não performarem tão bem em dados não normalizados, graças a importância dada pela sua escala. Com isso, pode-se entender que o objetivo principal da etapa é, que todas as *features* contribuam aproximadamente de forma proporcional para a distância final.

8 *Federated Learning* ou Centralized Learning

Para nosso cenário de aplicação temos um ambiente de FL, porém considerando técnicas mais comuns, pode-se ter um servidor central para processar toda e qualquer informação.

A partir de todas essas etapas explicitadas nesta parte introdutória, iremos denotar onde cada uma delas entrará em nosso ambiente proposto, bem como a etapa do FL mais detalhada.

4.3 Ambiente proposto

Para a parte de processamento, estamos utilizando um computador com processador 2,2 GHz Intel Core i7 Dual-Core com 8GB RAM. Como teremos um limitador de somente um

(a) *WUSTL-IIOT-2021*

features	**Importância**
dintpkt	0.1646
dstjitter	0.1196
dstpkts	0.0965
dappbytes	0.0892
sttl	0.0834
synack	0.0776
tcprtt	0.0391
dstloss	0.0388
dstbytes	0.0333
sport	0.0323
ploss	0.0249
dport	0.0247
dstrate	0.0238
dstload	0.0234
loss	0.0219
sappbytes	0.0149
proto	0.0103
srcload	0.0102
srcloss	0.01
load	0.0084
max	0.0079
dttl	0.0078
rate	0.0077
idletime	0.0074
srcrate	0.0045

(b) *CIC-IDS2017*

features	**Importância**
packet length std	0.0957
bwd packet length std	0.067
max packet length	0.0599
packet length variance	0.0588
bwd packet length mean	0.0535
average packet size	0.0514
avg bwd segment size	0.0466
packet length mean	0.0465
bwd packet length max	0.0448
total fwd packets	0.0287
fwd iat std	0.0253
total length of fwd packets	0.0252
subflow fwd packets	0.0248
flow iat max	0.024
subflow fwd bytes	0.0223
init_win_bytes_ forward	0.0196
psh flag count	0.0193
fwd packet length mean	0.0186
act_data_pkt_fwd	0.0172
subflow bwd packets	0.0166
flow iat std	0.0153
idle min	0.0146
fwd packet length max	0.0133
init_win_bytes_ backward	0.0133
total backward packets	0.013

Tabela 4.7: Continuação do resumo das *Features* mais importantes presentes em cada *dataset* considerando o contexto binário

computador para processar todo nosso ambiente, iremos utilizar de etapas para reduzir ao máximo a complexidade.

Com isso dito, foi decidido utilizar a linguagem de programação Python. Para seu Integrated Development Environment (IDE), escolhemos a simplicidade do *Pycharm*, que consegue facilmente processar código dentro do ambiente criado e, com processamento ágil.

Com o IDE escolhido, foi definido as todas bibliotecas utilizadas, são elas: *Flwr, Matplotlib, Numpy, Scikit-learn, Torch, Pandas, Imblearn, Argparse, Time.* Depois de baixá-las

pudemos começar a explorar cada etapa do nosso ambiente, que pode ser visualizado na Figura 4.2. A figura apresenta o nosso processo de simulação, onde focamos em ilustrar os primeiros *rounds*, pois a partir do segundo *round* pode-se entender que o processo é consecutivamente repetido.

O ambiente base, chamado *Flower (Flwr)* [35] serviu de apoio para criação do ambiente descentralizado utilizando técnicas de FL. Pode-se entender que, se optou por utilizar *Flwr* (2020) por ser um ambiente recente, de fácil customização, e por ser *Framework-agnostic*, o que significa que, pode ser utilizado em conjunto com diferentes tipos de ML *frameworks*, por exemplo, *PyTorch, TensorFlow, scikit-learn, JAX, TFLite*, entre outros.

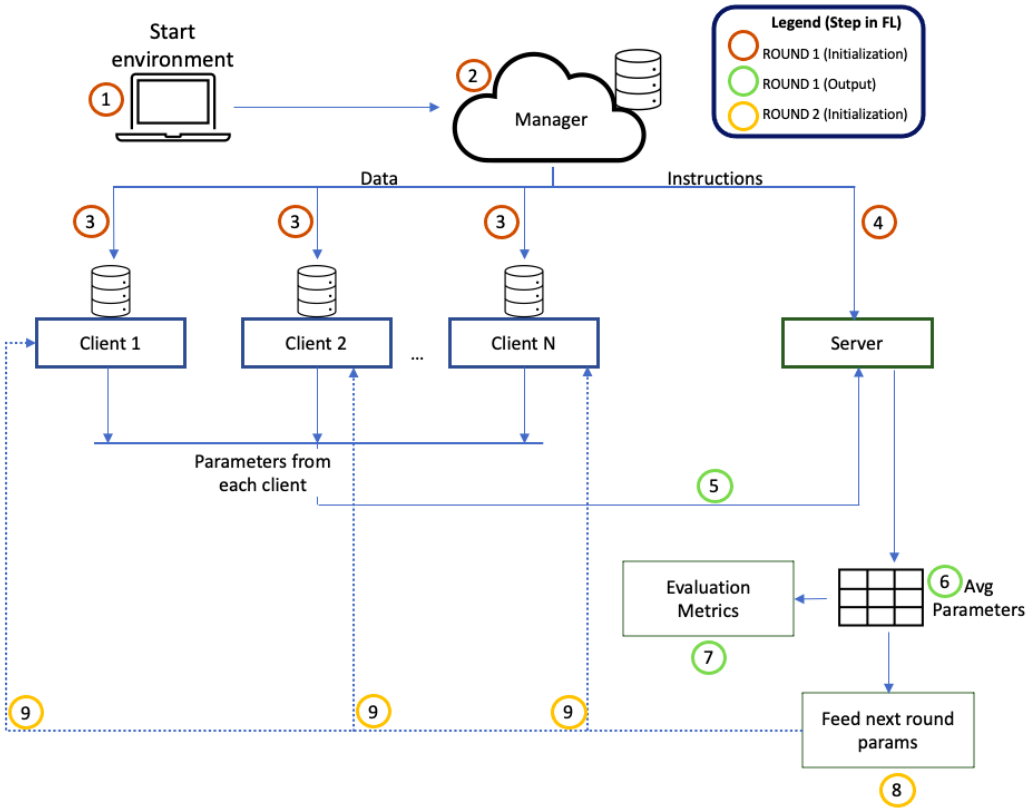

Figura 4.2: Arquitetura geral do ambiente proposto (baseado em [35]).

Para entendermos melhor como cada parte do ambiente funciona em cada etapa, iremos descrever a atividade e responsabilidade para cada integrante. Dito isso, vale ressaltar que na figura temos somente um exemplo simplificado, que mostra somente a inicialização de todo ambiente e o inicio do processo, visto que o número de *rounds* é um parâmetro a ser definido e, o número de *Clients* que também pode mudar, dependendo de qual a simulação proposta. Podemos acompanhar pela legenda, em qual etapa do nosso ambiente estamos lidando, ou seja, teremos a explicação mais a fundo de três etapas: *round* 1 inicialização, *round* 1 *output*, *round* 2 inicialização.

Com isso, temos a primeira etapa do *round* 1 inicialização, *Start Environment* (**Etapa ①**). Nessa etapa o utilizador deve fornecer algumas combinações de parâmetros obrigatórios, são elas:

- *dataset*: Escolher entre os quatro tipos disponíveis, as abreviações possíveis são:
  1. "nb15"corresponde ao *UNSW-NB15*
  2. "cic"corresponde ao *CIC-IDS2017*

3. "ton"correspondendo ao *ToN-IoT*
4. "wustl"que corresponde ao *WUSTL-IIOT-2021*

- *num_rounds*: Dependendo da experiência que deseja-se fazer. Indicado valores entre 2 e 100.
- *num_clients*: Decisão de quantos clientes vão participar da simulação. Indique um valor entre 2 a 40.
- *method*: Para a decisão do método de *Dimensionality Reduction* a ser utilizado, temos três opções:
  1. "regular": usaremos o dataset todo sem executar nenhum tipo de *Dimensionality Reduction*.
  2. "pca": aplicaremos o PCA para reduzir a dimensionalidade dos dados.
  3. "feat_selection": aplicaremos a *Dimensionality Reduction* baseada em *Feature Importance* a partir de uma aplicação de RF.
- *classification_type*: Para o tipo de classificação que vamos lidar temos duas opções, lidaremos com caso "binary"e com "multi"(perceba que para o cenário multi-classe temos a aplicação somente para dois *datasets* são eles "cic"e "ton").
- *model*: Qual o modelo de ML será aplicado para o teste. Decidir entre:
  1. "LogisticRegression": modelo treinado *SGDClassifier* com os parâmetros (*max_iter = 1*;*loss = 'log'*; *warm_start = True*)
  2. "SVM": modelo treinado *SGDClassifier* com os parâmetros (*max_iter = 1*; *penalty = 'l2'* ; *loss = 'hinge'*; *warm_start = True*)
  3. "PassiveAggressiveClassifier": modelo treinado *PassiveAggressiveClassifier* com os parâmetros (*warm_start = True*; *max_iter = 1*)
  4. "Perceptron": modelo treinado *Perceptron* com os parâmetros (*penalty = 'l2'*; *warm_start = True*; *max_iter = 1*)

A partir dos parâmetros iniciais definidos, partimos para a inicialização do *Manager* (**Etapa ②**) utilizando o *PyTorch multiprocessing* para garantir que o processamento ocorra em fila e a memória do sistema seja distribuída entre todos clientes e servidor, sem que haja comunicação por meio de memória compartilhada.

O *Manager* é responsável pela etapa inicial do ambiente, *load*. Será responsável também pelo processamento dos dados, bem como aplicação de técnicas para auxiliar no desbalanceamento dos dados, utilizando o SMOTE. Também, aplicará a etapa de *Feature Selection*, previamente escolhida.

Responsável pela divisão dos dados entre os clientes que estão a participar da simulação $\{n\_clients+1\}$, temos que dividir todas amostras entre os clientes para as etapas de treino e teste e, enviar também uma parte dos dados para testes no servidor, o que será usado para testar o modelo global, de acordo com métricas de avaliação.

É importante ressaltar que, a cada divisão de dados, se faz necessário levar a cabo o balanceamento das classes, para garantir que se aproximem o máximo possível do balanceamento inicial. Por fim, o *Manager* é responsável pela divisão de dados para cada cliente, considerando o número de *rounds* da simulação. Viabilizando assim, a cada novo *round* a inserção

de novas amostras de treino, se necessário. Iremos testar a evolução do modelo global no cenário de novos dados em cada *round*, para concluirmos a evolução do ambiente como um todo, com o compartilhamento da informação entre os clientes. Essas etapas iniciais foram apresentadas na Seção 4.2, porém com algumas alterações para a etapa de verificação do balanceamento dos dados, em todas as divisões feitas. Com isso, temos a esquematização real do ambiente na Figura 4.3.

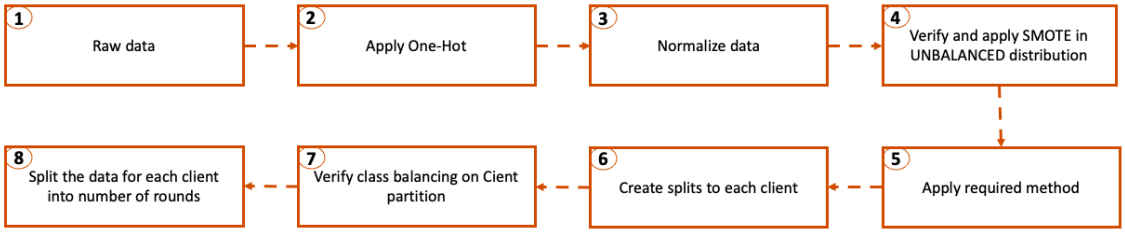

Figura 4.3: Arquitetura das etapas do *Manager*

Após o processamento dos dados e divisões em partições, devemos começar o compartilhamento da informação. Inicializando primeiro o Servidor (**Etapa ④**) enviando para o ramo, somente dados de teste. Do outro lado temos a inicialização de n clientes, em que cada um recebe informações iniciais de seus dados, de forma independente, cliente {1,2,3,...,n} (**Etapa ③**)

Dentro do ramo dos clientes vale ressaltar que, todos executam o mesmo processo, mas cada um com seus próprios dados, e de forma independente de forma simultânea. Para entendermos melhor como funciona, podemos pensar que os dados chegam ao cliente já particionados pelos *rounds*, ou seja, a cada novo *round* o cliente pode necessitar adicionar à sua etapa de treino novos dados.

Temos que, em primeiro momento, inicializar o modelo de ML indicado pelo *Manager*, com isso criamos uma etapa de *get_model_parameters* à qual, recebe também os parâmetros, e como é o primeiro *Round* estaremos com todos campos vazios, ou seja, respeitando:

- *coef_* : *array* do formato *(1, n_features)* para caso binário e, para caso multi-classe o formato *(n_classes, n_features)*
- *intercept_* : *array* do formato *(1,)* para caso binário, multi-classe formato *(n_classes,)*

Iniciando cada *round*, executa-se uma verificação do balanceamento dos dados. Essa verificação é feita internamente para cada cliente, e se necessário, aplica-se a técnica de *Oversampling* SMOTE. Ao chegarmos a etapa de treinar nosso modelo dentro dos clientes de forma independente, cada um separa seu grupamento de dados para treino, executa o processo de treino internamente, obtendo assim uma combinação de parâmetros, no formato [*coef_ , intercept_*], os quais serão posteriormente enviados ao servidor.

Outra importante etapa do *client* é a função de *evaluate* para verificar o desempenho do modelo, utilizando o grupamento de dados de teste para tal. Aqui, retornamos todas as métricas de avaliação, para verificarmos o desenvolvimento e evolução com o decorrer dos *rounds*, a saber: *{"f1-score", "ROC_curve", "accuracy"}*. A contabilização dessas métricas será utilizada para decisões nos próximos *rounds*, porém não são compartilhadas nem com outros clientes, nem com o servidor.

Após a execução de todos os passos dos clientes, cada um envia para o Servidor (**Etapa ⑤**) seus parâmetros obtidos. Iniciando a etapa verde na legenda, o *round* 1 *output*.

Como já dito, esses parâmetros possuem um formato de combinação de *Attributes* são eles: [*coef_ , intercept_*]. Onde o formato *coef_* corresponde aos pesos das *features* por classe, já em *intercept_* conhecido como o local onde o eixo y é cortado pela função.

Com isso, partimos para o Servidor (**Etapa ⑥**) onde esses parâmetros dos clientes serão todos utilizados para processar a estratégia de agregação chamada Fed Avg que já foi explicada na Subseção 2.4.1. Vale ressaltar que todos clientes participam de todos *rounds* durante nossas experiências, ou seja, todos parâmetros recebidos pelo servidor, serão utilizados. Após a aplicação do algoritmo de agregação ficamos com um modelo global, ou seja, o resultado dos parâmetros de cada cliente impacta no resultado final deste *round*. O formato do modelo global segue a configuração [*coef_ , intercept_*].

Ainda no lado do *Server* após a averiguação e determinação do modelo global, temos a avaliação dos parâmetros, utilizando dados de testes do servidor. Entrando na fase *Evaluation Metrics* (**Etapa ⑦**) onde, obtemos métricas de avaliação como obtemos nos clientes (*{"f1-score", "ROC_ curve", "accuracy"}*) essa fase de avaliação somente é utilizada para ser identificado, a evolução das métricas nos *rounds*, porém essas métricas globais ficam somente no lado do servidor.

Aqui se encerra a execução do *round* 1, onde pode-se identificar etapas de pre-processamento dos dados, a inicialização dos parâmetros vazios, e a orquestração do processo por parte do *Manager*. Obtendo para cada cliente as suas métricas de avaliação, e também a avaliação do modelo global. Já podemos aqui identificar a colaboratividade entre os clientes para a criação do modelo global.

A partir disso, iremos dar continuidade para o *round* seguinte, e pode-se dizer que do *round* $2, ..., n$ teremos um comportamento semelhante de processo. De seguida descreveremos as diferenças do novo *round*, começando pela etapa de inicialização, e pode-se entender que nos restantes *rounds* teremos uma replicação similar das etapas.

Para o último passo do diagrama temos as etapas da cor amarela, que representam na legenda o *round* 2 inicialização, temos a fase *Feed next rounds params* (**Etapa ⑧**) que corresponde a etapa em que o servidor envia a cada cliente o modelo global computado no *round* anterior. Agora daremos andamento para a nova inicialização do próximo *round*, que pode ser visualizado na Figura 4.2 pela linha pontilhada, caracterizando que o próximo *round*. Deste modo, os parâmetros do modelo global serão enviados para os clientes a partir do segundo round, deixando de se fazer a inicialização vazia do primeiro round.

No lado do cliente (**Etapa ⑨**) temos a realimentação dos clientes com os parâmetros globais. Dependendo do processo decidido inicialmente, os clientes podem ou não adicionar novas amostras para seu treino, já pré-divididas pelo *Manager*, isso depende do método escolhido.

A partir dos dados de treino já atualizados, partiremos para a etapa de computar um novo modelo do *round* 2. É importante perceber que o modelo já partirá de instruções do conjunto de parâmetros, do modelo global proveniente do *round* anterior. Esse treino, a partir de parâmetros já pré-definidos, se faz altamente necessário para a execução correta da simulação de FL. Denotando a importância de termos definido inicialmente, que todos modelos de ML utilizados, possuam nos seus parâmetros do modelo { $warm\_start = True$} ou seja, evitam que os pesos sejam apagados ao fazermos o novo treino dos dados.

É importante citar que, a cada novo round, verificamos se a distribuição das classes está condizente com a distribuição original, e se não estiver, aplicaremos novamente o SMOTE. Com os resultados do modelo novo obtido por cliente, temos uma nova etapa que não se faz necessária no *round* inicial, porém, em todos próximos, se faz, que diz respeito a

decisão de quais parâmetros o cliente irá enviar para o *Servidor*, e essa avaliação vem do armazenamento em cada *round* das métricas de avaliação por *round*. Com isso, no 2º *round* o cliente terá a etapa de verificar se o *f1-score* obtido no presente *round* ultrapassou o *f1-score* armazenado pelo cliente do seu *round* anterior.

A partir da análise feita internamente por cada cliente, ele terá que enviar ao Servidor os seus parâmetros coletados com maior performance, utilizando o *f1-score* como filtro. Esses parâmetros irão continuar todo o processo a partir da etapa ⑤, até chegar ao novo modelo global, calculando sua métrica de avaliação para o presente *round*. Consequentemente, pode-se perceber se o nosso ambiente simulado, consegue reter evolução, a partir da colaboração de todos clientes.

Com o fim dos *n rounds*, obtemos os resultados de evolução de cada cliente, bem como os parâmetros globais de cada *round* calculado pelo lado do servidor, o que viabilizará o entendimento da melhor combinação de parâmetros, modelos e processos para cada *dataset*. Deve-se entender que, possuímos algumas variações de possíveis combinações, visto que além dos processos de escolha, ainda trabalhamos com dois tipos de classificações, binária e multi-classe. Pode-se entender que as etapas consoantes a cada tipo de classificador é similar, o que as difere é direcionada com os parâmetros calculados (coef_, intercept_).

Em suma, neste capítulo explicitamos cada detalhe do ambiente de simulação construído, bem como, informações das fontes de dados selecionadas. Tivemos como objetivo descrever as etapas fundamentais, com suas respectivas atividades presentes na Figura 4.2.

This page is intentionally left blank.

Capítulo 5

Resultados

Neste capítulo, primeiramente denotaremos as variações de técnicas a serem testadas, as quais serão aplicadas para cada *dataset*. Seguido, performaremos testes para classificadores binários na Seção 5.1, que englobará todas as métricas para o cenário, bem como suas evoluções, devido à colaboração que o ambiente simulado propõe. Além disso, na Seção 5.2 temos as avaliações do cenário de identificação do tipo de ataque, onde abordaremos os desafios provenientes de alguns ciber ataques. A partir disso, concluiremos na Seção 5.2, as vantagens na aplicação de FL utilizando seu principal algoritmo de agregação Fed Avg. Na Seção 5.3 abordamos os resultados temporais, denotando o gasto computacional investido em cada simulação. Finalmente Seção 5.4 temos de forma resumida os resultados obtidos.

Para se compreender melhor sobre as simulações criadas, com cada um dos *datasets*, utilizando técnicas de FL, deve-se optar por alguns parâmetros antes de iniciarmos nosso ambiente simulado, os quais podem ser identificados abaixo:

1. *dataset*: "nb15", "cic", "ton"ou "wustl".
2. *classification_type*: "binário"ou "multi-classe".
3. *method*: "regular", "pca"ou "feat_selection".
4. *model*:"LogisticRegression", "SVM", "PassiveAggressiveClassifier"ou "Perceptron".
5. *num_rounds*: 20.
6. *num_clients*: 20.

Segmenta-se os resultados em duas partes, a primeira focará no contexto binário dos *datasets* e, na segunda parte, buscou-se tirar conclusões dos testes no contexto multi-classe. Com isso, pode-se entender a combinação de doze testes, para cada contexto proposto, onde pode-se identificar a variação de modelos ML, e diferentes formas de executarmos a redução de dimensionalidade, os quais, estão listados na Tabela 5.1.

5.1 Aplicação cenário Binário

Primeiro, para obtenção dos resultados, foi necessário, mantermos constante o número de *rounds*, e número de clientes, o que permite avaliar os resultados das experiências de forma comparativa. Para tal utilizamos 20 clientes e 20 *rounds*. Em cada teste feito, guardamos

Dimensionality Reduction	***Model***
Regular	*LogisticRegression*
Regular	SVM
Regular	*PassiveAggressiveClassifier*
Regular	*Perceptron*
PCA	*LogisticRegression*
PCA	SVM
PCA	*PassiveAggressiveClassifier*
PCA	*Perceptron*
Feat_selection	*LogisticRegression*
Feat_selection	SVM
Feat_selection	*PassiveAggressiveClassifier*
Feat_selection	*Perceptron*

Tabela 5.1: Conjunto de testes para as simulações em cada contexto proposto, binário e multi-classe

as métricas já abordadas nessa dissertação, como métrica principal temos *f1-score*, que foi considerada no critério de escolha de quais parâmetros devem ser enviados no lado do cliente.

Os resultados obtidos se dividem em duas partes: dados referentes ao desenvolvimento interno dos clientes, e os do servidor. Utilizamos como base a arquitetura proposta na Figura 4.2. Com isso, temos dois lados em constante adaptação dos parâmetros dos modelos de ML, ou seja, com a adaptação em cada *round* dos novos parâmetros, avaliados a partir de métricas de avaliação tanto interna de cada cliente, quanto na métrica individual do Servidor, coletada a partir da aplicação do Fed Avg. Pode-se entender que coletaremos métricas a cada *round*, são elas:

1. **X_train** : Número de amostras que cada cliente utiliza para treino no *round*
2. **Min_f1** : Valor mínimo do *f1_score* dentre todos clientes
3. **Max_f1** : Valor máximo obtido *f1_score* dentre todos clientes
4. **Avg_f1** : Valor médio do *f1_score* dentre todos clientes
5. **Avg_acc** : Valor médio do *accuracy* dentre todos clientes
6. **Avg_roc** : Valor médio da *ROC-Curve* dentre todos clientes
7. **Server_f1** : Valor proveniente do modelo global calculado, *f1_score*

***Dataset WUSTL* 2021:**

Iniciaremos por demonstrar um exemplo da coleta feita para o *dataset wustl*. Para tal, foi passado para cada cliente aproximadamente 13.948 amostras de treino e 3.484 de teste. Dentre as amostras de treino, são divididas em 20 *rounds*, com a proporção entre as classes a ser mantida durante toda a simulação é {"Normal": 90.79%, "Ataque": 9.21%}. O servidor, como será utilizado somente para avaliar o modelo global, receberá somente 3.484 amostras de teste. Deve-se entender que inicialmente cada cliente no *round* 0 começa com 698 amostras de treino e, a cada novo *round* adiciona-se 698 novas amostras, até ficar com aproximadamente 13.948 amostras no último *round*. Importante ressaltar que, no início

de cada *round*, onde os clientes recebem novas amostras para adicionar aos seus dados de treino, se faz necessário a execução de uma verificação do balanceamento das classes, com o objetivo de manter a proporção entre as distribuições em todos clientes.

Com isso, para o cenário com adição constante de 698 dados por novo *round* obtivemos como resultado a Tabela 5.2, onde pode-se visualizar a evolução de diferentes métricas para cada *round*, são elas, *Min_f1, Max_f1, Avg_f1* que corresponde aos valores no lados dos clientes de menor, maior e a média da métrica *f1-score* respectivamente. Na tabela, leva-se em conta todos clientes disponíveis, identificando a evolução ao passar dos *rounds*. Por fim, temos a coluna do *Server_f1* que corresponde aos resultados obtidos do *f1-score* do lado do Servidor, após a aplicação da técnica Fed Avg com os parâmetros de todos clientes do *Round*.

Os resultados obtidos do cálculo da métrica de avaliação *f1-score* para todos clientes, podem ser visualizados por todos experimentos em Figura 5.1. Pode-se perceber que a combinação de {feat_selection, PassiveAggressiveClassifier} obteve a métrica de avaliação mais alta.

A partir da melhor combinação de parâmetros, criamos Tabela 5.2, que corresponde aos valores das métricas para o melhor cenário teste. Para interpretarmos, pode-se focar nas informações em sublinhado na tabela. Primeiramente temos *Min_ Clients*, pode-se identificar que obteve valores iniciais de 53% no *Round* 0 obtendo 92% no *Round* 19, denotando a evolução dos parâmetros. Também se identifica, a evolução do lado dos maiores valores do *f1-score* nos *rounds*, *Max_ Clients* partindo de 93% para aproximadamente 98%. Obtendo de forma similar a evolução do *Avg_ Clients* partindo de 86% para 94% no último *round*.

Avançando na tabela (Tabela 5.2), temos a evolução do lado do *Server* (métrica *Server_f1*) que por ser alimentado com o modelo global, proveniente dos parâmetros de todos clientes, já obtém, desde o primeiro *round* valores elevados do *f1-score*. Para a avaliação do lado do servidor, é importante relembrar que utilizamos dados de teste provenientes do próprio servidor. Com isso, temos no *Round* 0 com 91%, porém, pode-se visualizar o ganho de informação se comparado ao *Round* 19 alcançando 96% do *f1-score*.

A Tabela 5.2 configura uma rodada completa da aplicação do cenário de FL, ou seja, todos *rounds* executados. Para se perceber melhor os resultados do lado do cliente e servidor temos a Figura 5.1. Na cor azul temos a evolução da métrica *Avg_ Clients* e em laranja o desenvolvimento do f1-score do lado do servidor.

***Dataset NB*-15:**

Para o *dataset NB-15* simulamos o processo análogo descrito no Capítulo 4. Cada cliente parte do recebimento de 1.428 amostras para treino, e a cada novo *round* recebe novas amostras da mesma quantidade. Para a etapa de teste, todos clientes recebem 7.121 e o servidor, que também é responsável por reter informação do modelo global, recebe a mesma quantidade de dados. A proporção entre as classes a ser mantida em todos *rounds* é de {"Normal": 88,77%, "Ataque": 11,23}.

Com isso, iniciamos o entendimento dos resultados obtidos, os quais denotam que o cenário de coleta dos dados, se trata de uma *testbed* mais simples, se comparada ao *dataset* anterior. Isso se dá, por termos obtido dentro das nossas doze simulações iniciais, um *f1-score* aproximadamente 100%, ou seja, um classificador com muita precisão e poucos ou quase nenhum, erro. Essa conclusão inicial, vai de encontro com as expectativas, visto que, é tido por artigos um dos *datasets* mais comuns em cenários de ciber ataque aplicados como *Supervised* ML, porém utilizado de forma centralizada [62].

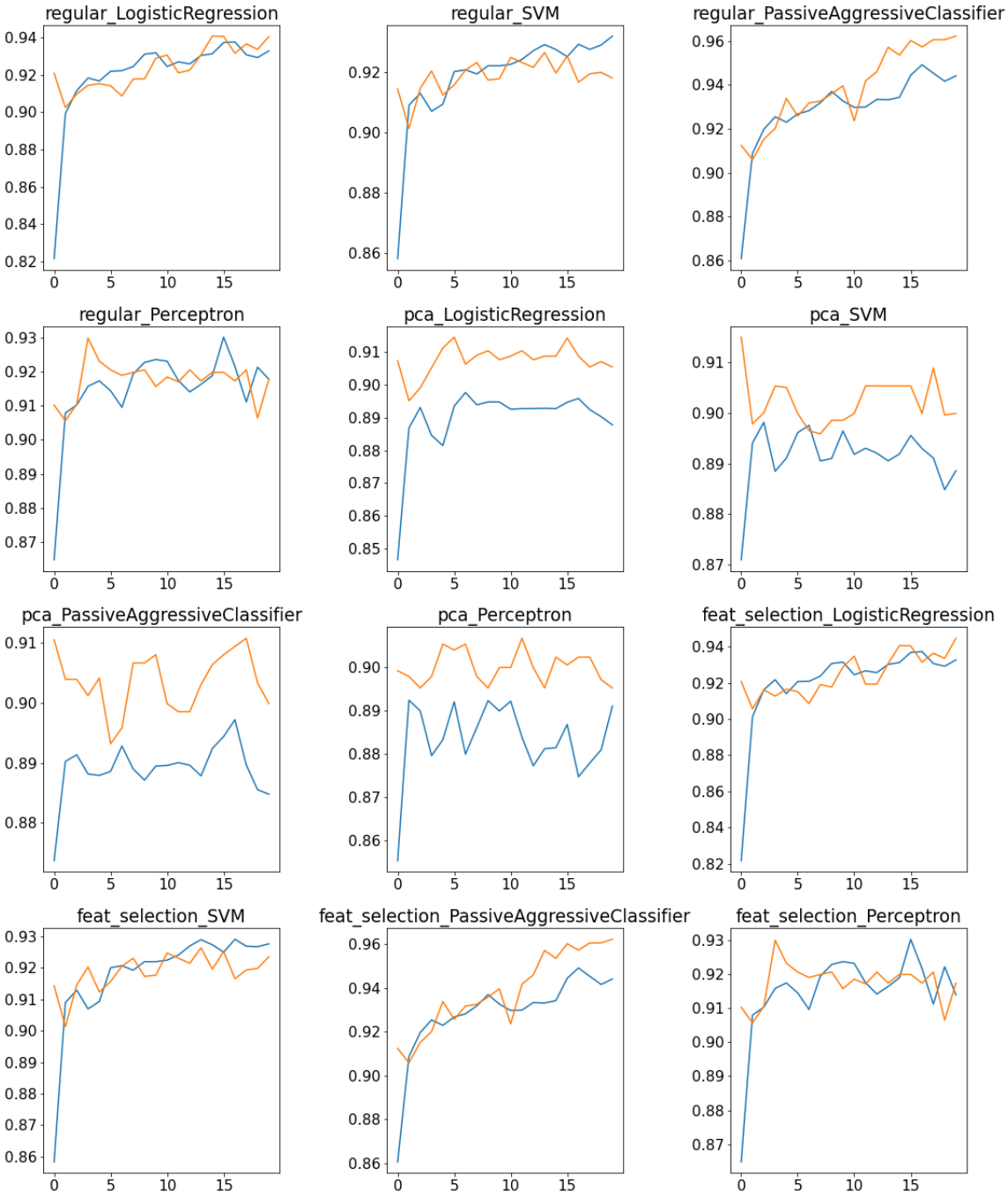

Figura 5.1: Resultado dos testes para *WUSTL* com adição de dados em todos *rounds*. No eixo X temos cada *Round* (0 a 19) e no eixo Y pode-se identificar os valores da evolução da métrica *f1-score*. A linha em azul representa a evolução dos clientes média. O que nos permite verificar que a combinação de {feat_selection, PassiveAggressiveClassifier} chega a 94,4% de f1_score, por parte dos clientes, enquanto os outros modelos ficam abaixo dos 93%. Em laranja temos a evolução por parte do *Server* podemos identificar que a mesma combinação obteve melhores resultados, chegando a obter f1-score de 96%.

Round	X_train	Min_f1	Max_f1	Avg_f1	Avg_acc	Avg_roc	Server_f1
0	698	0.531	0.931	0.861	0.976	0.985	0.912
1	1396	0.833	0.946	0.909	0.981	0.991	0.906
2	2094	0.862	0.95	0.92	0.983	0.992	0.915
3	2792	0.896	0.95	0.925	0.983	0.992	0.92
4	3490	0.887	0.947	0.923	0.984	0.992	0.934
5	4188	0.904	0.956	0.927	0.985	0.993	0.926
6	4886	0.904	0.966	0.928	0.936	0.993	0.932
7	5584	0.904	0.966	0.932	0.986	0.993	0.933
8	6281	0.904	0.963	0.937	0.985	0.993	0.936
9	6978	0.89	0.956	0.933	0.986	0.993	0.94
10	7675	0.868	0.956	0.93	0.982	0.993	0.924
11	8372	0.889	0.955	0.93	0.985	0.993	0.942
12	9069	0.899	0.971	0.933	0.986	0.993	0.946
13	9766	0.899	0.971	0.933	0.942	0.993	0.957
14	10463	0.905	0.961	0.934	0.986	0.994	0.953
15	11160	0.906	0.976	0.944	0.989	0.994	0.96
16	11857	0.923	0.976	0.949	0.988	0.994	0.957
17	12554	0.917	0.976	0.945	0.988	0.994	0.961
18	13251	0.916	0.976	0.942	0.987	0.994	0.961
19	13948	0.918	0.978	0.944	0.988	0.994	0.962

Tabela 5.2: Resultado das métricas para a simulação utilizando o *dataset wustl* para uma classificação binária. No cenário temos adição de dados a cada novo *round*

Inicialmente, podemos entender que o melhor resultado obtido de *f1-score* foi da simulação {regular, PassiveAggressiveClassifier}. Pode-se identificar as métricas da simulação na Tabela 5.3. Onde se identificam altas métricas desde o primeiro *Round*, porém, se faz possível identificar a evolução de acordo com cada novo *Round*.

No que diz respeito aos campos sombreados, estes são de grande importância para explorarmos a evolução das métricas. Temos o *Min_f1* no *round* 0 com 96% do *f1-score*, e ao final da simulação todos os clientes consolidam sua evolução, alcançando 98%. Analogamente temos o *Max_f1* onde podemos perceber uma evolução, alcançando 99% na última rodada da simulação. Outra importante métrica é, a *Avg_f1* de cada *Round*, a qual é responsável pela média encontrada pelos clientes, com isso identificamos que assume inicialmente o valor de 97,9% e no último *round* de 99,3%, denotando a evolução média no lado dos clientes.

Partindo do entendimento das métricas do lado dos clientes, pode-se perceber o ramo do *Server_f1*, onde o valor do *f1-score* quase se equivale ao *Avg_f1*, porém, obtendo ao final 99,2% no último *Round*. Também, pode-se verificar durante os *Rounds*, os valores para a métrica *Server_f1*, a qual, alcança o maior *f1_score* no *Round* 8, o que chama atenção é que os clientes continuam a ganhar informação, ou seja, clientes conseguem obter melhoria a cada *Round* a ponto de chegarem a melhor métrica possível no *Round* 19.

Sabe-se que, a tabela (Tabela 5.3) corresponde ao melhor resultado obtido para a simulação, porém, para chegarmos a esse resultado executamos os doze testes propostos, os quais podem ser visualizados em Figura 5.2. Em azul temos a evolução da *Avg_f1* que é a média dos clientes, temos um comparativo direto com o *f1-score* computado do lado do servidor *Server_f1*.

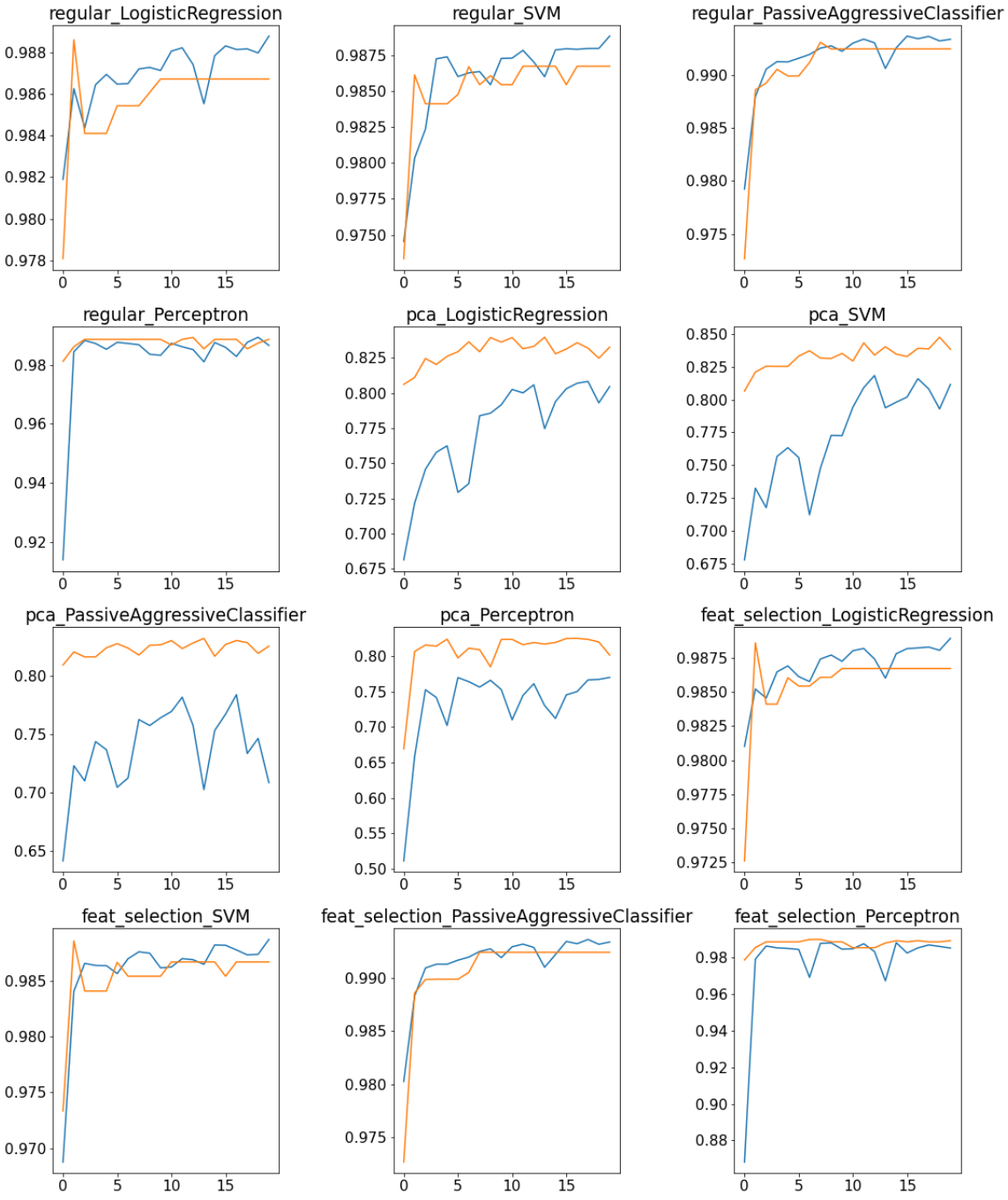

Figura 5.2: Resultado dos testes para *NB-15* com adição de dados em todos *rounds*. No eixo X temos cada *Round* (0 a 19) e no eixo Y pode-se identificar os valores da evolução da métrica *f1-score*. A linha em azul representa a evolução dos clientes média. O que nos permite verificar que a combinação de {regular, PassiveAggressiveClassifier} chega a 99,3% de *f1-score* médio, por parte dos clientes. Em laranja temos a evolução por parte do *Server* podemos identificar que a mesma combinação obteve melhores resultados, chegando a obter *f1-score* de 99,2%.

Round	X_train	Min_f1	Max_f1	Avg_f1	Avg_acc	Avg_roc	Server_f1
0	1428	0.969	0.992	0.979	0.995	0.997	0.973
1	2856	0.969	0.992	0.988	0.993	0.998	0.989
2	4284	0.984	0.994	0.991	0.997	0.999	0.989
3	5712	0.988	0.994	0.991	0.997	0.999	0.991
4	7140	0.988	0.995	0.991	0.997	0.999	0.99
5	8568	0.989	0.995	0.992	0.997	0.999	0.99
6	9996	0.989	0.995	0.992	0.998	0.999	0.991
7	11424	0.99	0.995	0.993	0.998	0.999	0.993
8	12852	0.99	0.995	0.993	0.997	0.999	0.992
9	14280	0.982	0.995	0.992	0.997	1.0	0.992
10	15708	0.989	0.996	0.993	0.998	1.0	0.992
11	17136	0.991	0.996	0.993	0.998	0.999	0.992
12	18564	0.987	0.996	0.993	0.997	0.999	0.992
13	19992	0.96	0.996	0.991	0.993	0.999	0.992
14	21420	0.978	0.996	0.993	0.998	0.999	0.992
15	22848	0.99	0.996	0.994	0.998	1.0	0.992
16	24276	0.991	0.996	0.993	0.997	1.0	0.992
17	25703	0.991	0.996	0.994	0.998	1.0	0.992
18	27130	0.983	0.996	0.993	0.997	1.0	0.992
19	28557	0.983	0.997	0.993	0.998	1.0	0.992

Tabela 5.3: Resultado das métricas para a simulação utilizando o *dataset NB-15* voltado para classificação binária. No cenário temos adição de dados por todos *rounds*

***Dataset CIC* 2017:**

Para as simulações em *dataset CIC-17* no mesmo ambiente que os testes citados anteriormente. Definiu-se que cada cliente recebe inicialmente 1.488 amostras para treino, e 7.429 dados para teste. Como conseguinte, o servidor também recebe a mesma quantidade de amostras de teste. Para a etapa de verificação constante da distribuição das classes, temos que, deve-se seguir {"Normal": 80,5%, "Ataque": 19,5%}.

Com isso, ao executarmos as simulações obtivemos resultados para todas os testes propostos, podem ser visualizados em Figura 5.3, onde temos em azul a evolução da métrica *Avg_f1* e em laranja temos a métrica *Server_f1*. Pode-se concluir que o melhor resultado obtido é correspondente a simulação {feat_selection, PassiveAggressiveClassifier}.

Observar-se que, a aplicação do modelo *PassiveAggressiveClassifier* obtém melhores métricas do que os outros modelos testados, chegando a obter mais de 8% que os outros testes. Para o lado do *Server_f1*, podemos identificar que a *Dimensionality Reduction* tanto a *regular feat_selection* conseguem obter valores quase semelhantes, porém o modo *feat_selection* consegue alcançar melhor *f1-score* com 90,3%.

Interpretando os melhores resultados obtidos, temos a evolução de todas as métricas por *round* em Tabela 5.4. Pode-se identificar a evolução do *Min_f1* saindo de 47,7% para 75,3%. Percebe-se, também, a evolução do *Avg_f1* partindo de aproximadamente 67% e chegando a 84%. Para as métricas coletadas no servidor, pode-se identificar a evolução da aplicação da agregação Fed Avg, podemos identificar a evolução de 73,9% para 90%.

***Dataset TON-IOT*:**

Por fim, temos o maior *dataset* utilizado para as simulações, um *dataset* com uma gama

Figura 5.3: Resultado dos testes para *CIC-17* com adição de dados em todos *rounds*. No eixo X temos cada *Round* (0 a 19) e no eixo Y pode-se identificar os valores da evolução da métrica *f1-score*. A linha em azul representa a evolução dos clientes média, o que nos permite verificar que a combinação de {feat_selection, PassiveAggressiveClassifier} chega a 84,5% para o *f1-score* médio, por parte dos clientes. Em laranja temos a evolução por parte do *Server* podemos identificar que a mesma combinação obteve melhores resultados, chegando a obter *f1-score* de 90,2%.

Round	X_train	Min_f1	Max_f1	Avg_f1	Avg_acc	Avg_roc	Server_f1
0	1488	0.477	0.789	0.67	0.869	0.949	0.74
1	2976	0.64	0.795	0.714	0.897	0.956	0.742
2	4464	0.64	0.809	0.727	0.894	0.965	0.787
3	5952	0.652	0.874	0.732	0.892	0.968	0.786
4	7440	0.668	0.874	0.765	0.903	0.975	0.817
5	8928	0.675	0.857	0.781	0.91	0.972	0.882
6	10416	0.683	0.857	0.779	0.903	0.978	0.803
7	11904	0.718	0.881	0.791	0.916	0.974	0.873
8	13392	0.705	0.881	0.798	0.912	0.978	0.891
9	14880	0.726	0.881	0.822	0.917	0.979	0.889
10	16368	0.735	0.895	0.842	0.927	0.981	0.898
11	17856	0.735	0.895	0.832	0.923	0.984	0.9
12	19344	0.734	0.89	0.822	0.915	0.981	0.899
13	20832	0.738	0.89	0.851	0.94	0.982	0.886
14	22320	0.734	0.89	0.852	0.922	0.983	0.891
15	23808	0.764	0.89	0.854	0.936	0.982	0.891
16	25296	0.802	0.893	0.864	0.937	0.984	0.897
17	26784	0.783	0.906	0.864	0.928	0.985	0.901
18	28271	0.763	0.906	0.845	0.928	0.985	0.904
19	29758	0.754	0.897	0.845	0.934	0.984	0.902

Tabela 5.4: Resultado das métricas para a simulação utilizando o *dataset cic-2017* voltado para classificação binária. No cenário temos adição de dados por todos *rounds*

vasta de *features*. Com isso, pode-se ser considerado a análise mais complexa, para o cenário binário. Para iniciar as simulações, se distribuiu a cada cliente 2.109 dados de treino e, a cada novo *round* envia-se essa mesma quantidade de novos dados. Para os dados de teste distribui-se 10.529, o que também foi enviado para etapa de avaliação do servidor. Importante ressaltar que neste *dataset* temos uma distribuição de ataques mais expressiva do que nos outros, obtendo assim o percentual de classes de {"Normal": 4,62%, "Ataque": 95,38%}. Pode-se perceber que, é a única *testbed* que coletou um maior número de amostras com ataque do que com funcionamento normal.

Com isso, ao executarmos as simulações obtivemos resultados para todos os testes propostos, os quais podem ser visualizados em Figura 5.4. Onde temos, doze gráficos, temos no eixo x a evolução dos *rounds* e no eixo y a evolução do *f1-score*. Para a interpretação das linhas, temos em azul a evolução da métrica *Avg_f1* e em laranja temos a métrica *Server_f1*. Pode-se concluir que o melhor resultado obtido é correspondente a simulação {feat_selection, PassiveAggressiveClassifier}.

Com a combinação de melhor resultado, podemos obter informações mais a fundo de cada uma das métricas coletadas na simulação. Na Tabela 5.5 temos em sombreado as informações mais relevantes. Pode-se perceber que a métrica *Min_f1* começa em 47% e no último *round* obtemos uma mínima de 75%, ou seja, quase 30% de melhoria. Para *Max_f1* temos inicialmente um valor de 78% alcançando por fim 89%.

Para a avaliação mais relevante da tabela, temos as métricas médias dos clientes *Avg_f1* que evolui mais de 20% durante a simulação. Por fim, temos a métrica mais relevante, considerando que é responsávela pela qualidade do modelo global *Server_f1* obtendo assim inicialmente *f1-score* de 74% e no último *round* chegamos a obter 90%.

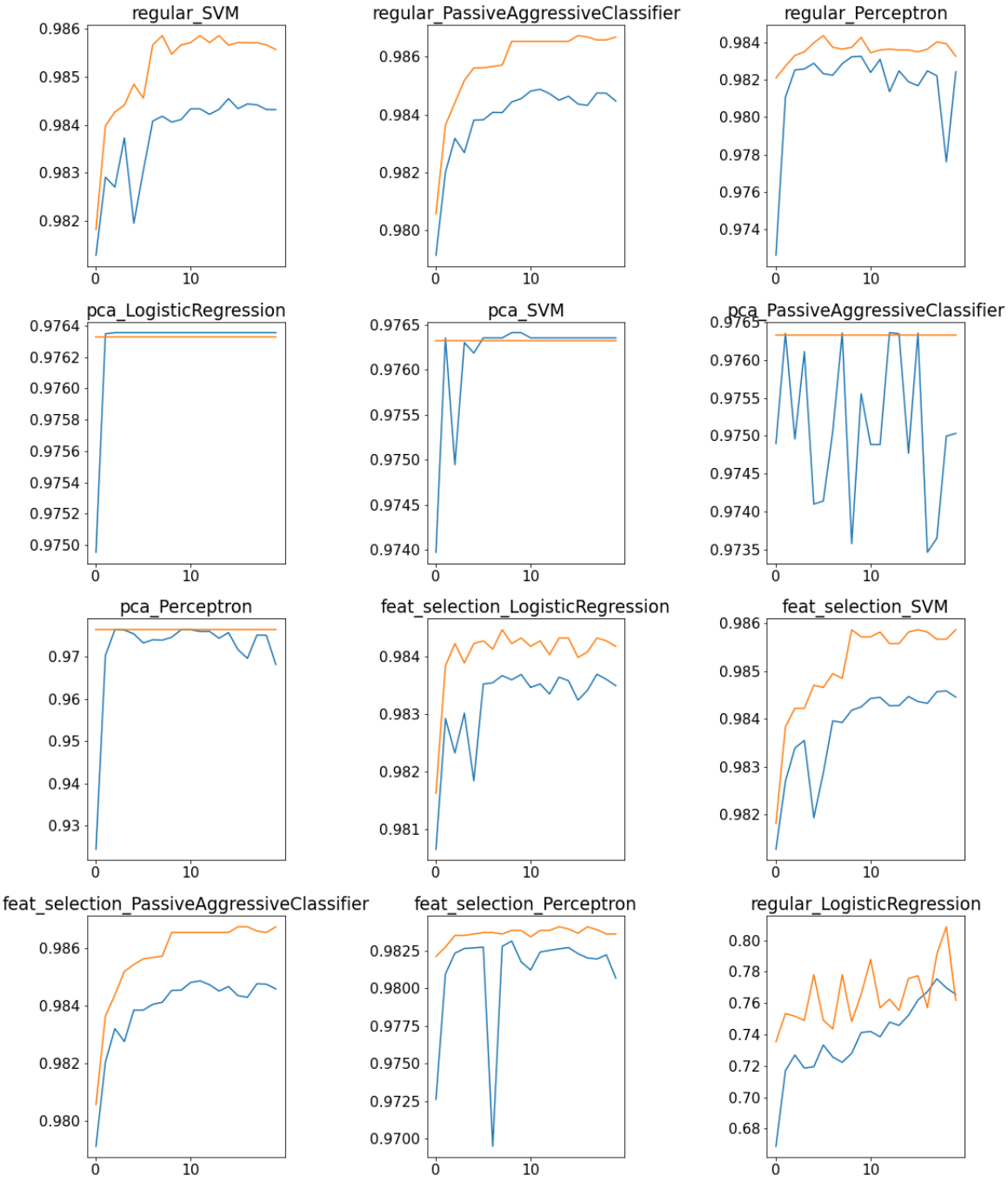

Figura 5.4: Resultado dos testes para *TON-IoT* com adição de dados em todos *rounds*. No eixo X temos cada *Round* (0 a 19) e no eixo Y pode-se identificar os valores da evolução da métrica *f1-score*. A linha em azul representa a evolução dos clientes média, o que nos permite verificar que a combinação de {feat_selection, PassiveAggressiveClassifier} chega a 98,4% para o *f1-score* médio, por parte dos clientes. Em laranja temos a evolução por parte do *Server* podemos identificar que a mesma combinação obteve melhores resultados, chegando a obter *f1-score* de 98,6%.

Round	X_train	Min_f1	Max_f1	Avg_f1	Avg_acc	Avg_roc	Server_f1
0	2109	0.968	0.984	0.979	0.959	0.76	0.981
1	4218	0.979	0.984	0.982	0.961	0.764	0.984
2	6327	0.982	0.985	0.983	0.96	0.768	0.984
3	8436	0.964	0.985	0.983	0.918	0.768	0.985
4	10545	0.981	0.986	0.984	0.918	0.78	0.985
5	12654	0.982	0.986	0.984	0.959	0.777	0.986
6	14763	0.982	0.985	0.984	0.945	0.763	0.986
7	16872	0.981	0.986	0.984	0.964	0.779	0.986
8	18981	0.982	0.986	0.985	0.951	0.776	0.987
9	21090	0.982	0.986	0.985	0.948	0.766	0.987
10	23199	0.984	0.986	0.985	0.97	0.778	0.987
11	25308	0.984	0.986	0.985	0.97	0.788	0.987
12	27417	0.983	0.986	0.985	0.945	0.781	0.987
13	29525	0.983	0.986	0.985	0.964	0.773	0.987
14	31633	0.982	0.986	0.985	0.948	0.781	0.987
15	33741	0.981	0.986	0.984	0.968	0.78	0.987
16	35849	0.979	0.986	0.984	0.925	0.767	0.987
17	37957	0.983	0.986	0.985	0.938	0.776	0.987
18	40065	0.984	0.986	0.985	0.965	0.783	0.987
19	42173	0.983	0.986	0.985	0.969	0.765	0.987

Tabela 5.5: Resultado das métricas para a simulação utilizando o *dataset TON-IoT* voltado para classificação binária. No cenário temos adição de dados por todos *rounds*.

Resumo contexto binário:

Para finalizar as análises binárias, faremos um resumo das melhores métricas encontradas nas simulações feitas, com o objetivo de buscar o entendimento, a realidade do resultado obtido para cada *dataset*. Pode ser verificar na Tabela 5.6 que a aplicação do modelo *PassiveAggressiveClassifier* foi unânime em relação a avaliação utilizando *f1-score* como métrica, alcançando resultados melhores que os outros modelos em todas as simulações. A maior diferença dentre os testes, foi na decisão de aplicar um método de *Dimensionality Reduction* o que somente para uma simulação *NB-15* obteve melhores resultados ao utilizarmos todos dados de forma *regular*, ou seja, todas as *features* sendo necessárias. Porém, nos outros *datasets* tivemos melhores resultados ao aplicar *feat_selection*.

Em um cenário de *Offline* ML, que nesse cenário seria aplicado para um classificador performando de forma *offline*, o qual podemos entender como a aplicação de modelos que necessitem de grande poder computacional, que a cada novo *round* necessitariam reiniciar o modelo treinado, sem utilizar os parâmetros prévios, com isso, não consegue obter uma fácil adaptação em cenários de grande escala. Por outro lado, o *Online Learning* traz o benefício de aprender com o passar dos *rounds*, sem precisar reiniciar, buscando evoluir corrigindo possíveis erros. Logo, pode-se identificar a melhor adaptação do modelo *PassiveAggressive* durante os *Rounds* em todos diferentes *datasets* binários, já que se trata de uma aplicação de *Online Learning*.

Rounds	*Dataset*	Simulação	Melhoria média	Avg_f1	Server_f1
0-19	WUSTL	{feat_selection, PassiveAggressive}	7%	0.860 - 0.944	0.912 - 0.962
0-19	NB-15	{regular, PassiveAggressive}	2%	0.979 - 0.993	0.973 - 0.992
0-19	CIC	{feat_selection, PassiveAggressive}	16%	0.670 - 0.844	0.739 - 0.902
0-19	TON	{feat_selection, PassiveAggressive}	2%	0.979 - 0.984	0.980 - 0.986

Tabela 5.6: Resumo dos melhores resultados para cada *dataset* no cenário binário. Podemos identificar a evolução do *f1-score* do *Round* 0 e no 19 no lado dos clientes em *Avg_f1* e no *Server_f1*

5.2 Aplicação cenário Multi-classe

Para o contexto multi-classe, foi necessário adaptarmos algumas das métricas anteriores coletadas no contexto binário, visto que teremos que diferenciar algumas conclusões, considerando os tipos de classes de ataques. Com isso dito, tivemos que adaptar principalmente a métrica *ROC-Curve*, para que fosse calculada por classe. Detalhamos abaixo as métricas para o cenário multi-classe, considerando a aplicação de todas para cada *round*:

1. **X_train** : Número de amostras que cada cliente utiliza para treino no *round*
2. **Min_f1** : Valor mínimo do *f1_score* dentre todos clientes
3. **Max_f1** : Valor máximo obtido *f1_score* dentre todos clientes
4. **Avg_f1** : Valor médio do *f1_score* dentre todos clientes
5. **Svr_f1** : Valor do *f1_score* proveniente do modelo global calculado
6. **Svr_acc** : Valor do *accuracy* proveniente do modelo global
7. **Svr_rocN** : Valor da *ROC-curve* para a classe N proveniente do modelo global

***Dataset CIC* 2017:**

Primeiramente deve-se entender mais sobre o *dataset CIC*, que será utilizado para criação de um classificador com quatro classes diferentes (funcionamento normal e três tipos de ciber ataque). Definiu-se que cada cliente recebe inicialmente 1.488 amostras para treino, para a etapa de testes foi distribuído 7.429 amostras. Deve-se entender que o servidor executa a agregação ao final de cada *round*, ou seja, também recebe a mesma quantidade de amostras de teste que os clientes. Para se perceber melhor sobre a distribuição das quatro classes analisadas temos que, {"Normal": 80,5%, "Ataque": 19,5%}, das amostras de ciber ataque, temos a divisão em {8,6%:"DoS Hulk", 6,3%:"PortScan", 4,6%:"DDoS"}.

Com as distribuições bem definidas, executamos todos os testes propostos, os quais podem ser visualizados na Figura 5.5. Pode-se entender que no eixo X temos o número de *rounds*

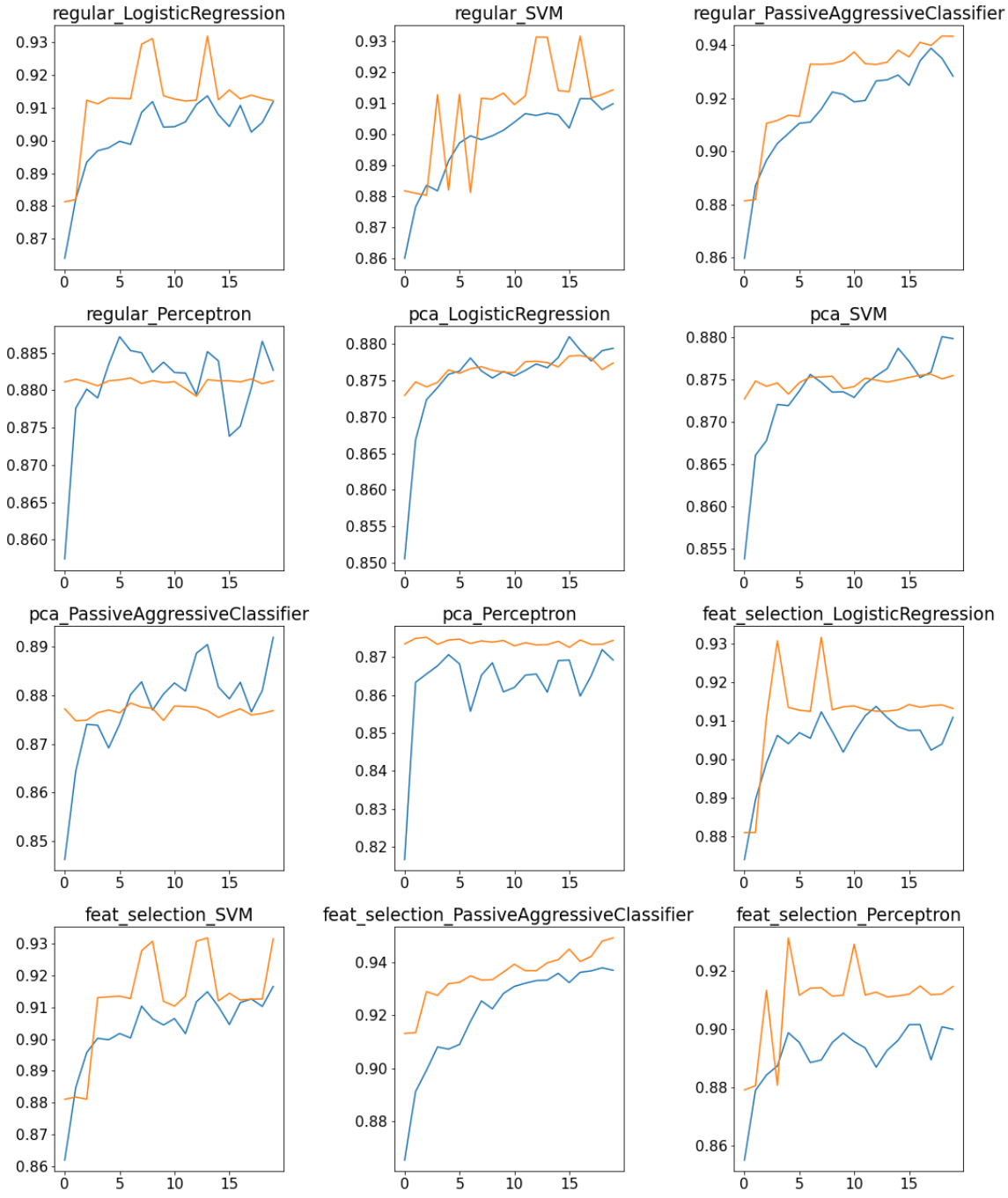

Figura 5.5: Resultado dos testes para *CIC* com adição de dados em todos *rounds*, contexto multi-classe. No eixo X temos cada *Round* (0 a 19) e no eixo Y pode-se identificar os valores da evolução da métrica *f1-score*. A linha em azul representa a evolução dos clientes média, o que nos permite verificar que a combinação de {feat_selection, PassiveAggressiveClassifier} alcança a 93,7% para o *f1-score* médio, por parte dos clientes. Em laranja temos a evolução por parte do *Server* podemos identificar que a mesma combinação obteve melhores resultados, chegando a obter *f1-score* de 94,9%.

executados, no eixo Y temos o valor da métrica *f1-score*, com isso, podemos identificar a sua evolução a partir da colaboração entre os clientes. Para bom entendimento, vale ressaltar que a linha em azul temos a evolução da métrica *Avg_f1* que corresponde a média dos clientes para aquele *round* e, em laranja, temos a evolução do *f1-score* calculado, proveniente do modelo global avaliado pelo servidor com seus dados de teste, *Svr_f1*. Conseguimos, assim, concluir que a combinação de {feat_selection, PassiveAggressiveClassifier} obteve melhores resultados.

Consequentemente, construímos a Tabela 5.7 e Tabela 5.8, que correspondem ao comportamento do melhor resultado coletadas para o *dataset* no contexto de multi-classificação. Para visualizarmos melhor a evolução dos clientes e do modelo global, podemos perceber que na Tabela 5.7 possuímos a evolução do *Min_f1* que inicialmente assume um valor de 75% e, ao fim das rodadas alcança aproximadamente 89%, denotando a presença no ambiente de clientes com *f1-score* maior que 89% na última rodada. A evolução do *Max_f1* também é expressiva, passando de 91% para 96%. *Avg_f1* compõe a métrica média coletada entre todos os vinte clientes em cada rodada, inicialmente de 86% e evoluindo para 94%. Pelo lado do servidor, podemos perceber que o *Svr_f1* parte de 91% no primeiro *round*, alcançando ao final, 95% no *f1-score*. Para o contexto, coletamos também o *Svr_ acc* que é amplamente utilizado como métrica principal pela academia, no contexto partimos de 92% e chegamos a 95%.

Para melhor entendimento sobre o classificador e, para percebermos mais a fundo qual das classes se faz mais complexa em quesito de avaliação para o cenário, coletamos na Tabela 5.8 as métricas da *ROC-Curve*. Pode-se perceber uma evolução em todas as classes com o passar dos *rounds*, grandes evoluções em relação à métrica para as classes 0 e 1. Porém, as classes 2 e 3 que se mostram em menor quantidade amostral, acarretaram uma maior dificuldade de serem identificadas com precisão, obtendo uma melhoria da métrica de menos de 1% entre o *round* 0 e 19.

***Dataset TON-IOT*:**

Para o segundo cenário multi-classe, temos o *dataset TON-IoT*, que providencia uma gama variada de amostras, o que vai de encontro com a busca por classes com representatividade maior que 1%. Optamos por selecionar oito classes diferentes, dentre elas dividimos em, sistema normal e diferentes ciber ataques, tendo uma distribuição de 4,4%, 95,6% respectivamente. Para os ciber ataques temos a divisão do percentual em {"Scanning": 30,4%, "DDoS": 26,3%, "DoS": 14,4%, "XSS": 11,6%, "Password": 7,6%, "Backdoor": 2,8%, "Injection": 2,5%}.

Com isso, podemos dar início a simulação, considerando o envio inicial de 2.109 amostras de treino para os clientes e, a cada novo *round* adicionam-se essa mesma quantidade de dados novos. Por outro lado, distribui-se dados de teste 10.529 amostras, o que também é enviado para etapa de avaliação do servidor.

Por fim, executou-se os doze testes, os quais podem ser identificados na Tabela 5.1. A partir desses cruzamentos de parâmetros construiu-se a Figura 5.6. Visualizamos que o modelo SVM obteve melhores resultados tanto para a *Dimensionality Reduction* para o caso "Regular"e "Feat_selection". Com a similaridade de resultados, optamos pela simulação mais ágil, com isso a melhor alternativa foi a combinação {regular, SVM}.

Explorando os resultados obtidos no ambiente simulado, criamos uma análise de todas as métricas para entendermos melhor o desenvolvimento da colaboratividade. Pode-se perceber na Tabela 5.9 e Tabela 5.10 a evolução de tais métricas. Tabela 5.9 demonstra algumas métricas relevantes como *Min_f1* com evolução de 43% para 67%, da mesma

Figura 5.6: Resultado dos testes para *TON* com adição de dados em todos *rounds*, contexto multi-classe. No eixo X temos cada *Round* (0 a 19) e no eixo Y pode-se identificar os valores da evolução da métrica *f1-score*. A linha em azul representa a evolução dos clientes média, o que nos permite verificar que a combinação de {regular, SVM} alcança a 70% para o *f1-score* médio, por parte dos clientes. Em laranja temos a evolução por parte do *Server* podemos identificar que a mesma combinação obteve melhores resultados, chegando a obter *f1-score* de 71%.

Round	X_train	Min_f1	Max_f1	Avg_f1	Svr_f1	Svr_acc
0	1488	0.756	0.911	0.865	0.913	0.918
1	2976	0.854	0.933	0.891	0.913	0.919
2	4464	0.844	0.933	0.899	0.929	0.932
3	5952	0.871	0.933	0.908	0.927	0.93
4	7440	0.88	0.933	0.907	0.932	0.935
5	8928	0.877	0.932	0.909	0.932	0.936
6	10416	0.878	0.94	0.918	0.935	0.937
7	11904	0.884	0.94	0.925	0.933	0.936
8	13392	0.883	0.945	0.922	0.933	0.936
9	14880	0.884	0.958	0.928	0.936	0.939
10	16368	0.889	0.958	0.931	0.939	0.941
11	17856	0.879	0.954	0.932	0.937	0.939
12	19344	0.886	0.954	0.933	0.937	0.939
13	20832	0.892	0.948	0.933	0.94	0.941
14	22320	0.893	0.953	0.936	0.941	0.942
15	23808	0.886	0.954	0.932	0.945	0.946
16	25296	0.88	0.954	0.936	0.94	0.942
17	26784	0.882	0.958	0.937	0.942	0.944
18	28271	0.904	0.958	0.938	0.948	0.948
19	29758	0.888	0.96	0.937	0.949	0.949

Tabela 5.7: Resultado das métricas para a simulação utilizando o *dataset CIC-17* voltado para classificação multi-classe. No cenário temos adição de dados por todos *rounds*.

forma pode-se perceber o aumento da *Max_f1* indo de 68% para 72%. A evolução geral dos clientes está diretamente demonstrada no *Avg_f1* indo de 56% inicialmente para 70%. Para o lado do *Svr_f1* pode-se identificar que, a evolução foi mais concisa, levando a um aumento de menos de 1% com a performance das simulações, porém devido à complexidade do *dataset* pode-se considerar uma grande evolução, visto a não exposição dos dados entre os integrantes do ambiente.

Na Tabela 5.10 denota-se uma maior complexidade do algoritmo identificar algumas classes, tendo em conta a quantidade de amostras enviadas, pode-se perceber que a evolução da *ROC-Curve* foi mais constante, devido à diversidade de amostras e classes analisadas. Pode-se perceber que a identificação do sistema normal foi a classe que obteve pior resultado avaliando o *trade-off* entre the *true positive rate (TPR)* e *false positive rate (FPR)*. Isso se da por mesmo tendo mais representatividade que duas outras classes, obteve mais dificuldades de ser identificada corretamente pelo modelo.

Resumo contexto multi-classe:

Por fim, pode-se concluir quais as melhores combinações de parâmetros para cada *dataset* no contexto multi-classe. De forma geral, pode-se perceber que o modelo *PassiveAggressiveClassifier* obteve melhores resultados tratando do *dataset CIC-17*, por outro lado, o *TON-IoT* obteve melhores resultados ao aplicarmos o SVM. Comparativamente em relação à questão da evolução da aplicação do FL, podemos perceber que nos dois cenários existe uma troca de informação que leva a um aperfeiçoamento do modelo. Porém, em um cenário com uma maior diversidade de classes, deve-se utilizar o máximo possível de amostras em cada cliente, visto que, a proporção das classes é altamente relevante, a qual se faz necessário ser seguida.

Round	Svr_roc0	Svr_roc1	Svr_roc2	Svr_roc3
0	0.96	0.966	0.983	0.987
1	0.966	0.968	0.983	0.987
2	0.97	0.978	0.984	0.994
3	0.972	0.979	0.985	0.993
4	0.977	0.98	0.986	0.993
5	0.978	0.983	0.987	0.993
6	0.981	0.987	0.984	0.993
7	0.984	0.988	0.986	0.993
8	0.985	0.992	0.985	0.993
9	0.986	0.992	0.987	0.993
10	0.986	0.992	0.987	0.993
11	0.987	0.99	0.988	0.993
12	0.987	0.993	0.987	0.993
13	0.987	0.994	0.988	0.993
14	0.988	0.994	0.988	0.993
15	0.988	0.995	0.988	0.993
16	0.988	0.994	0.988	0.993
17	0.988	0.995	0.988	0.993
18	0.988	0.995	0.988	0.993
19	0.988	0.995	0.988	0.993

Tabela 5.8: Complemento da Tabela 5.7. Abrangendo as métricas de *ROC-Curve* por classe 0 a 3, para a simulação utilizando o *dataset CIC-17* voltado para classificação multi-classe. No cenário temos adição de dados por todos *rounds*.

Portanto, para um resumo mais consolidado, pode-se identificar a Tabela 5.11 que demonstra a evolução presente em cada um dos *datasets* utilizados no contexto multi-classe. Pode-se perceber que obtivemos evoluções considerando os dois cenários propostos, 5% e 2% em média.

Percebe-se que quanto mais complexidade damos ao problema, mais difícil de obtermos métricas de classificação altas. De modo geral, alcançamos bons resultados, pois conseguimos criar um ambiente diverso, que proporcionou a exploração de diferentes combinações de parâmetros para buscarmos a evolução, contando com a colaboração dos indivíduos dentro da rede, devido à simulação utilizar a técnica descentralizada FL.

Rounds	*Dataset*	Classes	Simulação	Melhoria média	Avg_f1	Svr_f1
0-19	CIC	4	{feat_selection, PassiveAggressive}	5%	0.86 - 0.93	0.91 - 0.95
0-19	TON	8	{regular, SVM}	2%	0.56 - 0.70	0.70 - 0.71

Tabela 5.11: Resumo dos melhores resultados para cada *dataset* considerando o classificador multi-classe. Podemos identificar a evolução do *f1-score* do *Round* 0 e no 19 no lado dos clientes em *Avg_f1* e no Servidor em *Svr_f1*

Round	X_train	Min_f1	Max_f1	Avg_f1	Svr_f1	Svr_acc	Svr_roc0
0	2236	0.43	0.682	0.56	0.698	0.746	0.97
1	4472	0.582	0.682	0.639	0.695	0.745	0.97
2	6708	0.481	0.709	0.641	0.703	0.751	0.972
3	8944	0.545	0.709	0.642	0.635	0.681	0.973
4	11180	0.545	0.714	0.639	0.69	0.738	0.973
5	13416	0.58	0.719	0.656	0.712	0.76	0.973
6	15651	0.568	0.719	0.664	0.7	0.747	0.973
7	17886	0.577	0.716	0.662	0.704	0.751	0.973
8	20121	0.595	0.716	0.668	0.707	0.754	0.973
9	22356	0.595	0.712	0.671	0.7	0.75	0.973
10	24591	0.616	0.711	0.679	0.697	0.746	0.973
11	26826	0.613	0.715	0.679	0.698	0.746	0.973
12	29061	0.54	0.715	0.663	0.693	0.739	0.973
13	31296	0.627	0.713	0.678	0.691	0.737	0.973
14	33531	0.618	0.732	0.684	0.71	0.733	0.973
15	35766	0.626	0.732	0.684	0.694	0.727	0.973
16	38001	0.601	0.722	0.672	0.694	0.741	0.973
17	40236	0.606	0.722	0.681	0.703	0.75	0.973
18	42471	0.61	0.716	0.688	0.692	0.74	0.973
19	44706	0.666	0.716	0.7	0.707	0.756	0.973

Tabela 5.9: Resultado de métricas para a simulação utilizando o *dataset TON-IoT* voltado para classificação multi-classe. No cenário, temos adição de dados por todos *rounds*.

5.3 Percepção temporal

Para uma comparação temporal das simulações, temos na Tabela 5.12 a análise de todos os tempos obtidos durante os testes executados para cada *dataset*. Podemos identificar a presença da "Mngr(min)", que corresponde a etapa de pré-processamento que é executada pelo *Manager* (Figura 4.2) e, pode ser considerada o maior período de tempo da simulação em todos cenários testados.

Por outro lado, após a etapa de pré-processamento temos o tempo médio de treino/teste dos clientes e o tempo de teste do Servidor. Podem ser visualizado na coluna "Cl-Svr(s)"ilustrando o tempo por *round* por cada cliente em média, e no lado do *Server* corresponde a ao teste executado, após a averiguação do modelo global, tomando em média 0.3 segundos para ser executada. Pode-se entender que, para a etapa interna dos clientes, como necessita-se execução de treino com os dados de forma individual e teste torna um processo mais prolongado, toma-se mais do que 4x que o tempo de teste no *Server*.

5.4 Resumo

Por fim, podemos concluir que na seção dos resultados abordamos cada um dos *datasets*, para perceber mais sobre suas particularidades, bem como a evolução de suas métricas, o que nos permite assim apresentarmos uma visão geral de como executar alguns dos modelos de ML mais comuns, com suas devidas personalizações que permitem explorarmos o cenário descentralizado de aplicação do FL.

Round	Svr_roc1	Svr_roc2	Svr_roc3	Svr_roc4	Svr_roc5	Svr_roc6	Svr_roc7
0	0.909	0.985	0.871	0.747	0.849	0.908	0.938
1	0.91	0.986	0.875	0.755	0.848	0.905	0.941
2	0.913	0.99	0.878	0.767	0.847	0.908	0.939
3	0.913	0.99	0.879	0.774	0.847	0.909	0.939
4	0.913	0.989	0.879	0.778	0.853	0.91	0.939
5	0.913	0.99	0.88	0.775	0.849	0.908	0.939
6	0.913	0.991	0.881	0.78	0.848	0.909	0.939
7	0.913	0.99	0.879	0.773	0.852	0.91	0.938
8	0.913	0.99	0.879	0.781	0.852	0.909	0.937
9	0.914	0.99	0.88	0.771	0.849	0.91	0.938
10	0.913	0.99	0.88	0.763	0.853	0.91	0.94
11	0.913	0.991	0.881	0.773	0.851	0.911	0.938
12	0.914	0.991	0.88	0.77	0.839	0.911	0.937
13	0.915	0.991	0.881	0.771	0.85	0.91	0.937
14	0.914	0.991	0.881	0.771	0.851	0.91	0.937
15	0.915	0.991	0.881	0.77	0.856	0.909	0.937
16	0.914	0.991	0.881	0.772	0.856	0.911	0.937
17	0.914	0.991	0.881	0.768	0.853	0.911	0.938
18	0.914	0.991	0.882	0.775	0.858	0.911	0.937
19	0.914	0.991	0.881	0.767	0.857	0.911	0.938

Tabela 5.10: Complemento da Tabela 5.9. Abrangendo as métricas de *ROC-Curve* por classe 1 a 7, para a simulação utilizando o *dataset TON-IoT* voltado para classificação multi-classe. No cenário temos adição de dados por todos *rounds*.

Vale ressaltar que o modelo *PassiveAggressive* trouxe melhores resultados em quase todas aplicações teste, somente no cenário multi-classe para o *dataset TON-IoT* obtivemos melhores resultados com a aplicação do modelo SVM. Para o parâmetro de redução de dimensionalidade, temos que *regular* e *feat_selection* foram os mais comuns, o que não é uma surpresa, visto que em seus processos não exigiram uma grande perda de informação. Com a aplicação *regular*, temos a utilização do *dataset* todo, e ao utilizarmos a técnica *feat_selection* verificamos a *Importância das features*, e conforme a *threshold* opta-se pelas *features* mais relevantes. Em contrapartida, tivemos de forma unânime que a aplicação do PCA obteve resultados bem abaixo das outras, isso se dá, pela generalização não ter conseguido ter uma boa representação da informação dos dados, com isso, podemos dizer que ao aplicarmos o PCA tivemos grandes perdas de informação.

Em suma, aplicamos testes para colocar a prova a metodologia proposta (Capítulo 4) utilizando FL como técnica base de aplicações para o ambiente distribuído. Podemos assim concluir que mesmo sem utilizamos todas *amostras* disponíveis de forma agrupada, conseguimos performar classificadores com alto desempenho, sem necessitar de um poder computacional elevado, visto que os clientes (*edge devices*) utilizam sua própria memória para treinar os dados, o que evita o compartilhamento de informações privadas.

Simulação	*Dataset*	Tipo	Mngr(min)	Cl - Svr(s)
regular, LogisticRegression	nb15	Bin	22	8.0 - 0.3
	wustl	Bin	4	1.6 - 0.3
	cic17	Bin-Mult	54	4.5 - 0.3
	ton	Bin-Mult	8	5.1 - 0.3
regular, SVM	nb15	Bin	22	8.9 - 0.3
	wustl	Bin	3	1.2 - 0.3
	cic17	Bin-Mult	55	3.6 - 0.3
	ton	Bin-Mult	8	3.8 - 0.3
regular, PassiveAggressiveClassifier	nb15	Bin	28	8.6 - 0.3
	wustl	Bin	3	1.2 - 0.3
	cic17	Bin-Mult	57	3.7 - 0.3
	ton	Bin-Mult	8	3.8 - 0.3
regular, Perceptron	nb15	Bin	23	7.0 - 0.3
	wustl	Bin	3	1.2 - 0.3
	cic17	Bin-Mult	56	3.8 - 0.3
	ton	Bin-Mult	9	4.0 - 0.3
PCA, LogisticRegression	nb15	Bin	23	2.0 - 0.3
	wustl	Bin	4	1.2 - 0.3
	cic17	Bin-Mult	56	2.2 - 0.3
	ton	Bin-Mult	8	3.7 - 0.3
PCA,SVM	nb15	Bin	23	1.9 - 0.3
	wustl	Bin	4	1.0 - 0.3
	cic17	Bin-Mult	56	1.6 - 0.3
	ton	Bin-Mult	8	2.6 - 0.3
PCA, PassiveAggressiveClassifier	nb15	Bin	23	1.9 - 0.3
	wustl	Bin	4	1.1 - 0.3
	cic17	Bin-Mult	56	1.9 - 0.3
	ton	Bin-Mult	9	2.7 - 0.3
PCA, Perceptron	nb15	Bin	23	1.9 - 0.3
	wustl	Bin	4	1.1 - 0.3
	cic17	Bin-Mult	58	1.8 - 0.3
	ton	Bin-Mult	8	2.4 - 0.3
Feat_ selection, LogisticRegression	nb15	Bin	32	3.6 - 0.3
	wustl	Bin	6	1.4 - 0.3
	cic17	Bin-Mult	77	4.1 - 0.3
	ton	Bin-Mult	28	4.5 - 0.3
Feat_ selection, SVM	nb15	Bin	32	2.7 - 0.3
	wustl	Bin	6	1.2 - 0.3
	cic17	Bin-Mult	75	3.3 - 0.3
	ton	Bin-Mult	28	3.3 - 0.3
Feat_ selection, PassiveAggressiveClassifier	nb15	Bin	31	3.0 - 0.3
	wustl	Bin	7	1.3 - 0.3
	cic17	Bin-Mult	75	3.3 - 0.3
	ton	Bin-Mult	28	3.4 - 0.3
Feat_ selection, Perceptron	nb15	Bin	31	2.8 - 0.3
	wustl	Bin	6	1.2 - 0.3
	cic17	Bin-Mult	77	3.3 - 0.3
	ton	Bin-Mult	29	3.6 - 0.3

Tabela 5.12: Resumo do tempo de cada simulação. "Mngr"entende-se como tempo de pré-processamento executado pelo *Manager*, "Cl-Svr"correspondem ao tempo do treino/teste no Cliente e teste no Servidor

This page is intentionally left blank.

Capítulo 6

Conclusão

Nesta dissertação, definimos alguns domínios como ML, *computação de ponta*, IoT e FL. Apresentamos várias aplicações, as suas diferentes fontes de dados, focando em que algoritmo de agregação foi utilizado e, as suas diferentes métricas de avaliação, informação onde pode ser fundamental num domínio FL aplicado em dispositivos IoT.

Descrevemos os casos de uso do presente trabalho, destinado ao cenário de ciber segurança, centrado na criação de um classificador que permita a identificação de ataques num contexto FL. Destacamos a utilização de quatro conjuntos de *datasets* relevantes para a academia, visando explorar a descentralização e o seu impacto nas avaliações.

Passamos por definições importantes e compreendemos brevemente cada abordagem relevante para esta dissertação. Denotando os problemas mais relevantes num contexto FL, centrado na privacidade do cliente e o *trade-off* entre o poder computacional e a *performance* das métricas de avaliação. Tal como se verificou, durante o estado da arte, existem diferentes métodos para abordar os novas *frameworks* de FL. Para a dissertação, escolhemos utilizar a biblioteca *Flower (Flwr)* como base, uma vez que se adapta facilmente aos modelos ML, e também permite uma alta personalização das etapas de construção e aplicação do ambiente descentralizado.

Seguindo de, uma compreensão das etapas da metodologia proposta, e como o processo se relaciona (Figura 4.2). Pode-se identificar que explicamos como se executa a recolha dos parâmetros durante os *rounds* do FL, considerando a estratégia de agregação Fed Avg, a qual auxilia na evolução das simulações, permitindo o ambiente descentralizado alcançar em cada novo *round* uma evolução constante. Conquistando assim, no último *round* o pico da métrica. Para que isto se torne viável, recolhemos as métricas de avaliação do lado do cliente e do lado do servidor.

A partir dessas métricas, podemos quantificar evolução dos dispositivos com pior desempenho inicialmente. Utilizamos *Min_f1*, que atinge valores expressivamente superiores no seu último *round*, denotando a evolução das simulações. Atingindo um aumento de quase 40% no *dataset WUSTL* e, com a menor evolução, de menos de 2% para o conjunto de dados *NB-15*. Isso ocorre, porque todos os clientes desde a primeira formação, mesmo com poucos dados, já conseguem alcançar um bom desempenho para o classificador, obtendo mais de 96% no *f1-score*.

Em primeiro lugar, temos o resultado no cenário binário, onde obtivemos o maior ganho de informação, ao aplicar o modelo *PassiveAggressive*, obtendo assim melhores combinações com o tipo de redução de dimensionalidade para o conjunto de dados *dataset NB-15* e *"feat_selection"* para *WUSTL, CIC-17, TON-IoT*. A evolução média no final dos *rounds*,

é entre 2 a 16% (Tabela 5.6).

Para os multi-classificadores, temos uma melhoria média de 2% quando aplicamos o modelo SVM, combinado com a redução da dimensionalidade *"regular"* para o texto (CIC-17). Para o multi-classificador que utiliza o *TON-IoT* obtivemos um ganho de informação no último *round* de 5% para o modelo do *PassiveAggressive*, aplicando o *"feat_selection"* (Tabela 5.11).

Assim sendo, constatamos a evolução para os testes que lidam com a identificação do tipo de ciber ataque. Com isto, podemos concluir que, obtivemos grandes melhorias para a métrica *f1-score*, que tomamos como a métrica principal para a tomada de decisões. Para a métrica do *ROC-Curve* recolhemos informação para cada classe individualmente, com isto, podemos identificar, para o conjunto de dados do *dataset CIC-17*, quase todas as classes conseguiram alcançar ao final das simulações de forma similar as classes presentes, alcançando em média 99%, pode-se notar que as classes com menor representação (classes: 2 e 3) têm uma evolução mais lenta se comparadas com as outras. Para o *dataset TON-IoT*, que partimos da divisão de oito classes principais, nota-se que, as classes estão distribuídas de forma desproporcionada, levando classes como, *Backdoor* (classe 6) e *Injection* (classe 7), a obterem uma lenta evolução no decorrer dos *rounds*, visto que estão presentes em aproximadamente 5% das amostras totais.

Por fim, pode-se concluir que maior barreira enfrentada, que já foi abordada durante a dissertação, é a necessidade de grande poder computacional, tomando grandes períodos para a execução das simulações. Finalmente, podemos concluir que o âmbito planejado foi executado, bem como a utilização de técnicas e *datasets* diferentes, que puderam enriquecer os testes durante as simulações. Podemos perceber que a aplicação do FL aparece como uma importante técnica, a qual favorece a troca de informação, fomentando a não exposição de dados dos clientes.

Para trabalhos futuros sugerimos replicar todas as simulações feitas em cenário IID, para cenários Non-IID, onde buscaria explorar mais a fundo o impacto do desbalanceamento entre as classes, porém, entre as amostras recebidas para cada cliente. Vale relembrar que optamos por classes com representatividade maior que 1% durante as nossas análises, mas poderia ser avaliado o impacto nos resultados ao utilizarmos todos *datasets*.

This page is intentionally left blank.

Bibliografia

[1] Dinh C Nguyen, Ming Ding, Pubudu N Pathirana, Aruna Seneviratne, Jun Li, and H Vincent Poor. Federated learning for internet of things: A comprehensive survey. *arXiv preprint arXiv:2104.07914*, 2021.

[2] General Data Protection Regulation. General data protection regulation (gdpr). *Intersoft Consulting, Accessed in October*, 24(1), 2018.

[3] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[4] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[5] Quoc-Viet Pham, Kapal Dev, Praveen Kumar Reddy Maddikunta, Thippa Reddy Gadekallu, Thien Huynh-The, et al. Fusion of federated learning and industrial internet of things: A survey. *arXiv preprint arXiv:2101.00798*, 2021.

[6] Mamoun Alazab, Swarna Priya RM, M Parimala, Praveen Kumar Reddy Maddikunta, Thippa Reddy Gadekallu, and Quoc-Viet Pham. Federated learning for cybersecurity: Concepts, challenges, and future directions. *IEEE Transactions on Industrial Informatics*, 18(5):3501–3509, 2021.

[7] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[8] Ivo Frazão, Pedro Henriques Abreu, Tiago Cruz, Hélder Araújo, and Paulo Simões. Denial of service attacks: Detecting the frailties of machine learning algorithms in the classification process. In *International Conference on Critical Information Infrastructures Security*, pages 230–235. Springer, 2018.

[9] Nour Moustafa and Jill Slay. Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In *2015 military communications and information systems conference (MilCIS)*, pages 1–6. IEEE, 2015.

[10] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. *ICISSp*, 1: 108–116, 2018.

[11] Nour Moustafa. New generations of internet of things datasets for cybersecurity applications based machine learning: Ton_iot datasets. In *Proceedings of the eResearch Australasia Conference, Brisbane, Australia*, pages 21–25, 2019.

[12] Maede Zolanvari, Marcio A Teixeira, Lav Gupta, Khaled M Khan, and Raj Jain. Machine learning-based network vulnerability analysis of industrial internet of things. *IEEE Internet of Things Journal*, 6(4):6822–6834, 2019.

[13] Maede Zolanvari. Wustl-iiot-2021, 2021. URL `https://dx.doi.org/10.21227/yftq-n229`.

[14] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37–37, 1996.

[15] Patricia A Patrician. Multiple imputation for missing data. *Research in nursing & health*, 25(1):76–84, 2002.

[16] Ariruna Dasgupta and Asoke Nath. Classification of machine learning algorithms. *International Journal of Innovative Research in Advanced Engineering (IJIRAE)*, 3 (3):6–11, 2016.

[17] Shuang Dai and Fanlin Meng. Addressing modern and practical challenges in machine learning: A survey of online federated and transfer learning. *arXiv preprint arXiv:2202.03070*, 2022.

[18] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168, 2006.

[19] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[20] Stephan Dreiseitl and Lucila Ohno-Machado. Logistic regression and artificial neural network classification models: a methodology review. *Journal of biomedical informatics*, 35(5-6):352–359, 2002.

[21] BV Kiranmayee, Chalumuru Suresh, and S SreeRakshak. Classification of the suicide-related text data using passive aggressive classifier. In *Sentimental Analysis and Deep Learning*, pages 439–449. Springer, 2022.

[22] Zhuo Chen, Na Lv, Pengfei Liu, Yu Fang, Kun Chen, and Wu Pan. Intrusion detection for wireless edge networks based on federated learning. *IEEE Access*, 8:217463–217472, 2020.

[23] Yuwei Sun, Hideya Ochiai, and Hiroshi Esaki. Intrusion detection with segmented federated learning for large-scale multiple lans. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

[24] Ruei-Hau Hsu, Yi-Cheng Wang, Chun-I Fan, Bo Sun, Tao Ban, Takeshi Takahashi, Ting-Wei Wu, and Shang-Wei Kao. A privacy-preserving federated learning system for android malware detection based on edge computing. In *2020 15th Asia Joint Conference on Information Security (AsiaJCIS)*, pages 128–136. IEEE, 2020.

[25] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[26] Alaa Alhowaide, Izzat Alsmadi, and Jian Tang. Pca, random-forest and pearson correlation for dimensionality reduction in iot ids. In *2020 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, pages 1–6, 2020. doi: 10.1109/IEMTRONICS51293.2020.9216388.

[27] A. Malhi and R.X. Gao. Pca-based feature selection scheme for machine defect classification. *IEEE Transactions on Instrumentation and Measurement*, 53(6):1517–1525, 2004. doi: 10.1109/TIM.2004.834070.

[28] Somayya Madakam, R Ramaswamy, and Siddharth Tripathi. Internet of things (iot): A literature review. *Journal of Computer and Communications*, 3:164–173, 04 2015. doi: 10.4236/jcc.2015.35021.

[29] Enrique Mármol Campos, Pablo Fernández Saura, Aurora González-Vidal, José L Hernández-Ramos, Jorge Bernal Bernabe, Gianmarco Baldini, and Antonio Skarmeta. Evaluating federated learning for intrusion detection in internet of things: Review and challenges. *Computer Networks*, page 108661, 2021.

[30] Samuel Tweneboah-Koduah, Knud Erik Skouby, and Reza Tadayoni. Cyber security threats to iot applications and service domains. *Wireless Personal Communications*, 95(1):169–185, 2017.

[31] Georgi Tsochev, Roumen Trifonov, Ognian Nakov, Slavcho Manolov, and Galya Pavlova. Cyber security: Threats and challenges. In *2020 International Conference Automatics and Informatics (ICAI)*, pages 1–6, 2020. doi: 10.1109/ICAI50593.2020.9311369.

[32] Nour Moustafa. A new distributed architecture for evaluating ai-based security systems at the edge: Network ton_iot datasets. *Sustainable Cities and Society*, 72: 102994, 2021.

[33] Jakub Konečnỳ, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[34] H Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. Federated learning of deep networks using model averaging. *arXiv preprint arXiv:1602.05629*, 2016.

[35] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[36] Shaoqi Chen, Dongyu Xue, Guohui Chuai, Qiang Yang, and Qi Liu. Fl-qsar: a federated learning-based qsar prototype for collaborative drug discovery. *Bioinformatics*, 36(22-23):5492–5498, 2021.

[37] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2021.106775. URL `https://www.sciencedirect.com/science/article/pii/S0950705121000381`.

[38] Ce Ju, Dashan Gao, Ravikiran Mane, Ben Tan, Yang Liu, and Cuntai Guan. Federated transfer learning for eeg signal classification. *IEEE Engineering in Medicine and Biology Society (EMBC)*, 2020.

[39] Kunal Chandiramani, Dhruv Garg, and N Maheswari. Performance analysis of distributed and federated learning models on private data. *Procedia Computer Science*, 165:349–355, 2019.

[40] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[41] Qiong Wu, Kaiwen He, and Xu Chen. Personalized federated learning for intelligent iot applications: A cloud-edge based framework. *IEEE Open Journal of the Computer Society*, 1:35–44, 2020.

[42] George Vavoulas, Charikleia Chatzaki, Thodoris Malliotakis, Matthew Pediaditis, and Manolis Tsiknakis. The mobiact dataset: Recognition of activities of daily living using smartphones. In *ICT4AgeingWell*, 2016.

[43] Wenyuan Xu, Weiwei Fang, Yi Ding, Meixia Zou, and Naixue Xiong. Accelerating federated learning for iot in big data analytics with pruning, quantization and selective updating. *IEEE Access*, 9:38457–38466, 2021.

[44] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[45] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[46] Georgios Damaskinos, Rachid Guerraoui, Anne-Marie Kermarrec, Vlad Nitu, Rhicheek Patra, and Francois Taiani. Fleet: Online federated learning via staleness awareness and performance prediction. In *Proceedings of the 21st International Middleware Conference*, pages 163–177, 2020.

[47] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-100 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[48] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre Van Schaik. Emnist: Extending mnist to handwritten letters. In *2017 international joint conference on neural networks (IJCNN)*, pages 2921–2926. IEEE, 2017.

[49] Mohamad Mestoukirdi, Matteo Zecchin, David Gesbert, Qianrui Li, and Nicolas Gresset. User-centric federated learning. *arXiv preprint arXiv:2110.09869*, 2021.

[50] Nickolaos Koroniotis, Nour Moustafa, Elena Sitnikova, and Benjamin Turnbull. Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset. *Future Generation Computer Systems*, 100:779–796, 2019.

[51] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17(3):12–22, 2018.

[52] Alejandro Guerra-Manzanares, Jorge Medina-Galindo, Hayretdin Bahsi, and Sven Nõmm. Medbiot: Generation of an iot botnet dataset in a medium-sized iot network. In *ICISSP*, pages 207–218, 2020.

[53] Imtiaz Ullah and Qusay Mahmoud. A scheme for generating a dataset for anomalous activity detection in iot networks. pages 508–520, 05 2020. doi: 10.1007/978-3-030-47358-7_52.

[54] Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 15–24. IEEE, 2020.

[55] Jianmo Ni, Larry Muhlstein, and Julian McAuley. Modeling heart rate and activity data for personalized fitness recommendation. In *The World Wide Web Conference*, pages 1343–1353, 2019.

[56] Liang Zhao, Jiaming Li, Qi Li, and Fangyu Li. A federated learning framework for detecting false data injection attacks in solar farms. *IEEE Transactions on Power Electronics*, 37(3):2496–2501, 2021.

[57] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[58] Yang Qin, Hiroki Matsutani, and Masaaki Kondo. A selective model aggregation approach in federated learning for online anomaly detection. In *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pages 684–691. IEEE, 2020.

[59] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[60] Viraaji Mothukuri, Prachi Khare, Reza M Parizi, Seyedamin Pouriyeh, Ali Dehghantanha, and Gautam Srivastava. Federated-learning-based anomaly detection for iot security attacks. *IEEE Internet of Things Journal*, 9(4):2545–2554, 2021.

[61] Mohamed Amine Ferrag, Othmane Friha, Djallel Hamouda, Leandros Maglaras, and Helge Janicke. Edge-iiotset: A new comprehensive realistic cyber security dataset of iot and iiot applications for centralized and federated learning. *IEEE Access*, 10: 40281–40306, 2022.

[62] Jevgenijus Toldinas, Algimantas Venčkauskas, Robertas Damaševičius, Šarūnas Grigaliūnas, Nerijus Morkevičius, and Edgaras Baranauskas. A novel approach for network intrusion detection using multistage deep learning image recognition. *Electronics*, 10(15):1854, 2021.